View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Glasgow Theses Service

Olaosebikan, Sofiat (2020) *The Student-Project Allocation Problem: structure and algorithms.* PhD thesis.

http://theses.gla.ac.uk/81514/

Copyright and moral rights for this work are retained by the author

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge

This work cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given

Enlighten: Theses
https://theses.gla.ac.uk/
research-enlighten@glasgow.ac.uk

The Student-Project Allocation Problem: Structure and Algorithms

Sofiat Olaosebikan

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

July 9, 2020

© Sofiat Olaosebikan

Abstract

In this thesis we study the *Student-Project Allocation problem* (SPA), which is a matching problem based on the allocation of students to projects and lecturers. Students have preferences over projects, where each project is offered by one lecturer; whilst lecturers have preferences over students, or over the projects that they offer. We seek stable matchings of students to projects, which guarantee that no student and lecturer have an incentive to deviate from the matching by forming a private arrangement involving some project. We present new structural and algorithmic results for four problems related to SPA.

We begin by characterising the stable matchings in an instance of the *Student-Project Allocation problem with Lecturer preferences over Students* (SPA-S) where the preferences are strictly ordered, in the special case that for each student in the instance, all of the projects in her preference list are offered by different lecturers. We achieve this characterisation by showing that, under this restriction, the set of stable matchings in an instance of SPA-S is a distributive lattice with respect to a natural dominance relation.

Next, we study a variant of SPA-S where the preferences may involve ties — the *Student-Project Allocation problem with Lecturer preferences over Students with Ties* (SPA-ST). The presence of ties in the preference lists gives rise to three different concepts of stability, namely, *weak stability, strong stability*, and *super-stability*. We investigate stable matchings under the super-stability (respectively strong stability) concept. We present the first polynomial-time algorithm to find a super-stable (respectively strongly stable) matching or to report that no such matching exists, given an instance of SPA-ST. We also prove some structural results concerning the set of super-stable (respectively strongly stable) matchings in a given instance of SPA-ST. Further, we present results obtained from an empirical evaluation of our algorithms based on randomly-generated SPA-ST instances.

Moving away from variants of SPA with lecturer preferences over students, we study the *Student-Project Allocation problem with lecturer preferences over Projects* (SPA-P). In this context it is known that stable matchings can have different sizes and the problem of finding a maximum size stable matching, denoted MAX-SPA-P, is NP-hard. There are two known approximation algorithms for MAX-SPA-P, with performance guarantees 2 and $\frac{3}{2}$.

We show that MAX-SPA-P is polynomial-time solvable if there is only one lecturer involved, and NP-hard to approximate within some constant $c > 1$ if there are two lecturers involved. We also show that this problem remains NP-hard if each preference list is of length at most 3, with an arbitrary number of lecturers. We then describe an Integer Programming (IP) model to enable MAX-SPA-P to be solved optimally in the general case. Following this, we present results arising from an empirical evaluation that investigates how the solutions produced by the approximation algorithms compare to optimal solutions obtained from the IP model, with respect to the size of the stable matchings constructed, on instances that are both randomly-generated and derived from real datasets.

Acknowledgements

AlhamduliLlahi Robil ‘Alamin!

Some appreciations are difficult to express not because I cannot find the right words but because all of the “available words” are insufficient to express them. However, these words are all I have.

Firstly, I would like to thank Professor David Manlove for the outstanding supervision. David’s expertise in the area of matching problems transformed my research experience, which has shaped so much of my career. His passion for the research area and his ability to quickly understand mathematical proofs gave me the confidence to explore so many difficult problems, some of which are included in this thesis. I am very fortunate to have had David as my first supervisor and I could not have wished for someone better. Thank you David, for believing in me even when I doubted myself, for listening to all my ideas (incuding the stupid ones), for thoroughly reading all my drafts, and for your kindness and generosity. *There is a saying among David’s research students that when we grow up, we would like to become “DM 2.0”, but then that would mean we will be better than David. However, since no one can be as amazing as “DM 1.0”, we will settle for “DM 0.X”, where X is a constant* > 1.

I would also like to thank my second supervisor, Kitty Meeks: for reading drafts of my conference papers, some of which have now been published; and for her detailed comments and suggestions during my annual progression viva, all of which have improved the quality of this thesis. I am particularly grateful to Kitty for observing that an error I initially flagged in Chapter 3 of this thesis was flawed, and that the structural results actually holds.

Others who have reviewed a part of this thesis during my annual progression viva include Alice Miller and Patrick Prosser. I am grateful to you both for reading my reports, and for the helpful suggestions. Further, I would like to convey my sincere gratitude to: Adam Kunysz, for valuable discussions concerning a part of the work in Chapter 5; Bettina Klaus, for carefully observing that a claim I made in Chapter 3 of this thesis is not true in general; and Elias Koutsoupias, for suggesting to consider the restriction of SPA-P in which the number of lecturers is a constant. Thanks to the anonymous reviewers of submitted versions of parts

of the work contained in this thesis for their valuable comments. Also, thanks to Matthew Johnson and Jessica Enright for an enjoyable PhD defence.

My research was supported by a College of Science and Engineering Scholarship, University of Glasgow. I cannot express how grateful and fortunate I am to have received this Scholarship. Considering my financial background, it would have been impossible for me to do a PhD without the support. To the College and the University – thank you for investing in me, I hope I made you proud!

I would also like to express my gratitude to colleagues and office mates who have supported me throughout the course of my research: to Ben, for all the free hugs, it got me through the tough times; to Craig, for teaching me how to effectively use the linux command line and for helping to set up most of my experiments on eglinton; to "super-duper" Frances, for the numerous "free" therapy sessions, for the PhD survival kits, and for proof-reading parts of this thesis; to Michael, for always letting me distract him in the office whenever my code breaks or whenever a theoretical proof goes through; to Ivaylo, for always wearing a smiling face; and to Will, for regularly checking up on me and for proof-reading parts of this thesis. You all made my PhD experience a memorable one and I am grateful!

Outside work, I am grateful to friends who became family. It is almost impossible to name everyone, so I will approach this differently. During my PhD: did I have a sleep over at your flat? did you cook a meal for me? did you take care of me like your own? If you answered yes to any of these questions then you played a significant role during this journey. Thank you!

Lastly and most importantly, I am grateful to my parents and siblings, for their support, understanding, and unending love. Producing this thesis was very difficult and time-consuming. I could remember informing my family when I started writing to not worry about me if they do not hear from me in weeks. I lost count of all their calls that I could not attend to, yet they still love me. Mum, Dad – thank you for giving me life, I am convinced that I can never fully repay you both for all your sacrifice, but Bi'idhniLlah, I will spend the rest of my time on earth trying.

Declaration

This thesis is submitted in accordance with the rules for the degree of Doctor of Philosophy at the University of Glasgow. None of the material contained herein has been submitted for any other degree.

Some parts of Chapter 6 of this thesis are the product of collaboration with my PhD supervisor, David Manlove and a former Level 4 undergraduate student, Duncan Milne. In particular, the proof of Theorems 6.3.5 and 6.4.1 are based on ideas due to David Manlove. Also, the Integer Programming (IP) model described in Section 6.5 originated from David Manlove. Further, the Java implementation of the approximation algorithms and the IP-based algorithm that led to the experimental results in Section 6.6 was done by Duncan Milne. All other results therein are claimed as original.

Publications

The following publications contain some of the results presented in this thesis.

1. Sofiat Olaosebikan and David Manlove. An algorithm for strong stability in the Student-Project Allocation problem with Ties. Accepted for MATCH-UP 2019: International Workshop on Matching Under Preferences (no proceedings). In proceedings of CALDAM 2020: International Conference on Algorithms and Discrete Applied Mathematics, volume 12016 of Lecture Notes in Computer Science, pages 384 - 399, Springer, 2020. (This paper is based on some of the results in Chapter 5.)

2. Sofiat Olaosebikan and David Manlove. Super-stability in the Student-Project Allocation Problem with Ties. In Proceedings of COCOA 2018: International Conference on Combinatorial Optimization and Applications, volume 11346 of Lecture Notes in Computer Science, pages 357 - 371, Springer, 2018. (This paper is based on some of the results in Chapter 4.)

3. David Manlove, Duncan Milne and Sofiat Olaosebikan. An Integer Programming Approach to the Student-Project Allocation Problem with Preferences over Projects. In proceedings of ISCO 2018: International Symposium on Combinatorial Optimization, volume 10856 of Lecture Notes in Computer Science, pages 313 - 325, Springer, 2018. (This paper is based on some of the results in Chapter 6.)

4. Sofiat Olaosebikan and David Manlove. Super-stability in the Student-Project Allocation Problem with Ties. Accepted for publication in a special issue of Journal of

Combinatorial Optimization featuring selected papers from COCOA 2018. (This paper is based on Chapter 4.)

5. David Manlove, Duncan Milne and Sofiat Olaosebikan. Student-Project Allocation with Preferences over Projects: Algorithmic and Experimental Results. Invited for submission to a special issue of Discrete Applied Mathematics featuring selected papers from ISCO 2018. (This paper is based on Chapter 6.)

Table of Contents

Chapter 1

Introduction

1.1 Matching problems

In this thesis we look at matching problems with preferences which commonly arise when we need to allocate a set A of agents to another set B of agents/resources (e.g., when assigning pupils to schools [9, 10, 23, 78], allocating junior doctors to hospitals [30, 52, 63, 105, 106], allocating teachers to regions [29, 32, 117], assigning lawyers to legal internship positions [35], or allocating Israeli youths to gap-year programs [42]). A typical trend is that the agents involved have ordinal preferences (i.e., the notion of first choice, second choice, and so on) over the possible outcomes. Also each agent and resource in one set has a specified *capacity*, which is the maximum number of agents/resources from the other set that they can accommodate. The goal is to find a *matching*, i.e., an allocation of the agents in A to the agents/resources in B, which is optimal in some sense according to the stated preferences, and which does not violate the agents' capacities.

This class of problems was first studied in 1962 by Gale and Shapley [37]. In their seminal paper, they described the *College Admissions problem* which involves applicants (each of whom seeks to match with a single college) and colleges (that seek to match with one or more applicants), where each applicant (respectively college) provides a strictly-ordered preference list over the colleges (respectively applicants) that they find acceptable. The goal here is to find a matching of applicants to colleges, such that no applicant and college that are not matched together would prefer each other to their partner/s (if any). Gale and Shapley [37] also described a special case of the *College Admissions problem* — the *Stable Marriage problem*, which involves a one-to-one pairing of n men to n women in such a way that no man and woman who are not assigned to one another would prefer each other to their actual partners. The main result of their paper is a linear-time algorithm for solving these problems, which is commonly referred to as the Gale-Shapley algorithm among computer scientists (or the Deferred Acceptance mechanism among economists).

The practical applications of matching problems can be seen in various centralised matching schemes around the world, including the National Resident Matching Program (NRMP) in the USA [1], the Canadian Resident Matching Service (CaRMS) [2], the Japan Residency Matching Program (JRMP) [3], and until 2012, the Scottish Foundation Allocation Scheme (SFAS) [52]. These centralised matching schemes deal with the allocation of graduating medical students to hospital posts for their medical training programs, in their respective countries. Other matching schemes can be found in the context of allocating donor kidneys to transplant patients [88], where a patient who needs a kidney transplant can obtain a compatible donor by swapping her own willing but incompatible donor with another patient in a similar situation. Kidney exchange programmes have already been established in several countries, for example in the USA [4], the Netherlands [68, 72], and the UK [99]. For a comprehensive survey of further applications, we refer the interested reader to [24].

Why we need Algorithms

Due to the large number of agents that typically appear in practical applications, it is usually not feasible to find the desired solution by hand. For instance, the Chinese Higher Education matching scheme involves over 10 million students annually [122], and the NRMP currently involves over $40,000$ junior doctors each year [96]. It is clearly obvious why we need to employ algorithms to compute these allocations. In addition, the allocation that these agents receive may have a significant impact on their quality of life; thus it is important that the algorithms produce matchings that are optimal in a precise sense according to the agents' preferences.

Now, suppose we are given an instance of a matching problem, we can design an algorithm that will search through all the possible solutions that the problem admits and output the *best* solution which suits our goal. However, depending on the problem size, the time it will take to obtain a solution via this approach could be longer than the age of the universe. To see this, consider Algorithm A and Algorithm B that solves a given instance I of a problem, with time complexities $O(n^2)$ and $O(n!)$ respectively, where n is the problem size. Suppose a computer performs 10^9 operations per second. If $n = 10$, it will take Algorithms A and B 10^{-7} second and 0.004 second, respectively, to terminate. However, if $n = 50$, Algorithm A will terminate in 2.5×10^{-6} second, while Algorithm B will take longer than the age of the universe to terminate.

Algorithm B clearly suffers from a "combinatorial explosion" meaning that the running time grows massively relative to a small increase in the problem size. As a consequence, the algorithm is not practical, even for the problem size of the SFAS matching scheme that ran until 2012 which involved about 750 applicants. Thus, it is of great practical interest that the algorithms we design terminate in a reasonable amount of time (in seconds or minutes rather

than hours or days).

As a consequence, our focus in this thesis will be on the design of efficient (polynomial-time) algorithms for a class of matching problems that underpins these centralised matching schemes, and where such algorithms does not exist, we will provide results to prove this and we will describe other solution techniques.

1.2 Classification of matching problems

The underlying mathematical structure that is widely used to model matching problems is a graph. The vertices represent the agents and resources, and there is an edge between two vertices v_i, v_j if either the resource corresponding to v_j appears on the preference list of the agent corresponding to v_i, or the agents corresponding to v_i and v_j both appear on each other's preference lists. Matching problems in the literature [44, 83] are commonly classified according to whether they involve (i) two disjoint sets of agents (bipartite), (ii) a set of agents and a set of resources (bipartite), or (iii) one set of agents (non-bipartite). In addition, the classification also considers the existence (or otherwise) of preference lists. We discuss each category further in what follows (see Figure 1.1 for an illustration).

(i) *Bipartite matching problems with two-sided preferences.* In this category, there are two disjoint sets of agents involved, and agents in one set rank a subset of agents in the other set (and vice-versa). The fundamental problem here is the *Stable Marriage problem* [44], which has applications in the placement of pupils to schools [9, 10, 11], allocation of junior doctors to hospitals [105] and allocation of students to projects in a university department [13, 67].

(ii) *Bipartite matching problems with one-sided preferences.* Here, there are two disjoint sets involved – a set of agents and a set of resources, and only the agents rank a subset of the resources to form their preference list. The fundamental problem in this category is the *House Allocation problem* [12, 123], which has applications to campus housing allocation. Application of problems in this category also extends to the assignment of reviewers to conference papers [40], and variants of allocating students to projects [31, 76].

(iii) *Non-bipartite matching problems with preferences.* There is only one set of agents involved in this category, and each agent ranks a subset of the other agents in order of preference. The fundamental problem in this category is the *Stable Roommates problem* [44, 50], which has applications to P2P networking [79], and pairing players in a chess tournament [74]. Another example application appears in the context of the kidney exchange problem [88, 109].

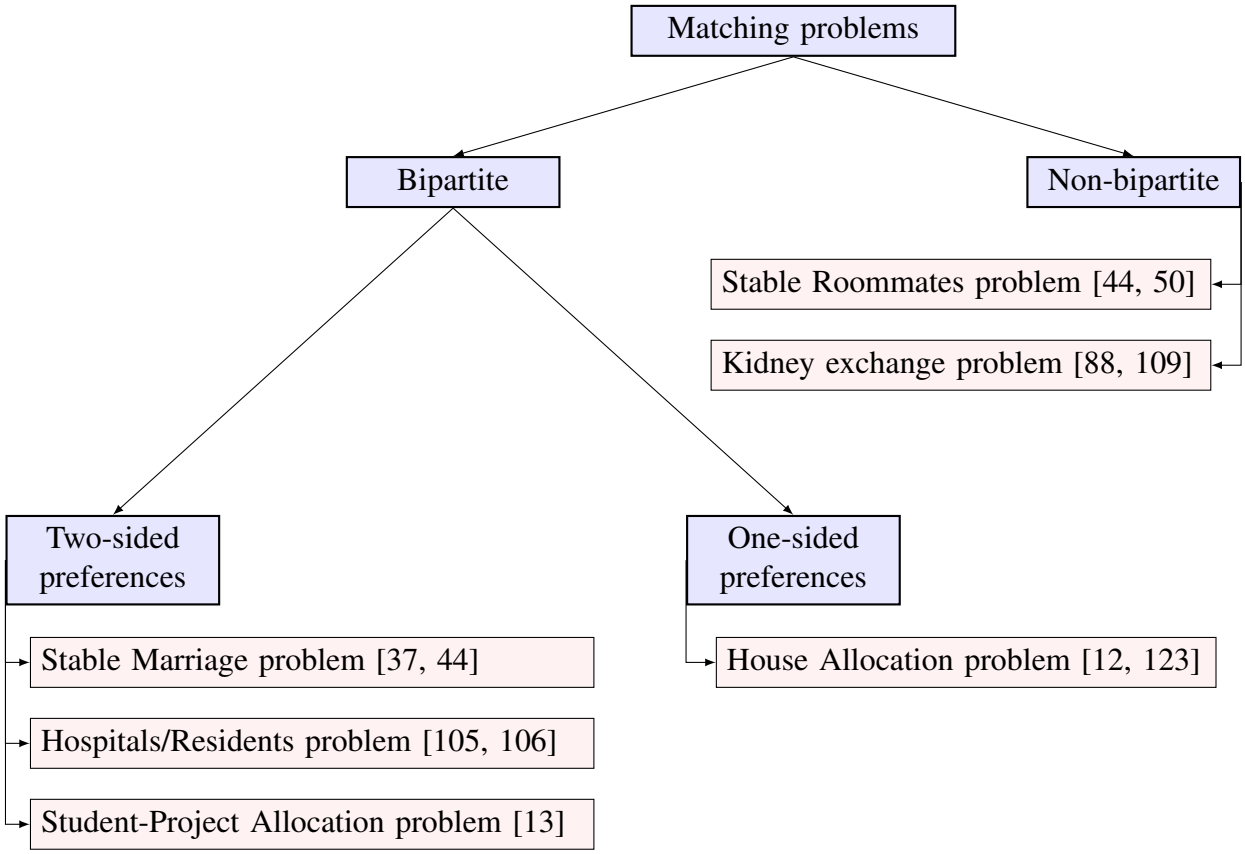


Figure 1.1: Classification of matching problems.

1.3 Two-sided preferences and stability

In this thesis our focus will be on matching problems that fall under category (i) above. We recall that the fundamental problem in this category is the *Stable Marriage problem* (SM) [37, 44], which we define formally in Section 2.1. Informally, an instance of SM involves a set of men and a set of women, each of whom has a strictly-ordered preference list over each member of the opposite set. A matching under SM is a one-one pairing of the men to the women. The *Hospitals/Residents problem* (HR) [105, 106] involves the assignment of residents to hopitals, and it is commonly referred to as a many-one generalisation of SM because many residents may be assigned to one hospital. Other generalisations of HR in this category are the *Workers/Firms problem* (WF) [83] and the *Student-Project Allocation problem* (SPA) [13, 83].

Our contributions in this thesis will focus on problems that arise within the context of SPA, which we define formally in Section 2.3. Informally, an instance of SPA involves three sets of entities: students, projects and lecturers. Each project is proposed by one lecturer and each student has preferences over a subset of the available projects that she finds acceptable. Further, each lecturer may have preferences over the students that find her projects acceptable and/or the projects that she offers. In addition, each project and lecturer has a capacity.

Variants of SPA in the literature involve: (i) lecturer preferences over (a) students (SPA-S) [13, 83], (b) projects (SPA-P) [61, 86, 87], and (c) (student, project) pairs (SPA-(S,P)) [14]; and (ii) no lecturer preference at all [76]. Our main contributions in this thesis will be on problems belonging to category (a) and (b). Applications of SPA exist at the universities of Glasgow [76], Southampton [17, 48], Southern Denmark [31], Singapore [116] and York [36, 67, 118].

For each of these bipartite matching problems with two-sided preferences (i.e., SM, HR, WF, SPA-S, SPA-P, and SPA-(S,P)), the goal is to find a *stable matching*. We recall that a *matching* is an assignment of agents in one set to agents from the other set, which respects the preferences and capacities of the agents involved. Informally, a *stable matching* ensures that no two agents who are not matched together would rather be assigned to each other than remain with their current assignees. If such a pair were to exist, then the agents involved would have an incentive to form a private arrangement outside of the matching; thus, undermining its integrity. The importance of *stability* as a solution concept in this context was highlighted in [105, 107, 108, 111].

1.4 Hard matching problems

Many of the practically interesting matching problems tend to have more than one feasible solution. For example, let I be a variant of HR where the preference lists of residents and hospitals can include ties (HRT) (see Section 2.2.4 for a formal definition of this problem variant). Stable matchings in I can have different sizes, with respect to the total number of residents that are assigned to a hospital. However, in practical applications, we are mostly interested in picking the *best* among these set of solutions. For instance, a notion of best solution in I could be to find a *maximum size stable matching*, i.e., a stable matching that assigns as many residents to hospitals as possible. The class of problems that involves finding the "best" solution among the finite set of feasible solutions that the problem admits are known as *optimisation problems*.

For many optimisation problems, the number of feasible solutions admitted grows very fast with respect to even a small increase in the problem size. Indeed, it turns out that these problems are NP-hard, so assuming $\mathsf{P} \neq \mathsf{NP}$, we cannot construct a polynomial-time algorithm to find the best solution for every instance of the problem. For an in-depth explanation of complexity classes, we refer the interested reader to [39]. We briefly discuss some common techniques of coping with these hard problems in what follows.

1.5 Coping with hard problems

The following are the most common techniques that have been explored for coping with NP-hard matching problems.

1. **Approximation Algorithms:** Given an instance I of a problem that is NP-hard, one could settle for a polynomial-time algorithm (with respect to the input size of I) that will output a feasible solution whose measure is guaranteed to not be more than a certain *ratio* below (respectively above) that of an optimal solution depending on whether I is a maximisation (respectively minimisation) problem. Such algorithms are referred to as *approximation algorithms*. Let Π be a maximisation (respectively minimisation) problem, let I be an instance of Π, and let T be an approximation algorithm for Π. The *approximation ratio* of T is $\max\{\frac{OPT(I)}{T(I)}\}$ (respectively $\max\{\frac{T(I)}{OPT(I)}\}$) over all instances I of Π, where $OPT(I)$ and $T(I)$ are the measures of an optimal solution and the algorithm's solution, respectively. As an illustration, the best known approximation algorithm for finding a maximum size stable matching given an instance of HRT has an approximation ratio of $\frac{3}{2}$ [71, 93, 100]. This implies that the algorithm will output a stable matching whose size is at least two-thirds of that of the optimal solution.

2. **Fixed Parameter Tractable (FPT) Algorithms:** In contrast to approximation algorithms, FPT algorithms are "exact algorithms", except that they are exponential with respect to a fixed parameter value while polynomial with respect to the input size. Let I be a decision problem with input size n and a parameter k, an FPT algorithm for I is an algorithm with time complexity $f(k) \cdot g(n)$, for some computable functions f and g, where f depends solely on k (for instance, 2^k or 1.1^k) and g is a polynomial function of n only. The fundamental idea behind FPT algorithms is to explore how various problem-specific parameters affect the computational complexity of the problem. For example, given an instance I of HRT, one such parameter could be the total length of ties in I. On one hand, if the parameter is small in practice, an FPT algorithm can be efficient. The drawback of this approach is that a large parameter could lead to an impractical algorithm. Marx and Schlotter [89] investigated the parameterised complexity of NP-hard stable matching problems.

3. **Linear Programming (LP):** LP is a widely-studied technique used to find optimal solutions to a range of optimisation problems, including matching problems [65, 120]. Given an instance I of a problem, it involves constructing a set of linear constraints over a set of real-valued variables and defining a specific objective function in such a way that an optimal solution in the constructed model corresponds to an optimal

(fractional) solution of I. We remark that such a formulation leads to an efficient algorithm for I given the polynomial-time solvability of LP [64, 69].

4. **Integer Programming (IP):** To solve some matching problems, restrictions need to be imposed on the domain of the variables involved in an LP model. One such restriction is that all the variables are integer-valued, thus forming an IP model. Despite the fact that, in general, solving an IP problem is NP-hard [39], commercial optimisation solvers, for example Gurobi [5], GLPK [6] and CPLEX [7] allow for these models to be solved in a reasonable amount of time, with respect to a moderately-sized input. As a result, there has been a lot of research into the formulation of IP models for matching problems [15, 17, 26, 31, 77, 92].

1.6 Contribution and thesis outline

Our contribution in this thesis is to present new structural and algorithmic results for four different problems within the context of SPA (i.e., the *Student-Project Allocation problem*). In Chapter 2, we give a literature survey on bipartite matching problems with two-sided preferences. We focus our attention on problems and known results that will be relevant in subsequent chapters of this thesis. Building on this, we present our own contributions in Chapters 3 - 6, which we outline in what follows.

In Chapter 3 we study SPA-S, a variant of SPA where lecturers have preferences over students, and the preference lists of students and lecturers are strictly ordered (we formally define this variant in Section 2.3.2). We recall that the solution concept we seek under this variant is that of a stable matching. Abraham *et al.* [13] described two linear-time algorithms to find a stable matching given an instance of SPA-S. The first algorithm finds the *student-optimal* stable matching, in the sense that each assigned student is allocated to the best project that she could obtain in any stable matching; while the second algorithm outputs the *lecturer-optimal* stable matching, in the sense that each assigned lecturer is allocated the best set of students that she could obtain in any stable matching (we discuss this further in Section 2.3.2). As it turns out, an arbitrary instance of SPA-S may admit many other stable matchings in addition to the student-optimal and lecturer-optimal stable matchings.

Our contribution in Chapter 3 is to characterise the stable matchings given an instance of SPA-S, in the special case that for each student in the instance, all of the projects in her preference list are offered by different lecturers. We achieve this characterisation by showing that, under this restriction, the set of stable matchings in an instance of SPA-S is a distributive lattice under a natural dominance relation, with the student-optimal and lecturer-optimal stable matchings representing the maximum and minimum elements of the lattice respectively.

Moving away from the notion of strictly-ordered preference lists, in Chapter 4 we consider SPA-ST, a variant of SPA-S where the preference lists of students and lecturers may admit indifference in the form of ties (we formally define this variant in Section 2.3.3). As a result of the presence of ties in the preference lists, three different forms of stability arise, namely *weak stability, strong stability*, and *super-stability*. These concepts were first defined and studied by Irving [51] in the context of SMT, a variant of SM with ties, and subsequently extended to HRT [57, 59]. We remark that HRT is the special case of SPA-ST in which each lecturer offers only one project, and the capacity of each project is the same as the capacity of the lecturer offering the project; in turn SMT is a one-to-one restriction of HRT. Under the weak stability concept, stable matchings in an instance of SPA-ST can have different sizes and the problem of finding a maximum size weakly stable matching is NP-hard [60, 84].

Our contribution in Chapter 4 is to present theoretical and experimental results for SPA-ST under super-stability. We present the first polynomial-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of SPA-ST. We also prove the correctness of our algorithm and we show that the algorithm can be implemented to run in $O(L)$ time, where L is the total length of the preference lists. We give some structural properties satisfied by the set of super-stable matchings in an instance of SPA-ST. For experimental purposes, we present an IP model for SPA-ST under super-stability. Further, we present results from an empirical evaluation of an implementation of our linear-time algorithm. For the experiments, we investigate how the nature of the preference lists affects the likelihood of a super-stable matching existing, with respect to randomly-generated SPA-ST instances. Our main finding from the empirical evaluation is that super-stable matchings are very elusive with ties in the students' and lecturers' preference lists. However, if the preference lists of the students are strictly ordered and only the lecturers express ties in their preference lists, the probability of a super-stable matching existing is significantly higher.

In Chapter 5 we present theoretical and experimental results for SPA-ST under strong stability. We describe the first polynomial-time algorithm to find a strongly stable matching or report that no such matching exists, given an instance of SPA-ST. We then move on to prove the correctness of our algorithm and we show that the algorithm can be implemented to run in $O(m^2)$ time, where m is the total length of the students' preference lists. In addition, we give some structural properties satisfied by the set of strongly stable matchings in an instance of SPA-ST. Similar to the super-stability setting, we also describe an IP model for SPA-ST under strong stability.

On the experimental side, we present results from an empirical evaluation based on an implementation of our algorithm, which investigates the proportion of randomly-generated instances that admit strongly stable matchings but no super-stable matchings. With respect to the datasets that we used for our super-stability experiments, we observed that the proportion of instances that admitted a strongly stable matching is exactly the same as those that

admitted a super-stable matching. However, when we varied the size of the instance between 10 and 50, there was a slight increase in the proportion of instances that admitted a strongly stable matching but no super-stable matching.

In Chapter 6 we study SPA-P, a variant of SPA where lecturers have preferences over their proposed projects (we formally define this variant in Section 2.3.4). In contrast to the other SPA variants considered in Chapters 4 and 5, stable matchings in this context can have different sizes. It is known that MAX-SPA-P, i.e., the problem of finding a stable matching that assigns as many students to projects as possible, is NP-hard, even if each project and lecturer has capacity 1, and all preference lists are of constant length [87]. There are two known approximation algorithms for MAX-SPA-P with approximation ratios 2 [87] and $\frac{3}{2}$ [61].

Our contribution in Chapter 6 is to present new theoretical and experimental results for SPA-P. First, we present a polynomial-time algorithm for MAX-SPA-P where the instance only involves one lecturer. In contrast to this, if there are two lecturers involved, we show that MAX-SPA-P remains NP-hard and is not approximable within some constant $c > 1$, unless P= NP. Further, we show that this problem remains NP-hard if each preference list is of length at most 3, with an arbitrary number of lecturers. We then move on to describe an IP model to enable MAX-SPA-P to be solved optimally, in the general case where there are no restrictions on the problem instance.

Finally, we present results from an empirical evaluation that investigates how the solutions produced by the approximation algorithms for SPA-P compare to the optimal solutions obtained from our IP model, with respect to the size of the stable matchings constructed, on instances that are both randomly generated and derived from real datasets. These real datasets are based on actual student preference data and manufactured lecturer preference data from previous runs of student-project allocation processes at the School of Computing Science, University of Glasgow. We also present results showing the time taken by the IP model to solve the problem instances optimally. Our main finding is that the $\frac{3}{2}$-approximation algorithm finds stable matchings that are very close to having maximum cardinality over the tested instances.

Chapter 2

Literature Review

In this chapter, we give formal definitions and a review of the structural and algorithmic results for bipartite matching problems with two-sided preferences. To be specific, we give a review of the literature on the *Stable Marriage problem* (SM), the *Hospitals/Residents problem* (HR) and the *Student-Project Allocation problem* (SPA) in Sections 2.1, 2.2 and 2.3 respectively.

2.1 The Stable Marriage problem: SM

2.1.1 Problem definition

An instance of the classical *Stable Marriage problem* (SM) involves two sets of agents – a set $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ of *men* and a set $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ of *women*, each of whom has a *strictly-ordered* preference list over *all* the members of the opposite set, thus giving rise to *complete* preference information. For each man m_i, we say that m_i *prefers* woman w_{j_1} to w_{j_2} if w_{j_1} precedes w_{j_2} on m_i's preference list. An analogous definition holds for each woman. In this context, a *matching* M is a one-one pairing between all the men and all the women. If a man m_i is assigned in M to a woman w_j, we say that m_i and w_j are *partners* in M (or M-partners) and write $m_i = M(w_j)$ and $w_j = M(m_i)$.

A pair (m_i, w_j) is said to *block* M, or be a *blocking pair* for M, if m_i prefers w_j to $M(m_i)$ and w_j prefers m_i to $M(w_j)$. Intuitively, m_i and w_j forms a blocking pair for M if they would rather be assigned with each other than remain with their M-partners. If a matching M admits at least one blocking pair, then we say that M is *unstable*; otherwise, if M admits no blocking pair then we say that M is *stable*. Given an arbitrary instance of SM, a typical goal is to find a stable matching in the instance, as this property guarantees that the matching cannot be undermined by any man and woman pair. We recall again that the importance of

stability as a solution concept in this setting was highlighted in [105, 107, 108, 111]. An example instance I of SM is shown in Figure 2.1, which involves 4 men and 4 women.

Men's preferences					Women's preferences				
m_1 :	w_2	w_4	w_1	w_3	w_1 :	m_2	m_1	m_4	m_3
m_2 :	w_3	w_1	w_4	w_2	w_2 :	m_4	m_3	m_1	m_2
m_3 :	w_2	w_3	w_1	w_4	w_3 :	m_1	m_4	m_3	m_2
m_4 :	w_4	w_1	w_3	w_2	w_4 :	m_2	m_1	m_4	m_3

Figure 2.1: An instance I of SM, due to Gusfield and Irving [44].

It can be verified that matchings $M_1 = \{(m_1, w_4), (m_2, w_3), (m_3, w_2), (m_4, w_1)\}$ and $M_2 = \{m_1, w_4), (m_2, w_1), (m_3, w_2), (m_4, w_3)\}$ are stable matchings in I. Clearly, each of M_1 and M_2 cannot be undermined since no man and woman have an incentive to break their assignment and become assigned to each other. On the other hand, matching $M_3 = \{(m_1, w_1)$, (m_2, w_3), (m_3, w_2), $(m_4, w_4)\}$ is unstable because it admits the blocking pair (m_1, w_4): m_1 prefers w_4 to his M_3-partner (i.e., w_1); likewise w_4 prefers m_1 to her M_3-partner (i.e., m_4).

One might wonder how likely it is for this desired solution concept (i.e., a stable matching) to exist in an arbitrary instance of SM. Gale and Shapley [37] cleared this doubt by providing an elegant theorem which establishes that every instance of SM admits at least one stable matching. To prove this result, they described a polynomial-time algorithm for actually finding such a matching. Their algorithm, which is referred to as the Gale-Shapley algorithm, involves a number of iterative proposals from the men to the women. Initially, every man who is unassigned proposes (simultaneously) to the most-preferred woman on his list, and each woman rejects all but her most-preferred man among the proposers. In subsequent steps, every man who is unassigned proposes to the most-preferred woman on his list whom he has not previously proposed to, and then again each woman who receives one or more proposals rejects all but her most-preferred man among the proposers and her current partner (if any). The algorithm proceeds in this fashion until everyone has a partner. It is clear that the number of proposals in the algorithm is bounded by the total length of the preference lists, and since all operations in a loop iteration can be implemented to run in $O(1)$ time, the overall time complexity of the algorithm is $O(n^2)$ [73].

In general, a given SM instance may admit many stable matchings, for example, the SM instance in Figure 2.1 admits two stable matchings, i.e., M_1 and M_2. This raises the question as to which stable matching is output by the Gale-Shapley algorithm? To answer this question, we note that the version of the algorithm that involves the men offering the proposals is referred to as *man-oriented*, and the resulting stable matching is *man-optimal* because each man obtains the best partner that he could possibly have in any stable matching [37]. Anal-

ogously, if the roles are reversed, i.e., if the women are the ones offering the proposals, the algorithm is *woman-oriented*, and the resulting stable matching is *woman-optimal* because each woman obtains the best partner that she could possibly have in any stable matching. Perhaps most interesting is the fact that the order in which the men or the women proposes is inconsequential to the final result.

For a given instance of SM, we will denote its man-optimal and woman-optimal stable matchings by M_0 and M_z respectively. It was observed by McVitie and Wilson [95] that M_0 (respectively M_z) is also a *woman-pessimal* (respectively *man-pessimal*) stable matching because each woman (respectively man) obtains the worst partner that she (respectively he) could possibly have in any stable matching. Going back to the SM instance given in Figure 2.1 (page 11), $M_0 = M_1 = \{(m_1, w_4), (m_2, w_3), (m_3, w_2), (m_4, w_1)\}$ and $M_z = M_2 = \{m_1, w_4), (m_2, w_1), (m_3, w_2), (m_4, w_3)\}$. In the case where $M_0 = M_z$, the underlying SM instance admits a unique stable matching.

An extended version of the Gale-Shapley algorithm was described in [44]. In addition to finding a stable matching, the aim of the extended algorithm is also to *reduce* the preference lists by eliminating certain pairs that cannot belong to any stable matching. By the term *delete* (m_i, w_j), we mean the removal of w_j from m_i's preference list and the removal of m_i from w_j's preference list; further, if m_i is assigned to w_j, we break the assignment. We define the *successor of* m_i in w_j's preference list as those men that are worse than m_i. An analogous definition holds for the *successor of* w_j in m_i's preference list.

Similar to the man-oriented version of the Gale-Shapley algorithm, the extended algorithm involves each man who is unassigned proposing to the most-preferred woman on his list. If woman w_i receives a proposal from man m_j, then w_j's partner in the resulting man-optimal (woman-pessimal) stable matching cannot be worse than m_i. This implies that for each successor $m_{i'}$ of m_i in w_j's preference list, the pair $(m_{i'}, w_j)$ cannot belong to any stable matching; hence, such pairs are deleted from the preference lists. We note that in the extended algorithm, if m_i proposes to w_j then the proposal will be accepted. This is because if w_j is already assigned to a man whom she prefers to m_i then the pair (m_i, w_j) would have been deleted. As before, the extended algorithm terminates when everybody has a partner; and its overall time complexity is $O(n^2)$. Finally, we note that in the man-optimal stable matching, each man is assigned to the first woman on her reduced list, and each woman is assigned to the last man on her reduced list. For further reading on the concept of reduced list, see [44].

2.1.2 Structure of stable matchings in SM

We recall that depending on the orientation (man- or woman-oriented) of the Gale-Shapley algorithm that is being used, the algorithm can generate either the man-optimal (M_0) or the

woman-optimal (M_z) stable matching. If $M_0 = M_z$ then the instance admits a unique stable matching. However, if $M_0 \neq M_z$ then the instance may admit many other stable matchings different from M_0 and M_z. As it turns out, the set of all stable matchings has some form of structure [44], which we discuss in what follows.

Let I be an instance of SM and let $\mathcal{M}$ denote the set of all stable matchings in I. First, we extend the notion of preferences over agents to preferences over matchings. Given $M, M' \in \mathcal{M}$, we say that man m_i *prefers* M to M' if m_i prefers $M(m_i)$ to $M'(m_i)$. Also, we say that m_i *is indifferent* between M and M' if $M(m_i) = M'(m_i)$. Next, we define a man-oriented dominance relation on $\mathcal{M}$. We say that M *dominates* M', denoted $M \preceq M'$, if for each man m_i, either m_i is indifferent between M and M' or m_i prefers M to M'. With respect to this definition of dominance relation, $\mathcal{M}$ is a partial order, denoted $(\mathcal{M}, \preceq)$.

The structure of stable matchings in I follows from a classical result which states that the partial order $(\mathcal{M}, \preceq)$ forms a distributive lattice[1] with the man-optimal and woman-optimal stable matchings representing the maximum and minimum element of the lattice, respectively. Knuth [73] attributes this structural result to John Conway, and a formal proof can be found in [44, Section 1.3.1]. As highlighted in [82], this lattice structure has been exploited to design efficient algorithms for a range of extensions of SM, some of which includes finding all stable pairs [43], generating all stable matchings [43], counting stable matchings [53], and finding stable matchings that satisfy additional optimality criteria [43, 54]. In what follows, we briefly summarise the key results that lead to the establishment of the lattice structure.

Lemma 2.1.1 ([44])**.** *Let M and M' be two (distinct) stable matchings in I. Let M^* be a set of man-woman pairs formed by assigning each man to the better of his partners in M and M'. Then M^* is a stable matching in I.*

Lemma 2.1.2 ([44])**.** *Let M and M' be two (distinct) stable matchings in I. Let M^* be a set of man-woman pairs formed by assigning each man to the poorer of his partners in M and M'. Then M^* is a stable matching in I.*

We denote by $M \wedge M'$ (respectively $M \vee M'$) the set of man-woman pairs in which each man is assigned to the better (respectively poorer) of his partners in M and M'. It follows from Lemma 2.1.1 (respectively Lemma 2.1.2) that $M \wedge M'$ (respectively $M \vee M'$) is a stable matching. These two operations give rise to a lattice structure for $\mathcal{M}$, which we state as follows.

Theorem 2.1.3 ([44])**.** *Let I be an instance of* SM*, and let $\mathcal{M}$ be the set of stable matchings in I. Let $\preceq$ be the dominance partial order on $\mathcal{M}$ and let $M, M' \in \mathcal{M}$. Then $(\mathcal{M}, \preceq)$ forms a distributive lattice, with $M \wedge M'$ representing the meet of M and M', and $M \vee M'$ the join of M and M'.*

[1] See Definition 3.3.1 for a formal definition of a distributive lattice.

So far, we have assumed that in an instance of SM, (i) the sets of men and women are of equal size, (ii) the preference lists are complete, and (iii) the preference lists are strictly ordered. In what follows, we consider the extensions of SM that have been studied in the literature [44] as a result of relaxing these three assumptions.

2.1.3 Preferences with Incomplete lists: SMI

The *Stable Marriage problem with Incomplete lists* (SMI) represents relaxations (i) and (ii) in the sense that the size of the sets of men and women need not be equal, and the preference lists need not be complete (i.e., the preference list of an agent may contain a proper subset of agents from the opposite set). An instance of SMI consists of a set $\mathcal{M} = \{m_1, m_2, \ldots, m_{n_1}\}$ of *men* and a set $\mathcal{W} = \{w_1, w_2, \ldots, w_{n_2}\}$ of *women* (possibly $n_1 \neq n_2$) each of whom has a strictly-ordered preference list over a subset of the members of the opposite set. This problem model allows men (women) to declare some of the women (men) to be *unacceptable*, meaning that such men (women) would rather be unassigned than be assigned to a woman (man) who they did not include in their preference lists. If woman w_j appears on man m_i's list then we say that m_i finds w_j *acceptable*. Similarly, if man m_i appears on woman w_j's list then we say that w_j finds m_i *acceptable*. If m_i and w_j find each other acceptable, we refer to the pair (m_i, w_j) as an *acceptable pair*.

A *matching* M in this context is a one-one pairing between a subset of the men and a subset of the women such that $(m_i, w_j) \in M$ only if m_i and w_j find each other acceptable. If $(m_i, w_j) \in M$, we say that m_i *is assigned to* w_j and we write $M(m_i) = w_j$; also, we say that w_j *is assigned to* m_i and we write $M(w_j) = m_i$. We note that it may not be possible to assign every man and woman in this setting. If a man m_i (woman w_j) is not involved in any pair in M, we say that m_i (w_j) is *unassigned* in M and $M(m_i)$ ($M(w_j)$) is undefined. As a consequence, we redefine the notion of a blocking pair as follows.

Definition 2.1.4 ([44])**.** *Let I be an instance of* SMI *and let M be a matching in I. An acceptable pair (m_i, w_j) is said to* block *M, or be a* blocking pair *for M, if (a) and (b) holds as follows:*

(a) either m_i is unassigned in M, or m_i prefers w_j to $M(m_i)$;

(b) either w_j is unassigned in M, or w_j prefers m_i to $M(w_j)$.

Again, M is *stable* if it admits no blocking pair. A stable matching always exists in this context and Gusfield and Irving [44] noted that such a matching can be obtained by extending the Gale-Shapley algorithm [37]. Moreover, Gusfield and Irving [44] noted that the results concerning the man-optimal (i.e., woman-pessimal) and woman-optimal (i.e., man-pessimal) stable matchings extend to SMI. While an instance of SMI may admit many stable matchings,

Gale and Sotomayor [38] observed that all stable matchings assign exactly the same subset of agents. Hence, if an agent is assigned (respectively unassigned) in one stable matching, they are assigned (respectively unassigned) in all of the stable matchings, and as such the lattice structure result for SM holds for SMI. They also noted that all stable matchings in an instance of SMI are of the same size.

2.1.4 Preferences with Ties and Incomplete lists: SMTI

If we relax the assumption that the agents' preference lists are strictly ordered, so that each agent can express some form of *indifference* in their preference list, then we obtain a generalisation of SMI referred to as the *Stable Marriage problem with Ties and Incomplete lists* (SMTI). In the preference list of man m_i, a set T of r women forms a *tie of length* r (where $r \geq 1$) if m_i does not prefer w_j to w_k for any $w_j, w_k \in T$. A tie in a woman's list is defined in a similar fashion. Formally, an instance I of SMTI consists of a set $\mathcal{M} = \{m_1, m_2, \ldots, m_{n_1}\}$ of *men* and a set $\mathcal{W} = \{w_1, w_2, \ldots, w_{n_2}\}$ of *women* (possible $n_1 \neq n_2$) each of whom has a strictly-ordered preference list over tied batches of agents in the opposite set. Thus an agent is indifferent between the members of each tie, and prefers each member of a given tie to each member of any successor tie. The definition of an *acceptable pair* in I is similar to that given in the SMI setting. We give an example instance I_1 of SMTI in Figure 2.2, which involves the set of men $\mathcal{M} = \{m_1, m_2, m_3\}$ and the set of women $\mathcal{W} = \{w_1, w_2\}$. Ties in the preference lists are indicated by round brackets.

Men's preferences	Women's preferences
m_1: w_1 w_2	w_1: m_2 m_1
m_2: (w_1 w_2)	w_2: m_2 m_1 m_3
m_3: w_2	

Figure 2.2: An instance I_1 of SMTI, adapted from [44, Figure 1.13, page 30]. Man m_1 prefers w_1 to w_2; while man m_2 is indifferent between w_1 and w_2.

The intuition behind the earlier definitions of blocking pair for a matching in the SM and SMI setting is that if m_i and w_j are both involved in a blocking pair, both of them would be better off by becoming assigned to each other. However, this does not directly translate to the SMTI setting; for example, what happens if m_i is indifferent between his assigned partner and w_j? To clarify this, Gusfield and Irving [44] established that an acceptable (man, woman) pair who are not matched to one another can form a blocking pair for a matching if, by becoming assigned to one another, either (i), (ii) or (iii) holds as follows:

(i) both of them would improve;

(ii) one of them would improve and the other would not be worse off;

(iii) neither of them would be worse off.

Further, they defined a matching M in an instance of SMTI as *weakly stable*, *strongly stable*, or *super-stable* if it does not admit a blocking pair of type (i), (ii) or (iii), respectively. Based on this informal definition, we note that a super-stable matching is strongly stable, and a strongly stable matching is weakly stable. Gusfield and Irving [44] showed that we can always obtain a weakly stable matching in an instance of SMTI by breaking the ties arbitrarily and applying the Gale-Shapley algorithm to the resulting SMI instance to find any stable matching. Subsequently, Manlove *et al.* [84] showed that the manner in which these ties are resolved could lead to the algorithm generating weakly stable matchings with different *sizes*. They further showed that MAX-SMTI (i.e., the problem of finding a maximum size weakly stable matching) is NP-hard, even under the restriction that the preference lists on one side are strictly ordered, and each preference list on the opposite side is either strictly ordered or comprises one tie of length two.

Irving *et al.* [56] showed that MAX-SMTI remains NP-hard even if the length of each agent's list is at most three, and they gave a polynomial-time algorithm for the restriction where the length of each man's list is at most two and the length of each woman's list is unbounded. Several approximation algorithms for MAX-SMTI exist in the literature [47, 55, 62, 70, 71, 84, 93, 100], with the best known for the unrestricted case having an approximation ratio of $\frac{3}{2}$ [71, 93, 100]. See [28] for a recent survey on approximability results for MAX-SMTI.

In contrast to the guaranteed existence of weakly stable matchings, it was observed in [44] that an instance of SMTI need not admit a strongly stable or super-stable matching. To see this, consider the instance I_1 given in Figure 2.2. The reader can verify that any matching will be undermined under strong stability or super-stability. For instance, matching $M_1 = \{(m_1, w_1), (m_2, w_2)\}$ is blocked by the pair (m_2, w_1), since m_2 would be no worse off and w_1 would improve relative to M_1. A similar argument can be made for the matching $M_2 = \{(m_1, w_2), (m_2, w_1)\}$ which is blocked by the pair (m_2, w_2).

Under the restriction of SMTI where the preference lists are complete (SMT), Irving [51] gave $O(n^4)$ and $O(n^2)$ algorithms to find a strongly stable and a super-stable matching respectively, or to report that no such matching exists, where $n = \max\{n_1, n_2\}$. Subsequently, Manlove [81] extended these two algorithms to the SMTI setting with the same time complexity. Later, Kavitha *et al.* [66] described an improved strong stability algorithm for SMTI with running time $O(nm)$, where n is the number of agents and m is the number of acceptable pairs.

2.2 The Hospitals/Residents Problem: HR

2.2.1 Introduction

The *Hospitals/Residents problem* (HR) [37, 44, 110] involves assigning one or more agents in one set (residents) to an agent in the opposite set (hospitals). In view of the fact that many residents may be assigned to one hospital, HR is commonly referred to as a many-one generalisation of SMI. The HR model was first introduced by Gale and Shapley [37], although in the context of the *College Admissions problem*. For more than five decades, this problem model has been applied by centralised matching schemes in different countries, including NRMP [1], CaRMS [2], JRMP [3], and SFAS [52] which ran until 2012 (see paragraph 3 of Section 1.1 for a description of these acronyms and more details on the centralised matching schemes). These matching schemes handle the allocation of graduating medical students (henceforth residents) to residency positions in hospitals in their respective countries. To make the allocations, the residents are required to provide strictly-ordered preference lists over the available hospitals that they would like to be assigned to. Each hospital also provides a strictly-ordered preference list over the residents that find them acceptable, as well as a *capacity*, which indicates the maximum number of residents that the hospital can accommodate. We recall that the goal in this setting is to find a stable matching of residents to hospitals.

2.2.2 Problem definition

Formally, an instance I of HR involves two sets of agents – a set $R = \{r_1, r_2, \ldots, r_{n_1}\}$ of *residents* and a set $H = \{h_1, h_2, \ldots, h_{n_2}\}$ of *hospitals*. Each resident $r_i \in R$ ranks a subset of hospitals in H in strict order of preference, which forms r_i's *preference list*. If a hospital h_j is in resident r_i's preference list, we say that r_i finds h_j *acceptable*. Let $h_j, h_k \in H$ such that r_i finds both h_j and h_k acceptable. If h_j precedes h_k on r_i's preference list, we say that r_i *prefers* h_j to h_k. Each hospital $h_j \in H$ ranks a subset of residents in R that find h_j acceptable, which forms h_j's preference list. If r_i is in h_j's preference list we say that h_j finds r_i *acceptable*. The *prefers* relation is defined similarly for a hospital. If r_i and h_j both find each other acceptable, we call (r_i, h_j) an *acceptable pair*. Also, h_j has a *capacity* $c_j \in \mathbb{Z}^+$, which indicates the maximum number of residents that h_j can accommodate.

An *assignment* M is a collection of acceptable pairs in $R \times H$. If $(r_i, h_j) \in M$, we say that r_i is *assigned to* h_j in M and we denote by $M(r_i)$ the set of hospitals that are assigned to r_i in M. Similarly, if $(r_i, h_j) \in M$ we say that h_j is *assigned* r_i in M and we denote by $M(h_j)$ the set of residents assigned to h_j in M. We denote by $|M(h_j)|$ the *size* of $M(h_j)$ (i.e., the number of residents assigned to h_j in M), and we say that h_j is *undersubscribed*, *full* or

oversubscribed according as $|M(h_j)|$ is less than, equal to, or greater than c_j, respectively. A *matching* M is an assignment such that $|M(r_i)| \leq 1$ for each $r_i \in R$ and $|M(h_j)| \leq c_j$ for each $h_j \in H$ (i.e., each resident is assigned to at most one hospital and no hospital is oversubscribed). For notational convenience, if r_i is assigned to h_j in M, we denote h_j by $M(r_i)$.

Definition 2.2.1 ([37]). *Let I be an instance of* HR *and let M be a matching in I. We say that M is* stable *in I if it admits no* blocking pair*, where a* blocking pair *for M is an acceptable pair $(r_i, h_j) \in (R \times H) \setminus M$ such that (a) and (b) holds as follows:*

(a) either r_i is unassigned in M, or r_i prefers h_j to $M(r_i)$;
(b) either h_j is undersubscribed in M, or h_j prefers r_i to some resident in $M(h_j)$.

Gale and Shapley [37] showed that every instance of HR admits a stable matching and they described a polynomial-time algorithm to find such matching. Analogous to the man-optimal and woman-optimal stable matchings in the SMI setting, the algorithm can output a stable matching that is either *resident-optimal* (*hospital-pessimal*) or *hospital-optimal* (*resident-pessimal*), depending on the orientation of the algorithm that is being used (i.e., resident-oriented or hospital-oriented, respectively).

2.2.3 Structure of stable matchings in HR

Given an instance I of HR, as in the case of SMI, there may be many other stable matchings in addition to the resident-optimal and hospital-optimal stable matchings. However, there are some interesting properties satisfied by assigned and unassigned residents and by undersubscribed hospitals in all stable matchings in I. These properties are collectively referred to as the *Rural Hospitals Theorem* in the literature [38, 105, 106], which we state as follows.

Theorem 2.2.2 (*Rural Hospitals Theorem* [38, 105, 106]). *For a given instance I of* HR*, the following properties holds:*

(i) the same residents are assigned in all stable matchings; thus, all stable matchings in I are of the same size;
(ii) each hospital is assigned the same number of residents in all stable matchings;
(iii) any hospital that is undersubscribed in one stable matching is assigned exactly the same set of residents in all stable matchings.

The structural results established for the set of stable matchings in an instance of SM in Section 2.1.2 also extend naturally to the HR setting [44]. Let I be an instance of HR. Let $\mathcal{M}$ denote the set of all stable matchings in I, and let $M, M' \in \mathcal{M}$. We say that a resident r_i

prefers M to M' if r_i is assigned in both M and M', and r_i prefers $M(r_i)$ to $M'(r_i)$. Also, we say that r_i is *indifferent between* M and M' if either (i) r_i is unassigned in both M and M', or (ii) r_i is assigned in both M and M', and $M(r_i) = M'(r_i)$. We say that M *dominates* M', denoted $M \preceq M'$, if each resident either prefers M to M' or is indifferent between them. Similar to the SM setting, $(\mathcal{M}, \preceq)$ forms a distributive lattice under this dominance relation with $M \wedge M'$ and $M \vee M'$ representing the *meet* and *join* of the lattice, respectively, where $M \wedge M'$ ($M \vee M'$) is a stable matching formed by giving each assigned resident the better (poorer) of her hospitals in M and M' (each resident who is unassigned in M and in M' is unassigned in $M \wedge M'$ and in $M \vee M'$). Moreover, the resident-optimal and hospital-optimal stable matchings represent the maximum and minimum elements of the lattice, respectively [44].

Example 2.2.3. We present an example to illustrate some of the results in this section. The HR instance I shown in Figure 2.3 involves the set of residents $\mathcal{R} = \{r_i : 1 \leq i \leq 12\}$ and the set of hospitals $\mathcal{H} = \{h_j : 1 \leq j \leq 5\}$. The instance admits a total of seven stable matchings, which are illustrated in Figure 2.4. To illustrate Theorem 2.2.2, we observe that

(i) each resident in $\mathcal{R} \setminus \{r_{12}\}$ is assigned in all of the stable matchings, and r_{12} is not assigned in any stable matching;

(ii) each hospital in $\mathcal{H}$ is assigned the same number of residents in all stable matchings; and

(iii) hospital h_2 is undersubscribed in all of the stable matchings, and $M_k(h_2) = \{r_7\}$ $(1 \leq k \leq 7)$.

For (ii), the reader can easily verify that for each k $(1 \leq k \leq 7)$, $|M_k(h_1)| = 4$, $|M_k(h_2)| = |M_k(h_5)| = 1$, $|M_k(h_3)| = 3$, and $|M_k(h_4)| = 2$. Further, the Hasse diagram of the lattice structure representing the set of all stable matchings in I is illustrated in Figure 2.5, with the resident-optimal (M_1) and hospital-optimal (M_7) stable matchings at the top and bottom of the lattice respectively. This structure is a directed graph with each vertex representing a stable matching, and there is a directed edge from vertex M to M' (where $M \neq M'$) if $M \preceq M'$ and there exists no M^* (distinct from M and M') such that $M \preceq M^* \preceq M'$. For example, it can be verified that $M_1 = M_2 \wedge M_3$ and $M_4 = M_2 \vee M_3$. We note that all the edges representing precedence implied by transitivity are suppressed in the Hasse diagram.

Residents' preferences	Hospitals' preferences
r_1: h_3 h_1 h_5 h_4	h_1: r_3 r_7 r_9 r_{11} r_5 r_4 r_{10} r_8 r_6 r_1 r_2
r_2: h_1 h_3 h_4 h_2 h_5	h_2: r_5 r_7 r_{10} r_6 r_8 r_2 r_3 r_{11}
r_3: h_4 h_5 h_3 h_1 h_2	h_3: r_{11} r_6 r_8 r_3 r_2 r_4 r_7 r_1 r_{10} r_{12}
r_4: h_3 h_4 h_1 h_5	h_4: r_{10} r_1 r_2 r_{11} r_4 r_9 r_5 r_3 r_6 r_8 r_{12}
r_5: h_1 h_4 h_2	h_5: r_2 r_4 r_{10} r_7 r_6 r_1 r_8 r_3 r_{11} r_9 r_{12}
r_6: h_4 h_3 h_2 h_1 h_5	
r_7: h_2 h_5 h_1 h_3	
r_8: h_1 h_3 h_2 h_5 h_4	
r_9: h_4 h_1 h_5	
r_{10}: h_3 h_1 h_5 h_2 h_4	Hospital capacities: $c_1 = 4, c_2 = c_3 = 3, c_4 = 2, c_5 = 1$
r_{11}: h_5 h_4 h_1 h_3 h_2	
r_{12}: h_3 h_4 h_5	

Figure 2.3: An instance of HR due to Gusfield and Irving [44].

Matching	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
M_1	h_3	h_1	h_4	h_3	h_1	h_3	h_2	h_1	h_4	h_1	h_5
M_2	h_1	h_3	h_4	h_3	h_1	h_3	h_2	h_1	h_4	h_1	h_5
M_3	h_3	h_1	h_5	h_3	h_1	h_3	h_2	h_1	h_4	h_1	h_4
M_4	h_1	h_3	h_5	h_3	h_1	h_3	h_2	h_1	h_4	h_1	h_4
M_5	h_5	h_3	h_3	h_4	h_1	h_3	h_2	h_1	h_1	h_1	h_4
M_6	h_5	h_4	h_3	h_1	h_1	h_3	h_2	h_3	h_1	h_1	h_4
M_7	h_4	h_4	h_3	h_1	h_1	h_3	h_2	h_3	h_1	h_5	h_1

Figure 2.4: The stable matchings in the HR instance illustrated in Figure 2.3.

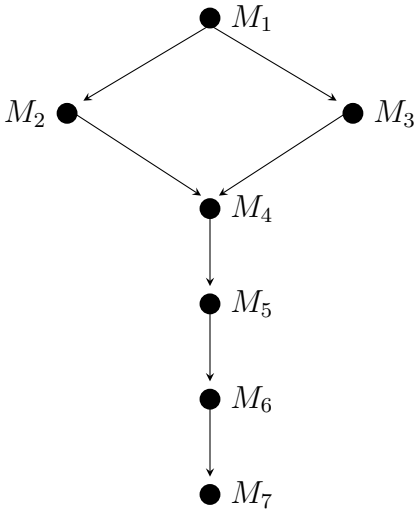


Figure 2.5: The lattice structure for the stable matchings in Figure 2.4, with M_1 (the resident-optimal stable matching) as the maximum element of the lattice and M_7 (the hospital-optimal stable matching) as the minimum element of the lattice.

2.2.4 Preferences with Ties: HRT

As mentioned earlier, the centralised matching scheme that handles the allocation of residents to hospitals in the United States (NRMP [1]) involves over $40,000$ residents annually [96]. In large-scale matching schemes of this kind, popular hospitals may not be able to provide a strictly-ordered preference over what may be a very large number of residents that find them acceptable. These hospitals may prefer to express *indifference* in their preference lists, for instance, ranking two or more residents equally in a *tie*. Similarly, a resident may wish to rank two or more hospitals equally. This leads to a generalisation of HR known as the *Hospitals/Residents problem with Ties* (HRT) [57, 59]. The formal definition of a tie in a resident's or hospital's preference list is analogous to that given in the SMTI setting (see Section 2.1.4). Irving *et al.* [57, 59] extended the three stability definitions for SMTI, namely *weak stability, strong stability* and *super-stability*, to the HRT setting. Thus a matching M in I is *weakly stable*, *strongly stable* or *super-stable* if it does not admit a *blocking pair* with respect to the corresponding stability criteria. For a formal definition of these terms in the context of HRT, see [83, Section 1.3.5].

Every instance of HRT admits a weakly stable matching [57]. This can be obtained by breaking each tie in the preference lists arbitrarily and subsequently applying the Gale-Shapley [37] algorithm to the resulting HR instance. However, similar to the SMTI setting, in general, the manner in which the ties are resolved yields weakly stable matchings with different sizes, and the problem of finding a maximum size weakly stable matching given an instance of HRT (MAX-HRT) is NP-hard [84]. We note that some of the approximability results and approximation algorithms for SMTI also extend to the HRT setting (we refer the interested reader to [83] for a summary). Other techniques that have been explored to enable MAX-HRT to be solved to optimality on reasonably-sized instances include IP formulation [77] and parameterised complexity [89].

Similar to the SMTI setting, an instance of HRT need not admit a strongly stable or super-stable matching. To find a super-stable (respectively strongly stable) matching or to report that no such matching exists, Irving *et al.* [57] (respectively [59]) described a polynomial-time algorithm with time complexity $O(L)$ (respectively $O(m^2)$), where L is the total length of all the preference lists and m is the total number of acceptable pairs. Subsequently, Kavitha *et al.* [66] described an improved strong stability algorithm for HRT, which builds on the one described in [59], with running time $O(m \sum_{h \in H} c_j)$. Furthermore, counterparts of the Rural Hospitals Theorem for HR (Theorem 2.2.2, page 18) hold for HRT under super-stability [57] and strong stability [113].

2.2.5 Couples with joint preference lists: HRC

A generalisation of HR where some of the residents may apply jointly in couples is known as the *Hospitals-Residents problem with Couples* (HRC) [25, 44, 83, 94, 105]. This extension is important in practical applications as it gives large-scale matching schemes, for example the NRMP [1], the possibility of matching couples to geographically close hospitals, essentially keeping partners together. A formal definition of this problem model and the notion of a stable matching can be found in [85]. In what follows, we give a general overview of the existing algorithmic results for HRC in the literature.

Roth [105] showed that an instance I of HRC need not admit a stable matching, while Ronn [104] proved that the problem of deciding whether an instance of HRC admits a stable matching is NP-complete, even if there are no single residents and each hospital has capacity 1. It was shown in [90] that this decision problem is also W[1]-hard when parameterised by the number of couples. In a further restricted case in which each single resident's preference list contains at most α hospitals, each couple's preference list contains at most β pairs of hospitals and each hospital's preference list contains at most γ residents, referred to as (α, β, γ)-HRC, Manlove and McDermid [94] showed that deciding whether an instance of $(3, 2, 4)$-HRC admits a stable matching is NP-complete. A similar result also holds for an instance of $(0, 2, 2)$-HRC [26].

Outwith the hardness results for HRC, Aldershof and Carducci [16] showed that, should an HRC instance admit a stable matching, such matchings could be of different sizes. Furthermore, Biro *et al.* [26] described an IP model for finding a maximum size stable matching or reporting that no such matching exists. They also presented an empirical evaluation of an implementation of their model, showing that their model is capable of solving instances of the magnitude of those arising in the SFAS [52] application. Further algorithmic results for HRC are given in [25, 83, 91].

2.3 The Student-Project Allocation Problem: SPA

2.3.1 Introduction

The *Student-Project Allocation problem* (SPA) [13, 31, 83] involves three sets of entities: students, projects and lecturers. Each project is supervised by one lecturer and each student has preferences over a subset of the available projects that she finds acceptable. Further, each lecturer may have preferences over the students that find her projects acceptable and/or the projects that she offers. Typically there may be upper bounds on the number of students that each project and lecturer can accommodate. In general, the goal is to allocate students to

projects by respecting the stated preferences such that each student is assigned to at most one project, and the capacity constraints on projects and lecturers are not violated.

Two variants of SPA arise based on the existence (or otherwise) of lecturers' preferences. In some centralised matching schemes that deal with the allocation of students to projects, only students express preferences. Examples of these schemes exist at the School of Computing Science, University of Glasgow [76], Department of Civil and Environmental Engineering, University of Southampton [17, 48], and the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore [116].

Under the SPA model where preferences are permitted from both the students and lecturers, three different variants have been studied based on the nature of the lecturers' preferences. These include (i) the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S) [13, 83], (ii) the *Student-Project Allocation problem with lecturer preferences over Projects* (SPA-P) [61, 86, 87], and (iii) the *Student-Project Allocation problem with lecturer preferences over Student-Project pairs* (SPA-(S,P)) [13, 14]. The problems that we will be considering in Chapters 3 - 6 belong to category (i) and (ii). In what follows, we give a survey of the three SPA variants mentioned above, with a specific focus on variants (i) and (ii).

2.3.2 Lecturer preferences over Students: SPA-S

A variant of SPA where lecturers have preferences over students is known as the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S) [13, 83]. In this model, each lecturer provides a strictly-ordered preference list over the students who rank at least one of the projects that the lecturer proposed. The Department of Computing Science, University of York [36, 67, 118] is an example of a centralised matching scheme where this variant is in use.

Formally, an instance I of SPA-S involves a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of *students*, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of *projects* and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of *lecturers*. Each student s_i ranks a subset of $\mathcal{P}$ in strict order, which forms s_i's *preference list*. We say that s_i finds p_j *acceptable* if p_j is in s_i's preference list. We denote by A_i the set of projects that s_i finds acceptable. Let $p_j, p_{j'} \in \mathcal{P}$ such that s_i finds both p_j and $p_{j'}$ acceptable. If p_j precedes $p_{j'}$ in A_i, we say that s_i *prefers* p_j to $p_{j'}$.

Each lecturer $l_k \in \mathcal{L}$ offers a non-empty set of projects P_k, where $P_1, P_2, \ldots, P_{n_3}$ partitions $\mathcal{P}$. Also, l_k ranks in strict order of preference those students who find at least one project in P_k acceptable. We say that l_k finds s_i *acceptable* if s_i is in l_k's preference list. We denote by $\mathcal{L}_k$ the set of students that l_k finds acceptable. The *prefers* relation is defined similarly for

lecturer l_k. For any pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$, where p_j is offered by l_k, we refer to (s_i, p_j) as an *acceptable pair* if s_i and l_k both find each other acceptable, i.e., if $p_j \in A_i$ and $s_i \in \mathcal{L}_k$.

Further, l_k has a capacity $d_k \in \mathbb{Z}^+$, indicating the maximum number of students that l_k is willing to supervise. Similarly each project $p_j \in \mathcal{P}$ has a capacity $c_j \in \mathbb{Z}^+$ indicating the maximum number of students that p_j can accommodate. We assume that for any lecturer l_k, $\max\{c_j : p_j \in P_k\} \leq d_k \leq \sum\{c_j : p_j \in P_k\}$, i.e., the capacity of l_k is (i) at least the highest capacity of the projects offered by l_k, and (ii) at most the sum of the capacities of all the projects that l_k is offering. We denote by $\mathcal{L}_k^j$, the *projected preference list* of lecturer l_k for p_j, which can be obtained from $\mathcal{L}_k$ by removing those students that do not find p_j acceptable (thereby retaining the order of the remaining students from $\mathcal{L}_k$).

An *assignment* M is a collection of acceptable pairs in $\mathcal{S} \times \mathcal{P}$. We define the *size* of M as the number of (student, project) pairs in M, denoted $|M|$. If $(s_i, p_j) \in M$, we say that s_i is *assigned to* p_j and p_j is *assigned* s_i. Furthermore, we denote by $M(s_i)$ the set of projects that are assigned to s_i in M. Similarly, we denote the set of students assigned to p_j in M as $M(p_j)$. For ease of exposition, if s_i is assigned to a project p_j offered by lecturer l_k, we may also say that s_i is *assigned to* l_k, and l_k is *assigned* s_i. Thus we denote the set of students assigned to l_k in M as $M(l_k)$. A project $p_j \in \mathcal{P}$ is *undersubscribed*, *full* or *oversubscribed* in M if $|M(p_j)|$ is less than, equal to or greater than c_j, respectively. We say that p_j is *non-empty* if $|M(p_j)| > 0$, i.e., there exists some student who is assigned to p_j in M. Similarly, each lecturer $l_k \in \mathcal{L}$ is *undersubscribed*, *full* or *oversubscribed* if $|M(l_k)|$ is less than, equal to or greater than d_k, respectively.

A *matching* M is an assignment such that $|M(s_i)| \leq 1$ for each $s_i \in \mathcal{S}$, $|M(p_j)| \leq c_j$ for each $p_j \in \mathcal{P}$, and $|M(l_k)| \leq d_k$ for each $l_k \in \mathcal{L}$ (i.e., each student is assigned to at most one project in M, and no project or lecturer is oversubscribed in M). For notational convenience, if s_i is assigned to p_j in M, we denote p_j by $M(s_i)$.

Definition 2.3.1 ([13])**.** *Let I be an instance of* SPA-S *and let M be a matching in I. We say that M is* stable *in I if it admits no blocking pair, where a* blocking pair *for M is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that (a) and (b) holds as follows:*

(a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$;

(b) either (i), (ii) or (iii) holds as follows:

- *(i) each of p_j and l_k is undersubscribed in M;*
- *(ii) p_j is undersubscribed in M, l_k is full in M and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$;*
- *(iii) p_j is full in M and l_k prefers s_i to the worst student in $M(p_j)$.*

We remark that HR is a special case of SPA-S where each lecturer offers only one project and the capacity of each project is equal to the capacity of the lecturer offering the project,

i.e., $P_k = \{p_k\}$ and $c_k = d_k$ ($1 \leq k \leq n_3$), respectively. Abraham *et al.* [13] proposed two linear-time algorithms to find a stable matching in an instance of SPA-S. The first algorithm involves students applying to projects (the *student-oriented* version) and it outputs the *student-optimal* stable matching, in the sense that each assigned student is allocated to the best project that she could obtain in any stable matching. In contrast, the second algorithm involves lecturers offering projects to students (*the lecturer-oriented version*) and it outputs the *lecturer-optimal* stable matching, in the sense that each lecturer is allocated the best set of students that she could obtain in any stable matching. The set of stable matchings in an arbitrary instance of SPA-S satisfy several properties, analogous to the Rural Hospitals Theorem for HR (Theorem 2.2.2), which we refer to as the *Unpopular Projects Theorem*.

Theorem 2.3.2 (Unpopular Projects Theorem [13])**.** *For a given instance of* SPA-S*, the following holds:*

1. *each lecturer is assigned the same number of students in all stable matchings;*
2. *exactly the same students are unassigned in all stable matchings;*
3. *a project offered by an undersubscribed lecturer is assigned the same number of students in all stable matchings.*

2.3.3 Lecturer preferences over Students with Ties: SPA-ST

The classical SPA-S model assumes that preferences are strictly ordered. However, this might not be achievable in practice. For instance, a lecturer may be unable or unwilling to provide a strict ordering of all the students who find her projects acceptable. Such a lecturer may be happier to rank two or more students equally in a tie, which indicates that the lecturer is indifferent between the students concerned. This leads to a generalisation of SPA-S known as the *Student-Project Allocation problem with lecturer preferences over Students with Ties* (SPA-ST) [13, 83]. Abraham *et al.* [13] proposed this problem and observed that the three different stability definitions for SMTI and HRT, namely *weak stability*, *strong stability* and *super-stability*, can be extended to the SPA-ST setting.

Further, Abraham *et al.* [13] noted that under the weakest of these three stability concepts, every instance of SPA-ST admits a weakly stable matching (this follows by breaking the ties in an arbitrary fashion and applying the stable matching algorithm described in [13] to the resulting SPA-S instance). However, the manner in which the ties are resolved could lead to weakly stable matchings of different sizes in general [84]. Thus weak stability poses a new problem denoted by MAX-SPA-ST, which is the problem of finding a weakly stable matching that matches as many students to projects as possible. This problem is known to be NP-hard even for SMTI [60, 84], which is the special case of HRT in which each project (hospital) has

capacity 1. To cope with the hardness of MAX-SPA-ST, Cooper and Manlove [33] described a $\frac{3}{2}$-approximation algorithm which finds a weakly stable matching that is at least two-thirds the size of a maximum weakly stable matching. Under the two other stability criteria, an instance of SPA-ST need not admit a strongly stable or super-stability matching (again, this follows by restriction to the SMTI and HRT settings). We study the super-stability and strong stability concepts in the SPA-ST setting in Chapters 4 and 5 respectively.

2.3.4 Lecturer Preferences over Projects: SPA-P

Manlove and O'Malley [87] were the first to study the variant of SPA where lecturers have preferences over their proposed projects. A motivation for this variant is that a lecturer might prefer to supervise projects that are closely related to or in-line with her research interests, while the remaining projects that she proposed, perhaps to ensure that the students have some sense of choice, are of lesser priority to her. The formal definition of an instance of SPA-P follows from that of SPA-S, except that each lecturer $l_k \in \mathcal{L}$ ranks the projects in P_k in strict order of preference instead of ranking the students in $\mathcal{S}$ (see Figure 6.1 on page 135 for an example instance of SPA-P). All the notation and terminology defined for SPA-S also holds for SPA-P, except that of stability. Informally, a matching in an instance of SPA-P is *stable* if it admits no (i) student and lecturer who would rather be assigned together than remain with their current assignment, and (ii) group of students acting together to undermine the integrity of the matching by swapping their assigned projects, in order to be better off.

Let I be an arbitrary instance of SPA-P. Manlove and O'Malley [87] established that I admits at least one stable matching, and when there is more than one stable matching in I they may have different sizes. Further, they proved that the problem of finding a stable matching that matches as many students to projects as possible, denoted MAX-SPA-P, is NP-hard. In addition, they showed that MAX-SPA-P is not approximable within δ_1, for some constant $\delta_1 > 1$, unless P = NP. Moreover, the result holds even if each project and lecturer has capacity 1, and all preference lists are of length at most 4. To cope with this hardness, they gave a polynomial-time 2-approximation algorithm for MAX-SPA-P.

Subsequently, Iwama *et al.* [61] described an improved approximation algorithm with an upper bound of $\frac{3}{2}$, which builds on the one described in [87]. In addition, the authors showed that MAX-SPA-P is not approximable within $\frac{21}{19} - \epsilon$, for any $\epsilon > 0$, unless P = NP. For the upper bound, they modified Manlove and O'Malley's algorithm [87] using ideas from Király's approximation algorithm for MAX-SMTI [70].

O'Malley [98] formulated another definition of stability for SPA-P, which is different from the one given in [87]. He referred to this form of stability as *strong stability*, and he showed that given an instance of SPA-P, a strongly stable matching need not exist. In addition, he described a linear-time algorithm to find a strongly stable matching or report that none exists.

2.3.5 Lecturer preferences over Student-Project pairs: SPA-(S,P)

A setting where lecturers' preference lists involve student-project pairs is a generalisation of each of SPA-S and SPA-P, and is known as the *Student-Project Allocation problem with lecturer preferences over Student-Project pairs* (SPA-(S,P)) [13, 14]. This variant models the possibility that a lecturer may believe that a given student is better suited to one project that she offers compared to another project. An instance of SPA-(S,P) is defined in the same way as an instance of SPA-S, except that each lecturer ranks, in strict order of preference, the set of student-project pairs (s_i, p_j) where s_i finds acceptable a project p_j that l_k offers.

Abraham *et al.* [13] suggested the study, and Abu El-Atta and Moussa [14] formally defined the problem. A suitable blocking pair definition was also established in [14]. Further, the authors extended the student-oriented algorithm for SPA-S [13] to SPA-(S,P). As a consequence, for every instance of SPA-(S,P), a stable matching can be constructed in $O(m)$ time, where m is the total length of the students' preference lists [14].

2.3.6 Other applications of SPA with two-sided preferences

Although the SPA problem and its variants were formulated and motivated in the context of allocating students to projects, they have applications in an engineering context where network users seek to be associated with base stations in downlink multi-cell non-orthogonal multiple-access (NOMA) networks [18, 19, 20].

The problem model described in [19] is equivalent to SPA-S with the students, projects and lecturers corresponding to users, channels and base stations, respectively. Users have strictly-ordered preference lists over the channels available at each base station. Also, base stations have strictly-ordered preference lists over the users that want to be associated with their channels. In addition, there is a capacity constraint on the number of users that can be associated to a channel and to each base stations. Considering the preferences and constraints, the goal is to associate users to channels at base stations, such that no user or base station would deviate from its assignment. This is similar to the concept of a stable matching in the SPA-S setting. Further, the student-oriented and lecturer-oriented algorithms described for SPA-S in [13] were adapted to find user-optimal and station-optimal stable matchings respectively.

The problem model described in [20] is equivalent to SPA-P with the students, projects and lecturers corresponding to users, relays and base stations, respectively. The users and base stations both have preferences over the relays, and similar to above, there is a notion of capacity constraints. Further, the 2- and $\frac{3}{2}$- approximation algorithms described for SPA-P in [61, 87] were adapted to find stable associations of users to relays at base stations.

Finally, the problem model described in [18] is equivalent to SPA-(S,P) with the students projects and lecturers corresponding to device-to-device (D2D) groups, users and base sta-

tions, respectively. Each D2D group has a strictly-ordered preference list over the users and each base station has a strictly-ordered preference list over (D2D group, user) pairs. The two polynomial-time algorithms for finding a stable matching described for SPA-(S,P) in [14] ~~was~~ were extended to this setting.

2.3.7 Other SPA models and approaches

The SPA variants that have been considered in the previous sections allow some form of preference from the lecturers over the students that finds their projects acceptable and/or the projects that they offer. However, some practical applications of SPA consider lecturer preferences to be undesirable or unnecessary [31, 76]. In these applications, preferences are only allowed from the students; thus the underlying SPA model falls under the category of bipartite matching problems with one-sided preferences. The reader may recall that the fundamental matching problem in this category is the *House Allocation problem* (HA) [12, 123].

Informally, an instance of HA consists of a set A of *applicants* and a set H of *houses*. Each applicant has a strictly-ordered preference list over a subset of the houses that she finds acceptable. In this setting, houses do not have preferences over applicants. A *matching* M in this context is a set of (applicant, house) pairs such that a house is paired with an applicant in M only if the applicant finds the house acceptable; and, each applicant and house is involved in at most one pair. Clearly, HA is similar to SMI with the applicants representing the men and the houses representing the women, except that women have no preferences over men in the HA setting. Extensions of HA that have been studied in the literature [83] include the *House Allocation problem with Ties* (HAT) in which applicants' preference lists may include ties, the *Capacitated House Allocation problem* (CHA) in which houses can accommodate more than one applicant up to a fixed capacity, and a hybrid of HAT and CHA which is the *Capacitated House Allocation problem with Ties* (CHAT).

The HA model and its extensions arise in several practical applications for which allocating students to projects is one of them (see [83] for more applications). For bipartite matching problems where preference lists are restricted to agents in one set, the notion of stability as a desired solution concept becomes irrelevant. Other optimality criteria that have been considered in the literature include: *Pareto optimality*, where a matching is Pareto optimal if no agent can be better off without requiring another agent to be worse off; *popularity*, where a matching is popular if there is no other matching that is preferred by a majority of the agents; and *profile-based optimality*, where the profile of a matching is a vector indicating the number of agents who are assigned in the matching to their first choice, second choice, third choice, and so on. In terms of optimising the profile of a matching, several types of

optimal matching have been considered, including *rank maximal matching*, *greedy maximum matching*, and *generous maximum matching*.

Informally, a *rank maximal matching* is a matching that has *lexicographically maximum profile*, i.e., the maximum number of agents are assigned to their first choice and subject to this, the maximum number of agents are assigned to their second choice, and so on. A *greedy maximum matching* is a matching of maximum cardinality that has lexicographically maximum profile. A *generous maximum matching* is a matching of maximum cardinality whose reverse profile is lexicographically minimum, i.e., the minimum number of agents are assigned to their R-th choice (where R is the maximum length of the preference lists taken over all the agents) and subject to this, the minimum number of agents are assigned to their $(R-1)$-th choice, and so on. For a formal definition of these optimality criteria, and for algorithmic results under the HA model and its extensions, we refer the interested reader to [83].

For a given instance I of SPA, Kwanashie *et al.* [76] described an algorithm to find a greedy maximum and generous maximum matching in I, both with time complexity $O(n_1^2 Rm)$, where n_1 is the number of students, R is the maximum length of the preference lists, and m is the total length of the students' preference lists.

We now move on to another variant of SPA with one-sided preferences. We denote by SPA-LQ$_P$ an instance of SPA where each project, in addition to its capacity, has a *lower quota*, which is the minimum number of students that must be assigned to the project. Let I be an arbitrary instance of SPA-LQ$_P$, two different variants of I exist in the literature [75] with respect to the definition of a feasible solution. We give the two definitions as follows.

1. A matching M in I is a *feasible solution* if for each project p_j, either p_j meets its lower quota in M or p_j is not assigned to any student in M, i.e., p_j remains closed.

2. A matching M in I is a *feasible solution* if for each project p_j, p_j meets its lower quota in M, i.e., p_j can still run even if the number of students assigned to p_j in M is less than p_j's lower quota.

Kwanashie [75] showed that for the first definition, the problem of finding a greedy maximum or generous maximum matching in I is NP-hard, and for the second definition, a feasible solution need not exist in I.

Several other variants of SPA have been studied in the literature, many of which take account of specific problem-constraints arising within the application context. For example, Chiarandini *et al.* [31] studied the allocation of students to projects which exists at the Faculty of Science, University of Southern Denmark. In their model, students have preferences over projects, and students who want to be in the same team can register together as a group.

Also, each lecturer provides lower and upper bounds on the sizes of each team, as well as a capacity constraint on the number of teams of students she can supervise for each project. To find an allocation of students to projects, the authors [31] described a Mixed Integer Linear Programming (MILP) formulation which computes allocations that are Pareto optimal, fair, envy-free and stable.

Other centralised matching schemes that are based on different formulations of SPA include student-project allocation mechanisms at the Department of Civil and Environmental Engineering, University of Southampton [17, 48], and the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore [116]. To find optimal allocations, techniques that have been used in the literature include constraint programming [36, 118], evolutionary algorithm [115], genetic algorithm [48], goal programming [101], integer programming [17, 65, 103, 112, 119], and local search [41]. See [31] for a recent survey.

Chapter 3

Structural Result for SPA-S

3.1 Introduction

In this chapter we consider the variant of SPA where students have preferences over projects, lecturers have preferences over students, and the preference lists of students and lecturers are strictly ordered. We introduced this variant as the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S) in Section 2.3.2; see that section for the problem definition along with key algorithmic results. We recall that an arbitrary instance of SPA-S can have many stable matchings, similar to the SM and HRT settings [53]. Our goal is to characterize the structure of the set of stable matchings for an instance of SPA-S under the restriction that for each student, all of the projects in her preference list are offered by different lecturers. In the remainder of this chapter, it should be clear that any usage of SPA-S assumes this restriction.

To achieve our goal, we show that the set of stable matchings in an instance of SPA-S forms a distributive lattice with respect to a dominance relation that we will define. A similar structure holds for stable matchings in the SM and HRT settings as mentioned in Sections 2.1.2 and 2.2.3 respectively.

3.2 Preliminary definitions and results

First, we extend the notion of preferences over individuals for a student and lecturer to preferences over matchings. We recall from Theorem 2.3.2 that (i) a student who is assigned in one stable matching is assigned in all stable matchings, and (ii) a lecturer is assigned the same number of students in all stable matchings. Hence, if a student and a lecturer prefers one matching over another matching then it is clear that both the student and the lecturer are assigned in both matchings.

Students' preferences	Lecturers' preferences	offers
s_1: p_3 p_1	l_1: s_1 s_2 s_3 s_4	p_1, p_2
s_2: p_1 p_3	l_2: s_2 s_1 s_4 s_3	p_3, p_4
s_3: p_4 p_2		
s_4: p_2 p_4	Project capacities: $c_1 = c_2 = c_3 = c_4 = 1$	
	Lecturer capacities: $d_1 = d_2 = 2$	

Figure 3.1: An instance I_1 of SPA-S, due to Abraham *et al.* [13].

Let I be an arbitrary instance of SPA-S. Let $\mathcal{M}$ denote the set of all stable matchings in I and let $M, M' \in \mathcal{M}$. We say that a student s_i *prefers* M to M' if s_i is assigned in both M and M', and s_i prefers $M(s_i)$ to $M'(s_i)$. Also, we say that s_i is *indifferent between* M and M' if either (i) s_i is unassigned in both M and M' or (ii) s_i is assigned in both M and M', and $M(s_i) = M'(s_i)$. It is not immediately clear how to compare two stable matchings from the point of view of a lecturer. To see this, consider the instance I_1 of SPA-S illustrated in Figure 3.1. The reader can verify that I_1 admits two stable matchings: $M_1 = \{(s_1, p_1), (s_2, p_3), (s_3, p_2), (s_4, p_4)\}$; and $M_2 = \{(s_1, p_3), (s_2, p_1), (s_3, p_4), (s_4, p_2)\}$. However, it is not the case that in M_1, each of l_1 and l_2 is assigned the best two students that she can be assigned in any stable matching. Abraham *et al.* [13] described what it means for a lecturer to prefer one stable matching to another stable matching, which we restate as follows. Let M and M' be two stable matchings in I. For a given lecturer l_k who is not assigned identical set of students in M and M', let

$$M(l_k) \setminus M'(l_k) = \{s_1, \ldots, s_r\},$$

and let

$$M'(l_k) \setminus M(l_k) = \{s'_1, \ldots, s'_r\},$$

where in each case, the students are enumerated in the order in which they appear in $\mathcal{L}_k$. We say that l_k *prefers* M to M' if l_k prefers s_i to s'_i for all i $(1 \leq i \leq r)$. It immediately follows from this definition that if l_k prefers M to M', then l_k prefers the worst student in $M(l_k) \setminus M'(l_k)$ to the worst student in $M'(l_k) \setminus M(l_k)$. We illustrate this using the instance I_1 in Figure 3.1. With respect to the stable matchings M_1 and M_2, and considering lecturer l_1, we have that

$$M_1(l_1) \setminus M_2(l_1) = \{s_1, s_3\},$$

and

$$M_2(l_1) \setminus M_1(l_1) = \{s_2, s_4\}.$$

Since l_1 prefers s_1 to s_2, as well as s_3 to s_4, we say that l_1 prefers M_1 to M_2. Building on the definition of students' preferences over matchings, we define a dominance partial order on

the set of stable matchings in I.

Definition 3.2.1 (Dominance relation: student-oriented)**.** *Let M and M' be two stable matchings in I, we say that M* dominates *M', denoted $M \preceq M'$, if and only if each student either prefers M to M' or is indifferent between them.*

Proposition 3.2.2. *Let $\mathcal{M}$ be the set of all stable matchings in I, $\mathcal{M}$ is a partial order under the dominance relation, denoted $(\mathcal{M}, \preceq)$.*

Proof. We will show that the dominance relation $\preceq$ on $\mathcal{M}$ is (i) reflexive, (ii) antisymmetric and (iii) transitive.

(i) Let $M \in \mathcal{M}$; clearly, $M \preceq M$. Thus $\preceq$ on M is reflexive.

(ii) Let $M, M' \in \mathcal{M}$ such that $M \preceq M'$ and $M' \preceq M$. Then $M = M'$. Suppose on the contrary that $M \neq M'$. Clearly each of M and M' is non-empty. Thus there exists some student, say s_i, such that s_i is assigned in both M and M', and $M(s_i) \neq M'(s_i)$. Now, $M \preceq M'$ implies that s_i prefers $M(s_i)$ to $M'(s_i)$; and $M' \preceq M$ implies that s_i prefers $M'(s_i)$ to $M(s_i)$; a contradiction. Hence, $M = M'$; and hence $\preceq$ on $\mathcal{M}$ is antisymmetric.

(iii) Let $M, M', M'' \in \mathcal{M}$ such that $M \preceq M'$ and $M' \preceq M''$; we claim that $M \preceq M''$. We prove our claim as follows. First, we know that exactly the same students are unassigned in all stable matchings (from Theorem 2.3.2). Thus every student who is unassigned in M is unassigned in M''; and thus every unassigned student is indifferent between M and M''. Clearly, every student who is assigned to the same project in M and M'' is indifferent between M and M''. Now, let s_i be a student such that s_i is assigned in both M and M'', and suppose $M(s_i) \neq M''(s_i)$. First, suppose $M(s_i) \neq M'(s_i)$; since $M \preceq M'$, it follows that s_i prefers $M(s_i)$ to $M'(s_i)$. Further, (a) if $M'(s_i) = M''(s_i)$ then s_i prefers $M(s_i)$ to $M''(s_i)$; and (b) if $M'(s_i) \neq M''(s_i)$, $M' \preceq M''$ implies that s_i prefers $M'(s_i)$ to $M''(s_i)$, and since the preference lists are strictly ordered, s_i prefers $M(s_i)$ to $M''(s_i)$. Now, suppose $M(s_i) = M'(s_i)$. It follows that $M'(s_i) \neq M''(s_i)$; thus $M' \preceq M''$ implies that s_i prefers $M'(s_i)$ to $M''(s_i)$. This implies that s_i prefers $M(s_i)$ to $M''(s_i)$. Hence our claim holds; and hence $\preceq$ on $\mathcal{M}$ is transitive.

□

Students' preferences	Lecturers' preferences	offers
s_1: p_1 p_2	l_1: s_3 s_1 s_2 s_4	p_1, p_2
s_2: p_2 p_3	l_2: s_2 s_4 s_3	p_3, p_4
s_3: p_3 p_1		
s_4: p_4 p_1	Project capacities: $c_1 = c_2 = c_3 = c_4 = 1$	
	Lecturer capacities: $d_1 = d_2 = 2$	

Figure 3.2: An instance I_2 of SPA-S in the general case, where s_2 ranks two projects (i.e., p_2 and p_3) that are offered by different lecturers (i.e., l_1 and l_2 respectively).

3.3 Stable matchings in SPA-S form a distributive lattice

To establish the structure of the set of stable matchings in I, our aim is to show that $(\mathcal{M}, \preceq)$ is a distributive lattice. We define this concept in what follows.

Definition 3.3.1 (Distributive lattice [44])**.** *Let A be a set and let $\preceq$ be an ordering relation defined on A. We say that the partial order $(A, \preceq)$ is a* distributive lattice *if:*

(i) each pair of element $x, y \in A$ has a greatest lower bound, or meet*, denoted $x \wedge y$, such that $x \wedge y \preceq x$, $x \wedge y \preceq y$, and there is no element $z \in A$ for which $z \preceq x$, $z \preceq y$ and $x \wedge y \preceq z$;*

(ii) each pair of element $x, y \in A$ has a least upper bound, or join*, denoted $x \vee y$, such that $x \preceq x \vee y$, $y \preceq x \vee y$, and there is no element $z \in A$ for which $x \preceq z$, $y \preceq z$ and $z \preceq x \vee y$;*

(iii) the join and meet distribute over each other, i.e., for $x, y, z \in A$, $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

The next assumption follows from our restriction that for each student s_i in I, all the projects in s_i's preference list are offered by different lecturers.

Assumption 3.3.2. *Let M and M' be two stable matchings in I. If a student is assigned to two different projects in M and M' then the two projects are offered by different lecturers.*

Initially, our aim was to show that the lattice structure holds for stable matchings in an arbitrary instance of SPA-S in the general case where there are no restrictions on the problem instance. To achieve this, we assumed that Assumption 3.3.2 holds and we used it as a tool in the proof of the lemmas that lead to the establishment of the structure. However, Bettina Klaus observed that Assumption 3.3.2 is not true in general, as can be seen in the instance I_2 in Figure 3.2. The stable matchings in I_2 are $M_1 = \{(s_1, p_1), (s_2, p_2), (s_3, p_3), (s_4, p_4)\}$

and $M_2 = \{(s_1, p_2), (s_2, p_3), (s_3, p_1), (s_4, p_4)\}$; and clearly, s_1 is assigned to p_1 and p_2 in M_1 and M_2 respectively, and these projects are both offered by l_1. We were unable to prove the results in this section without Assumption 3.3.2 holds. Hence, our reason for adapting this restriction for our characterisation.

Next, we present some lemmas which will serve as a tool in our characterisation.

Lemma 3.3.3. *Let M and M' be two stable matchings in I. Let l_k be a lecturer such that $M(l_k) \neq M'(l_k)$. Then either l_k prefers M to M' or l_k prefers M' to M.*

Proof. Let l_k be a lecturer in I for which $M(l_k) \neq M'(l_k)$. This implies that l_k is not assigned an identical set of students in M and M'. Let $M(l_k) \setminus M'(l_k) = \{s_1, \ldots, s_r\}$ and let $M'(l_k) \setminus M(l_k) = \{s'_1, \ldots, s'_r\}$, where in each case, the students are enumerated in the order in which they appear in $\mathcal{L}_k$. In order to prove the lemma, it suffices to show that either (a) for all i ($1 \leq i \leq r$) l_k prefers s_i to s'_i or (b) for all i ($1 \leq i \leq r$), l_k prefers s'_i to s_i. Suppose otherwise. Then, without loss of generality, there exists some q ($2 \leq q \leq r$) such that l_k prefers s'_q to s_q, and for every $i < q$, l_k prefers s_i to s'_i. For this to hold, we note that $|M(l_k) \setminus M'(l_k)| = |M'(l_k) \setminus M(l_k)| \geq 2$.

First, suppose $M' \preceq M$ (i.e., each student who is assigned to different projects in M and M' prefers her assigned project in M' to her assigned project in M). Let $(s'_q, p_{j_1}) \in M' \setminus M$, where p_{j_1} is offered by l_k. By the stability of M, either (i) or (ii) holds as follows:

(i) p_{j_1} is full in M and l_k prefers the worst student in $M(p_{j_1})$ to s'_q;

(ii) p_{j_1} is undersubscribed in M, l_k is full in M, and l_k prefers the worst student in $M(l_k)$ to s'_q.

Otherwise (s'_q, p_{j_1}) blocks M. However, we may ignore case (ii) in view of the fact that $s_q \in M(l_k)$ and l_k prefers s'_q to s_q. Hence, case (i) applies. Now, since p_{j_1} is full in M and $(s'_q, p_{j_1}) \in M' \setminus M$, there exists some student, say s_{z_1}, such that $(s_{z_1}, p_{j_1}) \in M \setminus M'$. We remark that $z_1 < q$; also, l_k prefers s_{z_1} to s'_q. Moreover, by Assumption 3.3.2, $s_{z_1} \notin M'(l_k)$. We have identified $s_{z_1} \in M(l_k) \setminus M'(l_k)$ where $z_1 < q$. Now, since $|M(l_k) \setminus M'(l_k)| = |M'(l_k) \setminus M(l_k)|$, there exists some student, say s'_{z_2}, such that $s'_{z_2} \in M'(l_k) \setminus M(l_k)$, where $z_2 < q$. This implies that l_k prefers s'_{z_2} to s'_q .

Again, let $(s'_{z_2}, p_{j_2}) \in M' \setminus M$, where p_{j_2} is offered by l_k. By the stability of M, p_{j_2} is full in M and l_k prefers every student in $M(p_{j_2})$ to s'_{z_2}; otherwise, (s'_{z_2}, p_{j_2}) blocks M. We note that since l_k prefers s'_{z_2} to a student in $M(l_k)$, namely s_q, we may again ignore case (ii). Now let s_{z_3} be a student such that $(s_{z_3}, p_{j_2}) \in M \setminus M'$. Then l_k prefers s_{z_3} to s'_{z_2}. Moreover, by Assumption 3.3.2, $s_{z_3} \notin M'(l_k)$. Again, since $|M(l_k) \setminus M'(l_k)| = |M'(l_k) \setminus M(l_k)|$, there is a student $s'_{z_4} \in M'(l_k) \setminus M(l_k)$, where $z_4 < z_2$. This implies that l_k prefers s'_{z_4} to s'_{z_2}.

We continue in this way to generate a sequence $\langle s'_{z_{2t}} \rangle_{t \geq 1}$ of distinct students, such that: for $1 \leq t < q$, $s'_{z_{2t}} \in M'(l_k) \setminus M(l_k)$; and for $2 \leq t < q$, l_k prefers $s'_{z_{2t}}$ to $s'_{z_{2t-2}}$, also, l_k prefers s'_{z_2} to s'_q. Additionally, we need to ensure that, if the same project is assigned to more than one of the students in the sequence $\langle s'_{z_{2t}} \rangle_{t \geq 1}$, we can choose a unique student who is assigned to that project in $M \setminus M'$ on each occasion. For, suppose p_u is such project that arises x times from pairs in $M' \setminus M$. Following our argument from above, p_u must be full in M with students whom l_k prefers to all of those x students. Clearly, there must be at least x such students in $M(p_u) \setminus M'(p_u)$ from which to choose. Finally, as the sequence of distinct students is finite, we reach an immediate contradiction.

Next, suppose $M \preceq M'$ (i.e., each student who is assigned to different projects in M and M' prefers her assigned project in M to her assigned project in M'). Let $(s_1, p_{j_1}) \in M \setminus M'$. We note that s_1 prefers p_{j_1} to $M'(s_1)$. Again, by the stability of M', either (i) or (ii) holds as follows:

(i) p_{j_1} is full in M' and l_k prefers the worst student in $M'(p_{j_1})$ to s_1;

(ii) p_{j_1} is undersubscribed in M', l_k is full in M', and l_k prefers the worst student in $M'(l_k)$ to s_1.

Otherwise, (s_1, p_{j_1}) will form a blocking pair for M'. We reach an immediate contradiction in cases (i) and (ii), since l_k prefers s_1 to every student in $M'(l_k) \setminus M(l_k)$. □

Lemma 3.3.4. *Let M and M' be two stable matchings in I. Suppose that student s_i is assigned to lecturer l_k in M but not in M'. If s_i prefers M to M' then l_k prefers M' to M. Likewise, if s_i prefers M' to M then l_k prefers M to M'.*

Proof. Let S (respectively S') denote the set of students who prefer M to M' (respectively M' to M). We note that $S \cap S' = \emptyset$. Similarly, let L (respectively L') denote the set of lecturers who prefer M to M' (respectively M' to M). Again, $L \cap L' = \emptyset$. Let s_i be a student assigned to a lecturer l_k in M but not in M', i.e., $s_i \in M(l_k) \setminus M'(l_k)$. In order to prove the lemma, it suffices to show that (a) if $s_i \in S$ then $l_k \in L'$, and (b) if $s_i \in S'$ then $l_k \in L$.

To show (a). Suppose otherwise, i.e., $s_i \in S$ and $l_k \notin L'$. Then l_k does not prefer M' to M, which, by Lemma 3.3.3, implies that l_k prefers M to M'. Hence $l_k \in L$. It follows that l_k prefers the worst student in $M(l_k) \setminus M'(l_k)$ to the worst student in $M'(l_k) \setminus M(l_k)$. Moreover, either s_i is the worst student in $M(l_k) \setminus M'(l_k)$ or l_k prefers s_i to the worst student in $M(l_k) \setminus M'(l_k)$. In any case, we have that l_k prefers s_i to the worst student in $M'(l_k) \setminus M(l_k)$. Now, $s_i \in S$ implies that s_i prefers $M(s_i)$ to $M'(s_i)$; and by Assumption 3.3.2, $s_i \notin M'(l_k)$.

Let $l_{z_0} = l_k, s_{q_0} = s_i$, $p_{t_0} = M'(s_{q_0})$ and $p_{t_1} = M(s_{q_0})$. By the stability of M', p_{t_1} is full in M' and l_{z_0} prefers the worst student in $M'(p_{t_1})$ to s_{q_0}; for otherwise, (s_{q_0}, p_{t_1}) will form

a blocking pair for M'. Now, since $(s_{q_0}, p_{t_1}) \in M \setminus M'$, and since p_{t_1} is full in M', there exists some student, say s_{q_1}, such that $(s_{q_1}, p_{t_1}) \in M' \setminus M$; for otherwise, p_{t_1} would be oversubscribed in M. Clearly, l_{z_0} prefers s_{q_1} to s_{q_0}; and thus, $s_{q_1} \neq s_{q_0}$. Let $M(s_{q_1}) = p_{t_2}$. We note that $p_{t_1} \neq p_{t_2}$ since $(s_{q_1}, p_{t_2}) \in M$ and $(s_{q_1}, p_{t_1}) \notin M$. Moreover, s_{q_1} prefers p_{t_2} to p_{t_1}; for otherwise, (s_{q_1}, p_{t_1}) will form a blocking pair for M. Let l_{z_1} be the lecturer who offers p_{t_2}. Again, by the stability of M', either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M' and l_{z_1} prefers the worst student in $M'(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M', l_{z_1} is full in M', $s_{q_1} \notin M'(l_{z_1})$, and l_{z_1} prefers the worst student in $M'(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M'. In case (i), there exists some student, say s_{q_2}, such that $(s_{q_2}, p_{t_2}) \in M' \setminus M$; for otherwise, p_{t_2} would be oversubscribed in M. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M'(l_{z_1}) \setminus M(l_{z_1})$. Let $M'(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$, since l_{z_1} prefers s_{q_2} to s_{q_1}. Applying similar reasoning as for s_{q_1}, s_{q_2} is assigned in M a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M'$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

First we claim that for each new project that we identify, $p_{t_{2i}} \neq p_{t_{2i-1}}$ for $i \geq 1$. Suppose $p_{t_{2i}} = p_{t_{2i-1}}$ for some $i \geq 1$. From above s_{q_i} was identified by $l_{z_{i-1}}$ such that $(s_{q_i}, p_{t_{2i-1}}) \in M' \setminus M$. Moreover $(s_{q_i}, p_{t_{2i}}) \in M$. Hence we reach a contradiction. Clearly, for each student s_{q_i} that we identify, for $i \geq 1$, s_{q_i} must be assigned to distinct projects in M and in M'.

Next we claim that for each new student s_{q_i} that we identify, $s_{q_i} \neq s_{q_t}$ for $1 \leq t < i$. We prove this by induction on i. For the base case, clearly $s_{q_2} \neq s_{q_1}$. We assume that the claim holds for some $i \geq 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}$ consists of distinct students. We show that the claim holds for $i + 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}, s_{q_{i+1}}$ also consists of distinct students. Clearly $s_{q_{i+1}} \neq s_{q_i}$ since l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}. Thus, it suffices to show that $s_{q_{i+1}} \neq s_{q_j}$ for $1 \leq j \leq i - 1$. Now, suppose $s_{q_{i+1}} = s_{q_j}$ for $1 \leq j \leq i - 1$. This implies that s_{q_j} was identified by l_{z_i} and clearly l_{z_i} prefers s_{q_j} to $s_{q_{j-1}}$. Now since $s_{q_{i+1}}$ was also identified by l_{z_i} to avoid the blocking pair $(s_{q_i}, p_{t_{2i}})$ in M', it follows that either (i) $p_{t_{2i}}$ is full in M' or (ii) $p_{t_{2i}}$ is undersubscribed in M' and l_{z_i} is full in M'. We consider each cases further as follows.

(i) If $p_{t_{2i}}$ is full in M', we know that $(s_{q_i}, p_{t_{2i}}) \in M \setminus M'$. Moreover s_{q_j} was identified by $l_{z_{i+1}}$ because of case (i). Furthermore $(s_{q_{j-1}}, p_{t_{2i}}) \in M \setminus M'$. In this case, $p_{t_{2i+1}} = p_{t_{2i}}$ and we have that

$$(s_{q_i}, p_{t_{2i+1}}) \in M \setminus M' \text{ and } (s_{q_{i+1}}, p_{t_{2i+1}}) \in M' \setminus M,$$

$$(s_{q_{j-1}}, p_{t_{2i+1}}) \in M \setminus M' \text{ and } (s_{q_j}, p_{t_{2i+1}}) \in M' \setminus M.$$

By the inductive hypothesis, the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_{j-1}}, s_{q_j}, \ldots, s_{q_i}$ consists of distinct students. This implies that $s_{q_i} \neq s_{q_{j-1}}$. Thus since $p_{t_{2i+1}}$ is full in M', l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ to avoid the blocking pairs $(s_{q_{j-1}}, p_{t_{2i+1}})$ and $(s_{q_i}, p_{t_{2i+1}})$ respectively in M', a contradiction.

(ii) $p_{t_{2i}}$ is undersubscribed in M' and l_{z_i} is full in M'. Similarly as in case (i) above, we have that

$$s_{q_i} \in M(l_{z_i}) \setminus M'(l_{z_i}) \text{ and } s_{q_{i+1}} \in M'(l_{z_i}) \setminus M(l_{z_i}),$$

$$s_{q_{j-1}} \in M(l_{z_i}) \setminus M'(l_{z_i}) \text{ and } s_{q_j} \in M'(l_{z_i}) \setminus M(l_{z_i}).$$

Since $s_{q_i} \neq s_{q_{j-1}}$ and l_{z_i} is full in M', l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ corresponding to students $s_{q_{j-1}}$ and s_{q_i} respectively, a contradiction.

This completes the induction step. As the sequence of distinct students and projects we are identifying is infinite, we reach an immediate contradiction. Hence, $s_i \in S$ implies that $l_k \in L'$, and it follows that

$$|S| \leq \sum_{l_k \in L'} |M(l_k) \setminus M'(l_k)|. \tag{3.1}$$

This completes the proof of (a). Next we show that (b) holds. Suppose there is a student $s_{i'} \in M'(l_{k'}) \setminus M(l_{k'})$. As shown above, if $s_{i'}$ prefers M' to M then $l_{k'}$ prefers M to M', and it follows that

$$|S'| \leq \sum_{l_{k'} \in L} |M'(l_{k'}) \setminus M(l_{k'})|. \tag{3.2}$$

But

$$|S| + |S'| = \sum_{l_k \in L'} |M(l_k) \setminus M'(l_k)| + \sum_{l_{k'} \in L} |M'(l_{k'}) \setminus M(l_{k'})|, \tag{3.3}$$

since each side represents the number of students that are assigned to different lecturers in M and in M'. Clearly, this is true for the LHS. It is also true for the RHS, because as already

observed, every lecturer that is not assigned identical sets of students in M and M' is either in L or in L' (but not both). From Inequalities 3.1 and 3.2, it follows that

$$|S| = \sum_{l_k \in L'} |M(l_k) \setminus M'(l_k)| \text{ and } |S'| = \sum_{l_{k'} \in L} |M'(l_{k'}) \setminus M(l_{k'})|.$$

Therefore, in M, every student in S' is assigned to a lecturer in L. Hence (b) holds. □

We begin the demonstration that $(\mathcal{M}, \preceq)$ is a distributive lattice by presenting two lemmas that will provide the foundations for the definitions of the "meet" and "join" operations for two stable matchings in I.

Lemma 3.3.5. *Let M and M' be two stable matchings in I. Let M^* be a set of (student, project) pairs formed as follows: for each student s_i, s_i is unassigned in M^* if she is unassigned in both M and M', otherwise s_i is assigned to the better of her projects in M and M'. Then M^* is a stable matching in I.*

Proof. Firstly, we show that M^* is a matching. It is clear from the hypothesis of the lemma that no student is multiply assigned. Suppose that there exists some project p_j such that p_j is oversubscribed in M^*. This implies that $M(p_j) \neq M'(p_j)$. Now, as p_j is oversubscribed in M^*, there exists some $s_i \in M^*(p_j) \setminus M'(p_j)$; thus $s_i \in M(p_j)$. Also, since p_j has at least c_j students in M^*, there exists some $s_{i'} \in M^*(p_j)$ such that $s_{i'} \in M'(p_j) \setminus M(p_j)$. Now, $(s_i, p_j) \in M \setminus M'$ implies that s_i prefers M to M', and $(s_{i'}, p_j) \in M' \setminus M$ implies that $s_{i'}$ prefers M' to M, since each of s_i and $s_{i'}$ is assigned in M^* to the better of her project in M and M'. Let l_k be the lecturer who offers p_j. By Assumption 3.3.2, we have that (a) $s_i \in M(l_k) \setminus M'(l_k)$ and (b) $s_{i'} \in M'(l_k) \setminus M(l_k)$. However, Lemma 3.3.4 applied to (a) implies that l_k prefers M' to M, and applied to (b) implies that l_k prefers M to M'. This gives a contradiction. Hence no project can be oversubscribed in M^*.

Next, suppose that there exists some lecturer l_k such that l_k is oversubscribed in M^*. As l_k is oversubscribed in M^*, there exists some $s_i \in M^*(l_k) \setminus M'(l_k)$; thus $s_i \in M(l_k)$. Further, since l_k has at least d_k students in M^*, there exists some student $s_{i'} \in M^*(l_k)$ such that $s_{i'} \in M'(l_k) \setminus M(l_k)$. Let $M^*(s_i) = p_j$ and let $M^*(s_{i'}) = p_{j'}$, with $p_j, p_{j'} \in P_k$. Then s_i prefers p_j to $M'(s_i)$ and $s_{i'}$ prefers $p_{j'}$ to $M(s_{i'})$. It follows that s_i prefers M to M' and s_i prefers M' to M. A similar argument follows from above, and thus we reach a contradiction. Hence M^* is a matching.

Next, we show that M^* is stable. Suppose that (s_i, p_j) forms a blocking pair for M^*, where l_k is the lecturer who offers p_j. Firstly, suppose that s_i is unassigned in M^*. By the hypothesis of the lemma, we have that s_i is unassigned in both M and M'. Now, since (s_i, p_j) forms a blocking pair for M^*, either (i) or (ii) holds as follows:

(i) p_j is full in M^* and l_k prefers s_i to the worst student in $M^*(p_j)$;

(ii) p_j is undersubscribed in M^*, l_k is full in M^*, and l_k prefers s_i to the worst student in $M^*(l_k)$.

In case (i), suppose $s_{i'}$ is the worst student in $M^*(p_j)$. If $s_{i'} \in M(p_j)$, then l_k prefers s_i to the worst student in $M(p_j)$; and (s_i, p_j) blocks M, a contradiction. Now, suppose that $s_{i'} \in M'(p_j) \setminus M(p_j)$. Then there exists some student $s_{q_1} \in M(p_j) \setminus M'(p_j)$; for otherwise p_j would be oversubscribed in M'. Since M is stable, p_j is full in M and l_k prefers every student in $M(p_j)$ to s_i. Hence, l_k prefers s_{q_1} to s_i; and thus, l_k prefers s_{q_1} to $s_{i'}$. Moreover, s_{q_1} is assigned in M' to a project p_{t_1} such that s_{q_1} prefers p_{t_1} to p_j; for otherwise (s_{q_1}, p_j) would block M'. Let l_{z_1} be the lecturer that offers p_{t_1}. Following a similar argument as in the proof of Lemma 3.3.4, we can identify a sequence of distinct students and projects, and as this sequence is infinite, we reach a contradiction.

In case (ii), suppose $s_{i'}$ is the worst student in $M^*(l_k)$. If $s_{i'} \in M(l_k)$, then l_k prefers s_i to the worst student in $M(l_k)$; and (s_i, p_j) blocks M, a contradiction. If $s_{i'} \in M'(l_k) \setminus M(l_k)$, then there exists some student $s_{q_1} \in M(l_k) \setminus M'(l_k)$; for otherwise l_k would be oversubscribed in M'. Since M is stable, l_k is full in M and l_k prefers every student in $M(l_k)$ to s_i. Hence, l_k prefers s_{q_1} to s_i; and thus, l_k prefers s_{q_1} to $s_{i'}$. Suppose $M(s_{q_1}) = p_{t_0}$ (possibly $p_{t_0} = p_j$). We have that s_{q_1} is assigned in M' to a project p_{t_1} such that s_{q_1} prefers p_{t_1} to p_{t_0}; for otherwise (s_{q_1}, p_{t_0}) would block M'. Let l_{z_1} be the lecturer that offers p_{t_1}. Again, we will need to identify a sequence of distinct students and projects, a contradiction.

Now, suppose that s_i is assigned to a project in M^*. This implies that s_i is assigned to a project in both M and M^*. Since (s_i, p_j) forms a blocking pair for M^*, it follows that s_i prefers p_j to both $M(s_i)$ and $M'(s_i)$ (possibly $M(s_i) = M'(s_i)$). Further, we have that either (i) or (ii) holds as follows:

(i) p_j is full in M^* and l_k prefers s_i to the worst student in $M^*(p_j)$

(ii) p_j is undersubscribed in M^*, l_k is full in M^*, and either $s_i \in M^*(l_k)$ or l_k prefers s_i to the worst student in $M^*(l_k)$.

In case (i), suppose $s_{i'}$ is the worst student in $M^*(p_j)$. If $s_{i'} \in M(p_j)$, then l_k prefers s_i to the worst student in $M(p_j)$; and (s_i, p_j) blocks M, a contradiction. We arrive at a similar contradiction if $s_{i'} \in M'(p_j)$. In case (ii), suppose first that $s_i \in M^*(l_k)$. If $s_i \in M(l_k)$, then (s_i, p_j) blocks M; and if $s_i \in M'(l_k)$, then (s_i, p_j) blocks M', a contradiction. Hence $s_i \notin M^*(l_k)$. Now suppose $s_{i'}$ is the worst student in $M^*(l_k)$. If $s_{i'} \in M(l_k)$, then l_k prefers s_i to the worst student in $M(l_k)$; and (s_i, p_j) blocks M, a contradiction. We arrive at a similar contradiction if $s_{i'} \in M'(l_k)$. Hence M^* is stable. □

We denote by $M \wedge M'$ the set of (student, project) pairs in which each student is assigned the better of her projects in M and M'; and it follows from Lemma 3.3.5 that $M \wedge M'$ is a stable matching. It also follows from Lemma 3.3.5 that if each student is given the better of her projects in any fixed set of stable matchings, then the resulting assignment is a stable matching. For the case where $\mathcal{M}$ is the set of all stable matchings in I, we denote by $\bigwedge_{M \in \mathcal{M}} M$, or simply $\bigwedge \mathcal{M}$, the resulting stable matching, which, obviously, is student-optimal, and thus lecturer-pessimal.

Lemma 3.3.6. *Let M and M' be two stable matchings in I. Let M^* be a set of (student, project) pairs formed as follows: for each student s_i, s_i is unassigned in M^* if she is unassigned in both M and M', otherwise s_i is assigned to the poorer of her projects in M and M'. Then M^* is a stable matching in I.*

Proof. By the hypothesis of the lemma, clearly no student is multiply assigned. Suppose that there exists some project p_j such that p_j is oversubscribed in M^*. Following a similar argument as in the proof of Lemma 3.3.5, there exists two students s_i and $s_{i'}$ such that $s_i \in M^*(p_j) \setminus M'(p_j)$ and $s_{i'} \in M^*(p_j) \setminus M(p_j)$. So we have that $s_i \in M(p_j)$ and $s_{i'} \in M'(p_j)$. Now, since each student is assigned in M^* to the poorer of her projects in M and M', it follows that s_i prefers M' to M and $s_{i'}$ prefers M to M'. Let l_k be the lecturer who offers p_j. Again, by Assumption 3.3.2, we have that (a) $s_i \in M(l_k) \setminus M'(l_k)$ and (b) $s_{i'} \in M'(l_k) \setminus M(l_k)$. However, Lemma 3.3.4 applied to (a) implies that l_k prefers M to M', and applied to (b) implies that l_k prefers M' to M. This gives a contradiction. Hence no project can be oversubscribed in M^*.

Next, suppose that there exists some lecturer l_k such that l_k is oversubscribed in M^*. Similar to above, there exists two students s_i and $s_{i'}$ such that $s_i \in M^*(l_k) \setminus M'(l_k)$ and $s_{i'} \in M^*(l_k) \setminus M(l_k)$. So we have that $s_i \in M(l_k)$ and $s_{i'} \in M'(l_k)$. Let $M^*(s_i) = p_j$ and let $M^*(s_{i'}) = p_{j'}$, with $p_j, p_{j'} \in P_k$. Again, since each student is assigned in M^* to the poorer of her projects in M and M', clearly s_i prefers $M'(s_i)$ to p_j, and thus s_i prefers M' to M. Similarly, $s_{i'}$ prefers M to M'. Following a similar argument as above, we reach a contradiction. Hence M^* is a matching.

Next, we show that M^* is stable. Suppose that (s_i, p_j) forms a blocking pair for M^*, where l_k is the lecturer who offers p_j. Firstly, suppose that s_i is unassigned in M^*. Then we arrive at a similar contradiction as in the proof of Lemma 3.3.5 (paragraph 3). Now, suppose that s_i is assigned to a project p_{j^*} in M^*. First, suppose $(s_i, p_{j^*}) \in M$, and let $M'(s_i) = p_{j'}$. Since (s_i, p_j) forms a blocking pair for M^*, we have the following three possibilities:

1. s_i prefers p_j to both $p_{j'}$ and p_{j^*} (possibly $p_{j'} = p_{j^*}$);
2. $p_{j'} = p_j$ and s_i prefers p_j to p_{j^*};
3. s_i prefers $p_{j'}$ to p_j to p_{j^*}.

Suppose firstly that (1) holds. Then we arrive at a similar contradiction as in the proof of Lemma 3.3.5 (paragraph 4). Next, suppose (2) or (3) holds. We have that s_i is assigned to different projects in M and M'. By Assumption 3.3.2, it follows that p_{j^*} cannot be offered by l_k, and thus $s_i \notin M^*(l_k)$. Now, since (s_i, p_j) forms a blocking pair for M^*, either (i) or (ii) holds as follows:

(i) p_j is full in M^* and l_k prefers s_i to the worst student in $M^*(p_j)$;
(ii) p_j is undersubscribed in M^*, l_k is full in M^*, and l_k prefers s_i to the worst student in $M^*(l_k)$.

In cases (i) and (ii), we arrive at a similar contradiction as in the proof of Lemma 3.3.5 (paragraph 3). We remark that if $(s_i, p_{j^*}) \in M'$, the argument is similar. Hence M^* is stable. □

We denote by $M \vee M'$ the set of (student, project) pairs in which each student is assigned the poorer (i.e., least-preferred) of her projects in M and M'; and it follows from Lemma 3.3.6 that $M \vee M'$ is a stable matching. It also follows from Lemma 3.3.6 that if each student is given the poorer of her projects in any fixed set of stable matchings, then the resulting assignment is a stable matching. For the case where $\mathcal{M}$ is the set of all stable matchings in I, we denote by $\bigvee_{M \in \mathcal{M}} M$, or simply $\bigvee \mathcal{M}$, the resulting stable matching, which, obviously, is student-pessimal, and thus lecturer-optimal.

Theorem 3.3.7. *Let I be an instance of* SPA-S*, and let $\mathcal{M}$ be the set of stable matchings in I. Let $\preceq$ be the dominance partial order on $\mathcal{M}$ and let $M, M' \in \mathcal{M}$. Then $(\mathcal{M}, \preceq)$ is a distributive lattice, with $M \wedge M'$ representing the meet of M and M', and $M \vee M'$ the join of M and M'.*

Proof. Let M and M' be two stable matchings in $\mathcal{M}$. By Lemma 3.3.5, we have that $M \wedge M'$ is a stable matching; and by the definition of $M \wedge M'$, it follows that $M \wedge M' \preceq M$ and $M \wedge M' \preceq M'$. Further if M^* is an arbitrary stable matching satisfying $M^* \preceq M$ and $M^* \preceq M'$, then each student must be assigned in M^* to a project that is at least as good as her assigned project in each of M and M', so that $M^* \preceq M \wedge M'$. Thus $M \wedge M'$ is the meet of M and M'. Similarly, by Lemma 3.3.6, we have that $M \vee M'$ is a stable matching; and by the definition of $M \vee M'$, it follows that $M \preceq M \vee M'$ and $M' \preceq M \vee M'$. Further, by a similar argument to above, $M \vee M'$ is the join of M and M'. Hence $(\mathcal{M}, \preceq)$ is a lattice.

Next, we show that the join and meet operation distribute over each other. Let M, M' and M'' be stable matchings in $\mathcal{M}$. First, let $X = M \vee (M' \wedge M'')$ and let $Y = (M \vee M') \wedge (M \vee M'')$; we need to show that $X = Y$. Let s_i be an arbitrary student. If s_i is unassigned in each of M, M' and M'', it is clear that s_i is unassigned in both X and Y. Now, suppose s_i is assigned to some project in each of M, M' and M''. We consider the following cases.

(i) If $M(s_i) = M'(s_i) = M''(s_i)$, clearly $X(s_i) = Y(s_i)$.

(ii) Suppose that either (a) $M(s_i) = M'(s_i)$ and $M(s_i) \neq M''(s_i)$ or (b) $M(s_i) \neq M'(s_i)$ and $M(s_i) = M''(s_i)$ holds. Irrespective of how we express s_i's preference over $M(s_i), M'(s_i)$ and $M''(s_i)$ in cases (a) and (b), we have that s_i is assigned to $M(s_i)$ in both X and Y.

(iii) If $M'(s_i) = M''(s_i)$ and $M'(s_i) \neq M(s_i)$. If s_i prefers $M'(s_i)$ to $M(s_i)$ then s_i is assigned to $M(s_i)$ in both X and Y. Otherwise, if s_i prefers $M(s_i)$ to $M'(s_i)$ then s_i is assigned to $M'(s_i)$ in both X and Y.

(iv) Suppose that $M(s_i)$, $M'(s_i)$ and $M''(s_i)$ are distinct projects. There are six different ways to express s_i's preference over $M(s_i), M'(s_i)$ and $M''(s_i)$. If s_i prefers $M(s_i)$ to $M'(s_i)$ to $M''(s_i)$ then s_i is assigned to $M'(s_i)$ in both X and Y. If s_i prefers $M(s_i)$ to $M''(s_i)$ to $M'(s_i)$ then s_i is assigned to $M''(s_i)$ in both X and Y. We leave it to the reader to verify that in the remaining four cases, s_i is assigned to $M(s_i)$ in both X and Y.

Since s_i is an arbitrary student, it follows that $X = Y$; and thus the first distributive property holds. Next, we show that the second distributive property holds. Let $X = M \wedge (M' \vee M'')$ and let $Y = (M \wedge M') \vee (M \wedge M'')$. Let s_i be an arbitrary student. Again, if s_i is unassigned in each of M, M' and M'', it is clear that s_i is unassigned in both X and Y. Now, suppose s_i is assigned to some project in each of M, M' and M''. Following the same case analysis as above, we arrive at the same conclusion in cases (i) and (ii). We consider cases (iii) and (iv) below.

(iii) If $M'(s_i) = M''(s_i)$ and $M'(s_i) \neq M(s_i)$. If s_i prefers $M'(s_i)$ to $M(s_i)$ then s_i is assigned to $M'(s_i)$ in both X and Y. Otherwise, if s_i prefers $M(s_i)$ to $M'(s_i)$ then s_i is assigned to $M(s_i)$ in both X and Y.

(iv) Suppose that $M(s_i)$, $M'(s_i)$ and $M''(s_i)$ are distinct projects. Again, in this case, there are six different ways to express s_i's preference over $M(s_i), M'(s_i)$ and $M''(s_i)$. If s_i prefers $M'(s_i)$ to $M''(s_i)$ to $M(s_i)$ then s_i is assigned to $M''(s_i)$ in both X and Y. Further, if s_i prefers $M''(s_i)$ to $M'(s_i)$ to $M(s_i)$ then s_i is assigned to $M'(s_i)$ in both X and Y. Again, we leave it to the reader to verify that in the remaining four cases, s_i is assigned to $M(s_i)$ in both X and Y.

Again, since s_i is an arbitrary student, it follows that $X = Y$; and thus the second distributive property holds. Since each of M, M' and M'' is an arbitrary stable matching in $\mathcal{M}$, it follows that $(\mathcal{M}, \preceq)$ is a distributive lattice. □

Example 3.3.8. We illustrate some of the results we have presented in this section, with respect to the SPA-S instance I_3 shown in Figure 3.3. This instance involves the set of students $\mathcal{S} = \{s_i : 1 \leq i \leq 12\}$, the set of projects $\mathcal{P} = \{p_j : 1 \leq j \leq 6\}$ and the set of lecturers $\mathcal{L} = \{l_k : 1 \leq k \leq 4\}$. The instance admits a total of nine stable matchings, as illustrated in Figure 3.4. The lattice structure representing these stable matchings is illustrated in Figure 3.5.

Similar to Example 2.2.3 presented in the HR seting, the Hasse diagram illustrated in Figure 3.5 is a directed graph with each vertex representing a stable matching, and there is a directed edge from vertex M to M' if $M \preceq M'$ and there is no such M^* such that $M \preceq M^* \preceq M'$. The reader can easily verify the meet and join operations with respect to the stable matchings, for example, $M_1 = M_2 \wedge M_3$ (i.e., in M_1, each student is assigned the better of her projects in M_2 and M_3) and $M_7 = M_6 \vee M_4$ (i.e., in M_7, each student is assigned the poorer of her projects in M_6 and M_4). We note that all the edges representing precedence implied by transitivity are suppressed in the diagram.

3.4 Conclusions and open problems

In this chapter we have characterised the stable matchings in an instance I of SPA-S, under the assumption that the projects in each student's preference list are offered by different lecturers. We achieved this characterisation by showing that the set of stable matchings in I is a distributive lattice under the dominance relation order.

An obvious open question is: can we characterise the stable matchings in an instance of SPA-S in the general case, where there are no restrictions on the problem instance? Our initial thought is that it may be difficult to provide this characterisation, in terms of the lattice structure, by merely adapting the definitions outlined in this chapter. However, establishing new definitions, for example, redefining lecturer preferences over matchings, or redefining the "meet" and "join" operations, may yield positive results. If these ideas do not yield fruitful results, can we come up with a counterexample to show that the set of stable matchings in an instance of SPA-S does not admit a lattice structure? Such counterexample could be an instance of SPA-S with two distinct stable matchings M and M' (where a student is assigned to two different projects offered by the same lecturer in M and M') such that if we form another matching M^* by giving each assigned student the better of her projects in M and M', then either M^* is not stable or M^* does not dominate both M and M'.

Another interesting direction is to investigate the relationship between the rotation posets and the set of stable matchings in an instance of SPA-S. To explain what *rotations* are informally, consider the stable matchings in the SM instance in Figure 2.1, with stable matchings $M_1 =$

Students' preferences	Lecturers' preferences	offers
s_1: p_3 p_5	l_1: s_4 s_3 s_8 s_7	p_1, p_2
s_2: p_4 p_6	l_2: s_9 s_{10} s_{11} s_{12} s_5 s_6 s_2 s_1	p_3, p_4
s_3: p_2 p_6	l_3: s_7 s_8 s_5 s_1 s_9 s_{11}	p_5
s_4: p_2 p_6	l_4: s_3 s_4 s_6 s_2 s_{10} s_{12}	p_6
s_5: p_3 p_5		
s_6: p_4 p_6		
s_7: p_1 p_5		
s_8: p_1 p_5		
s_9: p_5 p_3		
s_{10}: p_6 p_4		
s_{11}: p_5 p_3	Project capacities: $c_1 = c_2 = c_3 = c_4 = c_5 = c_6 = 2$	
s_{12}: p_6 p_4	Lecturer capacities: $d_1 = d_2 = 4$, $d_3 = d_4 = 2$	

Figure 3.3: An instance I_3 of SPA-S.

Matching	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
M_1	p_3	p_4	p_2	p_2	p_3	p_4	p_1	p_1	p_5	p_6	p_5	p_6
M_2	p_3	p_6	p_2	p_2	p_3	p_4	p_1	p_1	p_5	p_6	p_5	p_4
M_3	p_5	p_4	p_2	p_2	p_3	p_4	p_1	p_1	p_5	p_6	p_3	p_6
M_4	p_5	p_6	p_2	p_2	p_3	p_4	p_1	p_1	p_5	p_6	p_3	p_4
M_5	p_5	p_4	p_2	p_2	p_5	p_4	p_1	p_1	p_3	p_6	p_3	p_6
M_6	p_3	p_6	p_2	p_2	p_3	p_6	p_1	p_1	p_5	p_4	p_5	p_4
M_7	p_5	p_6	p_2	p_2	p_3	p_6	p_1	p_1	p_5	p_4	p_3	p_4
M_8	p_5	p_6	p_2	p_2	p_5	p_4	p_1	p_1	p_3	p_6	p_3	p_4
M_9	p_5	p_6	p_2	p_2	p_5	p_6	p_1	p_1	p_3	p_4	p_3	p_4

Figure 3.4: The stable matchings in the SPA-S instance illustrated in Figure 3.3, with M_1 as the student-optimal stable matching and M_9 as the lecturer-optimal stable matching.

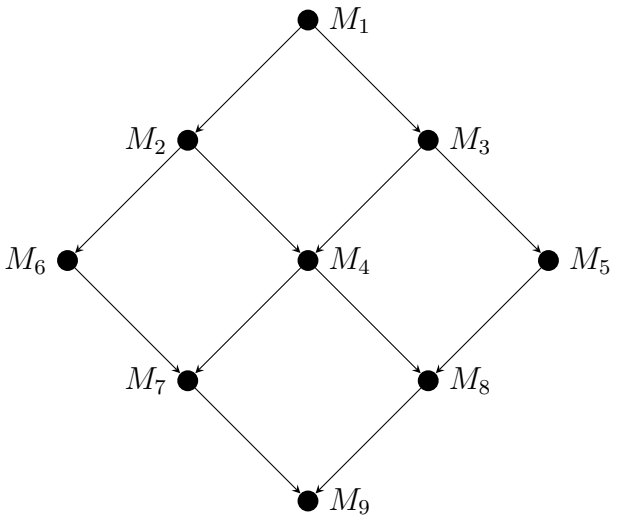


Figure 3.5: The lattice structure for the stable matchings in Figure 3.4, with M_1 as the maximum element of the lattice and M_9 as the minimum element of the lattice.

$\{(m_1, w_4), (m_2, w_3), (m_3, w_2), (m_4, w_1)\}$ and $M_2 = \{m_1, w_4), (m_2, w_1), (m_3, w_2), (m_4, w_3)\}$. The *rotation* in M_1 is a sequence $\rho = (m_2, w_3), (m_4, w_1)$ of pairs which allows us to move from M_1 to M_2 by assigning m_2 to w_1 (i.e., $M_1(m_4)$) and m_4 to w_3 (i.e., $M_1(m_2)$). We say that ρ is *exposed* in M_1 and by *eliminating* ρ from M_1, we obtain another stable matching M_2. See [53, 83] for a formal definition of rotation.

Let I be an instance of SM. Gusfield and Irving [44] showed that the set of rotations in I forms a poset under a natural ordering which they referred to as the *rotation poset*. Moreover, the authors established that there is a 1-1 correspondence between the stable matchings in I and the closed subsets of the rotation poset of I. These results have been extended to the HR setting [21, 30]. An open problem is to define a rotation in the SPA-S context, as well as establish the structural correspondence (if any) between the set of stable matchings and the set of rotations.

Furthermore, the characterisation of the stable matchings in I in terms of closed subsets of the rotation poset of I has been exploited to design efficient algorithms for a range of problems associated with SM. As the reader may recall, some of these problems include finding all stable pairs [43], generating all stable matchings [43], counting stable matchings [53], and finding stable matchings that satisfy additional optimality criteria [43, 54]. With a suitable structure, it is open as to whether a similar approach can be explored to solve similar problems in the SPA-S setting.

Chapter 4

Super-Stability in SPA-ST

4.1 Introduction

In Section 2.3.3, we introduced a variant of SPA where students have preferences over projects, lecturers have preferences over students, and preference lists of students and lecturers may admit indifference in the form of *ties*. We referred to this variant as the *Student-Project Allocation problem with lecturer preferences over Students with Ties* (SPA-ST). As the reader may recall, we mentioned that the three forms of stability for SMTI and HRT, namely *weak stability*, *strong stability* and *super-stability*, can be extended to this setting. We also mentioned that weakly stable matchings in an instance of SPA-ST can have different sizes and the problem of finding a maximum size weakly stable matching is NP-hard [60, 84].

We remark that super-stability is the most restrictive of these three stability concepts because it allows a student s and a lecturer l to form a blocking pair in the following situation: s is indifferent between her assigned project and one of l's projects, whilst l is indifferent between one of her assigned student/s and s. However, Irving *et al.* [57] argued that super-stability is a very natural solution concept in cases where agents have incomplete information. Central to their argument is the following, which we prove in Proposition 4.2.2: in a practical setting, suppose that an arbitrary student s_i has incomplete information about two or more projects and decides to rank these projects equally in a tie, so that s_i's preference list is a strict ordering over tied batches of projects. If a super-stable matching M exists in the corresponding SPA-ST instance I, then M is stable in every instance of SPA-S (obtained from I by breaking the ties) that represents the true preference of s_i.

Unfortunately, not every instance of SPA-ST admits a super-stable matching: for example, in the SPA-ST instance shown in Figure 4.1 (page 49), any matching will be undermined by a student and lecturer that are not matched together via some project that the lecturer offers.

However, as we will see later in this chapter, we can find a super-stable matching or report that no such matching exists in polynomial-time. Moreover, analogous to the HRT setting [57], if a super-stable matching M exists in an instance I of SPA-ST then all weakly stable matchings in I have the same size (equal to the size of M), and match exactly the same set of students (we prove this in Proposition 4.2.4). Thus the existence of a super-stable matching can steer us away from the hardness of finding a maximum size weakly stable matching.

Irving *et al*. [57] described an algorithm to find a super-stable matching or to report that no such matching exists, given an instance of HRT. However, merely reducing (using a "cloning" technique) an instance of SPA-ST to an instance of HRT and applying the algorithm described in [57] to the resulting HRT instance does not work in general (we explain this further in Section 4.3). Our contribution in this chapter is to present theoretical and experimental results for SPA-ST under super-stability.

On the theoretical side, we describe the first polynomial-time algorithm to find a super-stable matching or to report that no such matching exists, given an instance of SPA-ST – thus solving an open problem given in [13, 83]. Our algorithm runs in time linear in the total length of the preference lists. On the experimental side, we present results of an empirical evaluation based on an implementation of our linear-time algorithm that investigates how the nature of the preference lists affects the likelihood of a super-stable matching existing, with respect to randomly-generated SPA-ST instances. Our main finding from the empirical evaluation is that super-stable matchings are very elusive with ties on both sides; however, the probability of such matching existing is significantly higher if ties are restricted to the lecturers' preference lists.

The remainder of this chapter is structured as follows. We give some preliminary definitions and results in Section 4.2. We describe a natural strategy to clone an instance of SPA-ST to an instance of HRT in Section 4.3, and we give an intuition as to why this direction does not work in general. In Section 4.4, we describe our algorithm for SPA-ST under super-stability. Further, in Section 4.4, we illustrate an execution of our algorithm with respect to an instance of SPA-ST before moving on to present the algorithm's correctness and complexity results. To end this section, we give some structural properties satisfied by the set of super-stable matchings in an instance of SPA-ST.

In Section 4.5, we present an IP model for SPA-ST under super-stability. Our reason for doing this is purely experimental, as we intend to use an implementation of the IP model to test the correctness of our algorithm's implementation (i.e., to verify the existence or otherwise of a super-stable matching as reported by our implementation). In Section 4.6, we present the experimental results obtained from our algorithm's empirical evaluation. Finally, in Section 4.7, we present some concluding remarks and potential direction for future work.

Students' preferences	Lecturers' preferences	offers
s_1: p_3 p_2	l_1: (s_2 s_3) s_1	p_1, p_2
s_2: (p_1 p_2)	l_2: s_1	p_3
s_3: (p_1 p_2)		
	Project capacities: $c_1 = c_2 = c_3 = 1$	
	Lecturer capacities: $d_1 = 2,\ d_2 = 1$	

Figure 4.1: An instance I_1 of SPA-ST.

4.2 Preliminary definitions and results

Formally, an instance of SPA-ST involves a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of *students*, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of *projects* and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of *lecturers*. All the notation and terminology defined for SPA-S (see Section 2.3.2) also holds for SPA-ST except that the preference lists of students and lecturers in I can include *ties*. In the preference list of a student $s_i \in \mathcal{S}$, a set T of r projects forms a *tie of length* r (where $r \geq 1$) if s_i does not prefer p_j to $p_{j'}$ for any $p_j, p_{j'} \in T$. In addition, s_i's preference list is a strict ordering over tied batches of projects that she finds acceptable. Thus, s_i is indifferent between the projects that are tied together in her preference list, and she prefers each project in a given tie to each project in any successor tie. A tie in a lecturer's preference list is defined in a similar fashion.

An example SPA-ST instance I_1 is given in Figure 4.1, which involves the set $\mathcal{S} = \{s_1, s_2, s_3\}$ of students, the set $\mathcal{P} = \{p_1, p_2, p_3\}$ of projects and the set $\mathcal{L} = \{l_1, l_2\}$ of lecturers, with $P_1 = \{p_1, p_2\}$ and $P_2 = \{p_3\}$. Ties in the preference lists are indicated by round brackets, for example, s_2 is indifferent between p_1 and p_2.

Definition 4.2.1 (Super-stability)**.** *Let I be an instance of* SPA-ST*, and let M be a matching in I. We say that M is* super-stable *if it admits no blocking pair, where a* blocking pair *is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that (a) and (b) holds as follows:*

(a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or s_i is indifferent between them;

(b) either (i), (ii) or (iii) holds as follows:

- *(i) each of p_j and l_k is undersubscribed in M;*
- *(ii) p_j is undersubscribed in M, l_k is full in M, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$ or l_k is indifferent between them;*
- *(iii) p_j is full in M, and either l_k prefers s_i to the worst student in $M(p_j)$ or l_k is indifferent between them.*

The super-stability definition gives rise to the following proposition, stated for HRT in [57, Proposition 2], which extends naturally to SPA-ST as follows.

Proposition 4.2.2. *Let I be an instance of* SPA-ST *and let M be a matching in I. Then M is super-stable in I if and only if M is stable in every instance of* SPA-S *obtained from I by breaking the ties in some way.*

Proof. Let I be an instance of SPA-ST and let M be a matching in I. Suppose that M is super-stable in I. We want to show that M is stable in every instance of SPA-S obtained from I by breaking the ties in some way. Now, let I' be an arbitrary instance of SPA-S obtained from I by breaking the ties in some way, and suppose M is not stable in I'. This implies that M admits a blocking pair (s_i, p_j) in I'. Since I' is an arbitrary SPA-S instance obtained from I by breaking the ties in some way, it follows that in I:

(i) if s_i is assigned in M then s_i either prefers p_j to $M(s_i)$ or is indifferent between them;
(ii) if l_k is full in M then either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them; and
(iii) if p_j is full in M then l_k either prefers s_i to a worst student in $M(p_j)$ or is indifferent between them.

This implies that (s_i, p_j) forms a blocking pair for M in I, a contradiction to the super-stability of M.

Conversely, suppose M is stable in every instance of SPA-S obtained from I by breaking the ties in some way. Now suppose M is not super-stable in I. This implies that M admits a blocking pair (s_i, p_j) in I. We construct an instance I' of SPA-S from I by breaking the ties in the following way: (i) if s_i is assigned in M and s_i is indifferent between p_j and $M(s_i)$ in I then s_i prefers p_j to $M(s_i)$ in I', otherwise we break the ties in s_i's preference list arbitrarily; and (ii) if some student, say $s_{i'}$, different from s_i is assigned to l_k in M such that l_k is indifferent between s_i and $s_{i'}$ in I then l_k prefers s_i to $s_{i'}$ in I', otherwise we break the ties in l_k's preference list arbitrarily. Thus (s_i, p_j) forms a blocking pair for M in I', i.e., M is not stable in I', a contradiction to the fact that M is stable in every instance of SPA-S obtained from I by breaking the ties in some way. □

The reader may verify that the SPA-ST instance in Figure 4.1 admits two weakly stable matchings, namely $M_1 = \{(s_1, p_3), (s_2, p_1), (s_3, p_2)\}$ and $M_2 = \{(s_1, p_3), (s_2, p_2), (s_3, p_1)\}$. However, neither M_1 nor M_2 is super-stable, since (s_2, p_2) forms a blocking pair for M_1 and (s_2, p_1) forms a blocking pair for M_2. We remark that the formal definition of weak stability in SPA-ST follows from the definition of stability in SPA-S (see Definition 2.3.1 on page 24). Moreover, the existence of a weakly stable matching in I is guaranteed by breaking the ties in I arbitrarily, thus giving rise to an instance I' of SPA-S. Clearly, a stable matching in I' is weakly stable in I. Indeed a converse of sorts holds, which gives rise to the following proposition, which can be regarded as an analogue of Proposition 4.2.2 for weak stability.

Proposition 4.2.3. *Let I be an instance of* SPA-ST*, and let M be a matching in I. Then M is weakly stable in I if and only if M is stable in some instance I' of* SPA-S *obtained from I by breaking the ties in some way.*

Proof. Let I be an instance of SPA-ST and let M be a matching in I. Suppose that M is weakly stable in I. Let I' be an instance of SPA-S obtained from I by breaking the ties in the following way. For each student s_i in I such that the preference list of s_i includes a tie T containing two or more projects, we order the preference list of s_i in I' as follows: if s_i is assigned in M to a project p_j in T then s_i prefers p_j to every other project in T, otherwise we order the projects in T arbitrarily. For each lecturer l_k in I such that l_k's preference list includes a tie X, if X contains students that are assigned to l_k in M and students that are not assigned to l_k in M, then l_k's preference list in I' is ordered in such a way that each $s_i \in X \cap M(l_k)$ is preferred to each $s_{i'} \in X \setminus M(l_k)$, otherwise we order the students in X arbitrarily. Now, suppose (s_i, p_j) forms a blocking pair for M in I'. Given how the ties in I were removed to obtain I', this implies that (s_i, p_j) forms a blocking pair for M in I, a contradiction to our assumption that M is weakly stable in I. Thus M is stable in I'.

Conversely, suppose M is stable in some instance I' of SPA-S obtained from I by breaking the ties in some way. Now suppose that M is not weakly stable in I. Then some pair (s_i, p_j) forms a blocking pair for M in I. It is then clear from the definition of weak stability and from the construction of I' that (s_i, p_j) is a blocking pair for M in I', a contradiction. □

The following proposition, which is a consequence of Propositions 4.2.2 and 4.2.3, and Theorem 2.3.2, tells us that if a super-stable matching M exists in I then all weakly stable matchings in I are of the same size (equal to the size of M) and match exactly the same set of students.

Proposition 4.2.4. *Let I be an instance of* SPA-ST*, and suppose that I admits a super-stable matching M. Then the Unpopular Projects Theorem (i.e., Theorem 2.3.2) holds for the set of weakly stable matchings in I.*

Proof. Let I be an instance of SPA-ST. Let M be a super-stable matching in I and let M' be a weakly stable matching in I. Then by Proposition 4.2.3, M' is stable in some instance I' obtained from I by breaking the ties in some way. Also M is stable in I' by Proposition 4.2.2. By Theorem 2.3.2, each lecturer is assigned the same number of students in M and M', exactly the same students are unassigned in M and M', and a project offered by an undersubscribed lecturer is assigned the same number of students in M and M'. Hence, the Unpopular Projects Theorem holds for the set of weakly stable matchings in I. □

4.3 Cloning from SPA-ST to HRT does not work in general

As mentioned in Section 2.2.4, Irving *et al.* [57] described a polynomial-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of HRT. One might assume that reducing a given instance of SPA-ST to an instance of HRT (using a "cloning" technique) and subsequently applying the algorithm described in [57] to the resulting instance would solve our problem. However, this is not always true. In what follows, we describe a natural method to clone an instance of SPA-ST to an instance of HRT, and we show that applying the super-stable matching algorithm described in [57] to the resulting HRT instance does not work in general.

A method to derive an instance I' of HRT from an instance I of SPA-ST was described by Cooper and Manlove [34]. We explain this method as follows. The students and projects involved in I are converted into residents and hospitals respectively in I', i.e., each $s_i \in \mathcal{S}$ becomes r_i in the cloned instance, and each $p_j \in \mathcal{P}$ becomes h_j. Residents inherit their preference lists naturally from students, i.e., if r_i corresponds to s_i, then the preference list of r_i in I' is A_i, with each project in A_i being replaced by the associated hospital. Hospitals inherit their preference lists from the projected preference list of the associated project according to the lecturer offering the project, i.e., if p_j corresponds to h_j, where p_j is offered by l_k, then the preference list of h_j in I' is $\mathcal{L}_k^j$, with each student in $\mathcal{L}_k^j$ being replaced by the associated resident. Each hospital also inherits its capacity from the project, i.e., for each h_j associated with p_j, the capacity of h_j is c_j.

Let l_k be an arbitrary lecturer in I. In order to translate l_k's capacity into the HRT instance, we create n *dummy residents*[1] for each hospital h_j corresponding to a project $p_j \in P_k$, where n is the difference between the sum of the capacities of all the projects in P_k and the capacity of l_k (recall from Section 2.3.2 that $d_k \leq \sum\{c_j : p_j \in P_k\}$). The preference list for each of these dummy residents will be a single tie consisting of all the hospitals corresponding to a project in P_k. Further, the preference list for each hospital corresponding to a project in P_k will include a tie in its first position consisting of all the dummy residents associated with l_k.

Next, we describe how to map between matchings in I and in I'. Let M and M' be a matching in I and I' respectively. Let r_i be the resident associated with s_i and let h_j be the hospital associated with p_j. If s_i is assigned in M to project p_j, then r_i is assigned in M' to hospital h_j. In what follows, we give an example instance of SPA-ST as well as the corresponding cloned HRT instance, which adapts the cloning technique described above. Also, we give an intuition as to why this technique will not work in general.

[1] The dummy residents created for each hospital will offset the difference between the corresponding lecturer capacity and the total capacity of her proposed projects.

Students' preferences	Lecturers' preferences	offers
s_1: p_1	l_1: s_1 (s_2 s_3)	p_1, p_2
s_2: (p_1 p_2)	l_2: s_3	p_3
s_3: p_2 p_3		
	Project capacities: $c_1 = c_2 = c_3 = 1$	
	Lecturer capacities: $d_1 = d_2 = 1$	

Figure 4.2: An instance I of SPA-ST.

Residents' preferences	Hospitals' preferences
r_1: h_1	h_1: r_{d_1} r_1 r_2
r_2: (h_1 h_2)	h_2: r_{d_1} (r_2 r_3)
r_3: h_2 h_3	h_3: r_3
r_{d_1}: (h_1 h_2)	
	Hospital capacities: $c_1 = c_2 = c_3 = 1$

Figure 4.3: An instance I' of HRT cloned from the SPA-ST instance illustrated in Figure 4.2.

Each resident r_1, r_2 and r_3 in I' corresponds to student s_1, s_2 and s_3 in I, respectively; and the preference list of each resident is adapted from the preference list of the associated student. Also, each hospital h_1, h_2 and h_3 in I' corresponds to project p_1, p_2 and p_3 in I, respectively. The preference list of hospitals h_1 and h_2 is $\mathcal{L}_1^1$ and $\mathcal{L}_1^2$ respectively, since l_1 is the lecturer that offers both p_1 and p_2. Similarly, the preference list of hospital h_3 is $\mathcal{L}_2^3$, since l_2 is the lecturer that offers p_3. Further, for lecturer l_1 who offers both p_1 and p_2, since $c_1 + c_2 = 2 > 1 = d_1$, we add one dummy resident r_{d_1} to the cloned instance. The preference list of r_{d_1} is a single tie consisting of h_1 and h_2; and the preference list of both h_1 and h_2 includes r_{d_1} in first position.

The reader can easily verify that matching $M = \{(s_1, p_1), (s_3, p_3)\}$ is super-stable in the SPA-ST instance I illustrated in Figure 4.2. Now, following our description of how to map between matchings in I and in I', a matching in I' is $M' = \{(r_{d_1}, h_2), (r_1, h_1), (r_3, h_3)\}$, with $(s_1, p_1) \in M$ corresponding to $(r_1, h_1) \in M'$ and $(s_3, p_3) \in M$ corresponding to $(r_3, h_3) \in M'$. Clearly, M' is not super-stable in I' as (r_{d_1}, h_1) forms a blocking pair. In fact, the HRT instance I' admits no super-stable matching. The justification for this is as follows: irrespective of the hospital that the dummy resident r_{d_1} is assigned to in any matching in I', r_{d_1} will block this matching via one of the other hospitals tied in her preference list (since the hospital would be better off taking on r_{d_1}, and r_{d_1} would be no worse off).

One way to avoid this problem would be to strictly order the hospitals in r_{d_1}'s preference list; however, the order in which the hospitals appear will lead to different possi-

bilities. For instance: if r_{d_1} prefers h_1 to h_2, the reader can verify that the corresponding HRT instance admits no super-stable matching; however, if r_{d_1} prefers h_2 to h_1, again the reader can verify that the corresponding HRT instance admits the super-stable matching $\{(r_{d_1}, h_2), (r_1, h_1), (r_3, h_3)\}$. The downside of this strategy is that there is no obvious reason as to why r_{d_1} should prefer h_2 to h_1 in the cloned HRT instance in Figure 4.3 by merely looking at the original SPA-ST instance in Figure 4.2. Hence, in order to make this technique work in general, we will need to generate every HRT instance obtained by ordering the dummy residents' preference lists in some way. This is exponential in the problem instance.

4.4 A polynomial-time algorithm

4.4.1 Introduction

In this section we present our algorithm for SPA-ST under super-stability, which we will refer to as Algorithm SPA-ST-super. First, we note that our algorithm is a non-trivial extension of Algorithm HRT-Super-Res for finding a super-stable matching or reporting that no such matching exists, given an instance of HRT [57].

Before we proceed, we briefly describe Algorithm HRT-Super-Res. The algorithm involves a sequence of proposals from the residents to the hospitals. Each resident proposes in turn to all of the hospitals tied together in the first position on her preference list, and all proposals are provisionally accepted. If a hospital h becomes oversubscribed then none of h's worst assignees nor any resident tied with these assignees in h's preference list can be assigned to h in any super-stable matching – such pairs (r, h) are deleted from each other's preference lists. If a hospital h is full then no resident strictly worse than h's worst assignees can be assigned to h in any super-stable matching – again such (r, h) pairs are deleted from each other's preference lists. The proposal sequence terminates once every resident either is assigned to a hospital or has an empty preference list. At this point, if the constructed assignment of residents to hospitals is super-stable in the original HRT instance then the assignment is returned as a super-stable matching. Otherwise, the algorithm reports that no super-stable matching exists.

Due to the more general setting of SPA-ST, Algorithm SPA-ST-super requires some new ideas (precisely lines 27-34 of the algorithm on page 57), and the proofs of the correctness results are more complex than Algorithm HRT-Super-Res [57]. In Section 4.4.2, we give definitions relating to our algorithm. We give a description of our algorithm in Section 4.4.3, before presenting it in pseudocode form. In Section 4.4.4, we illustrate an example execution of our algorithm with respect to an instance of SPA-ST. We present the algorithm's correctness and complexity results in Section 4.4.5. Finally, in Section 4.4.6, we show that

the set of super-stable matchings in an instance of SPA-ST satisfy analogous properties to those given in Theorem 2.3.2 for SPA-S.

4.4.2 Definitions relating to the algorithm

In what follows, I is an instance of SPA-ST, (s_i, p_j) is an acceptable pair in I and l_k is the lecturer who offers p_j. Further, if (s_i, p_j) belongs to some super-stable matching in I, we call (s_i, p_j) a *super-stable pair*.

During the execution of the algorithm, students become *provisionally assigned* to projects. It is possible for a project to be provisionally assigned a number of students that exceeds its capacity. This holds analogously for a lecturer. The algorithm proceeds by deleting from the preference lists certain (s_i, p_j) pairs that cannot be super-stable. By the term *delete* (s_i, p_j), we mean the removal of p_j from s_i's preference list and the removal of s_i from $\mathcal{L}_k^j$.[2] In addition, if s_i is provisionally assigned to p_j at this point, we break the assignment. If s_i has been deleted from every projected preference list of l_k that she originally belonged to, we will implicitly assume that s_i has been deleted from l_k's preference list. By the *head* of a student's preference list at a given point, we mean the set of one or more projects, tied in her preference list after any deletions might have occurred, that she prefers to all other projects in her preference list.

For project p_j, we define the *tail* of $\mathcal{L}_k^j$ as the least-preferred tie in $\mathcal{L}_k^j$ after any deletions might have occurred (recalling that a tie can be of length one). In the same fashion, we define the *tail* of $\mathcal{L}_k$ (the preference list of lecturer l_k) as the least-preferred tie in $\mathcal{L}_k$ after any deletions might have occurred. If s_i is provisionally assigned to p_j, we define the *successors* of s_i in $\mathcal{L}_k^j$ as those students that are worse than s_i in $\mathcal{L}_k^j$. An analogous definition holds for the successors of s_i in $\mathcal{L}_k$.

4.4.3 Description of the algorithm

We now describe Algorithm SPA-ST-super, which is shown in pseudocode form as Algorithm 1 on page 57. Algorithm SPA-ST-super begins by initialising an empty set M which will contain the provisional assignments of students to projects (and implicitly to lecturers). We remark that such assignments can subsequently be broken during the algorithm's execution. Also, each project is initially assigned to be empty (i.e., not assigned to any student).

The `repeat-until` loop of the algorithm initialises the `while` loop which involves each student s_i who is not provisionally assigned to any project in M and who has a non-empty

[2]Recall that $\mathcal{L}_k^j$ is the projected preference list of lecturer l_k for p_j, which can be obtained from $\mathcal{L}_k$ by removing those students that do not find p_j acceptable (thereby retaining the order of the remaining students from $\mathcal{L}_k$).

preference list applying in turn to each project p_j at the head of her list. Immediately, s_i becomes provisionally assigned to p_j in M (and to l_k). If, by gaining a new student, p_j becomes oversubscribed, then no student s_t at the tail of $\mathcal{L}_k^j$ can be assigned to p_j in any super-stable matching – such pairs (s_t, p_j) are deleted. Similarly, if by gaining a new student, l_k becomes oversubscribed, none of the students s_t at the tail of $\mathcal{L}_k$ can be assigned to any project offered by l_k in any super-stable matching – the pairs (s_t, p_u), for each project $p_u \in P_k$ that s_t finds acceptable, are deleted.

Regardless of whether any deletions occurred as a result of the two conditionals described in the previous paragraph, we have two further (possibly non-disjoint) cases in which deletions may occur. If p_j becomes full, we let s_r be any worst student provisionally assigned to p_j (according to $\mathcal{L}_k^j$), and we delete (s_t, p_j) for each successor s_t of s_r in $\mathcal{L}_k^j$. Similarly if l_k becomes full, we let s_r be any worst student provisionally assigned to l_k, and we delete (s_t, p_u), for each successor s_t of s_r in $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable. As we will prove later, none of the (student, project) pairs that we delete can be a super-stable pair.

At the point where the `while` loop terminates (i.e., when every student is provisionally assigned to one or more projects or has an empty preference list), if some project p_j that was previously full ends up undersubscribed, we let s_r be any one of the most-preferred students (according to $\mathcal{L}_k^j$) who was provisionally assigned to p_j during some iteration of the algorithm but is not assigned to p_j at this point (for convenience, we henceforth refer to such s_r as the most-preferred student rejected from p_j according to $\mathcal{L}_k^j$). If the students at the tail of $\mathcal{L}_k$ are no better than s_r, it turns out that none of these students can be assigned to any project offered by l_k in any super-stable matching; thus for each s_t at the tail of $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable, we delete (s_t, p_u). The `while` loop is then potentially reactivated, and the entire process continues until every student is provisionally assigned to a project or has an empty preference list, at which point the `repeat-until` loop terminates.

Upon termination of the `repeat-until` loop, if the set M, containing the assignment of students to projects, is super-stable relative to the given instance I then M is output as a super-stable matching in I. Otherwise, the algorithm reports that no super-stable matching exists in I.

4.4.4 Example execution of the algorithm

We illustrate an execution of Algorithm SPA-ST-super with respect to the SPA-ST instance I_2 shown in Figure 4.4. We initialise $M = \emptyset$, which will contain the provisional assignment of students to projects. For each project $p_j \in \mathcal{P}$, we set `full`(p_j) = `false` (this will be set

Algorithm 1 Algorithm SPA-ST-super

Input: SPA-ST instance I

Output: a super-stable matching M in I or "no super-stable matching exists in I"

$M \leftarrow \emptyset$
for each $p_j \in \mathcal{P}$ **do**
    full(p_j) = false
repeat
    while some student s_i is unassigned and has a non-empty preference list **do**
        for each project p_j at the head of s_i's preference list **do**
            $l_k \leftarrow$ lecturer who offers p_j
            /* s_i applies to p_j */
            $M \leftarrow M \cup \{(s_i, p_j)\}$ /* provisionally assign s_i to p_j (and to l_k) */
            if p_j is oversubscribed **then**
                for each student s_t at the tail of $\mathcal{L}_k^j$ **do**
                    delete (s_t, p_j) /* if the pair is in M, remove it */
            else if l_k is oversubscribed **then**
                for each student s_t at the tail of $\mathcal{L}_k$ **do**
                    for each project $p_u \in P_k \cap A_t$ **do**
                        delete (s_t, p_u)
            if p_j is full **then**
                full(p_j) = true
                $s_r \leftarrow$ worst student assigned to p_j according to $\mathcal{L}_k^j$ {any if > 1}
                for each successor s_t of s_r on $\mathcal{L}_k^j$ **do**
                    delete (s_t, p_j)
            if l_k is full **then**
                $s_r \leftarrow$ worst student assigned to l_k according to $\mathcal{L}_k$ {any if > 1}
                for each successor s_t of s_r on $\mathcal{L}_k$ **do**
                    for each project $p_u \in P_k \cap A_t$ **do**
                        delete (s_t, p_u)
    for each $p_j \in \mathcal{P}$ **do**
        if p_j is undersubscribed and full(p_j) is true **then**
            $l_k \leftarrow$ lecturer who offers p_j
            $s_r \leftarrow$ most-preferred student rejected from p_j according to $\mathcal{L}_k^j$ {any if > 1}
            if the students at the tail of $\mathcal{L}_k$ are no better than s_r **then**
                for each student s_t at the tail of $\mathcal{L}_k$ **do**
                    for each project $p_u \in P_k \cap A_t$ **do**
                        delete (s_t, p_u)
until every unassigned student has an empty preference list
if M is a super-stable matching in I **then**
    return M
else
    return "no super-stable matching exists in I"

to `true` when p_j becomes full, so that we can easily identify any project that was full during an iteration of the `while` loop and ended up undersubscribed at the termination of this or at the termination of subsequent `while` loop iterations). We assume that the students become provisionally assigned to each project at the head of their preference list in subscript order. Table 4.1 illustrates how this execution of Algorithm SPA-ST-super proceeds with respect to I_2.

Students' preferences	Lecturers' preferences	offers
s_1: p_1	l_1: s_5 (s_1 s_2) s_3 s_4	p_1, p_2
s_2: (p_1 p_3)	l_2: s_4 s_5 s_2	p_3
s_3: p_2		
s_4: p_2 p_3	Project capacities: $c_1 = c_3 = 1,\ c_2 = 2$	
s_5: p_3 p_1	Lecturer capacities: $d_1 = 2,\ d_2 = 1$	

Figure 4.4: An instance I_2 of SPA-ST.

4.4.5 Correctness of the algorithm

We now present a series of results concerning the correctness of Algorithm SPA-ST-super. The first of these results deals with the fact that no super-stable pair is deleted during an execution of the algorithm. In what follows, I is an instance of SPA-ST, (s_i, p_j) is an acceptable pair in I and l_k is the lecturer who offers p_j.

Lemma 4.4.1. *If a pair (s_i, p_j) is deleted during an execution of Algorithm* SPA-ST-super*, then (s_i, p_j) does not belong to any super-stable matching in I.*

In order to prove Lemma 4.4.1, we present Lemmas 4.4.2 and 4.4.3.

Lemma 4.4.2. *If a pair (s_i, p_j) is deleted within the `while` loop during an execution of Algorithm* SPA-ST-super *then (s_i, p_j) does not belong to any super-stable matching in I.*

Proof. Without loss of generality, suppose that the first super-stable pair to be deleted within the `while` loop during an arbitrary execution E of the algorithm is (s_i, p_j), which belongs to some super-stable matching, say M^*. Suppose that M is the assignment immediately after the deletion. Let us denote this point in the algorithm where the deletion is made by ‡. During E, there are four cases that would lead to the deletion of any (student, project) pair within the `while` loop.

(1) *p_j is oversubscribed.* Suppose that (s_i, p_j) is deleted because some student (possibly s_i) became provisionally assigned to p_j during E, causing p_j to become oversubscribed. If p_j is full or undersubscribed at point ‡, since $s_i \in M^*(p_j) \setminus M(p_j)$ and no

Table 4.1: An execution of Algorithm SPA-ST-super with respect to Figure 4.4.

while loop iterations	Student applies to project	Consequence
1	s_1 applies to p_1	$M = \{(s_1, p_1)\}$. full(p_1) = true.
2	s_2 applies to p_1	$M = \{(s_1, p_1), (s_2, p_1)\}$. p_1 becomes oversubscribed. The tail of $\mathcal{L}_1^1$ contains s_1 and s_2 – thus we delete the pairs (s_1, p_1) and (s_2, p_1) (and we break the provisional assignments).
	s_2 applies to p_3	$M = \{(s_2, p_3)\}$. full(p_3) = true.
3	s_3 applies to p_2	$M = \{(s_2, p_3), (s_3, p_2)\}$.
4	s_4 applies to p_2	$M = \{(s_2, p_3), (s_3, p_2), (s_4, p_2)\}$. full($p_2$) = true.
5	s_5 applies to p_3	$M = \{(s_2, p_3), (s_3, p_2), (s_4, p_2), (s_5, p_3)\}$. p_3 becomes oversubscribed. The tail of $\mathcal{L}_2^3$ contains only s_2 – thus we delete the pair (s_2, p_3) (and we break the provisional assignment).
The first iteration of the while loop terminates since every unassigned student (i.e., s_1 and s_2) has an empty preference list. At this point, full(p_1) is true and p_1 is undersubscribed. Moreover, the student at the tail of $\mathcal{L}_1$ (i.e., s_4) is no better than s_1, where s_1 was previously assigned to p_1 and s_1 is also the most-preferred student rejected from p_1 according to $\mathcal{L}_1^1$; thus we delete the pair (s_4, p_2). The while loop is then reactivated.		
6	s_4 applies to p_3	$M = \{(s_3, p_2), (s_5, p_3), (s_4, p_3)\}$. p_3 becomes oversubscribed. The tail of $\mathcal{L}_2^3$ contains only s_5 – thus we delete the pair (s_5, p_3).
7	s_5 applies to p_1	$M = \{(s_3, p_2), (s_4, p_3), (s_5, p_1)\}$.
Again, every unassigned students has an empty preference list. We also have that full(p_2) is true and p_2 is undersubscribed; however no further deletion is carried out in line 34 of the algorithm, since the student at the tail of $\mathcal{L}_1$ (i.e., s_3) is better than s_4, where s_4 was previously assigned to p_2 and s_4 is also the most-preferred student rejected from p_2 according to $\mathcal{L}_1^2$. Hence, the repeat-until loop terminates and the algorithm outputs $M = \{(s_3, p_2), (s_4, p_3), (s_5, p_1)\}$ as a super-stable matching. It is clear that M is super-stable in the original instance I_2.		

project can be oversubscribed in M^*, then there is some student $s_r \in M(p_j) \setminus M^*(p_j)$ such that l_k prefers s_r to s_i or is indifferent between them. We note that s_r cannot be assigned to a project that she prefers to p_j in any super-stable matching. Otherwise, since p_j must have been in the head of s_r's preference list when she applied, this would mean that a super-stable pair was deleted before (s_i, p_j). Thus either s_r is unassigned in M^* or s_r prefers p_j to $M^*(s_r)$ or s_r is indifferent between them. Clearly, for any combination of l_k and p_j being full or undersubscribed in M^*, it follows that (s_r, p_j) blocks M^*, a contradiction.

(2) l_k *is oversubscribed.* Suppose that (s_i, p_j) is deleted because some student (possibly s_i) became provisionally assigned to a project offered by lecturer l_k during E, causing l_k to become oversubscribed. At point ‡, none of the projects offered by l_k is oversubscribed in M, otherwise we will be in case (1). Similar to case (1), if l_k is full or undersubscribed at point ‡, since $s_i \in M^*(p_j) \setminus M(p_j)$ and no lecturer can be oversubscribed in M^*, it follows that there is some project $p_{j'} \in P_k$ and some student $s_r \in M(p_{j'}) \setminus M^*(p_{j'})$ such that l_k prefers s_r to s_i or is indifferent between them. We consider two subcases.

(i) If $p_{j'} = p_j$ then $s_r \neq s_i$, since $s_i \in M^*(p_j)$ and $s_r \notin M^*(p_{j'})$. Moreover, as in case (1), either s_r is unassigned in M^* or s_r prefers $p_{j'}$ to $M^*(s_r)$ or s_r is indifferent between them. For any combination of l_k and $p_{j'}$ being full or undersubscribed in M^*, we have that $(s_r, p_{j'})$ blocks M^*, a contradiction.

(ii) If $p_{j'} \neq p_j$. Assume firstly that $s_r \neq s_i$. Then as $p_{j'}$ has fewer assignees in M^* than it has provisional assignees in M, and as in (i) above, $(s_r, p_{j'})$ blocks M^*, a contradiction. Finally assume $s_r = s_i$. Then s_i must have applied to $p_{j'}$ at some point during E before ‡. Clearly, either s_i prefers $p_{j'}$ to p_j or s_i is indifferent between them, since $p_{j'}$ must have been in the head of s_i's preference list when s_i applied. Since $s_i \in M^*(l_k)$ and $p_{j'}$ is undersubscribed in M^*, it follows that $(s_i, p_{j'})$ blocks M^*, a contradiction.

(3) p_j *is full.* Suppose that (s_i, p_j) is deleted because p_j became full during E. At point ‡, p_j is full in M. Thus at least one of the students in $M(p_j)$, say s_r, will not be assigned to p_j in M^*, for otherwise p_j will be oversubscribed in M^*. This implies that either s_r is unassigned in M^* or s_r prefers p_j to $M^*(s_r)$ or s_r is indifferent between them. For otherwise, we obtain a contradiction to (s_i, p_j) being the first super-stable pair to be deleted. Since l_k prefers s_r to s_i, it follows that (s_r, p_j) blocks M^*, a contradiction.

(4) l_k *is full.* Suppose that (s_i, p_j) is deleted because l_k became full during E. We consider two subcases.

(i) All the students assigned to p_j in M at point ‡ (if any) are also assigned to p_j in M^*. This implies that p_j has one more assignee in M^* than it has provisional assignees in M, namely s_i. Thus, some other project $p_{j'} \in P_k$ has fewer assignees in M^* than it has provisional assignees in M, for otherwise l_k would be oversubscribed in M^*. Hence there exists some student $s_r \in M(p_{j'}) \setminus M^*(p_{j'})$. It is clear that $s_r \neq s_i$, since s_i plays the role of s_t at some for loop iteration in line 24 of the algorithm. Also, s_r cannot be assigned to a project that she prefers to $p_{j'}$ in M^*, as explained in case (1). Moreover, since $p_{j'}$ is undersubscribed in M^* and l_k prefers s_r to s_i, it follows that $(s_r, p_{j'})$ blocks M^*, a contradiction.

(ii) Some student, say s_r, who is assigned to p_j in M is not assigned to p_j in M^*, i.e., $s_r \in M(p_j) \setminus M^*(p_j)$. Since s_r cannot be assigned in M^* to a project that she prefers to p_j and since l_k prefers s_r to s_i, it follows that (s_r, p_j) blocks M^*, a contradiction.

□

Lemma 4.4.3. *If a pair (s_i, p_j) is deleted in line 34 of Algorithm* SPA-ST-super *then (s_i, p_j) does not belong to any super-stable matching in I.*

Proof. Without loss of generality, suppose that the first super-stable pair to be deleted during an arbitrary execution E of the algorithm is (s_i, p_j), which belongs to some super-stable matching, say M^*. Then by Lemma 4.4.2, (s_i, p_j) was deleted in line 34 during E. Let l_k be the lecturer who offers p_j. Suppose that M is the assignment during the iteration of the `repeat-until` loop where (s_i, p_j) was deleted.

Let $p_{j'}$ be some other project offered by l_k which was full during a previous `repeat-until` loop iteration and subsequently ends up undersubscribed in the current `repeat-until` loop iteration, i.e., $p_{j'}$ plays the role of p_j in line 28. Suppose that $s_{i'}$ plays the role of s_r in line 30, i.e., $s_{i'}$ is the most-preferred student rejected from $p_{j'}$ according to $\mathcal{L}_k^{j'}$ (possibly $s_{i'} = s_i$). Moreover $s_{i'}$ was provisionally assigned to $p_{j'}$ during a previous `repeat-until` loop iteration but $(s_{i'}, p_{j'}) \notin M$ in the current `repeat-until` loop iteration. Thus $(s_{i'}, p_{j'})$ has been deleted before the deletion of (s_i, p_j) occurred; and thus, $(s_{i'}, p_{j'}) \notin M^*$, since (s_i, p_j) is the first super-stable pair to be deleted. Further, l_k either prefers $s_{i'}$ to s_i or is indifferent between them, since s_i plays the role of s_t at some for loop iteration in line 32.

We remark that no student who is provisionally assigned to some project in M can be assigned to a project better than her current assignment in any super-stable matching. For otherwise, this would mean a super-stable pair must have been deleted before (s_i, p_j), since each student who is assigned in M applies to projects in the head of her preference list. So, either $s_{i'}$ is unassigned in M^* or $s_{i'}$ prefers $p_{j'}$ to $M^*(s_{i'})$ or s_i is indifferent between them. By the super-stability of M^*, $p_{j'}$ is full in M^* and l_k prefers every student in $M^*(p_{j'})$ to $s_{i'}$; for otherwise, $(s_{i'}, p_{j'})$ blocks M^*, a contradiction.

Let $l_{z_0} = l_k$, $p_{t_0} = p_{j'}$ and $s_{q_0} = s_{i'}$. Just before the deletion of (s_i, p_j) occurred, p_{t_0} is undersubscribed in M. Since p_{t_0} is full in M^*, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus M(p_{t_0})$. We note that l_{z_0} prefers s_{q_1} to s_{q_0}; for otherwise, $(s_{i'}, p_{j'})$ blocks M^*, a contradiction. Let $p_{t_1} = p_{t_0}$. Since (s_i, p_j) is the first super-stable pair to be deleted, s_{q_1} is assigned in M to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as each student applies to projects at the head of her preference list, that would mean (s_{q_1}, p_{t_1}) must have been deleted before (s_i, p_j), a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in M$ and $(s_{q_1}, p_{t_1}) \notin M$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the super-stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefer the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

First we claim that for each new project that we identify, $p_{t_{2i}} \neq p_{t_{2i-1}}$ for $i \geq 1$. Suppose $p_{t_{2i}} = p_{t_{2i-1}}$ for some $i \geq 1$. From above s_{q_i} was identified by $l_{z_{i-1}}$ such that $(s_{q_i}, p_{t_{2i-1}}) \in M^* \setminus M$. Moreover $(s_{q_i}, p_{t_{2i}}) \in M$. Hence we reach a contradiction. Clearly, for each student s_{q_i} that we identify, for $i \geq 1$, s_{q_i} must be assigned to distinct projects in M and in M^*.

Next we claim that for each new student s_{q_i} that we identify, $s_{q_i} \neq s_{q_t}$ for $1 \leq t < i$. We prove this by induction on i. For the base case, clearly $s_{q_2} \neq s_{q_1}$. We assume that the claim holds for some $i \geq 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}$ consists of distinct students. We show that the claim holds for $i + 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}, s_{q_{i+1}}$ also consists of distinct students. Clearly $s_{q_{i+1}} \neq s_{q_i}$ since l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}. Thus, it suffices to show that $s_{q_{i+1}} \neq s_{q_j}$ for $1 \leq j \leq i - 1$. Now, suppose $s_{q_{i+1}} = s_{q_j}$ for $1 \leq j \leq i - 1$. This implies that s_{q_j} was identified by l_{z_i} and clearly l_{z_i} prefers s_{q_j} to $s_{q_{j-1}}$. Now since $s_{q_{i+1}}$ was also identified by l_{z_i} to avoid the blocking pair $(s_{q_i}, p_{t_{2_i}})$ in M^*, it follows that either (i) $p_{t_{2i}}$

is full in M^*, or (ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. We consider each cases further as follows.

(i) If $p_{t_{2i}}$ is full in M^*, we know that $(s_{q_i}, p_{t_{2i}}) \in M \setminus M^*$. Moreover s_{q_j} was identified by $l_{z_{i+1}}$ because of case (i). Furthermore $(s_{q_{j-1}}, p_{t_{2i}}) \in M \setminus M^*$. In this case, $p_{t_{2i+1}} = p_{t_{2i}}$ and we have that

$$(s_{q_i}, p_{t_{2i+1}}) \in M \setminus M^* \text{ and } (s_{q_{i+1}}, p_{t_{2i+1}}) \in M^* \setminus M,$$

$$(s_{q_{j-1}}, p_{t_{2i+1}}) \in M \setminus M^* \text{ and } (s_{q_j}, p_{t_{2i+1}}) \in M^* \setminus M.$$

By the inductive hypothesis, the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_{j-1}}, s_{q_j}, \ldots, s_{q_i}$ consists of distinct students. This implies that $s_{q_i} \neq s_{q_{j-1}}$. Thus since $p_{t_{2i+1}}$ is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ to avoid the blocking pairs $(s_{q_{j-1}}, p_{t_{2i+1}})$ and $(s_{q_i}, p_{t_{2i+1}})$ respectively in M^*, a contradiction.

(ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. Similarly as in case (i) above, we have that

$$s_{q_i} \in M(l_{z_i}) \setminus M^*(l_{z_i}) \text{ and } s_{q_{i+1}} \in M^*(l_{z_i}) \setminus M(l_{z_i}),$$

$$s_{q_{j-1}} \in M(l_{z_i}) \setminus M^*(l_{z_i}) \text{ and } s_{q_j} \in M^*(l_{z_i}) \setminus M(l_{z_i}).$$

Since $s_{q_i} \neq s_{q_{j-1}}$ and l_{z_i} is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ corresponding to students $s_{q_{j-1}}$ and s_{q_i} respectively, a contradiction.

This completes the induction step. As the sequence of distinct students and projects we are identifying is infinite, we reach an immediate contradiction. □

Lemmas 4.4.2 and 4.4.3 immediately give rise to Lemma 4.4.1. The next lemma will be used as a tool in the proof of the remaining lemmas.

Lemma 4.4.4. *Let M be the assignment at the termination of Algorithm* SPA-ST-super *and let M^* be any super-stable matching in I. Let l_k be an arbitrary lecturer: (a) if l_k is undersubscribed in M^* then every student who is assigned to l_k in M is also assigned to l_k in M^*; and (b) if l_k is undersubscribed in M then l_k has the same number of assignees in M^* as in M.*

Proof. Let l_k be an arbitrary lecturer. First, we show that (a) holds. Suppose otherwise, then there exists a student, say s_i, such that $s_i \in M(l_k) \setminus M^*(l_k)$. Moreover, there exists some project $p_j \in P_k$ such that $s_i \in M(p_j) \setminus M^*(p_j)$. By Lemma 4.4.1, s_i cannot be assigned to a project that she prefers to p_j in M^*. Also, by the super-stability of M^*, p_j is full in M^* and l_k prefers the worst student/s in $M^*(p_j)$ to s_i.

Let $l_{z_0} = l_k$, $p_{t_0} = p_j$, and $s_{q_0} = s_i$. As p_{t_0} is full in M^* and no project is oversubscribed in M, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus M(p_{t_0})$ such that l_{z_0} prefers s_{q_1} to s_{q_0}. Let $p_{t_1} = p_{t_0}$. By Lemma 4.4.1, s_{q_1} is assigned in M to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. We note that s_{q_1} cannot be indifferent between p_{t_2} and p_{t_1}; for otherwise, as each student applies to projects at the head of her preference list, since $(s_{q_1}, p_{t_1}) \notin M$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during the algorithm's execution, contradicting Lemma 4.4.1. It follows that $s_{q_1} \in M(p_{t_2}) \setminus M^*(p_{t_2})$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the super-stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, student s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 4.4.3, we can identify an infinite sequence of distinct students and projects, a contradiction. Hence, if l_k is undersubscribed in M^* then every student who is assigned to l_k in M is also assigned to l_k in M^*.

Next, we show that (b) holds. By the first claim, any lecturer who is full in M is also full in M^*, and any lecturer who is undersubscribed in M has as many assignees in M^* as she has in M. Hence

$$\sum_{l_k \in \mathcal{L}} |M(l_k)| \leq \sum_{l_k \in \mathcal{L}} |M^*(l_k)| \ . \tag{4.1}$$

We note that if a student s_i is unassigned in M, by Lemma 4.4.1, s_i is unassigned in M^*. Equivalently, if s_i is assigned in M^* then s_i is assigned in M. Let S_1 denote the set of students who are assigned to at least one project in M, and let S_2 denote the set of students

who are assigned to a project in M^*; it follows that $|S_2| \leq |S_1|$. Further, we have that

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| = |S_2| \leq |S_1| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)|, \tag{4.2}$$

From Inequalities (4.1) and (4.2), it follows that $|M(l_k)| = |M^*(l_k)|$ for each $l_k \in \mathcal{L}$. □

The next three lemmas deal with the case that Algorithm SPA-ST-super reports the non-existence of a super-stable matching in I.

Lemma 4.4.5. *If a student is assigned to two or more projects at the termination of Algorithm* SPA-ST-super *then I admits no super-stable matching.*

Proof. Let M be the assignment at the termination of the algorithm. Suppose for a contradiction that there exists a super-stable matching M^* in I. Suppose that a student is assigned to two or more projects in M. Then either (a) any two of these projects are offered by different lecturers or (b) all of these projects are offered by the same lecturer.

Firstly, suppose (a) holds. Then some lecturer has fewer assignees in M^* than in M. Suppose not, then

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| \geq \sum_{l_k \in \mathcal{L}} |M(l_k)| \ . \tag{4.3}$$

Let S_1 and S_2 be as defined in the proof of Lemma 4.4.4, it follows that $|S_2| \leq |S_1|$. Hence,

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| = |S_2| \leq |S_1| < \sum_{l_k \in \mathcal{L}} |M(l_k)|, \tag{4.4}$$

since some student in S_1 is assigned in M to two or more projects offered by different lecturers. Inequality (4.4) contradicts Inequality (4.3). Hence, our claim is established. As some lecturer l_k has fewer assignees in M^* than in M, it follows that l_k is undersubscribed in M^*, since no lecturer is oversubscribed in M. In particular, there exists some project $p_j \in P_k$ and some student, say s_i, such that p_j is undersubscribed in M^* and $(s_i, p_j) \in M \setminus M^*$. Since $(s_i, p_j) \in M$, then p_j must have been in the head of s_i's preference list when s_i applied to p_j during the algorithm's execution. By Lemma 4.4.1, either s_i is unassigned in M^* or s_i prefers p_j to $M^*(s_i)$ or s_i is indifferent between them. Hence (s_i, p_j) blocks M^*, a contradiction.

Next, suppose (b) holds. Then $|S_1| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)|$. As in case (a), since $|S_2| \leq |S_1|$, it follows that

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)| \ .$$

Suppose first that $|M^*(l_k)| < |M(l_k)|$ for some $l_k \in \mathcal{L}$. Then l_k has fewer assignees in M^* than in M, and following a similar argument as in case (a) above, we reach an immediate

contradiction. Hence, $|M^*(l_k)| = |M(l_k)|$ for all $l_k \in \mathcal{L}$. Further, for each $l_k \in \mathcal{L}$ we claim that every student who is assigned to l_k in M is also assigned to l_k in M^*. Suppose otherwise. Let l_{z_1} be an arbitrary lecturer in $\mathcal{L}$. Then there exists some student $s_{q_1} \in M(l_{z_1}) \setminus M^*(l_{z_1})$. Let $M(s_{q_1}) = p_{t_2}$. By Lemma 4.4.1, s_{q_1} is assigned in M^* to a project p_{t_1} such that s_{q_1} prefers p_{t_2} to p_{t_1}. Clearly, p_{t_1} is not offered by l_{z_1}, since $s_{q_1} \in M(l_{z_1}) \setminus M^*(l_{z_1})$. We also note that s_{q_1} cannot be indifferent between p_{t_2} and p_{t_1}. Otherwise, the argument follows from (a), since s_{q_1} is assigned in M to two projects offered by different lecturers, and we reach an immediate contradiction. By the super-stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers every student in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^* and l_{z_1} prefers every student in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise, (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}, and clearly $s_{q_2} \neq s_{q_1}$. Let $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). Applying similar reasoning as for s_{q_1}, student s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 4.4.3, we can identify an infinite sequence of distinct students and projects, a contradiction.

Now, let s_i be an arbitrary student such that s_i is assigned in M to two or more projects offered by a lecturer, say l_k. Then $s_i \in M^*(l_k)$. Moreover, there exists some project $p_j \in P_k$ such that $(s_i, p_j) \in M \setminus M^*$. We claim that p_j is undersubscribed in M^*. Suppose otherwise. Let $l_{z_0} = l_k$, $p_{t_0} = p_j$ and $s_{q_0} = s_i$. Then there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus M(p_{t_0})$, since p_{t_0} is not oversubscribed in M and $s_{q_0} \in M(p_{t_0}) \setminus M^*(p_{t_0})$. Again, by Lemma 4.4.1, s_{q_1} is assigned in M to a project p_{t_1} such that s_{q_1} prefers p_{t_1} to p_{t_0}. Let l_{z_1} be the lecturer who offers p_{t_1}. Following a similar argument as in the proof of Lemma 4.4.3, we can identify a sequence of distinct students and projects, and as this sequence is infinite, we reach a contradiction. Hence our claim holds, i.e., p_j is undersubscribed in M^*. Finally, since s_i cannot be assigned to any project that she prefers to p_j in M^* and since $(s_i, p_j) \in M^*(l_k)$, we have that (s_i, p_j) blocks M^*, a contradiction. □

Lemma 4.4.6. *If some lecturer l_k becomes full during some execution of Algorithm* SPA-ST-super *and l_k subsequently ends up undersubscribed at the termination of the algorithm, then I admits no super-stable matching.*

Proof. Let M be the assignment at the termination of the algorithm. Suppose for a contradiction that there exists a super-stable matching M^* in I. Let l_k be the lecturer who became full during some execution of the algorithm and subsequently ends up undersubscribed in M. By Lemma 4.4.4, $|M(l_k)| = |M^*(l_k)|$ and thus l_k is undersubscribed in M^*. At the point in the algorithm where l_k became full (line 22), we note that none of the projects offered by l_k is oversubscribed. Since l_k ended up undersubscribed in M, it follows that there is some project $p_j \in P_k$ that has fewer assignees in M at the termination of the algorithm than it had at some point during the algorithm's execution, thus p_j is undersubscribed in M.

We claim that each project offered by l_k has the same number of assignees in M^* as in M. Suppose otherwise, then there is some project $p_t \in P_k$ such that $|M^*(p_t)| < |M(p_t)|$; thus p_t is undersubscribed in M^*, since no project is oversubscribed in M. It follows that there exists some student $s_r \in M(p_t) \setminus M^*(p_t)$. By Lemma 4.4.1, s_r is either unassigned in M^* or prefers p_t to $M^*(s_r)$. Since l_k is undersubscribed in M^*, (s_r, p_t) blocks M^*, a contradiction. Hence $|M^*(p_t)| \geq |M(p_t)|$. Moreover, since $|M(l_k)| = |M^*(l_k)|$, we have that $|M(p_t)| = |M^*(p_t)|$ for all $p_t \in P_k$.

Hence p_j undersubscribed in M implies that p_j is undersubscribed in M^*. Moreover, there is some student s_i who was provisionally assigned to p_j at some point during the execution of the algorithm but s_i is not assigned to p_j in M. Thus, the pair (s_i, p_j) was deleted during the algorithm's execution, so that $(s_i, p_j) \notin M^*$ by Lemma 4.4.1. It follows that either s_i is unassigned in M^* or s_i prefers p_j to $M^*(s_i)$ or s_i is indifferent between them. Hence, (s_i, p_j) blocks M^*, a contradiction. □

Lemma 4.4.7. *If the pair (s_i, p_j) was deleted during some execution of Algorithm* SPA-ST-super*, and at the termination of the algorithm s_i is not assigned to a project better than p_j, and each of p_j and l_k is undersubscribed, then I admits no super-stable matching.*

Proof. Suppose for a contradiction that there exists a super-stable matching M^* in I. Let (s_i, p_j) be a pair that was deleted during an arbitrary execution E of the algorithm. This implies that $(s_i, p_j) \notin M^*$ by Lemma 4.4.1. Let M be the assignment at the termination of E. By the hypothesis of the lemma, l_k is undersubscribed in M. This implies that l_k is undersubscribed in M^*, by Lemma 4.4.4. Since p_j is offered by l_k, and p_j is undersubscribed in M, it follows from the proof of Lemma 4.4.6 that p_j is undersubscribed in M^*. Further, by the hypothesis of the lemma, either s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$ or is indifferent between them. By Lemma 4.4.1, this is true for s_i in M^*. Hence (s_i, p_j) blocks M^*, a contradiction. □

The next lemma shows that the final assignment may be used to determine the existence, or otherwise, of a super-stable matching in I.

Lemma 4.4.8. *If at the termination of Algorithm* SPA-ST-super*, the assignment M is not super-stable in I then no super-stable matching exists in I.*

Proof. Suppose M is not super-stable in I. If some student s_i is assigned to two or more projects in M then I admits no super-stable matching, by Lemma 4.4.5. Hence every student is assigned to at most one project in M. Moreover, since no project or lecturer is oversubscribed in M, it follows that M is a matching. Let (s_i, p_j) be a blocking pair for M, then s_i is either unassigned in M or prefers p_j to $M(s_i)$ or is indifferent between them. Whichever is the case, (s_i, p_j) has been deleted. Let l_k be the lecturer who offers p_j. In what follows, we identify the point in the algorithm at which (s_i, p_j) was deleted, and we arrive at a conclusion that no super-stable matching exists.

Firstly, suppose (s_i, p_j) was deleted as a result of p_j being full or oversubscribed (on lines 12 or 21). Suppose p_j is full in M. Then (s_i, p_j) cannot block M irrespective of whether l_k is undersubscribed or full in M, since l_k prefers the worst assigned student/s in $M(p_j)$ to s_i. Hence p_j is undersubscribed in M. As p_j was previously full, each pair (s_t, p_u), for each s_t that is no better than s_i at the tail of $\mathcal{L}_k$ and each $p_u \in P_k \cap A_t$, would have been deleted on line 34 of the algorithm. Thus, if l_k is full in M then (s_i, p_j) does not block M. Suppose l_k is undersubscribed in M. If l_k was full at some point during the execution of the algorithm then I admits no super-stable matching, by Lemma 4.4.6. Hence l_k was never full during the algorithm's execution. Recall that each of p_j and l_k is undersubscribed in M. As (s_i, p_j) is a blocking pair of M, s_i cannot be assigned in M to a project that she prefers to p_j. Hence I admits no super-stable matching, by Lemma 4.4.7.

Next, suppose (s_i, p_j) was deleted as a result of l_k being full or oversubscribed (on lines 16 or 26), (s_i, p_j) could only block M if l_k is undersubscribed in M. If this is the case then I admits no super-stable matching, by Lemma 4.4.6.

Finally, suppose (s_i, p_j) was deleted (on line 34) because some other project $p_{j'}$ offered by l_k was previously full and ended up undersubscribed on line 28. Then l_k must have identified the most-preferred student, say s_r, who was previously assigned to $p_{j'}$ but subsequently got rejected from $p_{j'}$. At this point, s_i is at the tail of $\mathcal{L}_k$ and s_i is no better than s_r in $\mathcal{L}_k$. Moreover, every project offered by l_k that s_i finds acceptable would have been deleted from s_i's preference list at the for loop iteration in line 34. If p_j is full in M then (s_i, p_j) does not block M. Hence p_j is undersubscribed in M. If l_k is full in M then (s_i, p_j) does not block M, since $s_i \notin M(l_k)$ and l_k prefers the worst student/s in $M(l_k)$ to s_i. Hence l_k is undersubscribed in M. Again by Lemma 4.4.7, I admits no super-stable matching.

Since (s_i, p_j) is an arbitrary pair, this implies that I admits no super-stable matching. □

The next lemma shows that Algorithm SPA-ST-super may be implemented to run in linear time.

Lemma 4.4.9. *Algorithm* SPA-ST-super *may be implemented to run in $O(L)$ time and $O(n_1 n_2)$ space, where n_1, n_2 and L are the number of students, number of projects and total length of the preference lists, respectively, in I.*

Proof. The algorithm's time complexity depends on how efficiently we can execute the operation of a student applying to a project and the operation of deleting a (student, project) pair, each of which occur once for any (student, project) pair. It turns out that both operations can be implemented to run in constant time, giving Algorithm SPA-ST-super an overall complexity of $\Theta(L)$, where L is the total length of all the preference lists. In what follows, we describe the non-trivial aspects of such an implementation. We remark that the data structures discussed here are inspired by, and extend, those detailed in [13, Section 3.3] for Algorithm SPA-student.

For each student s_i, build an array $position_{s_i}$, where $position_{s_i}(p_j)$ is the position of project p_j in s_i's preference list. For example, if s_i's preference list is $(p_2\ p_5\ p_3)\ p_7\ (p_6\ p_1)$ then $position_{s_i}(p_5) = 2$ and $position_{s_i}(p_1) = 6$. In general, position captures the order in which the projects appear in the preference list when read from left to right, ignoring any ties. Represent s_i's preference list by embedding doubly linked lists in an array $preference_{s_i}$. For each project $p_j \in A_i$, $preference_{s_i}(position_{s_i}(p_j))$ stores the list node containing p_j. This node contains two next pointers (and two previous pointers) – one to the next project in s_i's preference list (after deletions, this project may not be located at the next array position), and another pointer to the next project $p_{j'}$ in s_i's preference list, where $p_{j'}$ and p_j are both offered by the same lecturer. Construct the latter list by traversing through s_i's preference list, using a temporary array to record the last project in the list offered by each lecturer.

As highlighted in [13], we suggest using virtual initialisation [27] for these arrays, to avoid the $O(n_1 n_3)$ initialisation time dominating the total runtime of the algorithm. Very broadly, this technique allows us to create and access an initialised array in constant time. See [27, p. 149] for more details, and see [8] for an illustration.

To represent the ties in s_i's preference list, build an array $successor_{s_i}$. For each project p_j in s_i's preference list, $successor_{s_i}(position_{s_i}(p_j))$ stores the `true` boolean if p_j is tied with its successor in A_i and `false` otherwise. After the deletion of any (student, project) pair, update the successor booleans. As an illustration, with respect to s_i's preference list given in the previous paragraph, $successor_{s_i}$ is the array [`true, true, false, false, true, false`]. Now, suppose p_3 was deleted from s_i's preference list, since $successor_{s_i}(position_{s_i}(p_3))$ is `false` and $successor_{s_i}(position_{s_i}(p_5))$ is `true`, set $successor_{s_i}(position_{s_i}(p_5))$ to `false` (since p_5 is the predecessor of p_3). Clearly using these data structures, we can find the next project at the head of each student's preference list, find the next

project offered by a given lecturer on each student's preference list, as well as delete a project from a given student's preference list in constant time.

For each lecturer l_k, build two arrays $preference_{l_k}$ and $successor_{l_k}$, where $preference_{l_k}(s_i)$ is the position of student s_i in l_k's preference list, and $successor_{l_k}(preference_{l_k}(s_i))$ stores the position of the first strict successor (with respect to position) of s_i in $\mathcal{L}_k$ or a null value if s_i has no strict successor[3]. Represent l_k's preference list (i.e., $\mathcal{L}_k$) by the array $preference_{l_k}$, with an additional pointer, $last_{l_k}$. Initially, $last_{l_k}$ stores the index of the last position in $preference_{l_k}$. To represent the ties in l_k's preference list, build an array $predecessor_{l_k}$. For each $s_i \in \mathcal{L}_k$, $predecessor_{l_k}(preference_{l_k}(s_i))$ stores the `true` boolean if s_i is tied with its predecessor in $\mathcal{L}_k$ and `false` otherwise.

When l_k becomes full, make $last_{l_k}$ equivalent to l_k's worst assigned student through the following method. Perform a backward traversal through the array $preference_{l_k}$, starting at $last_{l_k}$, and continuing until l_k's worst assigned student, say $s_{i'}$, is encountered (each student stores a pointer to their assigned project, or a special null value if unassigned). Deletions must be carried out in the preference list of each student who is worse than $s_{i'}$ on l_k's preference list (precisely those students whose position in $preference_{l_k}$ is greater than or equal to that stored in $successor_{l_k}(preference_{l_k}(s_{i'}))$)[4].

When l_k becomes oversubscribed, we can find and delete the students at the tail of l_k by performing a backward traversal through the array $preference_{l_k}$, starting at $last_{l_k}$, and continuing until we encounter a student, say $s_{i'}$, such that $predecessor_{l_k}(preference_{l_k}(s_{i'}))$ stores the `false` boolean. If l_k becomes undersubscribed after we break the assignment of students encountered on this traversal (including $s_{i'}$) to l_k, rather than update $last_{l_k}$ immediately, which could be expensive, we wait until l_k becomes full again. The cost of these traversals taken over the algorithm's execution is thus linear in the length of l_k's preference list.

For each project p_j offered by l_k, build the arrays $preference_{p_j}$, $successor_{p_j}$ and $predecessor_{p_j}$ corresponding to $\mathcal{L}_k^j$, as described in the previous paragraph for $\mathcal{L}_k$. Represent the projected preference list of l_k for p_j (i.e., $\mathcal{L}_k^j$) by the array $preference_{p_j}$, with an additional pointer, $last_{p_j}$. These project preference arrays are used in much the same way as the lecturer preference arrays

Since we only visit a student at most twice during these backward traversals, once for the lecturer and once for the project, the asymptotic running time remains linear. □

Lemma 4.4.1 shows that there is an optimality property for each assigned student in any super-stable matching found by the algorithm, whilst Lemma 4.4.8 establishes the correct-

[3]For example, if l_k's preference list is s_5 $(s_3\ s_1\ s_6)$ s_7 $(s_2\ s_8)$ then $successor_{l_k}$ is the array $[2\ 5\ 5\ 5\ 6\ 0\ 0]$.

[4]For efficiency, we remark that it is not necessary to make deletions from the preference lists of lecturers or projected preference lists of lecturers for each project the lecturer offers, since the while loop of Algorithm SPA-ST-super involves students applying to projects in the head of their preference list.

ness of Algorithm SPA-ST-super. The following theorem collects together Lemmas 4.4.1, 4.4.8 and 4.4.9.

Theorem 4.4.10. *For a given instance I of* SPA-ST*, Algorithm* SPA-ST-super *determines, in $O(L)$ time and $O(n_1 n_2)$ space, whether or not a super-stable matching exists in I. If such a matching does exist, all possible executions of the algorithm find one in which each assigned student is assigned to the best project that she could obtain in any super-stable matching, and each unassigned student is unassigned in all super-stable matchings.*

Given the optimality property established by Theorem 4.4.10, we define the super-stable matching found by Algorithm SPA-ST-super to be *student-optimal*.

4.4.6 Properties of super-stable matchings in SPA-ST

In this section, we consider properties of the set of super-stable matchings in an instance of SPA-ST. We show that the Unpopular Projects Theorem for SPA-S (see Theorem 2.3.2) holds for SPA-ST under super-stability.

Theorem 4.4.11. *For a given instance I of* SPA-ST*, the following properties holds:*

1. *each lecturer is assigned the same number of students in all super-stable matchings;*
2. *exactly the same students are unassigned in all super-stable matchings;*
3. *a project offered by an undersubscribed lecturer has the same number of students in all super-stable matchings.*

Proof. Let M and M^* be two arbitrary super-stable matchings in I. Let I' be an instance of SPA-S obtained from I by breaking the ties in I in some way. Then by Proposition 4.2.2, each of M and M^* is stable in I'. Thus by Theorem 2.3.2, each lecturer is assigned the same number of students in M and M^*, exactly the same students are unassigned in M and M^*, and a project offered by an undersubscribed lecturer has the same number of students in M and M^*. □

To illustrate this, consider the SPA-ST instance I_3 given in Figure 4.5, which admits the super-stable matchings $M_1 = \{(s_3, p_3), (s_4, p_2), (s_5, p_3), (s_6, p_2)\}$ and $M_2 = \{(s_3, p_3), (s_4, p_3), (s_5, p_2), (s_6, p_2)\}$. The reader can easily verify that M_1 is the student-optimal super-stable matching in I_3. Each of l_1 and l_2 is assigned the same number of students in both M_1 and M_2, illustrating part (1) of Theorem 4.4.11. Also, each of s_1 and s_2 is unassigned in both M_1 and M_2, illustrating part (2) of Theorem 4.4.11. Finally, l_2 is undersubscribed in both M_1 and M_2, and each of p_3 and p_4 has the same number of students in both M_1 and M_2, illustrating part (3) of Theorem 4.4.11.

Students' preferences	Lecturers' preferences	offers
s_1: p_1	l_1: s_5 s_6 s_4 $(s_1$ $s_2)$ s_3	p_1, p_2
s_2: $(p_1$ $p_3)$	l_2: s_3 s_4 s_5 s_6 s_2	p_3, p_4
s_3: p_2 p_3		
s_4: p_2 p_3		
s_5: p_3 p_2	Project capacities: $c_1 = c_4 = 1,\ c_2 = c_3 = 2$	
s_6: p_2 p_4	Lecturer capacities: $d_1 = 2, d_2 = 3$	

Figure 4.5: An instance I_3 of SPA-ST.

4.5 An IP model for super-stability in SPA-ST

4.5.1 Introduction

In this section, we describe an IP model for super-stability in SPA-ST. Although a super-stable matching in an instance of SPA-ST can be found in polynomial-time (as illustrated by Theorem 4.4.10), we reiterate that our reason for this is purely experimental. Let I be an instance of SPA-ST involving a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of students, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of projects and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of lecturers. We construct an IP model J of I as follows. Firstly, we create binary variables $x_{i,j} \in \{0, 1\}$ $(1 \leq i \leq n_1, 1 \leq j \leq n_2)$ for each acceptable pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ such that $x_{i,j}$ indicates whether s_i is assigned to p_j in a solution or not. Henceforth, we denote by S a solution in the IP model J, and we denote by M the matching derived from S in the following natural way: if $x_{i,j} = 1$ under S then s_i is assigned to p_j in M, otherwise s_i is not assigned to p_j in M.

4.5.2 Constraints

In this section, we give the set of constraints to ensure that the assignment obtained from a feasible solution in J is a matching, and that the matching admits no blocking pair.

Matching constraints. The feasibility of a matching can be ensured with the following three set of constraints.

$$\sum_{p_j \in A_i} x_{i,j} \leq 1 \qquad (1 \leq i \leq n_1), \tag{4.5}$$

$$\sum_{i=1}^{n_1} x_{i,j} \leq c_j \qquad (1 \leq j \leq n_2), \tag{4.6}$$

$$\sum_{i=1}^{n_1} \sum_{p_j \in P_k} x_{i,j} \leq d_k \qquad (1 \leq k \leq n_3)\ . \tag{4.7}$$

Note that Inequality (4.5) ensures that each student $s_i \in \mathcal{S}$ is not assigned to more than one project, while Inequalities (4.6) and (4.7) ensure that the capacity of each project $p_j \in \mathcal{P}$ and each lecturer $l_k \in \mathcal{L}$ is not exceeded.

Given an acceptable pair (s_i, p_j), we define $\mathrm{rank}(s_i, p_j)$, the *rank* of p_j on s_i's preference list, to be $r+1$, where r is the number of projects that s_i prefers to p_j. Clearly, projects that are tied together on s_i's preference list have the same rank. Given a lecturer $l_k \in \mathcal{L}$ and a student $s_i \in \mathcal{L}_k$, we define $\mathrm{rank}(l_k, s_i)$, the *rank* of s_i on l_k's preference list, to be $r+1$, where r is the number of students that l_k prefers to s_i. Similarly, students that are tied together on l_k's preference list have the same rank. With respect to an acceptable pair (s_i, p_j), we define $S_{i,j} = \{p_{j'} \in A_i : \mathrm{rank}(s_i, p_{j'}) < \mathrm{rank}(s_i, p_j)\}$, the set of projects that s_i prefers to p_j. Let l_k be the lecturer who offers p_j. We also define $T_{i,j} = \{s_{i'} \in \mathcal{L}_k^j : \mathrm{rank}(l_k, s_{i'}) < \mathrm{rank}(l_k, s_i)\}$, the set of students that are better than s_i on the projected preference list of l_k for p_j. Finally, we define $D_{i,k} = \{s_{i'} \in \mathcal{L}_k : \mathrm{rank}(l_k, s_{i'}) < \mathrm{rank}(l_k, s_i)\}$, the set of students that are better than s_i on l_k's preference list.

In what follows, we fix an arbitrary acceptable pair (s_i, p_j) and we enforce constraints to ensure that (s_i, p_j) does not form a blocking pair for the matching M. Henceforth, l_k is the lecturer who offers p_j.

Blocking pair constraints. First, we define $\theta_{i,j} = 1 - x_{i,j} - \sum_{p_{j'} \in S_{i,j}} x_{i,j'}$. Intuitively, $\theta_{i,j} = 1$ if and only if s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$ or is indifferent between them. Henceforth, if (s_i, p_j) forms a blocking pair for M then we refer to (s_i, p_j) as a blocking pair of type (i), type (ii) or type (iii), according as (s_i, p_j) satisfies condition (i), (ii) or (iii) of Definition 4.2.1, respectively. We describe the constraints to avoid these types of blocking pair as follows.

Type (i). First, we create a binary variable α_j in J such that if p_j is undersubscribed in M then $\alpha_j = 1$. We enforce this condition by imposing the following constraint.

$$c_j \alpha_j \geq c_j - \sum_{i'=1}^{n_1} x_{i',j}, \tag{4.8}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$. If p_j is undersubscribed in M then the RHS of Inequality (4.8) is at least 1 and this implies that $\alpha_j = 1$, otherwise α_j is not constrained. Next, we create a binary variable β_k in J such that if l_k is undersubscribed in M then $\beta_k = 1$. We enforce this condition by imposing the following constraint:

$$d_k \beta_k \geq d_k - \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'}, \tag{4.9}$$

where $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} = |M(l_k)|$. If l_k is undersubscribed in M then the RHS of Inequality (4.9) is at least 1 and this implies that $\beta_k = 1$, otherwise β_k is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (i) blocking pair for M.

$$\boxed{\theta_{i,j} + \alpha_j + \beta_k \leq 2 \ .} \tag{4.10}$$

Type (ii). We create a binary variable η_k in J such that if l_k is full in M then $\eta_k = 1$. We enforce this condition by imposing the following constraint.

$$d_k \eta_k \geq \left(1 + \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} \right) - d_k \ . \tag{4.11}$$

If l_k is full in M then the RHS of Inequality (4.11) is at least 1 and this implies that $\eta_k = 1$, otherwise η_k is not constrained. Next, we create a binary variable $\delta_{i,k}$ in J such that if $s_i \in M(l_k)$, or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them, then $\delta_{i,k} = 1$. We enforce this condition by imposing the following constraint.

$$d_k \delta_{i,k} \geq \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} - \sum_{s_{i'} \in D_{i,k}} \sum_{p_{j'} \in P_k} x_{i',j'} \ . \tag{4.12}$$

Note that if $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or l_k is indifferent between them, then the RHS of Inequality (4.12) is at least 1 and this implies that $\delta_{i,k} = 1$, otherwise $\delta_{i,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (ii) blocking pair for M.

$$\boxed{\theta_{i,j} + \alpha_j + \eta_k + \delta_{i,k} \leq 3 \ .} \tag{4.13}$$

Type (iii). Next we create a binary variable γ_j in J such that if p_j is full in M then $\gamma_j = 1$. We enforce this condition by imposing the following constraint.

$$c_j \gamma_j \geq \left(1 + \sum_{i'=1}^{n_1} x_{i',j} \right) - c_j \ . \tag{4.14}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$. If p_j is full in M then the RHS of Inequality (4.14) is at least 1 and this implies that $\gamma_j = 1$, otherwise γ_j is not constrained. Next, we create a binary variable $\lambda_{i,j,k}$ in J such that if l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them,

then $\lambda_{i,j,k} = 1$. We enforce this condition by imposing the following constraint.

$$c_j \lambda_{i,j,k} \geq \sum_{i'=1}^{n_1} x_{i',j} - \sum_{s_{i'} \in T_{i,j}} x_{i',j} \ . \tag{4.15}$$

Note that if l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, then the RHS of Inequality (4.15) is at least 1 and this implies that $\lambda_{i,j,k} = 1$, otherwise $\lambda_{i,j,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (iii) blocking pair for M.

$$\boxed{\theta_{i,j} + \gamma_j + \lambda_{i,j,k} \leq 2 \ .} \tag{4.16}$$

4.5.3 Variables

We define a collective notation for each set of variables involved in J as follows:

$A = \{\alpha_j : 1 \leq j \leq n_2\}$, $\quad \Gamma = \{\gamma_j : 1 \leq j \leq n_2\}$,
$B = \{\beta_k : 1 \leq k \leq n_3\}$, $\quad \Delta = \{\delta_{i,k} : 1 \leq i \leq n_1, 1 \leq k \leq n_3\}$,
$N = \{\eta_k : 1 \leq k \leq n_3\}$, $\quad \Lambda = \{\lambda_{i,j,k} : 1 \leq i \leq n_1, 1 \leq j \leq n_2, 1 \leq k \leq n_3\}$,
$X = \{x_{i,j} : s_i \in \mathcal{S} \wedge p_j \in A_i\}$.

4.5.4 Objective function

On one hand, all super-stable matchings are of the same size, and thus nullifies the need for an objective function. On the other hand, optimization solvers require an objective function in addition to the variables and constraints in order to produce a solution. The objective function given below involves maximising the summation of all the $x_{i,j}$ binary variables.

$$\boxed{\max \sum_{i=1}^{n_1} \sum_{p_j \in A_i} x_{i,j} \ .} \tag{4.17}$$

Finally, we have constructed an IP model J of I comprising the set of integer-valued variables $A, B, N, X, \Gamma, \Delta,$ and Λ, the set of Inequalities (4.5) - (4.16) and an objective function (4.17). Note that J can then be used to construct a super-stable matching in I, should one exist.

4.5.5 Correctness of the IP model

Given an instance I of SPA-ST formulated as an IP model J using the above transformation, we present the following lemmas regarding the correctness of J.

Lemma 4.5.1. *A feasible solution S to J corresponds to a super-stable matching M in I.*

Proof. Assume firstly that J has a feasible solution S. Let $M = \{(s_i, p_j) \in \mathcal{S} \times \mathcal{P} : x_{i,j} = 1\}$ be the assignment in I generated from S. We note that Inequality (4.5) ensures that each student is assigned in M to at most one project. Moreover, Inequalities (4.6) and (4.7) ensures that the capacity of each project and lecturer is not exceeded in M. Thus M is a matching. We will prove that Inequalities (4.8) - (4.16) ensures that M admits no blocking pair.

Suppose for a contradiction that there exists some acceptable pair (s_i, p_j) that forms a blocking pair for M, where l_k is the lecturer who offers p_j. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. Thus $\sum_{p_{j'} \in S_{i,j}} x_{i,j'} = 0$. Moreover, since s_i is not assigned to p_j in M, we have that $x_{i,j} = 0$. Thus $\theta_{i,j} = 1$.

Now suppose (s_i, p_j) forms a type (i) blocking pair for M. Then each of p_j and l_k is undersubscribed in M. Thus $\sum_{i'=1}^{n_1} x_{i',j} < c_j$ and $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} < d_k$. This implies that the RHS of Inequality (4.8) and the RHS of Inequality (4.9) is strictly greater than 0. Moreover, since S is a feasible solution to J, $\alpha_j = \beta_k = 1$. Hence, the LHS of Inequality (4.10) is strictly greater than 2, a contradiction to the feasibility of S.

Now suppose (s_i, p_j) forms a type (ii) blocking pair for M. Then p_j is undersubscribed in M and as explained above, $\alpha_j = 1$. Also, l_k is full in M and this implies that the RHS of Inequality (4.11) is strictly greater than 0. Since S is a feasible solution, we have that $\eta_k = 1$. Furthermore, either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or l_k is indifferent between them. In any of these cases, the RHS of Inequality (4.12) is strictly greater than 0. Thus $\delta_{i,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (4.13) is strictly greater than 3, a contradiction to the feasibility of S.

Finally, suppose (s_i, p_j) forms a type (iii) blocking pair for M. Then p_j is full in M and thus the RHS of Inequality (4.14) is strictly greater than 0. Since S is a feasible solution, we have that $\gamma_j = 1$. In addition, l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them. This implies that the RHS of Inequality (4.15) is strictly greater than 0. Thus $\lambda_{i,j,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (4.16) is strictly greater than 2, a contradiction to the feasibility of S. Hence M admits no blocking pair; and hence, M is a super-stable matching in I. □

Lemma 4.5.2. *A super-stable matching M in I corresponds to a feasible solution S to J.*

Proof. Let M be a super-stable matching in I. First we set all the binary variables involved in J to 0. For each $(s_i, p_j) \in M$, we set $x_{i,j} = 1$. Since M is a matching, it is clear that Inequalities (4.5) - (4.7) is satisfied. For any acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them, we set

$\theta_{i,j} = 1$. For any project $p_j \in \mathcal{P}$ such that p_j is undersubscribed in M, we set $\alpha_j = 1$ and thus Inequality (4.8) is satisfied. For any lecturer $l_k \in \mathcal{L}$ such that l_k is undersubscribed in M, we set $\beta_k = 1$ and thus Inequality (4.9) is satisfied.

Now, for Inequality (4.10) not to be satisfied, its LHS must be strictly greater than 2. This would only happen if there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$ and $\beta_k = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them, and each of p_j and l_k is undersubscribed in M. Thus (s_i, p_j) forms a type (i) blocking pair for M, a contradiction to the super-stability of M. Hence, Inequality (4.10) is satisfied.

For any lecturer $l_k \in \mathcal{L}$ such that l_k is full in M, we set $\eta_k = 1$. Thus Inequality (4.11) is satisfied. Let (s_i, p_j) be an acceptable pair such that $p_j \in P_k$ and $(s_i, p_j) \notin M$. If $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them, we set $\delta_{i,k} = 1$. Thus Inequality (4.12) is satisfied. Suppose Inequality (4.13) is not satisfied. Then there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$, $\eta_k = 1$ and $\delta_{i,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. In addition, p_j is undersubscribed in M, l_k is full in M and either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them. Thus (s_i, p_j) forms a type (ii) blocking pair for M, a contradiction to the super-stability of M. Hence Inequality (4.13) is satisfied.

Finally, for any project $p_j \in \mathcal{P}$ such that p_j is full in M, we set $\gamma_j = 1$. Thus Inequality (4.14) is satisfied. Let l_k be the lecturer who offers p_j and let (s_i, p_j) be an acceptable pair. If l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, we set $\lambda_{i,j,k} = 1$. Thus Inequality (4.15) is satisfied. Suppose Inequality (4.16) is not satisfied. Then there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that $\theta_{i,j} = 1$, $\gamma_j = 1$ and $\lambda_{i,j,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. In addition, p_j is full in M and l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them. Thus (s_i, p_j) forms a type (iii) blocking pair for M, a contradiction to the super-stability of M. Hence, Inequality (4.16) is satisfied. Hence S, comprising the above assignments of values to the variables in $A \cup B \cup N \cup X \cup \Gamma \cup \Delta \cup \Lambda$, is a feasible solution to J. □

The following theorem is a consequence of Lemmas 4.5.1 and 4.5.2.

Theorem 4.5.3. *Let I be an instance of* SPA-ST *and let J be the IP model for I as described above. A feasible solution to J corresponds to a super-stable matching in I. Conversely, a super-stable matching in I corresponds to a feasible solution to J.*

4.6 Empirical evaluation

In this section, we evaluate an implementation of Algorithm SPA-ST-super empirically. We implemented our algorithm in Python[5], and we performed our experiments on a system with dual Intel Xeon CPU E5-2640 processors with 64GB of RAM, running Ubuntu 17.10. For our experiment, we were primarily concerned with the following question: how does the nature of the preference lists in a given SPA-ST instance affect the existence of a super-stable matching?

4.6.1 Datasets

When generating random datasets, there are clearly several parameters that can be varied, such as the number of students, projects and lecturers; the lengths of the students' preference lists as well as a measure of the density of ties present in the preference lists. We denote by t_d, the measure of the density of ties present in the preference lists. In each student's preference list, the tie density t_{d_s} $(0 \leq t_{d_s} \leq 1)$ is the probability that some project is tied to its successor. The tie density t_{d_l} in each lecturer's preference list is defined similarly. At $t_{d_s} = t_{d_l} = 1$, each preference list comprises a single tie while at $t_{d_s} = t_{d_l} = 0$, no tie would exist in the preference lists, thus reducing the problem to an instance of SPA-S.

4.6.2 Experimental setup

For each range of values for the aforementioned parameters, we randomly generated a set of SPA-ST instances, involving n_1 students (which we will henceforth refer to as the size of the instance), $0.5n_1$ projects, $0.2n_1$ lecturers and $1.2n_1$ total project capacity which was randomly distributed amongst the projects such that each project has capacity at least 1. The capacity for each lecturer l_k was chosen uniformly at random to lie between the highest capacity of the projects offered by l_k and the sum of the capacities of the projects that l_k offers. In each set, we measured the proportion of instances that admit a super-stable matching. We remark that the parameter values were chosen to ensure that projects could typically accommodate more than one student, that the total capacity of the projects exceeded the number of students, and that each lecturer typically offered multiple projects, without reflecting any specific real-world application. Further, we note that the instances that we used for our empirical analysis in this section, and in Sections 5.5 and 6.6 were generated using a simple strategy.

It is worth mentioning that when we varied the tie density on both the students' and lecturers' preference lists between 0.1 and 0.5, super-stable matchings were very elusive, even with an

[5]https://github.com/sofiatolaosebikan/spa-st-super

instance size of 100 students. Thus, for the purpose of our experiment, we decided to choose a low tie density.

4.6.3 Correctness testing

To test the correctness of our algorithm's implementation, we implemented our IP model for SPA-ST under super-stability using the Gurobi optimisation solver in Python. We randomly generated $10,000$ SPA-ST instances, each consisting of 100 students and a constant ratio of projects, lecturers, project capacities and lecturer capacities as described above. Also, each student's preference list was fixed at 10, with a tie density of 0.1. With this setup, we verified consistency between the outcomes of our implementation of Algorithm SPA-ST-super and our implementation of the IP-based algorithm in terms of the existence or otherwise of a super-stable matching.

4.6.4 Experimental results

Experiment 1

In our first experiment, we examined how the length of the students' preference lists affects the existence of a super-stable matching. We increased the number of students n_1 while maintaining a constant ratio of projects, lecturers, project capacities and lecturer capacities as described above. For various values of n_1 $(100 \leq n_1 \leq 1000)$ in increments of 100, we varied the length of each student's preference list for various values of x $(5 \leq x \leq 50)$ in increments of 5; and with each of these parameters, we randomly generated 1000 instances. For all the preference lists, we set $t_{d_s} = t_{d_l} = 0.005$ (on average, 1 out of 5 students has a single tie of length 2 in their preference list, and this holds similarly for the lecturers).

The result, which is displayed in Figure 4.6, shows that as we varied the length of the preference list, there was no significant uplift in the number of instances that admitted a super-stable matching. In most cases, we observed that the proportion of instances that admit a super-stable matching is slightly higher when the preference list length is 50 compared to when the preference list length is 5. The result also shows that the proportion of instances that admit a super-stable matching decreases as the number of students increases. Further, we recorded the time taken for our algorithm's implementation to terminate, and as can be seen in Table 4.2, for an instance size of 1000 and preference list length 50, the algorithm terminates in approximately 0.4 second.

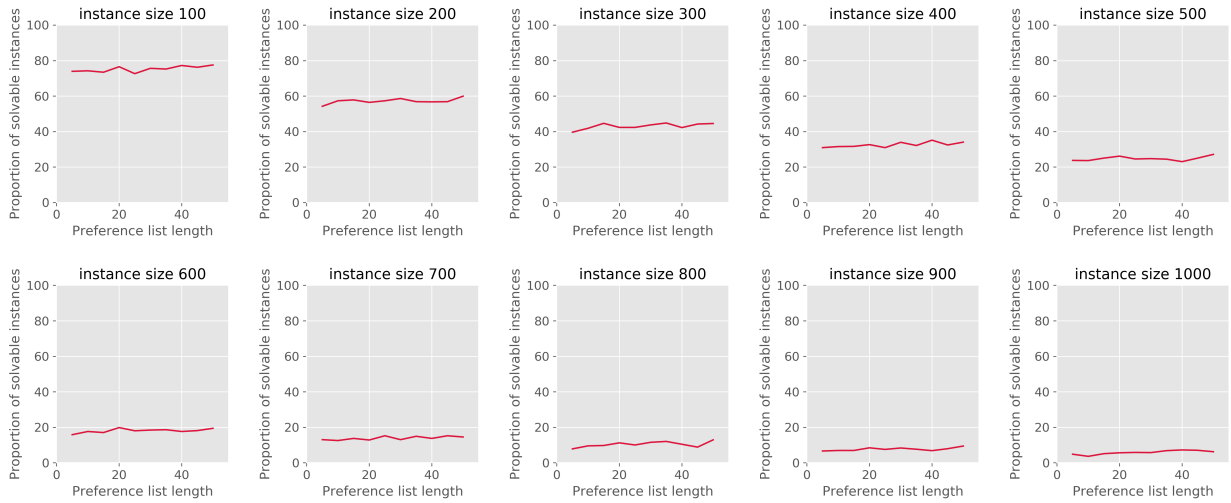


Figure 4.6: Proportion of instances that admit a super-stable matching as the size of the instance increases while varying the length of the preference lists with tie density fixed at 0.005 in both the students' and lecturers' preference lists.

Table 4.2: Time (in seconds) for our algorithm's implementation to terminate averaged over 1000 for each instance size, with the length of each student's preference list fixed at 50.

n_1	100	200	300	400	500	600	700	800	900	1000
Time	0.017	0.046	0.082	0.120	0.160	0.203	0.248	0.298	0.349	0.399

Experiment 2

In our second experiment, we investigated how the variation in tie density in both the students' and lecturers' preference lists affects the existence of a super-stable matching. To achieve this, we varied the tie density in the students' preference lists t_{d_s} $(0 \leq t_{d_s} \leq 0.05)$ and the tie density in the lecturers' preference lists t_{d_l} $(0 \leq t_{d_l} \leq 0.05)$, both in increments of 0.005. For each pair of tie densities in $t_{d_s} \times t_{d_l}$, we randomly-generated 1000 SPA-ST instances for various values of n_1 $(100 \leq n_1 \leq 1000)$ in increments of 100. For each of these instances, we maintained the same ratio of projects, lecturers, project capacities and lecturer capacities as in Experiment 1. Considering our discussion from Experiment 1, we fixed the length of each student's preference list at 50.

The result displayed in Figure 4.7 shows that increasing the tie density in both the students' and lecturers' preference lists reduces the proportion of instances that admit a super-stable matching. In fact, this proportion reduces further as the size of the instance increases. However, it was interesting to see that when we fixed the tie density in the students' preference lists at 0 and varied the tie density in the lecturers' preference lists, about 74% of

the randomly-generated SPA-ST instances involving 1000 students admitted a super-stable matching.

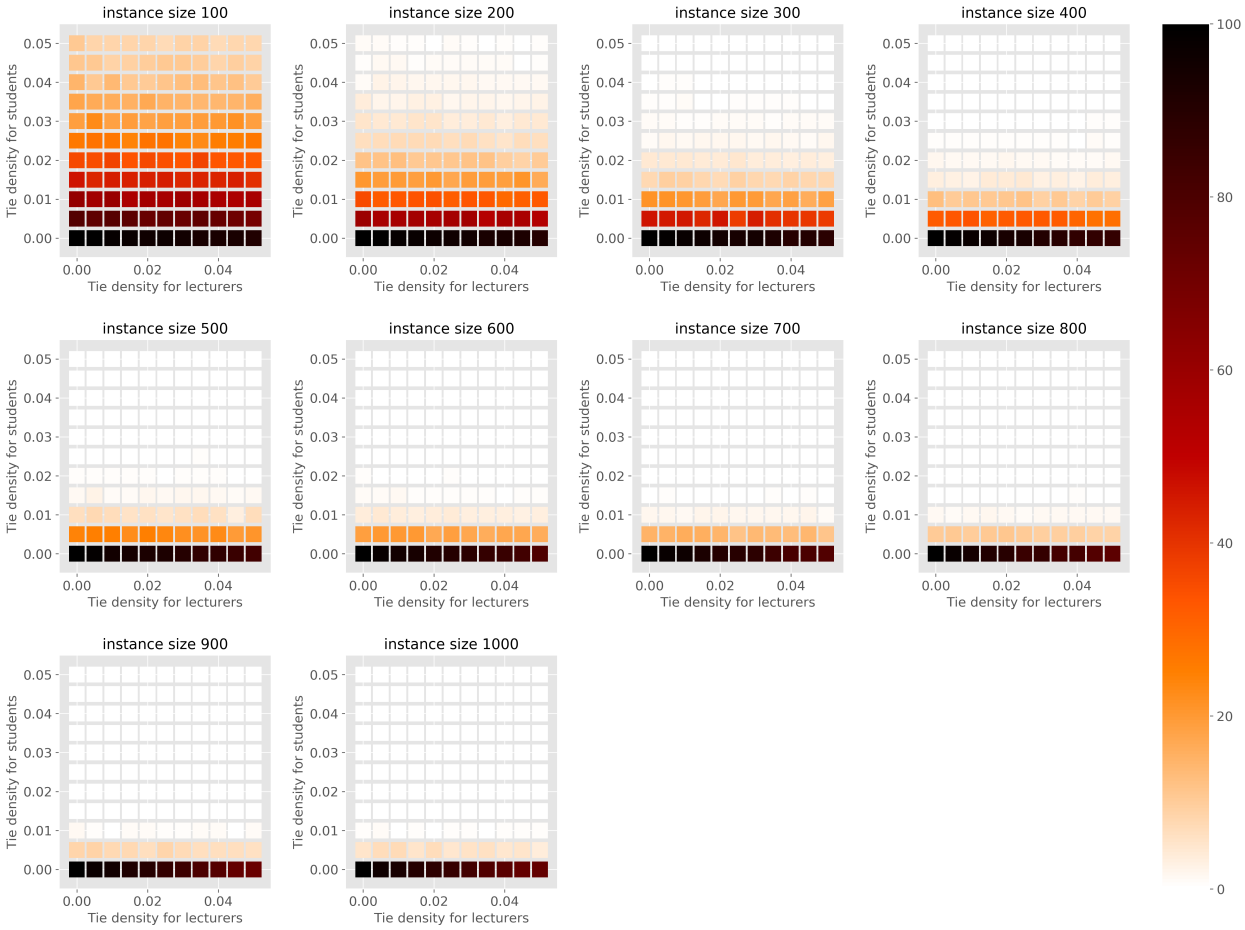


Figure 4.7: Result for Experiment 2. Each of the coloured square boxes represents the proportion of the 1000 randomly-generated SPA-ST instances that admit a super-stable matching, with respect to the tie density in the students' and lecturers' preference lists. See the colour bar transition, as this proportion ranges from dark (100%) to light (0%).

4.7 Conclusions and open problems

In this chapter, we have described a linear-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of SPA-ST. We established that for instances that do admit a super-stable matching, our algorithm produces the student-optimal super-stable matching, in the sense that each assigned student has the best project that she could obtain in any super-stable matching. We leave open the formulation of a lecturer-oriented counterpart to our algorithm.

We also described an IP model for super-stability in SPA-ST. Further, we carried out an

empirical evaluation of our algorithm's implementation. The purpose of our experiments was to investigate how the nature of the preference lists affects the existence (or otherwise) of super-stable matchings in an arbitrary instance of SPA-ST.

Based on the instances we generated randomly, the experimental results suggest that as we increase the size of the instance and the density of ties in the preference lists, the likelihood of a super-stable matching existing decreases. There was no significant uplift in this likelihood even as we increased the length of the students' preference lists. When the ties occur only in the lecturers' preference lists, we found that a significantly higher proportion of instances admit a super-stable matching. However, the reverse is the case when the ties occur only in the students' preference lists. We have no explanation for this outcome.

Given that there are typically more students than lecturers in practical applications, it could be that only lecturers are permitted to have some form of indifference over the students that they find acceptable, whilst each student might be able to provide a strict ordering over what may be a small number of projects that she finds acceptable. Further evaluation of our implementation could investigate how other parameters (e.g., the popularity of some projects, or the position of the ties in the preference lists) affect the existence of a super-stable matching. It would also be interesting to examine the existence (or otherwise) of super-stable matchings in real SPA-ST datasets.

From a theoretical perspective, the following are other directions for future work. Let I be an arbitrary instance of SPA-ST.

1. Can we formalise the results on the probability of a super-stable matching existing in I? This question has been partially explored for the Stable Roommates problem [102].

2. Is there a characterisation of the set of super-stable matchings in I in terms of a lattice structure? It is known that the set of super-stable matchings in an instance of SMT forms a distributive lattice [82, 114]. To generalise this structural result for SPA-ST, ideas from [82, 114], as well as Chapter 3 of this thesis, would certainly be useful.

Chapter 5

Strong Stability in SPA-ST

5.1 Introduction

The experimental results arising from our contribution in Chapter 4 suggest that super-stable matchings can be very elusive given an arbitrary instance of SPA-ST. To cope with the possible non-existence of matchings satisfying this stability concept, rather than settle for a weakly stable matching, a natural strategy would be to seek a strongly stable matching if one exists.

It was motivated in [59] that weakly stable matching may be undermined by bribery or persuasion, in practical applications of HRT. In what follows, we give a corresponding argument for an instance I of SPA-ST. Suppose that a matching M is a weakly stable matching in I, and suppose that a student s_i prefers project p_j (where p_j is offered by lecturer l_k) to her assigned project in M, say $p_{j'}$ (where $p_{j'}$ is offered by a lecturer different from l_k). Suppose further that p_j is full in M and l_k is indifferent between s_i and one of the worst student/s assigned to p_j in M, say $s_{i'}$. Clearly, the pair (s_i, p_j) does not constitute a blocking pair for M under weak stability as l_k would not improve by taking on s_i in the place of $s_{i'}$. However, s_i might be so invested in p_j that she is ready to persuade or even bribe l_k to reject $s_{i'}$ and accept her instead. Lecturer l_k, being indifferent between s_i and $s_{i'}$, may decide to accept s_i's proposal. We can reach a similar argument if the roles are reversed. However, if M is strongly stable, it cannot be potentially undermined by this type of student-project pair.

Throughout this chapter, if a SPA-ST instance admits a strongly stable matching, we say that such an instance is solvable. Unfortunately not every instance of SPA-ST is solvable. To see this, consider the case where there are two students, two projects and two lecturers, each lecturer offers one project, the capacity of each project and lecturer is 1, the students have exactly the same strictly-ordered preference list of length 2, and each lecturer's preference list is a single tie of length 2. Then any matching will be undermined by a student and lecturer

that are not matched together). However, it should be clear from the discussions above that in cases where a strongly stable matching exists, it should be preferred over a matching that is merely weakly stable.

Our contribution in this chapter is to present theoretical and experimental results for SPA-ST under strong stability. On the theoretical side, we present the first polynomial-time algorithm to find a strongly stable matching or report that no such matching exists, given an instance of SPA-ST – thus solving an open problem given in [13, 97]. Our algorithm is student-oriented because it involves the students applying to projects. Moreover, the algorithm returns the student-optimal strongly stable matching, in the sense that if the given instance is solvable then our algorithm will output a solution in which each student has at least as good a project as she could obtain in any strongly stable matching that the instance admits. We note that our algorithm is a non-trivial extension of the strong stability algorithms for SMT [51], SMTI [81], and HRT [59] (we discuss this further in Section 5.3.3).

On the experimental side, we present results of an empirical evaluation based on an implementation of our polynomial-time algorithm that investigates the proportion of randomly-generated SPA-ST instances that admit strongly stable matchings but no super-stable matchings. With respect to the datasets that we used for our super-stability experiments, we observed that the proportion of instances that admitted strongly stable matchings are exactly the same as those that admitted super-stable matchings. However, when we varied the size of the instance between 10 and 50, there was a slight increase in the proportion of instances that admitted strongly stable matchings but no super-stable matchings.

The remainder of this chapter is structured as follows. We give a formal definition for the strong stability concept in Section 5.2, followed by some justification for the definition. We describe our algorithm for SPA-ST under strong stability in Section 5.3. Further, in Section 5.3: we discuss the non-triviality of extending the strong stability algorithms in the literature [51, 59, 81] to the SPA-ST setting; we illustrate an execution of our algorithm with respect to an example instance of SPA-ST; and we present the algorithm's correctness and complexity results. To end Section 5.3, we give some structural properties satisfied by the set of strongly stable matchings in an instance of SPA-ST.

In Section 5.4, we present an IP model for SPA-ST under strong stability. Similar to the justification we gave in the super-stability setting, we intend to use an implementation of the IP model to test the correctness of our algorithm's implementation. In Section 5.5, we present the experimental results obtained from our algorithm's empirical evaluation. Finally, in Section 5.6, we present some concluding remarks and potential direction for future work.

Students' preferences	Lecturers' preferences	offers
s_1: $(p_1 \quad p_2)$	l_1: $s_3 \ (s_1 \quad s_2)$	p_1, p_2
s_2: $p_2 \quad p_3$	l_2: $(s_3 \quad s_2)$	p_3
s_3: $p_3 \quad p_1$		
	Project capacities: $c_1 = c_2 = c_3 = 1$	
	Lecturer capacities: $d_1 = 2, \ d_2 = 1$	

Figure 5.1: An instance I_1 of SPA-ST.

5.2 Preliminary definitions

5.2.1 Introduction

Let I be an instance of SPA-ST (as defined in Section 4.2), and let M be a matching in I. Let (s_i, p_j) be an acceptable pair in I and let l_k be the lecturer who offers p_j.

Definition 5.2.1 (Strong stability)**.** *We say that M is* strongly stable *in I if it admits no blocking pair, where a* blocking pair *is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that either (1a and 1b) or (2a and 2b) holds as follows:*

(1a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$;
(1b) either (i), (ii) or (iii) holds as follows:

- *(i) each of p_j and l_k is undersubscribed in M;*
- *(ii) p_j is undersubscribed in M, l_k is full in M, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student/s in $M(l_k)$ or l_k is indifferent between them;*
- *(iii) p_j is full in M, and l_k prefers s_i to the worst student/s in $M(p_j)$ or l_k is indifferent between them.*

(2a) s_i is indifferent between p_j and $M(s_i)$;
(2b) either (i), (ii) or (iii) holds as follows:

- *(i) each of p_j and l_k is undersubscribed in M;*
- *(ii) p_j is undersubscribed in M, l_k is full in M, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student/s in $M(l_k)$;*
- *(iii) p_j is full in M and l_k prefers s_i to the worst student/s in $M(p_j)$.*

In the SPA-ST instance shown in Figure 5.1, it may be verified that matching $M_1 = \{(s_1, p_2), (s_3, p_3)\}$ is weakly stable (for the weak stability definition, see Definition 2.3.1), since each of lecturer l_1 and l_2 would not be better off rejecting her assigned student for s_2. It may also be verified that matching $M_2 = \{(s_1, p_2), (s_2, p_3), (s_3, p_1)\}$ is weakly stable in I_1. Clearly, matchings M_1 and M_2 have different sizes. We note that I_1 admits the strongly stable matching $M_3 = \{(s_1, p_1), (s_2, p_2), (s_3, p_3)\}$. Clearly, a strongly stable matching is also weakly

stable. Further, M_3 should be preferred over M_2 in practical applications because each of s_2 and s_3 is assigned in M_3 to a project better than her assigned project in M_2. Moreover, strong stability will prevent s_2 and s_3 from undermining M_2 by persuading l_1 and l_2 respectively to take them on for p_2 and p_3 respectively. We remark that (s_1, p_2) forms a blocking pair for M_3 under super-stability, since s_1 and l_1 will be no worse off forming an arrangement for s_1 to take on p_1. In fact, I_1 does not admit a super-stable matching.[1]

In the remainder of this chapter, any usage of the term *blocking pair* refers to the version of this term for strong stability as defined in Definition 5.2.1.

5.2.2 Justification for the strong stability definition

In this section, we give a justification for our definition of how an acceptable pair (s_i, p_j) can form a blocking pair for a matching M under strong stability. Let l_k be the lecturer who offers p_j. It is clear from Definition 5.2.1 that if s_i seeks to become assigned to p_j outside of M, then either s_i improves relative to M (i.e., s_i is unassigned in M or prefers p_j to her assigned project in M) or s_i is indifferent to the switch.

First, we consider the case where s_i improves relative to M, i.e., Definition 5.2.1(1a). Following from the implication of strong stability, lecturer l_k must be no worse off relative to M. There are three different scenarios in which l_k will be willing to take on s_i for p_j. In Definition 5.2.1(1b)(i), l_k will be willing to take on s_i for p_j, since there is a free space. In Definition 5.2.1(1b)(ii), if s_i was already assigned in M to a project offered by l_k then l_k will agree to the switch, since the total number of students assigned to l_k remains the same. However, if s_i was not already assigned in M to a project offered by l_k, since l_k is full, l_k will need to reject some student assigned to her in order to take on s_i. Obviously, l_k will not reject a student that she prefers to s_i; thus l_k will either improve or be no worse off after the switch. Finally, in Definition 5.2.1(1b)(iii), since p_j is full, l_k will need to reject some student assigned to p_j in order to take on s_i. Again, l_k will either improve or be no worse off after the switch. Under this definition, as observed in [13, Section 2.2], if s_i was already assigned in M to a project offered by l_k, then the number of students assigned to l_k will decrease by 1 (the reason for adapting this definition was further justified in [13, Section 6.1]).

Next, we consider the case where s_i will be no worse off relative to M, i.e., Definition 5.2.1(2a). Again, following from the implication of strong stability, lecturer l_k must improve relative to M. Now, for this to happen, we would expect that either (A), (B) or (C) holds as follows.

[1] Let I be an instance of SPA-ST, and suppose I admits no super-stable matching. In contrast to Proposition 4.2.4, if I admits a strongly stable matching then weakly stable matchings in I may be of different sizes, as the example instance I_1 shown in Figure 5.1 illustrates.

Students' preferences	Lecturers' preferences	offers
s_1: $(p_1 \quad p_2)$	l_1: $s_1 \quad s_2 \quad s_3$	p_1, p_2
s_2: $(p_1 \quad p_2)$		
s_3: p_2	Project capacities: $c_1 = c_2 = 2$	
	Lecturer capacity: $d_1 = 3$	

Figure 5.2: An instance I_2 of SPA-ST.

Students' preferences	Lecturers' preferences	offers
s_1: $(p_1 \quad p_2)$	l_1: $s_1 \quad s_2$	p_1, p_2
s_2: p_1		
	Project capacities: $c_1 = c_2 = 1$	
	Lecturer capacity: $d_1 = 2$	

Figure 5.3: An instance I_3 of SPA-ST.

(A) p_j is undersubscribed, l_k is undersubscribed and $s_i \notin M(l_k)$: if p_j and l_k are both undersubscribed then the only way that l_k would improve is if s_i is not already assigned in M to a project offered by l_k, if this is the case, then l_k will agree to the switch since there is a free space and she will get one more student to supervise, namely s_i.

(B) p_j is undersubscribed, l_k is full, $s_i \notin M(l_k)$ and l_k prefers s_i to the worst student/s in $M(l_k)$: if p_j is undersubscribed and l_k is full then the only way that l_k could improve is first for s_i to not be assigned in M to a project offered by l_k, if this is the case then l_k will be happy to reject a student that is worse than s_i on her preference list.

(C) p_j is full, $s_i \notin M(l_k)$ and l_k prefers s_i to the worst student/s in $M(p_j)$: if p_j is full then the only way that l_k could improve is first for s_i to not be assigned in M to a project offered by l_k; if this is the case then l_k will be happy to reject some student assigned to p_j; obviously, this student must be worse than s_i on l_k's preference list.

The reader can observe that cases (A), (B) and (C) above are different from the cases laid out in Definitions 5.2.1(2b) (i), (ii) and (iii), respectively. Our reason for adapting Definition 5.2.1(2b) is because if we settle for cases (A) - (C), then strongly stable matchings in I may be of different sizes. We illustrate this with respect to the SPA-ST instances in Figures 5.2 and 5.3 as follows.

(A) Here, instance I_2 illustrated in Figure 5.2 admits the strongly stable matching $M_1 = \{(s_1, p_2), (s_2, p_2)\}$. Each of p_1 and l_1 is undersubscribed in M_1, but it is straightforward to see why neither s_1 nor s_2 forms a blocking pair for M_1 via p_1, since they are both assigned in M_1 to a project offered by l_1. Also, instance I_3 illustrated in Figure 5.3 admits the strongly stable matching $M_2 = \{(s_1, p_1)\}$. Again, each of p_2 and l_1 is undersubscribed in M_2; however, (s_1, p_2) is not a blocking pair for M_2, since $s_1 \in M_2(l_1)$.

Students' preferences	Lecturers' preferences	offers
s_1: $(p_1 \quad p_2)$	l_1: $s_3 \quad s_1 \quad s_2$	p_1, p_2, p_3
s_2: $(p_1 \quad p_2)$		
s_3: p_3	Project capacities: $c_1 = c_2 = c_3 = 1$	
	Lecturer capacity: $d_1 = 2$	

Figure 5.4: An instance of SPA-ST.

(B) Following this definition, instance I_2 illustrated in Figure 5.2 also admits the strongly stable matching $M_3 = \{(s_1, p_1), (s_2, p_2), (s_3, p_2)\}$. We observe that p_1 is undersubscribed in M_3 and l_1 is full in M_3; however, (s_2, p_1) does not form a blocking pair for M_3 since $s_2 \in M_3(l_1)$. Clearly, instance I_2 admits two strongly stable matchings of different sizes, namely M_1 and M_3.

(C) Here, instance I_3 illustrated in Figure 5.3 admits the strongly stable matching $M_4 = \{(s_1, p_2), (s_2, p_1)\}$. Here, p_1 is full in M_4 and l_1 prefers s_1 to a worst student assigned to p_1, namely s_2. However, (s_1, p_1) is not a blocking pair under M_4 since s_1 is assigned in M_4 to a project offered by l_1. It is clear that instance I_3 also admits two strongly stable matchings of different sizes, namely M_2 and M_4.

Going back to Definition 5.2.1(2b), we will now draw our conclusions by referring to Definition 5.2.1(2b)(ii). The reader can easily verify that Definition 5.2.1(2b) has enforced the condition that if s_i is already assigned in M to a project offered by l_k, then (s_i, p_j) could still potentially form a blocking pair for M. The implication of this is that, potentially, l_k does not improve after the switch. To see this, consider the SPA-ST instance in Figure 5.4 with the matching $M = \{(s_1, p_2), (s_2, p_2)\}$, where s_1 is indifferent between p_1 and her assigned project in M, p_1 is undersubscribed in M, l_1 is full in M and $s_1 \in M(l_1)$. Under Definition 5.2.1(2b)(ii), the pair (s_1, p_1) forms a blocking pair for M; and hence M is not strongly stable in the instance. Indeed, the instance admits no strongly stable matching.

We note that by allowing (s_1, p_1) to form a blocking pair, neither s_1 nor l_1 will improve. On one hand, one could argue that this does not satisfy the concept of strong stability. On the other hand, if we enforce that (s_1, p_1) cannot form a blocking pair in this case because $s_1 \in M(l_1)$, this will lead us back to strongly stable matchings having different sizes as argued in case (B). Moreover, in order to assign as many students to projects as possible, we would seek a maximum size strongly stable matching. However, we conjecture that this problem is NP-hard, following from related problems of finding maximum size stable matchings in the literature [60, 84]. As a consequence, in the remainder of this chapter, we will adapt Definition 5.2.1(2b).

5.3 A polynomial-time algorithm

In this section we present our algorithm for SPA-ST under strong stability, which we will refer to as Algorithm SPA-ST-strong. In Section 5.3.1, we give some definitions relating to the algorithm. In Section 5.3.2, we give a description of our algorithm and present it in pseudocode form. In Section 5.3.3, we briefly describe the non-trivial modifications that are involved in extending the existing strong stability algorithms for SMT [51], SMTI [81] and HRT [59] to our algorithm for the SPA-ST case. We illustrate an execution of our algorithm with respect to a SPA-ST instance in Section 5.3.4. In Section 5.3.5, we present our algorithm's correctness and complexity results. Finally, in Section 5.3.6, we give some structural properties satisfied by the set of strongly stable matchings in an instance of SPA-ST.

5.3.1 Definitions relating to the algorithm

Given a pair $(s_i, p_j) \in M$, for some strongly stable matching M in I, we call (s_i, p_j) a *strongly stable pair*. During the execution of the algorithm, students become *provisionally assigned* to projects (and implicitly to lecturers), and it is possible for a project (and lecturer) to be provisionally assigned a number of students that exceeds its capacity.

The *provisional assignment graph* is an undirected bipartite graph $G = (\mathcal{S} \cup \mathcal{P}, E)$ such that there is an edge $(s_i, p_j) \in E$ if and only if s_i is provisionally assigned to p_j. During the execution of the algorithm, it is possible for a student to be adjacent to more than one project in G. Thus, we denote by $G(s_i)$ the set of projects that are adjacent to s_i in G. Given a project $p_j \in P$, we denote by $G(p_j)$ the set of students who are provisionally assigned to p_j in G and we let $d_G(p_j) = |G(p_j)|$. Similarly, we denote by $G(l_k)$ the set of students who are provisionally assigned to a project offered by l_k in G and we let $d_G(l_k) = |G(l_k)|$.

As stated earlier, for a project p_j, it is possible that $d_G(p_j) > c_j$ at some point during the algorithm's execution. Thus, we denote by $q_{p_j} = \min\{c_j, d_G(p_j)\}$ the *quota of* p_j in G. Similarly, for a lecturer l_k, it is possible that $d_G(l_k) > d_k$ at some point during the algorithm's execution. At this point, we denote by $\alpha_k = \sum\{q_{p_j} : p_j \in P_k\}$ the total quota of projects offered by l_k in G, and we denote by $q_{l_k} = \min\{d_k, \alpha_k\}$ the *quota of* l_k in G.

The algorithm proceeds by deleting from the preference lists certain (s_i, p_j) pairs that are not strongly stable. By the term *delete* (s_i, p_j), we mean the removal of p_j from s_i's preference list and the removal of s_i from $\mathcal{L}_k^j$ (i.e., the projected preference list of lecturer l_k for p_j); in addition, if $(s_i, p_j) \in E$ we delete the edge from G. By the *head* and *tail* of a preference list at a given point we mean the first and last tie respectively on that list after any deletions might have occurred (recalling that a tie can be of length 1).

We now give four further definitions relating to the provisional assignment graph.

Definition 5.3.1 (Dominated in $\mathcal{L}_k^j$)**.** Given a project p_j, we say that a student s_i is *dominated in* $\mathcal{L}_k^j$ if s_i is worse than at least c_j students in $\mathcal{L}_k^j$ who are provisionally assigned to p_j.

Definition 5.3.2 (Dominated in $\mathcal{L}_k$)**.** Given a lecturer l_k, we say that a student s_i is *dominated in* $\mathcal{L}_k$ if $\sum_{p_j \in P_k} \min\{q_{p_j}, |\{s_r : (s_r, p_j) \in G \wedge l_k \text{ prefers } s_r \text{ to } s_i\}|\}$ is at least d_k.

Definition 5.3.3 (Lower rank edge)**.** Given an edge $(s_i, p_j) \in E$, let l_k be the lecturer who offers p_j and let α_k be as defined above. We define (s_i, p_j) as a *lower rank edge* if s_i is in the tail of $\mathcal{L}_k$ and $\alpha_k > d_k$.

Definition 5.3.4 (Bound)**.** Given an edge $(s_i, p_j) \in E$, we say that s_i is *bound to* p_j if (i) p_j is not oversubscribed or s_i is not in the tail of $\mathcal{L}_k^j$ (or both), and (ii) (s_i, p_j) is not a lower rank edge. If s_i is bound to p_j, we may also say that (s_i, p_j) is a *bound edge*; otherwise, we refer to it as an *unbound edge*.[2]

Reduced assignment graph. We form a *reduced assignment graph* $G_r = (S_r, P_r, E_r)$ from a provisional assignment graph G as follows. Initially, let $G_r = G$. For each edge $(s_i, p_j) \in E$ such that s_i is bound to p_j, we remove the edge (s_i, p_j) from G_r and we reduce the quota of p_j in G_r (and implicitly the quota of l_k in G_r) by 1.[3] Further, we remove all other unbound edges (s_i, p_t) incident to s_i in G_r, without reducing p_t's quota in G_r. Each isolated student vertex is then removed from G_r. Finally, if the quota of any project is reduced to 0, or if p_j becomes an isolated vertex, then p_j is removed from G_r. For each surviving p_j in G_r, we denote by $q^*_{p_j}$ the *revised quota of* p_j, where $q^*_{p_j}$ is the difference between p_j's quota in G (i.e., q_{p_j}) and the number of students that are bound to p_j. Similarly, we denote by $q^*_{l_k}$ the *revised quota of* l_k in G_r, where $q^*_{l_k}$ is the difference between l_k's quota in G (i.e., q_{l_k}) and the total number of bound edges adjacent to a project offered by l_k in G.[4]

Further, for each l_k who offers at least one project in G_r, we let

$$n_k = \sum \{q^*_{p_j} : p_j \in P_k \cap P_r\} - q^*_{l_k},$$

where n_k is the difference between the total revised quota of projects in G_r that are offered by l_k and the revised quota of l_k in G_r. Now, if $n_k > 0$ and $d_{G_r}(l_k) > q^*_{l_k}$ (i.e., if the revised quota of l_k in G_r is less than the number of students who are adjacent to a project offered by l_k in G_r), we extend G_r as follows. We add n_k dummy student vertices to S_r. For each of these dummy vertices, say s_{d_i}, and for each project $p_j \in P_k \cap P_r$ that is adjacent to a student

[2]Note that an edge $(s_i, p_j) \in E$ can change state from *bound* to *unbound*, but not vice versa.

[3]We note that we only remove this edge to form G_r, we do not delete the edge from G.

[4]We note that the revised quota of l_k in G_r is not in general the minimum between the capacity of d_k and the total revised quota of projects offered by l_k in G_r (for example, see the second paragraph in the description corresponding to Figure 5.6, page 98).

vertex in S_r via a lower rank edge, we add the edge (s_{d_i}, p_j) to E_r.[5] An intuition as to why we add dummy students to G_r is as follows. Let l_k be a lecturer who offers a project that is provisionally assigned to a student in G_r. If $d_{G_r}(l_k) > q^*_{l_k}$ and $n_k > 0$, then we need n_k dummy students to offset the difference between $\sum\{q^*_{p_j} : p_j \in P_k \cap P_r\}$ and $q^*_{l_k}$, so that we do not oversubscribe l_k in any maximum matching obtained from G_r. On the contrary, if $d_{G_r}(l_k) \leq q^*_{l_k}$ or $n_k \leq 0$, we do not add any dummy student for l_k.[6]

Given a set $X \subseteq S_r$ of students, define $\mathcal{N}(X)$, the *neighbourhood of* X, to be the set of project vertices adjacent in G_r to a student in X. Similarly, given a set $Y \subseteq P_r$ of projects, define $\mathcal{N}(Y)$, the *neighbourhood of* Y, to be the set of student vertices adjacent in G_r to a project in Y. For each project $p_j \in P_r$, we replace p_j with x clones of p_j in P_r, where $x = \min\{d_{G_r}(p_j), q^*_{p_j}\}$. For each cloned vertex p_{j_t} $(1 \leq t \leq x)$, and for each $s_i \in \mathcal{N}(p_j)$, we add the edge (s_i, p_{j_t}) to E_r. Hence the reduced assignment graph G_r is a bipartite graph whose vertex set $S_r \cup P_r$ consists of real student vertices, dummy student vertices and cloned project vertices, each with capacity 1.

It is well known in the literature [80] that if G_r does not admit a matching that saturates S_r, then there must exist a *deficient* subset $X \subseteq S_r$ such that $|X| > |\mathcal{N}(X)|$. Similarly, if G_r does not admit a matching that saturates P_r, then there must exist a *deficient* subset $Y \subseteq P_r$ such that $|Y| > |\mathcal{N}(Y)|$. To be precise, the *deficiency* of X is defined by $\delta(X) = |X| - |\mathcal{N}(X)|$; and the deficiency of Y is defined analogously.

Definition 5.3.5 (Left-critical set)**.** The *left deficiency* of G_r, denoted $\delta_L(G_r)$, is the maximum deficiency taken over all subsets of S_r. If $\delta_L(G_r) > 0$, then there exists a minimal subset $Z_s \subseteq S_r$ that is maximally deficient such that $\delta_L(G_r) = \delta(Z_s)$. We refer to Z_s as the *left-critical set*.

Definition 5.3.6 (Right-critical set)**.** Similar to above, the *right deficiency* of G_r, denoted $\delta_R(G_r)$, is the maximum deficiency taken over all subsets of P_r. If $\delta_R(G_r) > 0$, then there exists a minimal subset $Z_p \subseteq P_r$ that is maximally deficient such that $\delta_R(G_r) = \delta(Z_p)$. We refer to Z_p as the *right-critical set*.

Definition 5.3.7 (Feasible matching)**.** To form a feasible matching M in the final provisional assignment graph G, first we need to identify a subset of projects in $\mathcal{P}$ that must be full in M. We denote by P^* a subset of projects in $\mathcal{P}$ that is obtained as follows. For each project $p_j \in \mathcal{P}$, let l_k be the lecturer who offers p_j. We add p_j to P^* if for any student s_i, the pair (s_i, p_j) has been deleted, and (i) and (ii) holds as follows:

(i) either s_i is unassigned in G or $(s_i, p_{j'}) \in G$ where s_i prefers p_j to $p_{j'}$ or is indifferent between them;

[5] We only add edges between the dummy students and projects that are adjacent to students via lower rank edges because l_k is bound to have at least n_k lower rank edges.

[6] There is no need to add dummy students since in either of these two cases, the revised quota of l_k will not be exceeded in any maximum matching in G_r.

(ii) either l_k is undersubscribed in G or l_k is full and l_k prefers s_i to some student in $G(l_k)$.

Let G^* be the subgraph of G induced by the students who are adjacent to a project in P^*, and let E^* be the edge set of G^*. A *feasible matching* in G is a maximum matching M in G such that $M \cap E^*$ is a maximum matching in G^*.

5.3.2 Description of the algorithm

Algorithm SPA-ST-strong, described in pseudocode form as Algorithm 2, begins by initialising an empty bipartite graph G which will contain the provisional assignments of students to projects (and implicitly to lecturers). We remark that such assignments (i.e., edges in G) can subsequently be broken during the algorithm's execution.

The outer `repeat-until` loop of the algorithm initialises the inner `repeat-until` loop, which in turn initialises the `while` loop. The `while` loop involves each student s_i who is not adjacent to any project in G and who has a non-empty preference list applying in turn to each project p_j at the head of her preference list. Immediately, s_i becomes provisionally assigned to p_j in G (and to l_k). If by gaining a new provisional assignee, project p_j becomes full or oversubscribed, then for each student s_t in $\mathcal{L}_k^j$ such that s_t is dominated in $\mathcal{L}_k^j$, we delete the pair (s_t, p_j). As we will prove later, such pairs cannot belong to any strongly stable matching. Similarly, if by gaining a new provisional assignee, l_k becomes full or oversubscribed, then for each student s_t in $\mathcal{L}_k$, such that s_t is dominated in $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable, we delete the pair (s_t, p_u). This continues until every student is provisionally assigned to one or more project or has an empty preference list.

At the point where the `while` loop terminates, we form the reduced assignment graph G_r and we find the right-critical set Z_p of projects in G_r (Lemma 5.3.10 describes how to find Z_p). As we will see later, for each cloned project $p_{j_t} \in Z_p$, no student s_i in the tail of $\mathcal{L}_k^j$ (where p_j is the real project associated with p_{j_t}) can be assigned to p_j in any strongly stable matching, so all such pairs are deleted.[7] Next, we find the left-critical set Z_s of students in G_r (Lemma 5.3.11 describes how to find Z_s). For each cloned project $p_{j_t} \in \mathcal{N}(Z_s)$, the associated real project p_j cannot be assigned to any student in the tail of $\mathcal{L}_k^j$ in any strongly stable matching, so all such pairs are deleted.

At the termination of the inner `repeat-until` loop on line 26, i.e., when $Z_p \cup Z_s$ is empty, if some project p_j that has been deleted from a student's preference list is undersubscribed, we carry out some certain deletions.[8] We let s_r be any one of the most-preferred students

[7] We remark that Z_p consists of project clones. However, deletions are carried out in the projected preference list of the real projects associated with each project clones in G_r.

[8] A weaker form of this type of deletion was also carried out in Algorithm SPA-ST-super (page 57, lines 27 - 34).

(according to $\mathcal{L}_k^j$) who was provisionally assigned to p_j during some iteration of the inner `repeat-until` loop but is not assigned to p_j at this point. If the students at the tail of $\mathcal{L}_k$ are no better than s_r, it turns out that none of these students can be assigned to any project offered by l_k in any strongly stable matching – such pairs (s_t, p_u), for each s_t at the tail of $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable, are deleted. The outer `repeat-until` loop is then potentially reactivated, and the entire process continues until every student is provisionally assigned to a project or has an empty preference list.

At the termination of the outer `repeat-until` loop on line 35, if a student is adjacent in G to a project p_j via a bound edge, then we may potentially carry out extra deletions as follows. First, we let l_k be the lecturer who offers p_j and we let U be the set of projects that are adjacent to s_i in G via an unbound edge. For each project $p_u \in U$ such that p_u is not offered by l_k, it turns out that s_i cannot be assigned to p_u in any strongly stable matching, thus we delete all such pairs (s_i, p_u). Finally, we let M be any feasible matching in the provisional assignment graph G. If M is strongly stable relative to the given instance I then M is output as a strongly stable matching in I. Otherwise, the algorithm reports that no strongly stable matching exists in I.

Finding the right-critical and left-critical set. Consider the reduced assignment graph $G_r = (S_r, P_r, E_r)$ formed from G at a given point during the algorithm's execution (line 15). We recall that the vertex set of G_r consists of real student vertices, dummy student vertices and cloned project vertices, with each vertex having capacity 1. To find the right-critical set of projects and left-critical set of students in G_r, first we need to construct a maximum matching M_r in G_r, with respect to the unitary capacity for each $p_{j_t} \in P_r$. We describe how to construct M_r as follows:

1. Let G'_r be the subgraph of G_r induced by the dummy students adjacent to a project in G_r. First, find a maximum matching M'_r in G'_r.
2. Using M'_r as an initial solution, find a maximum matching M_r in G_r.[9]

An *alternating path* in G_r relative to M_r is any simple path in which edges are alternately in, and not in, M_r. An *augmenting path* in G_r is an alternating path from an unassigned student vertex to an unassigned cloned project vertex. The following lemmas are classical results with respect to matchings in bipartite graphs.

Lemma 5.3.8. *A matching M_r in a bipartite graph G_r has maximum cardinality if and only if there is no augmenting path relative to M_r in G_r.*

[9]By making sure that all the dummy students are matched in step 1, we are guaranteed that no lecturer is oversubscribed with non-dummy students in any maximum matching in G_r.

Algorithm 2 Algorithm SPA-ST-strong

Input: SPA-ST instance I
Output: a strongly stable matching in I or "no strongly stable matching exists in I"

$G \leftarrow \emptyset$
repeat
  repeat
    while some student s_i is unassigned and has a non-empty list **do**
      for each project p_j at the head of s_i's list **do**
        $l_k \leftarrow$ lecturer who offers p_j
        add the edge (s_i, p_j) to G
        if p_j is full or oversubscribed **then**
          for each student s_t dominated in $\mathcal{L}_k^j$ **do**
            delete (s_t, p_j)
        if l_k is full or oversubscribed **then**
          for each student s_t dominated in $\mathcal{L}_k$ **do**
            for each project $p_u \in P_k \cap A_t$ **do**
              delete (s_t, p_u)
    form the reduced assignment graph G_r (using the transformation on page 90)
    find the right-critical set Z_p of project clones (using Lemma 5.3.10)
    for each project p_j such that a clone of p_j is in Z_p **do**
      $l_k \leftarrow$ lecturer who offers p_j
      for each student s_i at the tail of $\mathcal{L}_k^j$ **do**
        delete (s_i, p_j)
    find the left-critical set Z_s of students (using Lemma 5.3.11)
    for each project p_j such that a clone of p_j is in $\mathcal{N}(Z_s)$ **do**
      $l_k \leftarrow$ lecturer who offers p_j
      for each student s_i at the tail of $\mathcal{L}_k^j$ **do**
        delete (s_i, p_j)
  until $Z_p \cup Z_s$ is empty
  for each $p_j \in \mathcal{P}$ **do**
    if there is some student s_i such that (s_i, p_j) has been deleted **then**
      $l_k \leftarrow$ lecturer who offers p_j
      $s_r \leftarrow$ most-preferred student rejected from p_j, who was previously assigned to p_j
      if p_j is undersubscribed and the students at the tail of $\mathcal{L}_k$ are no better than s_r **then**
        for each student s_t at the tail of $\mathcal{L}_k$ **do**
          for each project $p_u \in P_k \cap A_t$ **do**
            delete (s_t, p_u)
until every unassigned student has an empty list
for each student s_i in G **do**
  if s_i is adjacent in G to a project p_j via a bound edge **then**
    $l_k \leftarrow$ lecturer who offers p_j
    $U \leftarrow$ unbound projects adjacent to s_i in G
    for each $p_u \in U \setminus P_k$ **do**
      delete (s_i, p_u)
$M \leftarrow$ a feasible matching in G
if M is a strongly stable matching in I **then**
  return M
else
  return "no strongly stable matching exists in I"

Lemma 5.3.9. *Let M_r be a maximum matching in G_r. Then $|M_r| = |S_r| - \delta_L(G_r)$, where S_r is the set of left vertices. Likewise, $|M_r| = |P_r| - \delta_R(G_r)$, where P_r is the set of right vertices.*

The Hopcroft-Karp algorithm [49] can be used to obtain M_r, and this can be implemented to run in $O(\sqrt{n}m)$ time, where $n = |S_r| + |P_r|$ and $m = |E_r|$. Now that we have described how to construct a maximum matching in the reduced assignment graph, the next two lemmas tells us how to find the right-critical set of projects and the left-critical set of students.

Lemma 5.3.10. *Given a maximum matching M_r in the reduced assignment graph G_r, any right-critical set Z_p consists precisely of the set U_p of unassigned project clones together with the set U'_p of project clones reachable from a project clone in U_p via an alternating path. Hence, the right-critical set is unique.*

Proof. First, we note that $\delta_R(G_r) = |U_p|$. Let $C = U_p \cup U'_p$; we claim that $\delta(C) = |U_p|$. By the definition of $\delta_R(G_r)$, clearly $\delta(C) \leq \delta_R(G_r) = |U_p|$. Now suppose that $\delta(C) < |U_p|$, then

$$\begin{aligned} |U_p| &> \delta(C) \\ &= |U_p \cup U'_p| - |\mathcal{N}(C)| \\ &= |U_p| + |U'_p| - |\mathcal{N}(C)|, \qquad \text{(since } U_p \cap U'_p = \emptyset\text{)} \end{aligned}$$

which implies that

$$|U'_p| < |\mathcal{N}(C)| \ . \tag{5.1}$$

We claim that every project clone that is assigned in M_r to a student $s_{i'} \in \mathcal{N}(C)$ must be in U'_p. For suppose there is a project clone $p_{j_t} \notin U'_p$ such that p_{j_t} is assigned to $s_{i'}$ in M_r. Since $s_{i'} \in \mathcal{N}(C)$, then $s_{i'}$ must be adjacent to some project clone $p_{j'_t} \in C$. Now, if $p_{j'_t} \in U_p$, then there is an alternating path from $p_{j'_t}$ to p_{j_t} via $s_{i'}$, a contradiction. Hence, $p_{j'_t} \in U'_p$. Since $p_{j'_t}$ is reachable from a project clone in U_p via an alternating path, p_{j_t} is also reachable from the same project clone in U_p via an alternating path. Hence our claim is established.

Since $\mathcal{N}(C)$ contains $|U'_p|$ students, denoted by S', who are collectively assigned to project clones in U'_p, Inequality (5.1) implies that there is an additional student $s_i \in \mathcal{N}(C) \setminus S'$. If s_i is assigned in M_r, then the above claim implies that $s_i \in S'$, a contradiction. Hence s_i is unassigned in M_r. Now, since each project clone in U'_p is reachable from a project in U_p via an alternating path, and the project clones in $U_p \cup U'_p$ are collectively adjacent to students in $\mathcal{N}(C)$, we can find an alternating path from a project clone in U_p to s_i. Thus M_r admits an augmenting path, contradicting the maximality of M_r. Hence $\delta(C) = |U_p|$.

Next we show that the right-critical set is unique. We note that any right-critical set Z_p must contain every project clone that is unassigned in some maximum matching in G_r. For, suppose not. Let M_r^* be an arbitrary maximum matching in G_r, where $|M_r^*| = |P_r| - \delta_R(G_r)$. Suppose there is some project clone $p_{j_t} \in P_r \setminus Z_p$ such that p_{j_t} is unassigned in M_r^*. There must be $\delta_R(G_r)$ unassigned project clones, with at most $\delta_R(G_r) - 1$ of these clones contained in Z_p (since $p_{j_t} \notin Z_p$). Hence Z_p contains at least $|Z_p| - \delta_R(G_r) + 1$ assigned project clones. It follows that

$$|\mathcal{N}(Z_p)| \geq |Z_p| - \delta_R(G_r) + 1,$$

or

$$|Z_p| - |\mathcal{N}(Z_p)| \leq \delta_R(G_r) - 1,$$

contradicting the required deficiency of Z_p. Moreover, for every $p_{j_t} \in U_p'$, there is a maximum matching in which p_{j_t} is unassigned, obtainable from M_r via an alternating path from a project clone in U_p to p_{j_t}. Hence $C \subseteq Z_p$. Further, by the minimality of Z_p, and since

$$\begin{aligned}
\delta(C) &= |U_p| \\
&= |P_r| - |M_r| \\
&= |M_r^*| + \delta_R(G_r) - |M_r| \\
&= \delta_R(G_r) \qquad \text{(since both } M_r^* \text{ and } M_r \text{ are maximum matchings in } G_r\text{)} \\
&= \delta(Z_p),
\end{aligned}$$

it follows that $C = Z_p$. Hence Z_p is unique. □

Lemma 5.3.11. *Given a maximum matching M_r in the reduced assignment graph G_r, any left-critical set Z_s consists precisely of the set U_s of unassigned students together with the set U_s' of students reachable from a student in U_s via an alternating path. Hence, the left-critical set is unique.*

Proof. By symmetry, the proof is similar to that given for Lemma 5.3.10. □

5.3.3 The non-triviality of extending Algorithm HRT-strong to SPA-ST

Algorithm SPA-ST-strong is a non-trivial extension of Algorithm HRT-strong for HRT [59]. Here we outline the major distinctions between our algorithm and Algorithm HRT-strong, which indicate the challenges involved in extending the earlier approach to the SPA-ST setting.

1. Given a lecturer l_k, it is possible that during some iteration of our algorithm, some $p_j \in P_k$ is oversubscribed which causes l_k to become full or oversubscribed (see Figure 5.6(a) on page 98, at the point where s_3 applies to p_1). Finding the dominated students in $\mathcal{L}_k$ becomes more complex in SPA-ST – to achieve this we introduced the notion of quota for each $p_j \in P_k$ (i.e., q_{p_j}).

2. To form G_r in the SPA-ST case, we extended the approach described in the HRT case [59] by introducing the concept of lower rank edges for each lecturer who offers a project in G_r, and we also introduced dummy students and project clones.

3. In the HRT setting, the critical set is restricted to the resident vertex set. However, in our case, the critical set involves both the student and project vertex sets. See Figure 5.10 on page 101.

4. Lines 27 - 34 of Algorithm SPA-ST-strong refer to additional deletions that must be carried out in a certain situation. A weaker form of this type of deletion was also carried out in Algorithm SPA-ST-super (page 57, lines 27 - 34). See the description corresponding to Figure 5.7 on page 99 for why we may need to carry out this type of deletion in the strong stability context.

5. Lines 36 - 41 of Algorithm SPA-ST-strong are new. This section of the algorithm deletes additional (student, project) pairs that cannot be part of any strongly stable matching.

6. Constructing a feasible matching M from G in the SPA-ST setting is much more challenging: we first identify some projects that must be full in M, denoted by P^*. See the description corresponding to Figure 5.8 on page 100.

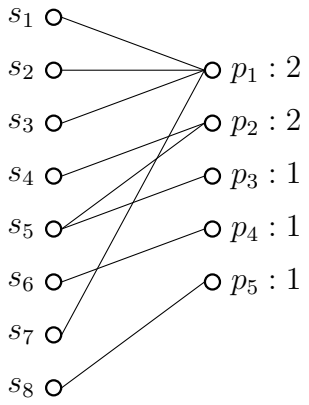
5.3.4 Example execution of the algorithm

In this section, we illustrate an execution of Algorithm SPA-ST-strong with respect to two SPA-ST instances shown in Figures 5.5 and 5.9. We begin with Figure 5.5, which involves the set of students $\mathcal{S} = \{s_i : 1 \leq i \leq 8\}$, the set of projects $\mathcal{P} = \{p_j : 1 \leq j \leq 6\}$ and the set of lecturers $\mathcal{L} = \{l_k : 1 \leq k \leq 3\}$. The algorithm starts by initialising the bipartite graph $G = \{\}$, which will contain the provisional assignment of students to projects. We assume that the students become provisionally assigned to each project at the head of their preference list in subscript order. Henceforth, the usage $G^{(i)}$ (for $i \geq 1$) denotes the provisional assignment graph at the i-th iteration of the inner `repeat-until` loop. A similar definition holds for $G_r^{(i)}$, $M_r^{(i)}$, $Z_p^{(i)}$, and $Z_s^{(i)}$, for $i \geq 1$.

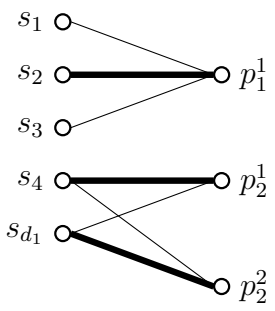
Figures 5.6, 5.7 and 5.8 illustrate how this execution of Algorithm SPA-ST-strong proceeds with respect to I_4.

Students' preferences	Lecturers' preferences	offers
s_1: p_1 p_6	l_1: s_8 s_7 $(s_1$ s_2 $s_3)$ $(s_4$ $s_5)$ s_6	p_1, p_2
s_2: p_1 p_2	l_2: s_6 s_5 $(s_7$ $s_3)$	p_3, p_4
s_3: $(p_1$ $p_4)$	l_3: $(s_1$ $s_4)$ s_8	p_5, p_6
s_4: p_2 p_5		
s_5: $(p_2$ $p_3)$		
s_6: $(p_2$ $p_4)$		
s_7: p_3 p_1	Project capacities: $c_1 = c_2 = c_6 = 2,\ c_3 = c_4 = c_5 = 1$	
s_8: p_5 p_1	Lecturer capacities: $d_1 = d_3 = 3,\ d_2 = 2$	

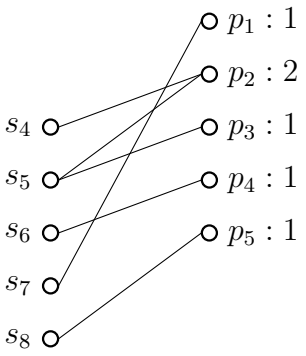
Figure 5.5: An instance I_4 of SPA-ST.



(a) The provisional assignment graph $G^{(1)}$ at the end of the `while` loop, with the quota of each project labelled beside it.



(b) The reduced assignment graph $G_r^{(1)}$ with a dummy student and project clones. The collection of the thick edges is the maximum matching $M_r^{(1)}$, with $Z_p^{(1)} = \emptyset$ and $Z_s^{(1)} = \{s_1, s_2, s_3\}$.



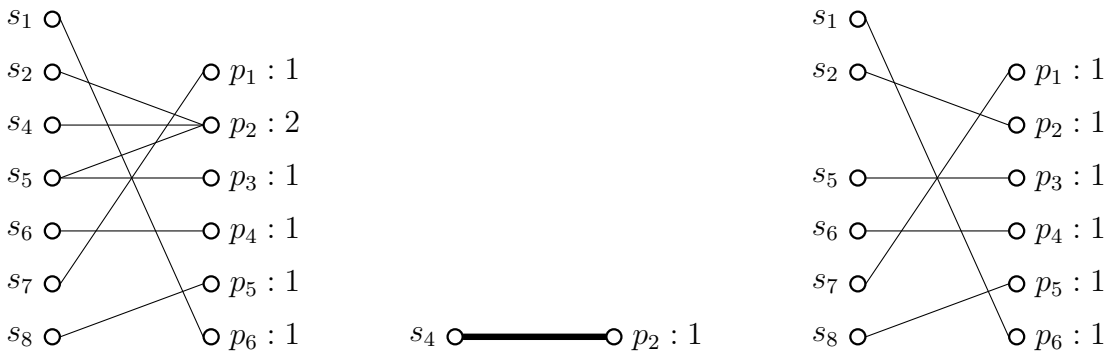
(c) The provisional assignment graph $G^{(1)}$ at the end of iteration (1).

Figure 5.6: Iteration (1).

Iteration 1: At the termination of the `while` loop during the first iteration of the inner `repeat-until` loop, every student, except s_3, s_6 and s_7, is provisionally assigned to every project in the first tie on their preference list. Edge $(s_3, p_4) \notin G^{(1)}$ because (s_3, p_4) was deleted as a result of s_6 becoming provisionally assigned to p_4, causing s_3 to be dominated in $\mathcal{L}_2^4$. Also, edge $(s_6, p_2) \notin G^{(1)}$ because (s_6, p_2) was deleted as a result of s_4 becoming provisionally assigned to p_2, causing s_6 to be dominated in $\mathcal{L}_1$. Finally, edge $(s_7, p_3) \notin G^{(1)}$ because (s_7, p_3) was deleted as a result of s_5 becoming provisionally assigned to p_3, causing s_7 to be dominated in $\mathcal{L}_2^3$.

To form $G_r^{(1)}$, the bound edges $(s_5, p_3), (s_6, p_4), (s_7, p_1)$ and (s_8, p_5) are removed from the graph. We can verify that edges (s_4, p_2) and (s_5, p_2) are unbound, since they are lower rank edges for l_1; however, $(s_5, p_2) \notin G_r^{(1)}$, since s_5 is bound to p_3. Now, since p_1 is oversubscribed, and each of s_1, s_2 and s_3 is at the tail of $\mathcal{L}_1^1$, edges (s_1, p_1), (s_2, p_1) and (s_3, p_1) are unbound. Further, the revised quota of l_1 in $G_r^{(1)}$ is 2, and the total revised quota of projects offered by l_1 (i.e., p_1 and p_2) is 3, hence $n_1 = 1$. Moreover, since the number of students that are adjacent to a project offered by l_1 in $G_r^{(1)}$ is more than the revised quota of l_1 (i.e., $d_{G_r^{(1)}}(l_1) = 4 > 2 = q_{l_1}^*$), we add n_1 (i.e., 1) dummy student vertex s_{d_1} to $G_r^{(1)}$, and we add an edge between s_{d_1} and p_2 (since p_2 is the only project in $G_r^{(1)}$ adjacent to a student in the tail of $\mathcal{L}_1$ via a lower rank edge).

Finally, we replace each project p_j in $G_r^{(2)}$ with $\min\{d_{G_r^{(1)}}, q_{p_j}^*\}$ clones, i.e., p_1 with $\min\{3, 1\}$ clone and p_2 with $\min\{2, 2\}$ clones. With respect to the maximum matching $M_r^{(1)}$ (i.e., the thick edges), it is clear that the right-critical set $Z_p^{(1)} = \emptyset$, and the left-critical set $Z_s^{(1)} = \{s_1, s_2, s_3\}$, thus we delete (s_1, p_1), (s_2, p_1) and (s_3, p_1); and the inner `repeat-until` loop is reactivated.



(a) The provisional assignment graph $G^{(2)}$ at the end of the `while` loop.

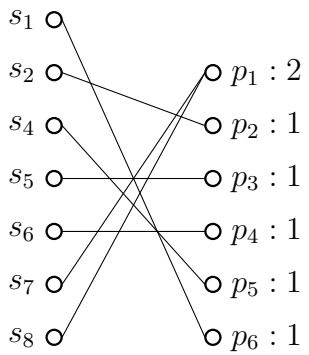
(b) The reduced assignment graph $G_r^{(2)}$.

(c) The provisional assignment graph $G^{(2)}$ at the end of iteration (2).

Figure 5.7: Iteration (2).

Iteration 2: At the beginning of this iteration, each of s_1 and s_2 is unassigned and has a non-empty preference list; thus we add edges (s_1, p_6) and (s_2, p_2) to the provisional assignment graph obtained at the end of iteration (1) to form $G_r^{(2)}$. It can be verified that every edge in $G_r^{(2)}$, except (s_4, p_2) and (s_5, p_2), is a bound edge. The revised quota of l_1 in $G_r^{(2)}$ is 1 and thus no dummy student is added at this point. Clearly, $Z_p^{(2)} \cup Z_s^{(2)} = \emptyset$, thus the inner `repeat-until` loop terminates. At this point, project p_1, which was deleted from s_3's preference list during iteration (1), is undersubscribed in iteration (2). Moreover, the students at the tail of $\mathcal{L}_1$ (i.e., s_4 and s_5) are no better than s_3, where s_3 is one of the most-preferred students rejected from p_1 according to $\mathcal{L}_1^1$; thus we delete (s_4, p_2) and (s_5, p_2). The outer `repeat-until` loop is then reactivated (since s_4 is unassigned and has a non-empty

preference list).



(a) The provisional assignment graph $G^{(3)}$ at the end of the `while` loop.

Figure 5.8: Iteration (3).

Iteration 3: At the beginning of this iteration, the only student that is unassigned and has a non-empty preference list is s_4; thus we add edge (s_4, p_5) to the provisional assignment graph obtained at the end of iteration (2) to form $G_r^{(3)}$. The provisional assignment of s_4 to p_5 led to p_5 becoming oversubscribed; thus (s_8, p_5) is deleted (since s_8 is dominated on $\mathcal{L}_3^5$). Further, s_8 becomes provisionally assigned to p_1. It can be verified that all the edges in $G_r^{(3)}$ are bound edges. Moreover, the reduced assignment graph $G_r^{(3)} = \emptyset$.

Again, every unassigned student has an empty preference list. We also have that project p_2, which was deleted from s_4's preference list in iteration (2), is undersubscribed in iteration (3). However, no further deletion is carried out on line 34 of the algorithm, since the student at the tail of $\mathcal{L}_1$ (i.e., s_2) is better than s_4 and s_5, where s_4 and s_5 are the most-preferred students rejected from p_2 according to $\mathcal{L}_1^2$. Hence, the outer `repeat-until` loop terminates. Also, no deletion is carried out on line 41 of the algorithm.

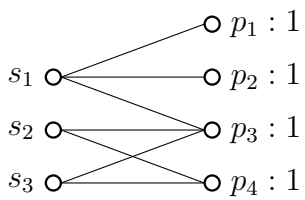
We observe that $P^* = \{p_5\}$, since (s_8, p_5) has been deleted, s_8 prefers p_5 to her provisional assignment in G and l_3 is undersubscribed. Thus we need to ensure that p_5 fills up in the feasible matching M constructed from G, so as to avoid (s_8, p_5) from blocking M. Finally, the algorithm outputs the feasible matching $M = \{(s_1, p_6), (s_2, p_2), (s_4, p_5), (s_5, p_3), (s_6, p_4), (s_7, p_1), (s_8, p_1)\}$ as a strongly stable matching. We leave it to the reader to verify that M is strongly stable in I_4.

Next, we consider Figure 5.9, which involves the set of students $\mathcal{S} = \{s_i : 1 \leq i \leq 3\}$, the set of projects $\mathcal{P} = \{p_j : 1 \leq j \leq 4\}$ which are all offered by a single lecturer l_1. This example highlights why we need the left-critical set of projects and the right-critical set of students. Similar to the previous example, the algorithm starts by initialising the bipartite graph $G = \{\}$, which will contain the provisional assignment of students to projects. Again,

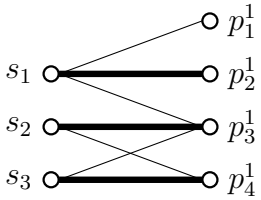
we assume that the students become provisionally assigned to each project at the head of their preference list in subscript order. Figure 5.10 illustrates how this execution of Algorithm SPA-ST-strong proceeds with respect to I_5.

Students' preferences	Lecturers' preferences	offers
s_1: $(p_1 \quad p_2 \quad p_3)$	l_1: $(s_1 \quad s_2 \quad s_3)$	p_1, p_2, p_3, p_4
s_2: $(p_3 \quad p_4)$		
s_3: $(p_3 \quad p_4)$	Project capacities: $c_j = 1$ for $1 \leq j \leq 4$	
	Lecturer capacity: $d_1 = 3$	

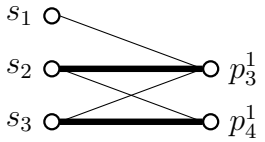
Figure 5.9: An instance I_5 of SPA-ST.



(a) The provisional assignment graph $G^{(1)}$ at the end of the `while` loop, with the quota of each project labelled beside it.



(b) The reduced assignment graph $G_r^{(1)}$ with project clones and no dummy student; also $Z_p^{(1)} = \{p_1, p_2\}$.



(c) The deletion of students at the tail of a project whose clones are in $Z_p^{(1)}$ led to s_1 becoming exposed in $M_r^{(1)}$, and thus $Z_s^{(1)} = \{s_1, s_2, s_3\}$.

Figure 5.10: Iteration (1).

Iteration 1: At the termination of the `while` loop during the first iteration of the inner `repeat-until` loop, every student is provisionally assigned to every project in the first tie on their preference list. All of the edges in $G^{(1)}$ are lower rank edges, since each of s_1, s_2 and s_3 is at the tail of $\mathcal{L}_1$ and the total quota of projects offered by l_1 is more than the capacity of l_1, i.e., $\alpha_1 = 4 > 3 = d_1$. Thus all of the edges in $G^{(1)}$ are unbound edges, and thus $G^{(1)} = G_r^{(1)}$. The number of students that are adjacent to a project offered by l_1 in $G_r^{(1)}$ is equal to the revised quota of l_1 in G, i.e., $d_{G_r^{(1)}}(l_1) = 3 = q_{l_1}^*$, and thus no dummy students are added. Further, each project in $G_r^{(1)}$ has a revised quota of 1, and thus each project has only 1 clone.

With respect to the maximum matching $M_r^{(1)}$ (i.e., the thick edges), it is clear that the right-critical set $Z_p^{(1)} = \{p_1, p_2\}$, thus we delete (s_1, p_1) and (s_1, p_2). As a consequence, s_1 became exposed in $M_r^{(1)}$, and thus the left-critical set $Z_s^{(1)} = \{s_1, s_2, s_3\}$. Then we delete (s_1, p_3), (s_2, p_3), (s_3, p_3), (s_2, p_4) and (s_4, p_4). Since $Z_p^{(1)} \cup Z_s^{(1)} \neq \emptyset$, the inner `repeat-until` loop is reactivated. This terminates immediately, since each unassigned student has an empty

preference list. We have that $P^* = \{p_1, p_2, p_3, p_4\}$; however, since the provisional assignment graph G is empty, the feasible matching M is also empty. Each of the deleted pairs forms a blocking pair for M, and the algorithm outputs that no strongly stable matching exists. The reader can verify that any matching obtained from instance I_5 will be undermined by a (student, project) pair that is not in the matching.

5.3.5 Correctness of the algorithm

We now present the following results regarding the correctness of Algorithm SPA-ST-strong. The first of these results deals with the fact that no strongly stable pair is deleted during the execution of the algorithm.

Lemma 5.3.12. *If a pair (s_i, p_j) is deleted during an execution of Algorithm* SPA-ST-strong*, then (s_i, p_j) does not belong to any strongly stable matching in I.*

In order to prove Lemma 5.3.12, we present Lemmas 5.3.13 - 5.3.17.

Lemma 5.3.13. *If a pair (s_i, p_j) is deleted within the* `while` *loop during an execution of Algorithm* SPA-ST-strong *then (s_i, p_j) does not belong to any strongly stable matching in I.*

Proof. Suppose that (s_i, p_j) is the first strongly stable pair to be deleted within the `while` loop during an arbitrary execution E of Algorithm SPA-ST-strong. Let M^* be some strongly stable matching in which s_i is assigned to p_j. Let l_k be the lecturer who offers p_j. Suppose that G is the provisional assignment graph immediately after the deletion of (s_i, p_j). There are two cases to consider.

1. Suppose that (s_i, p_j) is deleted (on line 10) because some other student became provisionally assigned to p_j during E, causing p_j to become full or oversubscribed, so that s_i is dominated in $\mathcal{L}_k^j$. Since $(s_i, p_j) \in M^* \setminus G$, there is some student, say s_r, such that l_k prefers s_r to s_i and $(s_r, p_j) \in G \setminus M^*$, for otherwise p_j would be oversubscribed in M^*. We note that s_r cannot be assigned to a project that she prefers to p_j in any strongly stable matching, for otherwise some strongly stable pair must have been deleted before (s_i, p_j), as p_j must be in the head of s_r's preference list when she applied. So s_r is either unassigned in M^* or s_r prefers p_j to $M^*(s_i)$ or is indifferent between them. Clearly for any combination of l_k and p_j being full or undersubscribed in M^*, it follows that (s_r, p_j) blocks M^*, a contradiction.

2. Suppose that (s_i, p_j) is deleted (on line 14) because some other student became provisionally assigned to a project offered by l_k during E, causing l_k to become full or

oversubscribed, so that s_i is dominated in $\mathcal{L}_k$. We denote by C_k the set of projects that are full or oversubscribed in G, which are offered by l_k. We denote by D_k the set of projects that are undersubscribed in G, which are offered by l_k. Clearly the projects offered by l_k that are provisionally assigned to a student in G at this point can be partitioned into C_k and D_k. We consider two subcases.

(i) Each student who is provisionally assigned in G to a project in D_k (if any) is also assigned to that same project in M^*. However, after the deletion of (s_i, p_j), we know that

$$\sum_{p_t \in C_k \cup D_k} q_{p_t} = \sum_{p_t \in C_k} c_t + \sum_{p_t \in D_k} d_G(p_t) \geq d_k; \tag{5.2}$$

i.e., the total quota of projects in $C_k \cup D_k$ is at least the capacity of l_k. Now, since p_j has one more assignee in M^* than it has provisional assignees in G, namely s_i, then some other project $p_{j'} \in C_k$ must have fewer than $c_{j'}$ assignees in M^*, for otherwise l_k would be oversubscribed in M^*. This implies that there is some student, say s_r, such that l_k prefers s_r to s_i and $(s_r, p_{j'}) \in G \setminus M^*$. Moreover, s_r cannot be assigned to a project that she prefers to $p_{j'}$ in M^*, as explained in (1) above. Hence, $(s_r, p_{j'})$ blocks M^*, a contradiction.

(ii) Each project in C_k at this point is full in M^*. This implies that there is some project $p_{j'} \in D_k$ with fewer assignees in M^* than provisional assignees in G, for otherwise l_k would be oversubscribed in M^*. Thus $p_{j'}$ is undersubscribed in M^* (since D_k is the set of undersusbcribed projects offered by l_k). Moreover, there is some student, say s_r, such that l_k prefers s_r to s_i and $(s_r, p_{j'}) \in G \setminus M^*$. Following a similar argument as in (i) above, $(s_r, p_{j'})$ blocks M^*, a contradiction. □

Lemma 5.3.14. *If a pair (s_i, p_j) is deleted on line 20 during an execution of Algorithm* SPA-ST-strong *then (s_i, p_j) does not belong to any strongly stable matching in I.*

Proof. Suppose that (s_i, p_j) is the first strongly stable pair to be deleted during E. Suppose further that (s_i, p_j) was deleted on line 20 of the algorithm. Let M^* be some strongly stable matching in which s_i is assigned to p_j, where l_k is the lecturer who offers p_j. We note that (s_i, p_j) is deleted at this point because some clone of p_j is in the right-critical set Z_p, and at that point s_i is in the tail of $\mathcal{L}_k^j$. We refer to the set of preference lists at that point as the current lists. Let Z'_p be the set of project clones in Z_p such that the associated real projects are assigned in M^* to a student from the tail of their current lists. We have that $p_{j_t} \in Z'_p$, for $1 \leq t \leq \min\{d_{G_r}(p_j), q^*_{p_j}\}$; thus, $Z'_p \neq \emptyset$. Let S' be the set of students in $\mathcal{N}(Z_p)$ who are assigned in M^* to at least one project from the head of their current list. Consider

$s_{i'} \in \mathcal{N}(Z_p)$. We note that $s_{i'}$ cannot be assigned in M^* to a project that she prefers to any project in the head of her current list, for otherwise some strongly stable pair must have been deleted before (s_i, p_j). Hence, any student $s_{i'}$ in $\mathcal{N}(Z_p)$ who is provisionally assigned to some clone of p_j in G_r must be in S'; otherwise, $(s_{i'}, p_j)$ would block M^*. Thus $S' \neq \emptyset$. Also $Z_p \setminus Z'_p \neq \emptyset$, because $|Z_p| - |\mathcal{N}(Z_p)| > 0$ and $|Z'_p| - |\mathcal{N}(Z_p)| \leq |Z'_p| - |S'| \leq 0$, since every project whose clone is in Z'_p is assigned in M^* to a student in S'.

We now claim that there must be an edge $(s_r, p_{j'_t})$ in G_r such that $p_{j'_t} \in Z_p \setminus Z'_p$ and $s_r \in S'$. For otherwise, $\mathcal{N}(Z_p \setminus Z'_p) \subseteq \mathcal{N}(Z_p) \setminus S'$, and

$$\begin{aligned} |Z_p \setminus Z'_p| - |\mathcal{N}(Z_p \setminus Z'_p)| &\geq |Z_p \setminus Z'_p| - |\mathcal{N}(Z_p) \setminus S'| \\ &= |Z_p| - |\mathcal{N}(Z_p)| - (|Z'_p| - |S'|) \\ &\geq |Z_p| - |\mathcal{N}(Z_p)|, \qquad \text{(since } |Z'_p| - |S'| \leq 0) \ . \end{aligned}$$

Hence $Z_p \setminus Z'_p$ has deficiency at least that of Z_p, contradicting the fact that Z_p is the right-critical set. Thus our claim is established, i.e., there is some project $p_{j'_t} \in Z_p \setminus Z'_p$ and some student $s_r \in S'$ such that s_r is adjacent to $p_{j'_t}$ in G_r and $(s_r, p_{j'}) \notin M^*$. We note that s_r is indifferent between $p_{j'}$ and $M^*(s_r)$, since S' is the set of students who are assigned in M^* to some project in the head of their current list.

Let $s_{q_0} = s_r$, $p_{t_0} = p_{j'}$, and let l_{z_0} be the lecturer who offers p_{t_0}. By the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0};[10]

(ii) p_{t_0} is undersubscribed in M^*, l_{z_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(l_{z_0})$ to s_{q_0} or is indifferent between them.

Otherwise (s_{q_0}, p_{t_0}) blocks M^*. In case (i), since $(s_{q_0}, p_{t_0}) \in G \setminus M^*$, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus G(p_{t_0})$. For suppose each student who is assigned to p_{t_0} in M^* is also assigned to p_{t_0} in G, then (s_{q_0}, p_{t_0}) would have been deleted in G. Let $p_{t_1} = p_{t_0}$.

In case (ii), we claim that l_{z_0} is either full or undersubscribed in G. Suppose otherwise, i.e., l_{z_0} is oversubscribed in G, then considering how G_r is formed from G, we have that the number of students assigned to a project offered by l_{z_0} in G_r exceeds the revised quota of l_{z_0}, i.e., $d_{G_r}(l_{z_0}) > q^*_{l_{z_0}}$. Now, if the total revised quota of projects offered by l_{z_0} in G_r is at most the revised quota of l_{z_0}, i.e., if $\sum\{q^*_{p_u} : p_u \in P_{z_0} \cap P_r\} \leq q^*_{l_{z_0}}$, then each project clone in G_r that is associated with a real project offered by l_{z_0} is matched in all maximum matchings that can be obtained from G_r, and thus $p_{j'_t} \notin Z_p$, a contradiction. Hence, $\sum\{q^*_{p_u} : p_u \in P_{z_0} \cap P_r\} > q^*_{l_{z_0}}$, which implies that $n_{z_0} = (\sum\{q^*_{p_u} : p_u \in P_{z_0} \cap P_r\} - q^*_{l_{z_0}})$ dummy

[10]We note that l_{z_0} cannot be indifferent between the worst student/s in $M^*(p_{t_0})$ and s_{q_0}, since p_{t_0} is not assigned in M^* to students from the tail of its current list.

student vertices are adjacent to the project clones corresponding to each lower rank projects offered by l_{z_0} in G_r. With this construction, again, each project clone in G_r that is associated with a real project offered by l_{z_0} is matched in all maximum matchings that can be obtained from G_r, and thus $p_{j'_t} \notin Z_p$, a contradiction. Hence, our claim holds, i.e., l_{z_0} is full or undersubscribed in G. Moreover, since $s_{q_0} \in G(l_{z_0}) \setminus M^*(l_{z_0})$, there exists some student $s_{q_1} \in M^*(l_{z_0}) \setminus G(l_{z_0})$. We note that l_{z_0} either prefers s_{q_1} to s_{q_0} or is indifferent between them; clearly $s_{q_1} \neq s_{q_0}$. Now, suppose $M^*(s_{q_1}) = p_{t_1}$ (possibly $p_{t_1} = p_{t_0}$).

Since (s_i, p_j) is the first strongly stable pair to be deleted, s_{q_1} is provisionally assigned in G to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as students apply to projects in the head of their preference list and since $(s_{q_1}, p_{t_1}) \notin G$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during an iteration of the `while` loop, a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in G$ and $(s_{q_1}, p_{t_1}) \notin G$. Let l_{z_1} be the lecturer who offers p_{t_2}. Again by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^* and l_{z_1} prefers the worst students in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus G(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus G(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}; again, it is clear that $s_{q_2} \neq s_{q_1}$. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). Applying similar reasoning as for s_{q_1}, s_{q_2} is provisionally assigned in G to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in G$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

First we claim that for each new project that we identify, $p_{t_{2i}} \neq p_{t_{2i-1}}$ for $i \geq 1$. Suppose $p_{t_{2i}} = p_{t_{2i-1}}$ for some $i \geq 1$. From above s_{q_i} was identified by $l_{z_{i-1}}$ such that $(s_{q_i}, p_{t_{2i-1}}) \in M^* \setminus G$. Moreover $(s_{q_i}, p_{t_{2i}}) \in G$. Hence we reach a contradiction. Clearly, for each student s_{q_i} for $i \geq 1$ that we identify, s_{q_i} must be assigned to distinct projects in G and in M^*.

Next we claim that for each new student s_{q_i} that we identify, $s_{q_i} \neq s_{q_t}$ for $1 \leq t < i$. We prove this by induction on i. For the base case, clearly $s_{q_2} \neq s_{q_1}$. We assume that the claim holds for some $i \geq 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}$ consists of distinct students. We show that the claim holds for $i + 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}, s_{q_{i+1}}$ also consists of distinct students. Clearly $s_{q_{i+1}} \neq s_{q_i}$ since l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}. Thus, it suffices to show

that $s_{q_{i+1}} \neq s_{q_j}$ for $1 \leq j \leq i-1$. Now, suppose $s_{q_{i+1}} = s_{q_j}$ for $1 \leq j \leq i-1$. This implies that s_{q_j} was identified by l_{z_i} and clearly l_{z_i} prefers s_{q_j} to $s_{q_{j-1}}$. Now since $s_{q_{i+1}}$ was also identified by l_{z_i} to avoid the blocking pair $(s_{q_i}, p_{t_{2_i}})$ in M^*, it follows that either (i) $p_{t_{2i}}$ is full in M^*, or (ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. We consider each cases further as follows.

(i) If $p_{t_{2i}}$ is full in M^*, we know that $(s_{q_i}, p_{t_{2i}}) \in G \setminus M^*$. Moreover s_{q_j} was identified by $l_{z_{i+1}}$ because of case (i). Furthermore $(s_{q_{j-1}}, p_{t_{2i}}) \in G \setminus M^*$. In this case, $p_{t_{2i+1}} = p_{t_{2i}}$ and we have that

$$(s_{q_i}, p_{t_{2i+1}}) \in G \setminus M^* \text{ and } (s_{q_{i+1}}, p_{t_{2i+1}}) \in M^* \setminus G,$$

$$(s_{q_{j-1}}, p_{t_{2i+1}}) \in G \setminus M^* \text{ and } (s_{q_j}, p_{t_{2i+1}}) \in M^* \setminus G.$$

By the inductive hypothesis, the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_{j-1}}, s_{q_j}, \ldots, s_{q_i}$ consists of distinct students. This implies that $s_{q_i} \neq s_{q_{j-1}}$. Thus since $p_{t_{2i+1}}$ is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ to avoid the blocking pairs $(s_{q_{j-1}}, p_{t_{2i+1}})$ and $(s_{q_i}, p_{t_{2i+1}})$ respectively in M^*, a contradiction.

(ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. Similarly as in case (i) above, we have that

$$s_{q_i} \in G(l_{z_i}) \setminus M^*(l_{z_i}) \text{ and } s_{q_{i+1}} \in M^*(l_{z_i}) \setminus G(l_{z_i}),$$

$$s_{q_{j-1}} \in G(l_{z_i}) \setminus M^*(l_{z_i}) \text{ and } s_{q_j} \in M^*(l_{z_i}) \setminus G(l_{z_i}).$$

Since $s_{q_i} \neq s_{q_{j-1}}$ and l_{z_i} is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ corresponding to students $s_{q_{j-1}}$ and s_{q_i} respectively, a contradiction.

This completes the induction step. As the sequence of distinct students and projects is infinite, we reach an immediate contradiction. □

Lemma 5.3.15. *If a pair (s_i, p_j) is deleted on line 25 during an execution of Algorithm* SPA-ST-strong *then (s_i, p_j) does not belong to any strongly stable matching in I.*

Proof. Suppose that (s_i, p_j) is the first strongly stable pair to be deleted during E. Suppose further that (s_i, p_j) was deleted on line 25 of the algorithm. Let M^* be some strongly stable matching in which s_i is assigned to p_j, where l_k is the lecturer who offers p_j. We note that (s_i, p_j) is deleted at this point because some clone of p_j is provisionally assigned to a student in the left-critical set Z_s, and at that point s_i is in the tail of $\mathcal{L}_k^j$. We refer to the set of preference lists at that point as the current lists. Let Z_s' be the set of students in Z_s who are assigned in M^* to a project from the head of their current lists, and let P' be the set of

project clones in $\mathcal{N}(Z_s)$ such that the associated real projects are assigned in M^* to at least one student from the tail of its current list. Since $(s_i, p_j) \in M^*$, then some clone of p_j, say p_{j_t} for $1 \leq t \leq \min\{d_{G_r}(p_j), q^*_{p_j}\}$, must be in P'. Thus $P' \neq \emptyset$. Consider $s_{i'} \in Z_s$. We note that $s_{i'}$ cannot be assigned in M^* to a project that she prefers to any project in the head of her current list, for otherwise some strongly stable pair must have been deleted before (s_i, p_j). Hence, any student $s_{i'}$ in Z_s who is provisionally assigned to p_{j_t} must be in Z'_s, otherwise $(s_{i'}, p_j)$ would block M^*. Thus $Z'_s \neq \emptyset$. Also $Z_s \setminus Z'_s \neq \emptyset$, because $|Z_s| - |\mathcal{N}(Z_s)| > 0$ and $|Z'_s| - |\mathcal{N}(Z_s)| \leq |Z'_s| - |P'| \leq 0$, since every student in Z'_s is assigned in M^* to a project in P'.

We now claim that there must be an edge $(s_r, p_{j'_t})$ in G_r such that $s_r \in Z_s \setminus Z'_s$ and $p_{j'_t} \in P'$. For otherwise, $\mathcal{N}(Z_s \setminus Z'_s) \subseteq \mathcal{N}(Z_s) \setminus P'$, and

$$\begin{aligned} |Z_s \setminus Z'_s| - |\mathcal{N}(Z_s \setminus Z'_s)| &\geq |Z_s \setminus Z'_s| - |\mathcal{N}(Z_s) \setminus P'| \\ &= |Z_s| - |\mathcal{N}(Z_s)| - (|Z'_s| - |P'|) \\ &\geq |Z_s| - |\mathcal{N}(Z_s)|, \qquad \text{(since } |Z'_s| - |P'| \leq 0) \ . \end{aligned}$$

Hence $Z_s \setminus Z'_s$ has deficiency at least that of Z_s, contradicting the fact that Z_s is the left-critical set. Thus our claim is established, i.e., there is some student $s_r \in Z_s \setminus Z'_s$ and some project $p_{j'_t} \in P'$ such that s_r is adjacent to $p_{j'_t}$ in G_r. Since s_r is either unassigned in M^* or prefers $p_{j'}$ to $M^*(s_r)$, and since the lecturer who offers $p_{j'}$ is indifferent between s_r and at least one student in $M^*(p_{j'})$, we have that $(s_r, p_{j'})$ blocks M^*, a contradiction. □

Lemma 5.3.16. *If a pair (s_i, p_j) is deleted on line 34 during an execution of Algorithm* SPA-ST-strong*, then (s_i, p_j) does not belong to any strongly stable matching in I.*

Proof. Suppose that (s_i, p_j) is the first strongly stable pair to be deleted during E. Suppose further that (s_i, p_j) was deleted on line 34 of the algorithm. Let M^* be some strongly stable matching in which s_i is assigned to p_j, where l_k is the lecturer who offers p_j. Let $p_{j'}$ be some other project offered by l_k, such that $p_{j'}$ was deleted from the preference list of some student during an iteration of the inner `repeat-until` loop and $p_{j'}$ is undersubscribed at the end of the loop, i.e., $p_{j'}$ plays the role of p_j on line 31 of the algorithm. Suppose that $s_{i'}$ plays the role of s_r on line 30, i.e., $s_{i'}$ was provisionally assigned to $p_{j'}$ at some point during an iteration of the inner `repeat-until` loop and $s_{i'}$ is the most-preferred student rejected from $p_{j'}$ according to $\mathcal{L}_k^{j'}$ (possibly $s_{i'} = s_i$). Then $(s_{i'}, p_{j'}) \notin G$ at the end of the loop. Thus $(s_{i'}, p_{j'}) \notin M^*$, since no strongly stable pair is deleted within the inner `repeat-until` loop, as proved in Lemmas 5.3.13 - 5.3.15. Moreover, l_k prefers $s_{i'}$ to s_i or is indifferent between them, since s_i plays the role of s_t at some for loop iteration on line 32.

Let $l_{z_0} = l_k$, $p_{t_0} = p_{j'}$ and $s_{q_0} = s_{i'}$. We remark that no student who is provisionally assigned to some project in G can be assigned to a project better than her current assignment

in any strongly stable matching. For otherwise, this would mean a strongly stable pair must have been deleted before (s_i, p_j), since each student who is assigned in G applies to all the project/s in the head of her preference list. So, either (a) s_{q_0} is unassigned in M^* or s_{q_0} prefers p_{t_0} to $M^*(s_{q_0})$, or (b) s_{q_0} is indifferent between p_{t_0} and $M^*(s_{q_0})$. If (a) holds, then by the strong stability of M^*, p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0}; for otherwise, (s_{q_0}, p_{t_0}) blocks M^*, since $s_i \in M^*(l_{z_0})$ and l_{z_0} prefers s_{q_0} to s_i or is indifferent between them, a contradiction.

If (b) holds, i.e., s_{q_0} is indifferent between p_{t_0} and $M^*(s_{q_0})$. First suppose that $M^*(s_{q_0})$ is offered by l_{z_0}, then the argument follows from (a) above. Next, suppose $M^*(s_{q_0})$ is not offered by l_{z_0}, we claim that (i) if p_{t_0} is full in M^* then l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0}, and (ii) if p_{t_0} is undersubscribed in M^* then l_{z_0} prefers s_{q_0} to s_i. To prove claim (i), suppose l_{z_0} is indifferent between a worst student in $M^*(p_{t_0})$ and s_{q_0}, since (s_{q_0}, p_{t_0}) has been deleted during an iteration of the inner `repeat-until` loop, for each student s_{q_a} tied together with s_{q_0} in $\mathcal{L}_{z_0}^{t_0}$, (s_{q_a}, p_{t_0}) would have been deleted at the same iteration, a contradiction.

To prove claim (ii), suppose l_{z_0} is indifferent between s_{q_0} and s_i, we will identify the point in the algorithm where (s_{q_0}, p_{t_0}) was deleted. We note that (s_{q_0}, p_{t_0}) was not deleted on line 10 (i.e., when p_{t_0} became full or oversubscribed) of the algorithm, otherwise s_{q_0} will not be the most-preferred student rejected from p_{t_0} for p_{t_0} to become undersubscribed on line 34 after being full on line 10. Next, we note that (s_{q_0}, p_{t_0}) was not deleted on line 14 (i.e., when l_{z_0} became full or oversubscribed), otherwise (s_i, p_j) would have been deleted during the same iteration, a contradiction. Thus, (s_{q_0}, p_{t_0}) must have been deleted (a) on line 20 (i.e., at the point where some clone of p_{t_0} was in the right-critical set Z_p, and at that point s_{q_0} was in the tail of $\mathcal{L}_{z_0}^{t_0}$), or (b) on line 25 (i.e., at the point where some clone of p_{t_0} was adjacent in G_r to some student in the left-critical set Z_s). Moreover, since s_{q_0} is indifferent between p_{t_0} and $M^*(s_{q_0})$, this implies that s_{q_0} was not in G_r during that iteration. Thus, some other student, say s_{q_a}, must have caused this deletion and such student would either be unassigned in M^* or be assigned to a project worse than p_{t_0} on her preference list. As a result, if p_{t_0} is undersubscribed in M^* and l_{z_0} is full in M^* with $s_i \in M^*(l_{z_0})$, then (s_{q_a}, p_{t_0}) blocks M^*, a contradiction.

To recall, either s_{q_0} is unassigned in M^* or s_{q_0} prefers p_{t_0} to $M^*(s_{q_0})$ or is indifferent between them. Moreover, p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0}. Just before the deletion of (s_i, p_j) occurred, p_{t_0} is undersubscribed in G. Since p_{t_0} is full in M^*, it follows that there exists some student, say s_{q_1}, such that $(s_{q_1}, p_{t_0}) \in M^* \setminus G$. We note that l_{z_0} prefers s_{q_1} to s_{q_0}. Let $p_{t_1} = p_{t_0}$. Since (s_i, p_j) is the first strongly stable pair to be deleted, s_{q_1} is provisionally assigned in G to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as students apply to projects in the head of their preference list, that would mean (s_{q_1}, p_{t_1}) must have been deleted during an iteration of the inner `repeat-until` loop, a

contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in G$ and $(s_{q_1}, p_{t_1}) \notin G$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus G(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus G(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, s_{q_2} is assigned in G to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in G$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 5.3.14, we can identify an infinite sequence of distinct students and projects, a contradiction. □

Lemma 5.3.17. *If a pair (s_i, p_j) is deleted on line 41 during an execution of Algorithm* SPA-ST-strong*, then (s_i, p_j) does not belong to any strongly stable matching in I.*

Proof. Suppose that (s_i, p_j) is the first strongly stable pair to be deleted during E. Suppose further that (s_i, p_j) was deleted on line 41. Let M^* be some strongly stable matching in which s_i is assigned to p_j. Let l_k be the lecturer who offers p_j. At this point in the algorithm where the deletion of (s_i, p_j) occured, s_i is adjacent to some other project $p_{j'} \in G$ via a bound edge, where $p_{j'}$ is offered by $l_{k'}$ (note that $l_k \neq l_{k'}$). By the definition of a bound edge, it follows that either $p_{j'}$ is not oversubscribed in G or $l_{k'}$ prefers s_i to some student in $G(p_{j'})$ (or both)[11]. Also, $(s_i, p_{j'})$ is not a lower rank edge.

Let $l_{z_0} = l_{k'}$, $p_{t_0} = p_{j'}$ and $s_{q_0} = s_i$. We note that s_{q_0} is indifferent between p_j and p_{t_0}. Now, since $(s_{q_0}, p_{t_0}) \in G \setminus M^*$, by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0} or is indifferent between them;

[11] This is similar to saying $l_{k'}$ prefers s_i to some student at the tail of $\mathcal{L}_k^j$ who is assigned to $p_{j'}$ in G

(ii) p_{t_0} is undersubscribed in M^*, l_{z_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(l_{z_0})$ to s_{q_0} or is indifferent between them.

Otherwise (s_{q_0}, p_{t_0}) blocks M^*. In case (i), since s_{q_0} is bound to p_{t_0} in G and since $(s_{q_0}, p_{t_0}) \in G \setminus M^*$, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus G(p_{t_0})$. Let $p_{t_1} = p_{t_0}$. In case (ii), since s_{q_0} is assigned in G to a project offered by l_{z_0} (i.e., p_{t_0}) via a bound edge, and since s_{q_0} is not assigned to l_{z_0} in M^*, there exists some student $s_{q_1} \in M^*(l_{z_0}) \setminus G(l_{z_0})$. We note that l_{z_0} either prefers s_{q_1} to s_{q_0} or is indifferent between them; clearly $s_{q_1} \neq s_{q_0}$. Now, suppose $M^*(s_{q_1}) = p_{t_1}$ (possibly $p_{t_1} = p_{t_0}$).

Since (s_{q_0}, p_j) is the first strongly stable pair to be deleted, s_{q_1} is provisionally assigned in G to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as students apply to projects in the head of their preference list and since $(s_{q_1}, p_{t_1}) \notin G$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during an iteration of the `repeat-until` loop, a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in G$ and $(s_{q_1}, p_{t_1}) \notin G$. Let l_{z_1} be the lecturer who offers p_{t_2}. Again by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};

(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus G(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus G(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}; again it is clear that $s_{q_2} \neq s_{q_1}$. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). Applying similar reasoning as for s_{q_1}, s_{q_2} is provisionally assigned in G to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in G$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 5.3.14, we can identify an infinite sequence of distinct students and projects, a contradiction. □

Lemmas 5.3.13 - 5.3.17 immediately give rise to Lemma 5.3.12. The next two lemmas will be used as a tool in the proof of the remaining lemmas.

Lemma 5.3.18. *Let M be a feasible matching in the final provisional assignment graph G. The following holds: (a) every student who is assigned to a project in G must be assigned in M; (b) every project that is undersubscribed in G has as many assignees in M as it has in G, and every project that is full or oversubscribed in G is full in M.*

Proof. First, we show that (a) holds. Suppose s_i is a student who is provisionally assigned to some project in G. If s_i is not assigned in a feasible matching M, then s_i must be in the left-critical set Z_s. Thus $Z_s \neq \emptyset$, a contradiction. Next, we show that (b) holds. Suppose p_j is a project that is provisionally assigned to some student in G. If p_j is undersubscribed in G then $|M(p_j)| = d_G(p_j)$; and if p_j is full or oversubscribed in G then $|M(p_j)| = c_j$. For otherwise, p_j must be in the right-critical set Z_p. Thus $Z_p \neq \emptyset$, a contradiction. □

We note that an equivalence of Lemma 4.4.4 for super-stability also holds in the strong stability setting, which we restate as follows without proof.

Lemma 5.3.19. *Let M be a feasible matching in the final provisional assignment graph G and let M^* be any strongly stable matching. Let l_k be an arbitrary lecturer. The following holds: (a) if l_k is undersubscribed in M^* then every student who is assigned to l_k in M is also assigned to l_k in M^*; and (b) if l_k is undersubscribed in M then l_k has the same number of assignees in M^* as in M.*

The next three lemmas deal with the case that Algorithm SPA-ST-strong reports the non-existence of a strongly stable matching in I.

Lemma 5.3.20. *Let M be a feasible matching in the final provisional assignment graph G. Suppose that (a) some lecturer who is undersubscribed in G has fewer assignees in M than provisional assignees in G, or (b) some lecturer who is full in G is not full in M. Then I admits no strongly stable matching.*

Proof. Suppose that M^* is a strongly stable matching for the instance. By Lemma 5.3.18, each student who is provisionally assigned to a project in G must be assigned in M. Moreover, any student who is not provisionally assigned to a project in G must have an empty preference list. It follows that these students are unassigned in M^*, since Lemma 5.3.12 guarantees that no strongly stable pairs are deleted. Thus $|M^*| \leq |M|$.

Suppose that condition (a) is satisfied. Then some lecturer $l_{k'}$ who is undersubscribed in G satisfies $|M(l_{k'})| < d_G(l_{k'})$, where $d_G(l_{k'})$ is the number of students that are provisionally assigned in G to a project offered by $l_{k'}$. As $l_{k'}$ is undersubscribed, it follows that $d_G(l_{k'}) < d_{k'}$. Now $|M(l_k)| \leq \min\{d_k, d_G(l_k)\}$ for all $l_k \in \mathcal{L}$. Hence

$$|M| = \sum_{l_k \in \mathcal{L}} |M(l_k)| < \sum_{l_k \in \mathcal{L}} \min\{d_k, d_G(l_k)\} \ . \tag{5.3}$$

Now, suppose that $|M^*(l_k)| \geq \min\{d_k, d_G(l_k)\}$ for all $l_k \in \mathcal{L}$. Then $|M^*| > |M|$ by Inequality (5.3), a contradiction. Hence $|M^*(l_k)| < \min\{d_k, d_G(l_k)\}$ for some $l_k \in \mathcal{L}$. This implies that l_k is undersubcribed in M^*. Moreover, l_k has fewer assignees in M^* than provisional assignees in G. Thus there exists some student s_i who is provisionally assigned to l_k in G but not in M^*. It follows that there exists some project $p_j \in P_k$ such that $(s_i, p_j) \in G \setminus M^*$. By Lemma 5.3.12, s_i is not assigned to a project that she prefers to p_j in M^*. Also, by the strong stability of M^*, p_j is full in M^* and l_k prefers the worst student/s in $M^*(p_j)$ to s_i; for if p_j is undersubscribed in M^* then (s_i, p_j) blocks M^*, a contradiction. Since p_j is full in M^* with students that are better than s_i in $\mathcal{L}_k^j$ and $(s_i, p_j) \in G \setminus M^*$, then there is some student, say $s_{i'}$, such that l_k prefers $s_{i'}$ to s_i and $(s_{i'}, p_j) \in M^* \setminus G$. For if all the students assigned to p_j in M^* are also assigned to p_j in G, then s_i would be dominated in $\mathcal{L}_k^j$ and thus (s_i, p_j) would have been deleted in G.

Let $l_{z_0} = l_k, p_{t_0} = p_{t_1} = p_j, s_{q_0} = s_i, s_{q_1} = s_{i'}$. Again, by Lemma 5.3.12, s_{q_1} is provisionally assigned in G to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as students apply to projects in the head of their preference list, since $(s_{q_1}, p_{t_1}) \notin G$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during the algorithm's execution, a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in G$ and $(s_{q_1}, p_{t_1}) \notin G$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus G(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus G(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, student s_{q_2} is assigned in G to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in G$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 5.3.14, we can identify an infinite sequence of distinct students and projects, a contradiction.

Now suppose condition (b) is satisfied, i.e., some lecturer who is full in G is not full in M. Let L_1 and L_2 be the set of lecturers who are full and undersubscribed in G respectively. Then

some $l_{k''} \in L_1$ satisfies $|M(l_{k''})| < d_{k''}$. Condition (a) cannot be satisfied, for otherwise the first part of the proof shows that M^* does not exist. Hence $|M(l_k)| = d_G(l_k) < d_k$ for all $l_k \in L_2$. Now $|M(l_k)| \leq d_k$ for all $l_k \in L_1$. Hence

$$|M| = \sum_{l_k \in L_1} |M(l_k)| + \sum_{l_k \in L_2} |M(l_k)| < \sum_{l_k \in L_1} d_k + \sum_{l_k \in L_2} d_G(l_k) \ . \tag{5.4}$$

Now suppose that $|M^*(l_k)| = d_k$ for all $l_k \in L_1$, and $|M^*(l_k)| \geq d_G(l_k)$ for all $l_k \in L_2$. Then $|M^*| > |M|$ by Inequality (5.4), a contradiction. Hence either (i) $|M^*(l_{k'})| < d_{k'}$ for some $l_{k'} \in L_1$, or (ii) $|M^*(l_{k'})| < d_G(l_{k'})$ for some $l_{k'} \in L_2$. In case (ii), we reach a similar contradiction to that arrived at for condition (a). In case (i), we have that $l_{k'}$ is undersubscribed in M^*. As $l_{k'}$ is full in G, there exists some student s_i who is provisionally assigned to some project offered by $l_{k'}$ in G, but s_i is not assigned to a project offered by $l_{k'}$ in M^*. That is, there is some project $p_j \in P_{k'}$ such that $(s_i, p_j) \in G \setminus M^*$. By Lemma 5.3.12, s_i is not assigned to a project in M^* that she prefers to p_j. Following a similar argument as above, we can identify a sequence of distinct students and projects, and as this sequence is infinite, we reach a contradiction. Thus I admits no strongly stable matching. □

Lemma 5.3.21. *Suppose that in the final provisional assignment graph G, a student is bound to two or more projects. Then I admits no strongly stable matching.*

Proof. Suppose that a strongly stable matching exists for the instance. Let M be a feasible matching in the final provisional assignment graph G. Denote by L_1 and L_2 the set of lecturers who are full and undersubscribed in G respectively. Denote by S_1 the set of students who are bound to one or more projects in G, and by S_2 the other students who are provisionally assigned to one or more projects in G. By Lemma 5.3.18,

$$|M| = |S_1| + |S_2| \ . \tag{5.5}$$

Also,

$$|M| = \sum_{l_k \in L_1} |M(l_k)| + \sum_{l_k \in L_2} |M(l_k)| \ . \tag{5.6}$$

Moreover, we have that

$$\sum_{l_k \in L_1} |M(l_k)| + \sum_{l_k \in L_2} |M(l_k)| = \sum_{l_k \in L_1} d_k + \sum_{l_k \in L_2} d_G(l_k); \tag{5.7}$$

for otherwise, no strongly stable matching exists by Lemma 5.3.20. Now, if some student is bound to two or more projects, by considering how the lecturers' quotas are reduced when

the students in S_1 are removed in forming G_r from G, it follows that

$$\begin{aligned}\sum_{l_k \in L_1}(d_k - q^*_{l_k}) + \sum_{l_k \in L_2}(d_G(l_k) - q^*_{l_k}) &\geq \sum_{l_k \in L_1}(\min\{\alpha_k, d_k\} - q^*_{l_k}) + \sum_{l_k \in L_2}(\min\{\alpha_k, d_k\} - q^*_{l_k}) \\ &= \sum_{l_k \in L_1}(q_{l_k} - q^*_{l_k}) + \sum_{l_k \in L_2}(q_{l_k} - q^*_{l_k}) \\ &= \sum_{l_k \in L_1 \cup L_2}(q_{l_k} - q^*_{l_k}) \\ &> |S_1|, \end{aligned} \tag{5.8}$$

where $\sum_{l_k \in L_1 \cup L_2}(q_{l_k} - q^*_{l_k})$ is the total number of bound edges that are adjacent to a project offered by a lecturer in $L_1 \cup L_2$. It is clear that $d_k - q^*_{l_k} \geq \min\{\alpha_k, d_k\} - q^*_{l_k}$ for each $l_k \in L_1$. We claim that $d_G(l_k) \geq \alpha_k$ for each $l_k \in L_2$. Suppose otherwise, i.e., there exists some $l_k \in L_2$ such that $d_G(l_k) < \alpha_k$, i.e., $d_G(l_k) < \{q_{p_j} : p_j \in P_k \cap P\}$. This implies that $d_G(l_k) < q_{l_k}$. Now, by considering how G_r is formed from G, we have that $d_{G_r}(l_k) < q^*_{l_k}$.[12] Thus, no dummy students will be adjacent to a project offered by l_k in G_r. Moreover, after replacing each project offered by l_k in G_r with clones, we have that $d_{G_r}(l_k) < |\mathcal{N}(G_r(l_k))|$.[13] This implies that, in G_r, some cloned project associated with a real project offered by l_k does not belong to some maximum matching. Thus, the right-critical set Z_p is non-empty, a contradiction. Hence, $d_G(l_k) - q^*_{l_k} \geq \min\{\alpha_k, d_k\} - q^*_{l_k}$ for each $l_k \in L_2$. Now, by substituting Equality 5.7 into Inequality 5.8, we obtain the following

$$\sum_{l_k \in L_1}|M(l_k)| - \sum_{l_k \in L_1} q^*_{l_k} + \sum_{l_k \in L_2}|M(l_k)| - \sum_{l_k \in L_2} q^*_{l_k} > |S_1| \ . \tag{5.9}$$

Combining Equalities (5.5) and (5.6) into Inequality (5.9), we have

$$\begin{aligned}|S_1| + |S_2| - \sum_{l_k \in L_1} q^*_{l_k} - \sum_{l_k \in L_2} q^*_{l_k} &> |S_1|, \\ \sum_{l_k \in L_1 \cup L_2} q^*_{l_k} &< |S_2| \ . \end{aligned} \tag{5.10}$$

Since the total revised quota of lecturers in $L_1 \cup L_2$ whose projects are provisionally assigned to a student in G_r is strictly less than the number of students in S_2, the preceding inequality suffices to establish that the left-critical set Z_s is non-empty, a contradiction. □

Lemma 5.3.22. *Let M be a feasible matching in the final provisional assignment graph G. If some project in P^* is not full in M then I admits no strongly stable matching.*

[12] Since for each bound edge which is adjacent to a project offered by l_k that we remove from G_r, we reduce the quota of l_k by 1.

[13] This is because before the project cloning takes place, the number of students adjacent to a project offered by l_k in G_r is fewer than the total quota of projects offered by l_k in G_r.

Proof. Suppose for a contradiction that there exists a strongly stable matching M^* in I. By the hypothesis of the lemma, P^* is non-empty. Thus there exists some project $p_j \in P^*$ and some student, say s_i, such that (s_i, p_j) has been deleted. Thus $(s_i, p_j) \notin G$. Let l_k be the lecturer who offers p_j. By the definition of P^*, either (a) l_k is undersubscribed in G or (b) l_k is full in G and l_k prefers s_i to the worst student/s in $G(l_k)$.

If l_k is undersubscribed in G then l_k is undersubscribed in M, since M is formed from G. By Lemma 5.3.19, we have that l_k is undersubscribed in M^*. Since p_j is offered by l_k, and p_j is undersubscribed in M, it follows from the proof of Lemma 4.4.6 (paragraph 2) that p_j is undersubscribed in M^*. Moreover, since (s_i, p_j) has been deleted, we have that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. By Lemma 5.3.12, this is true for s_i in M^*. Hence (s_i, p_j) blocks M^*, a contradiction.

If l_k is full in G and l_k prefers s_i to some student in $G(l_k)$. By Lemma 5.3.20, l_k is full in M, and by Lemma 5.3.19, l_k is full in M^*. Moreover, in this case, the only possibility is that s_i is indifferent between p_j and $M(s_i)$. For suppose otherwise, i.e., either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$, then (s_i, p_j) must have been deleted during some iteration of the outer `repeat-until` loop. Now, since p_j is undersubscribed in G, then every student that is no better than s_i according to $\mathcal{L}_k$ would have been deleted from l_k's preference list on line 34. However, l_k prefers s_i to the worst student/s in $G(l_k)$. Thus, (s_i, p_j) must have been deleted on line 41, which implies that (s_i, p_j) is an unbound edge. Further, we note that $s_i \notin G(l_k)$ because the deletion of (s_i, p_j) on line 41 implies that s_i is bound to some other project offered by a lecturer different from l_k. So, if $s_i \in G(l_k)$ at the termination of the algorithm, then s_i must be bound to two projects. Thus, no strongly stable matching exists by Lemma 5.3.21, a contradiction. Now, by Lemma 5.3.12, either (a) s_i prefers p_j to $M^*(s_i)$ or (b) s_i is indifferent between p_j and $M^*(s_i)$. Let $l_{z_0} = l_k$, $p_{t_0} = p_j$ and $s_{q_0} = s_i$. If (a) holds then by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0};
(ii) p_{t_0} is undersubscribed in M^*, l_{z_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(l_{z_0})$ to s_{q_0}.

Otherwise (s_{q_0}, p_{t_0}) blocks M^*. In case (i), since p_{t_0} is undersubscribed in G, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus G(p_{t_0})$. Let $p_{t_1} = p_{t_0}$. In case (ii), since l_{z_0} prefers s_{q_0} to some student in $G(l_{z_0})$, there exists some student $s_{q_1} \in M^*(l_{z_0}) \setminus G(l_{z_0})$. We note that l_{z_0} prefers s_{q_1} to s_{q_0}; clearly $s_{q_1} \neq s_{q_0}$. Now, suppose $M^*(s_{q_1}) = p_{t_1}$ (possibly $p_{t_1} = p_{t_0}$). If (b) holds, then by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(p_{t_0})$ to s_{q_0} or is indifferent between them;

(ii) p_{t_0} is undersubscribed in M^*, l_{z_0} is full in M^* and l_{z_0} prefers the worst student/s in $M^*(l_{z_0})$ to s_{q_0} or is indifferent between them.

Otherwise (s_{q_0}, p_{t_0}) blocks M^*. In case (i), since p_{t_0} is undersubscribed in G, there exists some student $s_{q_1} \in M^*(p_{t_0}) \setminus G(p_{t_0})$. Let $p_{t_1} = p_{t_0}$. In case (ii), since l_{z_0} prefers s_{q_0} to some student in $G(l_{z_0})$, there exists some student $s_{q_1} \in M^*(l_{z_0}) \setminus G(l_{z_0})$. We note that l_{z_0} either prefers s_{q_1} to s_{q_0} or is indifferent between them. Now, suppose $M^*(s_{q_1}) = p_{t_1}$ (possibly $p_{t_1} = p_{t_0}$).

By Lemma 5.3.12, s_{q_1} is provisionally assigned in G to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as students apply to projects in the head of their preference list during the algorithm's execution, and since $(s_{q_1}, p_{t_1}) \notin G$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during some iteration of the algorithm, a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in G$ and $(s_{q_1}, p_{t_1}) \notin G$. Let l_{z_1} be the lecturer who offers p_{t_2}. Again by the strong stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \setminus G(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \setminus G(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}; again it is clear that $s_{q_2} \neq s_{q_1}$. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). Applying similar reasoning as for s_{q_1}, s_{q_2} is provisionally assigned in G to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in G$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 5.3.14, we can identify an infinite sequence of distinct students and projects, a contradiction. □

The next lemma shows that the feasible matching M may be used to determine the existence, or otherwise, of a strongly stable matching in I.

Lemma 5.3.23. *Let M be a feasible matching in the final provisional assignment graph G. If M is not strongly stable then I admits no strongly stable matching.*

Proof. Suppose that M is not strongly stable. Let (s_i, p_j) be a blocking pair for M, and let l_k be the lecturer who offers p_j. We consider two cases.

1. Suppose $(s_i, p_j) \notin G$, i.e., (a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$, or (b) s_i is indifferent between p_j and $M(s_i)$. Then (s_i, p_j) has been deleted. First, suppose (s_i, p_j) forms a blocking pair for M because each of p_j and l_k is undersubscribed in M. If l_k is full in G then by Lemma 5.3.20, no strongly stable matching exists. Hence l_k is undersubscribed in G. In cases (a) and (b), we have that $p_j \in P^*$. Moreover, since p_j is not full in M, no strongly stable matching exists by Lemma 5.3.22.

   Next, suppose (s_i, p_j) forms a blocking pair for M because p_j is undersubscribed in M and l_k is full in M. We claim that $s_i \notin M(l_k)$ and l_k prefers the worst student/s in $M(l_k)$ to s_i. To prove our claim, we identify the point at which (s_i, p_j) was deleted within the inner `repeat-until` loop during the algorithm's execution, i.e., lines 10, 14, 20, 25 and 34. We treat the deletion outside the `repeat-until` loop separately, i.e., line 41, because it is only valid for case (b).

   Suppose (s_i, p_j) was deleted on line 10 as a result of p_j becoming full or oversubscribed. Let s_r be some student who was assigned to p_j at some point during the algorithm's execution, where s_r is the most-preferred student rejected from p_j according to $\mathcal{L}_k^j$. Then either (i) $s_r = s_i$, or (ii) l_k is indifferent between s_r and s_i, or (iii) l_k prefers s_r to s_i. As p_j is undersubscribed, each pair (s_t, p_u), for each s_t that is no better than s_r at the tail of $\mathcal{L}_k$ and each $p_u \in A_t \cap P_k$, would have been deleted on line 34. Thus our claim holds. Now, suppose (s_i, p_j) was deleted on line 14 as a result of l_k becoming full or oversubscribed, then for each s_t at the tail of $\mathcal{L}_k$ and for each $p_u \in A_t \cap P_k$, (s_t, p_u) would have been deleted on line 14. Thus our claim holds.

   Now, suppose (s_i, p_j) was deleted (i) on line 20, i.e., because s_i was in the tail of $\mathcal{L}_k^j$ at a point when some clone of p_j was in the right-critical set Z_p, or (ii) on line 25, i.e., because s_i was in the tail of $\mathcal{L}_k^j$ at a point when some clone of p_j was adjacent to a student in the left-critical set Z_s. Let s_r be some student who was adjacent to some clone of p_j in the reduced assignment graph at that point. The argument follows from the first part of the previous paragraph. Thus our claim holds.

   Finally, suppose (s_i, p_j) was deleted on line 34, because some other project $p_{j'}$ offered by l_k has been deleted from some student's preference list and $p_{j'}$ was undersubscribed at line 31. Then l_k must have identified her most-preferred student, say s_r, who was assigned to $p_{j'}$ at some point during the algorithm's execution but is not assigned to $p_{j'}$ on line 30. Moreover, for each s_t at the tail of $\mathcal{L}_k$ and for each $p_u \in A_t \cap P_k$, (s_t, p_u) would have been deleted on line 34. Thus our claim holds.

Now, suppose (s_i, p_j) was deleted on line 41. Then the only possibility is that s_i is indifferent between p_j and $M(s_i)$. In addition, s_i must be bound to some project $p_{j'}$ offered by a lecturer different from l_k. Further, s_i is not assigned to any project offered by l_k in G. For otherwise, s_i is bound to two projects offered by different lecturers; and thus, no strongly stable matching exists, by Lemma 5.3.21. Hence, $s_i \notin G(l_k)$, and hence $s_i \notin M(l_k)$. Again, recall that p_j is undersubscribed in M and l_k is full in M. Moreover, (s_i, p_j) can only form a blocking pair in this case if l_k prefers s_i to the worst student/s in $M(l_k)$. If this is the case, then $p_j \in P^*$. Since p_j is not full in M, no strongly stable matching exists by Lemma 5.3.22.

2. Suppose $(s_i, p_j) \in G$, i.e., s_i is indifferent between p_j and $M(s_i)$. Then (s_i, p_j) has not been deleted. First, suppose (s_i, p_j) forms a blocking pair for M because each of p_j and l_k is undersubscribed in M. If l_k is full in G then by Lemma 5.3.20, no strongly stable matching exists. Hence, l_k is undersubscribed in G. It follows that s_i is bound to p_j. Now, if $s_i \notin M(l_k)$ then l_k has fewer assignees in M than provisional assignees in G. Thus no strongly stable matching exists by Lemma 5.3.20. Hence, $s_i \in M(l_k)$. Let $M(s_i) = p_{j'}$, we claim that s_i is bound to $p_{j'}$. For otherwise, since l_k is undersubscribed in G, the only possibility is that $p_{j'}$ is oversubscribed in G and s_i is in the tail of $\mathcal{L}_k^{j'}$. By the construction of G_r, since s_i is bound to p_j, we have that s_i is not adjacent to any project in G_r. Thus, some other student $s_{i'}$ at the tail of $\mathcal{L}_k^{j'}$ is adjacent to clones of $p_{j'}$ in G_r. Since $(s_i, p_{j'}) \in M$, this implies that some clone of $p_{j'}$ is unassigned in some maximum matching obtained from G_r. Thus the right-critical set is non-empty, a contradiction. Hence our claim holds; and hence s_i is bound to two projects in G. Thus no strongly stable matching exists by Lemma 5.3.21.

   Next, suppose (s_i, p_j) forms a blocking pair for M because p_j is undersubscribed in M and l_k is full in M. Suppose firstly that $s_i \notin M(l_k)$. It follows that both (s_i, p_j) and $(s_i, M(s_i))$ are unbound edges.[14] Now, since p_j is undersubscribed in M and (s_i, p_j) is an unbound edge, this implies that (s_i, p_j) is a lower rank edge. Thus l_k is full in M with students that are no worse than s_i, and thus (s_i, p_j) cannot block M. Hence $s_i \in M(l_k)$. In this case, we claim that (s_i, p_j) is not a lower rank edge. For otherwise, (s_i, p_j) and $(s_i, M(s_i))$ are both in the reduced assignment graph G_r. Moreover, since $(s_i, p_j) \in G \setminus M$ and since p_j is undersubscribed in M, it follows that some clone of p_j is unassigned in some maximum matching obtained from G_r. Thus, the right-critical set is non-empty, a contradiction. Further, since p_j is undersubscribed and (s_i, p_j) is not a lower rank edge, it follows that s_i is bound to p_j. Let $M(s_i) = p_{j'}$, the argument that s_i is bound to $p_{j'}$ follows from the previous paragraph. Hence, s_i is bound to two projects in G; and hence, no strongly stable matching exists by Lemma 5.3.21.

[14] Justification for this is as follows: if (s_i, p_j) is a bound edge then $(s_i, M(s_i))$ would have been deleted on line 41; similarly, if $(s_i, M(s_i))$ is a bound edge then (s_i, p_j) would have been deleted on line 41.

Finally, suppose p_j is full in M. Then (s_i, p_j) cannot block M irrespective of whether l_k is full or undersubscribed in M, since the worst student/s in $M(p_j)$ are either better or no worse than s_i, according to $\mathcal{L}_k^j$.

Since (s_i, p_j) is an arbitrary pair, this implies that I admits no strongly stable matching. □

Lemma 5.3.24. *Algorithm* SPA-ST-strong *may be implemented to run in $O(m^2)$ time, where m is the total length of the students' preference lists.*

Proof. It is clear that the work done in the inner `repeat-until` loop other than in finding the maximum cardinality matchings and critical sets is bounded by a constant times the number of deleted pairs, and so is $O(m)$. We remark that the total amount of work done outside the inner `repeat-until` loop (i.e., in deleting pairs when a project that has been deleted from a student's preference list is undersubscribed on line 31) is bounded by the total length of the students' preference lists, and so is $O(m)$. Similarly, the total amount of work done on lines 36 - 41 (i.e., in deleting some set of unbound edges) is bounded by the total length of the students' preference lists, and so is $O(m)$. During each iteration of the inner `repeat-until` loop of Algorithm SPA-ST-strong, we need to form the reduced assignment graph G_r, which takes $O(m)$ time. Further, we need to find a maximum matching in G_r, which allows us to use Lemmas 5.3.10 and 5.3.11 to find the right-critical and left-critical set respectively. Next we show how we can bound the total amount of work done in finding the maximum matchings, which is inspired by that given in the HRT setting [58].

Suppose that Algorithm SPA-ST-strong finds a maximum matching $M_r^{(i)}$ in the reduced assignment graph $G_r^{(i)}$ at the ith iteration of the inner `repeat-until` loop. Suppose also that, during the ith iteration in this loop, a total of x_i pairs are deleted on lines 20 and 25 because they involve (i) students that are in the tail of a project whose clone is in the right-critical set $Z_p^{(i)}$, and (ii) students that are in the left-critical set $Z_s^{(i)}$ or students tied with them in the tail of a project whose clone is in $\mathcal{N}(Z_s^{(i)})$. Suppose further that in the $(i+1)$th iteration, y_i pairs are deleted before the reduced assignment graph $G_r^{(i+1)}$ is formed. Note that any edge in $G_r^{(i)}$ which is not one of these $x_i + y_i$ deleted pairs must be in $G_r^{(i+1)}$, since an edge in the provisional assignment graph cannot change state from unbound to bound. In particular, at least $|M_r^{(i)}| - x_i - y_i$ edges of $M_r^{(i)}$ remain in $G_r^{(i+1)}$ at the $(i+1)$th iteration. Hence we can start from these edges in that iteration and find a maximum matching $M_r^{(i+1)}$ in $O(\min\{nm, (x_i + y_i + z_i)m\})$ time, where z_i is the number of edges in $G_r^{(i+1)}$ which were not in $G_r^{(i)}$.

Suppose that a total of s iterations of the inner `repeat-until` loop are carried out, and that in t of these, $\min\{nm, (x_i + y_i + z_i)m\}) = nm$. Then the time complexity of maximum matching operations, taken over the entire execution of the algorithm, is $O(\min\{n, \sum c_j\}m +$

$tnm + m\sum(x_i + y_i + z_i))$ where the first term is for the maximum matching computation in the first iteration, and the sum in the third term is over the appropriate $s - t - 1$ values of i. We note that $\sum_{i=1}^{s}(x_i + y_i + z_i) \leq 2m$ (since each of the total number of deletions and provisional assignments is bounded by the total length of the students' preference lists), and since $x_i + y_i + z_i \geq n$ for the appropriate t values of i, it follows that

$$tnm + m\sum(x_i + y_i + z_i) \leq m\sum_{i=1}^{s}(x_i + y_i + z_i) \leq 2m^2.$$

Thus

$$m\sum(x_i + y_i + z_i) \leq 2m^2 - tnm.$$

To construct a feasible matching M on line 42, we first need to identify the set P^* of projects that needs to fill up in M. The total amount of work done in finding P^* is bounded by the total length of the students' preference lists, and so is $O(m)$. Moreover, M can be constructed by taking the final maximum matching at the termination of the outer `repeat-until` loop and augmenting it to a maximum matching in G by prioritising edges that are adjacent to a project in P^* over edges that are adjacent to a project in $\mathcal{P} \setminus P^*$.[15] This operation is clearly bounded by the number of edges, and so is $O(m)$. It follows that the overall complexity of Algorithm SPA-ST-strong is $O(m + \min\{n, \sum c_j\}m + 2m^2) = O(m^2)$.

We remark that the Hopcroft-Karp algorithm [49] can be used to find a maximum matching in G_r, since each student vertex and cloned project vertex has capacity 1. This can be implemented to run in $O(\sqrt{n}m)$ time, where n is the number of vertices in G_r and m is the number of edges. However, it is not clear how this algorithm can be used to improve the overall running time of Algorithm SPA-ST-strong. □

The following theorem collects together Lemmas 5.3.12 - 5.3.24 and establishes the correctness of Algorithm SPA-ST-strong.

Theorem 5.3.25. *For a given instance I of* SPA-ST*, Algorithm* SPA-ST-strong *determines in $O(m^2)$ time whether or not a strongly stable matching exists in I. If such a matching does exist, all possible executions of the algorithm find one in which each assigned student is assigned at least as good a project as she could obtain in any strongly stable matching, and each unassigned student is unassigned in every strongly stable matching.*

Given the optimality property established by Theorem 5.3.25, we define a strongly stable matching found by Algorithm SPA-ST-strong to be *student-optimal*. We note that a student-optimal strongly stable matching in an arbitrary instance of SPA-ST need not be unique.

[15] We remark that the final outcome is independent of the type of edges (i.e., bound or unbound) that are selected during the augmenting stage.

5.3.6 Properties of strongly stable matchings in SPA-ST

In this section, we consider properties of the set of strongly stable matchings in an instance of SPA-ST. We show that the Unpopular Projects Theorem for SPA-S (see Theorem 2.3.2), which holds for SPA-ST under super-stability (i.e., Theorem 4.4.11) also holds for SPA-ST under strong stability.

Theorem 5.3.26. *For a given instance I of* SPA-ST*, the following properties holds:*

1. *each lecturer is assigned the same number of students in all strongly stable matchings;*
2. *exactly the same students are unassigned in all strongly stable matchings;*
3. *a project offered by an undersubscribed lecturer has the same number of students in all strongly stable matchings.*

Proof. Let M be the student-optimal strongly stable matching returned by Algorithm SPA-ST-strong, and let M^* be any other strongly stable matching in I.

1. Let l_k be an arbitrary lecturer. By Lemma 5.3.19, if l_k is full in M then l_k is full in M^*. Moreover, if l_k is undersubscribed in M, then l_k has as many assignees in M^* as she has in M. Thus $|M(l_k)| \leq |M^*(l_k)|$. However, $|M^*(l_k)| \leq |M(l_k)|$, since M is student-optimal and Lemma 5.3.12 guarantees that no strongly stable pair is deleted during an execution of the algorithm. Hence $|M(l_k)| = |M^*(l_k)|$.
2. Let U and U^* be the set of students that are unassigned in M and M^* respectively. By Lemma 5.3.12, $U \subseteq U^*$, since each student who is unassigned in M is also unassigned in M^*. However, from (1) above, we have that $|M| = |M^*|$ and thus $|U| = |U^*|$. Hence, $U = U^*$.
3. Let l_k be an arbitirary lecturer who is undersubscribed in M. By Lemma 5.3.19, l_k is undersubscribed in M^*. We want to show that each project offered by l_k has the same number of assignees in M^* as in M. Suppose on the contrary that there exists some project $p_j \in P_k$ such that $|M^*(p_j)| < |M(p_j)|$, then p_j is undersubscribed in M^* since no project is oversubscribed in M. Moreover, there is some student $s_r \in M(p_j) \setminus M^*(p_j)$. By Lemma 5.3.12, either s_r is unassigned in M^* or s_r prefers p_j to $M^*(s_r)$ or is indifferent between them. Since each of p_j and l_k is undersubscribed in M^*, (s_r, p_j) blocks M^*, a contradiction. Therefore $|M^*(p_j)| \geq |M(p_j)|$. Moreover, from (1), we have shown that $|M(l_k)| = |M^*(l_k)|$. Hence, $|M(p_j)| = |M^*(p_j)|$ for all $p_j \in P_k$. This completes the proof.

□

To illustrate each of these properties, consider the SPA-ST instance I_6 given in Figure 5.11, which admits the strongly stable matchings $M_1 = \{(s_1, p_2), (s_2, p_1), (s_3, p_3), (s_4, p_3), (s_6, p_4),$

Student preferences	Lecturer preferences	offers
s_1: p_2 $(p_4$ $p_1)$	l_1: s_6 s_2 s_4 s_3 s_5	p_3
s_2: p_1 p_3 p_2	l_2: s_1 s_3 s_2 $(s_5$ $s_7)$	p_1, p_2
s_3: p_3 p_2 p_1	l_3: s_6 $(s_7$ $s_5)$ $(s_1$ $s_4)$	p_4, p_5
s_4: p_3 $(p_4$ $p_5)$		
s_5: p_4 p_3 p_1		
s_6: p_4 p_5 p_3	Project capacities: $c_1 = c_2 = c_3 = c_5 = 2$, $c_4 = 1$	
s_7: p_4 $(p_5$ $p_1)$	Lecturer capacities: $d_1 = d_2 = 2$, $d_3 = 3$	

Figure 5.11: An instance I_6 of SPA-ST.

$(s_7, p_5)\}$ and $M_2 = \{(s_1, p_2), (s_2, p_3), (s_3, p_2), (s_4, p_3), (s_6, p_4), (s_7, p_5)\}$. The reader can easily verify that M_1 is the student-optimal strongly-stable matching in I_6. Each of l_1, l_2 and l_3 is assigned the same number of students in both M_1 and M_2, illustrating part (1) of Theorem 5.3.26. Also, s_5 is unassigned in both M_1 and M_2, illustrating part (2) of Theorem 5.3.26. Finally, l_3 is undersubscribed in both M_1 and M_2, and the projects offered the l_3, i.e., p_4 and p_5, have the same number of students in both M_1 and M_2, illustrating part (3) of Theorem 5.3.26.

We note that l_2 is full in both M_1 and M_2; however, the projects offered by l_2 (i.e., p_1 and p_2) do not have the same number of students in both M_1 and M_2. We illustrate this in Table 5.12 below.

	M_1	M_2
l_2	$\{s_1, s_2\}$	$\{s_1, s_3\}$
p_1	$\{s_2\}$	$\emptyset$
p_2	$\{s_1\}$	$\{s_1, s_3\}$

Figure 5.12: The set of students assigned to each of l_2, p_1 and p_2 in both M_1 and M_2, with respect to the SPA-ST instance I_6 in Figure 5.11.

5.4 An IP model for strong stability in SPA-ST

In this section, we describe an IP model for SPA-ST under strong stability. Again, our reason for doing this is to test the correctness of our implementation of Algorithm SPA-ST-strong. Let I be an instance of SPA-ST involving a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of students, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of projects and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of lecturers. We construct an IP model J of I as follows.

Variables. The variables involved in this setting are similar to those given in the context of SPA-ST under super-stability. For completeness, we recall them in what follows. We

create binary variables $x_{i,j} \in \{0,1\}$ $(1 \leq i \leq n_1, 1 \leq j \leq n_2)$ for each acceptable pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ such that $x_{i,j}$ indicates whether s_i is assigned to p_j in a solution or not. Henceforth, we denote by S a solution in the IP model J, and we denote by M the matching derived from S in the following natural way: if $x_{i,j} = 1$ under S then s_i is assigned to p_j in M, otherwise s_i is not assigned to p_j in M.

5.4.1 Constraints

In this section, we give the set of constraints to ensure that the assignment obtained from a feasible solution in J is a matching, and that the matching admits no blocking pair.

Matching constraints. The constraints to ensure the feasibility of a matching are exactly the same as Inequalities (4.5), (4.6) and (4.7) for SPA-ST under super-stability (see page 72).

Given a lecturer $l_k \in \mathcal{L}$, a student $s_i \in \mathcal{L}_k$, and an acceptable pair (s_i, p_j), the definition of $\mathrm{rank}(s_i, p_j)$ and $\mathrm{rank}(l_k, s_i)$ follows from that given in the super-stability setting (see page 73). With respect to an acceptable pair (s_i, p_j), we define $S_{i,j} = \{p_{j'} \in A_i : \mathrm{rank}(s_i, p_{j'}) \leq \mathrm{rank}(s_i, p_j)\}$, the set of projects that s_i likes as much as p_j. We also define $S^*_{i,j} = \{p_{j'} \in A_i : \mathrm{rank}(s_i, p_{j'}) = \mathrm{rank}(s_i, p_j)\}$, the set of projects that s_i ranks equally with p_j in the same tie.

In what follows, we fix an arbitrary acceptable pair (s_i, p_j) and we impose constraints to ensure that (s_i, p_j) does not form a blocking pair for the matching M. Henceforth, l_k is the lecturer who offers p_j.

Blocking pair constraints. First, we define $\theta_{i,j} = 1 - \sum_{p_{j'} \in S_{i,j}} x_{i,j'}$. Intuitively, $\theta_{i,j} = 1$ if and only if s_i is unassigned in M or s_i prefers p_j to $M(s_i)$. Next, we define $\theta^*_{i,j} = \sum_{p_{j'} \in S^*_{i,j}} x_{i,j'} - x_{i,j}$. Intuitively, $\theta^*_{i,j} = 1$ if and only if s_i is indifferent between p_j and $M(s_i)$, where $p_j \neq M(s_i)$. Henceforth, if (s_i, p_j) forms a blocking pair for M then we refer to (s_i, p_j) as a blocking pair of type (1i), type (1ii), type (1iii), type (2i), type (2ii) or type (2iii), according as (s_i, p_j) satisfies condition (1bi), (1bii), (1biii), (2bi), (2bii) or (2biii), of Definition 5.2.1, respectively.

We remark that the constraints to avoid blocking pair of types (1i), 1(ii) and 1(iii) are exactly the same as those given for blocking pair of types (i), (ii) and (iii) respectively, in the super-stability setting (see Inequalities (4.10), (4.13) and (4.16)). The only exception is that the $\theta_{i,j}$ in each inequality is as defined for the strong stability setting. For completeness, we recall these blocking pair constraints below.

Type (1i). The following constraint ensures that (s_i, p_j) does not form a type (1i) blocking pair for M, where α_j and β_k are as defined for a type (i) blocking pair in the super-stability

setting (see page 73).

$$\boxed{\theta_{i,j} + \alpha_j + \beta_k \leq 2 \ .} \tag{5.11}$$

Type (1ii). The following constraint ensures that (s_i, p_j) does not form a type (1ii) blocking pair for M, where η_k and $\delta_{i,k}$ are as defined for a type (ii) blocking pair in the super-stability setting (see page 74).

$$\boxed{\theta_{i,j} + \alpha_j + \eta_k + \delta_{i,k} \leq 3 \ .} \tag{5.12}$$

Type (1iii). The following constraint ensures that (s_i, p_j) does not form a type (1iii) blocking pair for M, where γ_j and $\lambda_{i,j,k}$ are as defined for a type (iii) blocking pair in the super-stability setting (see page 74).

$$\boxed{\theta_{i,j} + \gamma_j + \lambda_{i,j,k} \leq 2 \ .} \tag{5.13}$$

Next, we give the constraints to avoid blocking pair of types (2i), 2(ii) and (2iii).

Type (2i). The following constraint ensures that (s_i, p_j) does not form a type (2i) blocking pair for M, where α_j and β_k are as defined for a type (i) blocking pair in the super-stability setting (see page 73).

$$\boxed{\theta^*_{i,j} + \alpha_j + \beta_k \leq 2 \ .} \tag{5.14}$$

Type (2ii). First, we define $\omega_{i,k} = \sum_{p_{j'} \in P_k} x_{i,j'}$. If s_i is assigned to a project offered by l_k in M then $\omega_{i,k} = 1$ in S; otherwise, $\omega_{i,k} = 0$ in S. We define

$$D^*_{i,k} = \{s_{i'} \in \mathcal{L}_k : \mathrm{rank}(l_k, s_{i'}) \leq \mathrm{rank}(l_k, s_i)\},$$

the set of students that l_k likes as much as s_i. Next, we create a binary variable $\mu_{i,k}$ in J such that if $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$, then $\mu_{i,k} = 1$. We enforce this condition by imposing the following constraint.

$$d_k \mu_{i,k} \geq \omega_{i,k} + \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} - \sum_{s_{i'} \in D^*_{i,k}} \sum_{p_{j'} \in P_k} x_{i',j'} \ , \tag{5.15}$$

where $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} = |M(l_k)|$ and $\sum_{s_{i'} \in D^*_{i,k}} \sum_{p_{j'} \in P_k} x_{i',j'}$ is the occupancy of l_k in M involving students that are at least as good as s_i (including s_i). Note that if $s_i \in M(l_k)$ or l_k prefers

s_i to a worst student in $M(l_k)$ then the RHS of Inequality (5.15) is at least 1, and this implies that $\mu_{i,k} = 1$; otherwise, $\mu_{i,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (2ii) blocking pair for M, where η_k is as defined in the super-stability setting (see page 74).

$$\boxed{\theta^*_{i,j} + \alpha_j + \eta_k + \mu_{i,k} \leq 3 \ .} \tag{5.16}$$

Type (2iii). We define $T^*_{i,j,k} = \{s_{i'} \in \mathcal{L}^j_k : \text{rank}(l_k, s_{i'}) \leq \text{rank}(l_k, s_i)\}$, the set of students that are at least as good as s_i on the projected preference list of l_k for p_j. Next, we create a binary variable $\tau_{i,j,k}$ in J such that if l_k prefers s_i to a worst student in $M(p_j)$, then $\tau_{i,j,k} = 1$. We enforce this condition by imposing the following constraint.

$$c_j \tau_{i,j,k} \geq \sum_{i'=1}^{n_1} x_{i',j} - \sum_{s_{i'} \in T^*_{i,j,k}} x_{i',j} \ , \tag{5.17}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$ and $\sum_{s_{i'} \in T^*_{i,j,k}} x_{i',j}$ is the occupancy of p_j in M involving students that l_k likes as much as s_i. Note that if l_k prefers s_i to a worst student in $M(p_j)$, then the RHS of Inequality (5.17) is at least 1, and this implies that $\tau_{i,j,k} = 1$; otherwise, $\tau_{i,j,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (2iii) blocking pair for M, where γ_j is as defined in the super-stability setting (see page 74).

$$\boxed{\theta^*_{i,j} + \gamma_j + \tau_{i,j,k} \leq 2 \ .} \tag{5.18}$$

5.4.2 Variables

We define a collective notation for each set of variables involved in J as follows:

$A = \{\alpha_j : 1 \leq j \leq n_2\}$, $\Delta = \{\delta_{i,k} : 1 \leq i \leq n_1, 1 \leq k \leq n_3\}$,
$B = \{\beta_k : 1 \leq k \leq n_3\}$, $M = \{\mu_{i,k} : 1 \leq i \leq n_1, 1 \leq k \leq n_3\}$,
$N = \{\eta_k : 1 \leq k \leq n_3\}$, $\Lambda = \{\lambda_{i,j,k} : 1 \leq i \leq n_1, 1 \leq j \leq n_2, 1 \leq k \leq n_3\}$,
$\Gamma = \{\gamma_j : 1 \leq j \leq n_2\}$, $T = \{\tau_{i,j,k} : 1 \leq i \leq n_1, 1 \leq j \leq n_2, 1 \leq k \leq n_3\}$,
$X = \{x_{i,j} : s_i \in \mathcal{S} \wedge p_j \in A_i\}$.

5.4.3 Objective function

Similar to the super-stability setting, all strongly stable matchings are of the same size, which nullifies the need for an objective function. However, optimisation solvers require an objective function in addition to the variables and constraints in order to produce a solution. The

objective function given below involves maximising the summation of all the $x_{i,j}$ binary variables.

$$\boxed{\max \sum_{i=1}^{n_1} \sum_{p_j \in A_i} x_{i,j} \ .} \tag{5.19}$$

Finally, we have constructed an IP model J of I comprising the set of integer-valued variables $A, B, N, \Gamma, X, \Delta, M, \Lambda$ and T, the set of constraints (4.5), (4.6), (4.7), (4.8), (4.9), (4.11), (4.12), (4.14), (4.15), (5.11) - (5.18) and an objective function (5.19). Note that J can then be used to construct a strongly stable matching in I, should one exist.

5.4.4 Correctness of the IP model

Given an instance I of SPA-ST formulated as an IP model J using the above transformation, we present the following lemmas regarding the correctness of J.

Lemma 5.4.1. *A feasible solution S to J corresponds to a strongly stable matching M in I.*

Proof. Assume firstly that J has a feasible solution S. Let $M = \{(s_i, p_j) \in \mathcal{S} \times \mathcal{P} : x_{i,j} = 1\}$ be the assignment in I generated from S. We note that Inequality (4.5) ensures that each student is assigned in M to at most one project. Moreover, Inequalities (4.6) and (4.7) ensures that the capacity of each project and lecturer is not exceeded in M. Thus M is a matching. We will prove that Inequalities (5.11) - (5.18) ensures that M admits no blocking pair.

Suppose for a contradiction that there exists some acceptable pair (s_i, p_j) that forms a blocking pair for M, where l_k is the lecturer who offers p_j. This implies that (a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$, or (b) s_i is indifferent between p_j and $M(s_i)$. If (a) holds then $\sum_{p_{j'} \in S_{i,j}} x_{i,j'} = 0$; and thus $\theta_{i,j} = 1$. Now suppose (s_i, p_j) forms a blocking pair of type (1i), type (1ii) or type (1iii) for M. Following the same argument as in the proof of Lemma 4.5.1, we arrive at a conclusion which contradicts the feasibility of S. Hence M does not admit any blocking pair of types (1i), (1ii) or (1iii).

If (b) holds, since s_i is not assigned to p_j in M, we have that $\sum_{p_{j'} \in S^*_{i,j}} x_{i,j'} - x_{i,j} = 1$. Thus $\theta^*_{i,j} = 1$. Now, suppose (s_i, p_j) forms a blocking pair of type (2i) for M. Then each of p_j and l_k is undersubscribed in M. Thus $\sum_{i'=1}^{n_1} x_{i',j} < c_j$ and $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} < d_k$. This implies that the RHS of Inequality (4.8) and the RHS of Inequality (4.9) is strictly greater than 0. Moreover, since S is a feasible solution to J, $\alpha_j = \beta_k = 1$. Hence, the LHS of Inequality (5.14) is strictly greater than 2, a contradiction to the feasibility of S.

Next, suppose (s_i, p_j) forms a type (2ii) blocking pair for M. Then p_j is undersubscribed in M and as explained above, $\alpha_j = 1$. Also, l_k is full in M and this implies that the RHS

of Inequality (4.11) is strictly greater than 0. Since S is a feasible solution, we have that $\eta_k = 1$. Furthermore, either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$. In any of these cases, the RHS of Inequality (5.15) is strictly greater than 0. Thus $\mu_{i,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (5.16) is strictly greater than 3, a contradiction to the feasibility of S.

Finally, suppose (s_i, p_j) forms a type (2iii) blocking pair for M. Then p_j is full in M and thus the RHS of Inequality (4.14) is strictly greater than 0. Since S is a feasible solution, we have that $\gamma_j = 1$. In addition, l_k prefers s_i to a worst student in $M(p_j)$, which implies that the RHS of Inequality (5.17) is strictly greater than 0. Thus $\tau_{i,j,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (5.18) is strictly greater than 2, a contradiction to the feasibility of S. Hence M admits no blocking pair and it follows that M is a strongly stable matching in I. □

Lemma 5.4.2. *A strongly stable matching M in I corresponds to a feasible solution S to J.*

Proof. Let M be a strongly stable matching in I. First we set all the binary variables involved in J to 0. For each $(s_i, p_j) \in M$, we set $x_{i,j} = 1$. Since M is a matching, it is clear that Inequalities (4.5) - (4.7) is satisfied. For any acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, if s_i is unassigned in M or prefers p_j to $M(s_i)$, we set $\theta_{i,j} = 1$; and if s_i is indifferent between p_j and $M(s_i)$, we set $\theta^*_{i,j} = 1$. For any project $p_j \in \mathcal{P}$ such that p_j is undersubscribed in M, we set $\alpha_j = 1$ and thus Inequality (4.8) is satisfied. For any lecturer $l_k \in \mathcal{L}$ such that l_k is undersubscribed in M, we set $\beta_k = 1$ and thus Inequality (4.9) is satisfied.

Now, for Inequality (5.11) not to be satisfied, its LHS must be strictly greater than 2. This would only happen if there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$ and $\beta_k = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$, and each of p_j and l_k is undersubscribed in M. Thus (s_i, p_j) forms a type (1i) blocking pair for M, a contradiction to the strong stability of M. Hence, Inequality (5.11) is satisfied.

For any lecturer $l_k \in \mathcal{L}$ such that l_k is full in M, we set $\eta_k = 1$. Thus Inequality (4.11) is satisfied. Let (s_i, p_j) be an acceptable pair such that $p_j \in P_k$ and $(s_i, p_j) \notin M$. If $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them, we set $\delta_{i,k} = 1$. Thus Inequality (4.12) is satisfied. Suppose Inequality (5.12) is not satisfied. Then there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$, $\eta_k = 1$ and $\delta_{i,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$. In addition, p_j is undersubscribed in M, l_k is full in M and either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them. Thus (s_i, p_j) forms a type (1ii) blocking pair for M, a contradiction to the strong stability of M. Hence Inequality (5.12) is satisfied.

For any project $p_j \in \mathcal{P}$ such that p_j is full in M, we set $\gamma_j = 1$. Thus Inequality (4.14) is satisfied. Let l_k be the lecturer who offers p_j and let (s_i, p_j) be an acceptable pair. If l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, we set $\lambda_{i,j,k} = 1$. Thus Inequality (4.15) is satisfied. Suppose Inequality (5.13) is not satisfied. Then there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that $\theta_{i,j} = 1$, $\gamma_j = 1$ and $\lambda_{i,j,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$. In addition, p_j is full in M and l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them. Thus (s_i, p_j) forms a type (1iii) blocking pair for M, a contradiction to the strong stability of M. Hence, Inequality (5.13) is satisfied.

Now, for Inequality (5.14) not to be satisfied, its LHS must be strictly greater than 2. This would only happen if there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta^*_{i,j} = 1$, $\alpha_j = 1$ and $\beta_k = 1$. This implies that s_i is indifferent between p_j and $M(s_i)$, and each of p_j and l_k is undersubscribed in M. Thus (s_i, p_j) forms a type (2i) blocking pair for M, a contradiction to the strong stability of M. Hence, Inequality (5.14) is satisfied.

Let (s_i, p_j) be an acceptable pair such that $p_j \in P_k$ and $(s_i, p_j) \notin M$. If $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$, we set $\mu_{i,k} = 1$. Thus Inequality (5.15) is satisfied. Suppose Inequality (5.16) is not satisfied. Then there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta^*_{i,j} = 1$, $\alpha_j = 1$, $\eta_k = 1$ and $\mu_{i,k} = 1$. This implies that s_i is indifferent between p_j and $M(s_i)$. In addition, p_j is undersubscribed in M, l_k is full in M and either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$. Thus (s_i, p_j) forms a type (2ii) blocking pair for M, a contradiction to the strong stability of M. Hence Inequality (5.16) is satisfied.

Let l_k be the lecturer who offers p_j and let (s_i, p_j) be an acceptable pair. If l_k prefers s_i to a worst student in $M(p_j)$, we set $\tau_{i,j,k} = 1$. Thus Inequality (5.17) is satisfied. Suppose Inequality (5.18) is not satisfied. Then there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that $\theta^*_{i,j} = 1$, $\gamma_j = 1$ and $\tau_{i,j,k} = 1$. This implies that s_i is indifferent between p_j and $M(s_i)$. In addition, p_j is full in M and l_k prefers s_i to a worst student in $M(p_j)$. Thus (s_i, p_j) forms a type (2iii) blocking pair for M, a contradiction to the strong stability of M. Hence, Inequality (5.18) is satisfied.

Hence S, comprising the above assignments of values to the variables in $A \cup B \cup N \cup \Gamma \cup X \cup \Delta \cup M \cup \Lambda \cup T$, is a feasible solution to J. □

The following theorem is a consequence of Lemmas 5.4.1 and 5.4.2.

Theorem 5.4.3. *Let I be an instance of* SPA-ST *and let J be the IP model for I as described above. A feasible solution to J corresponds to a strongly stable matching in I. Conversely, a strongly stable matching in I corresponds to a feasible solution to J.*

5.5 Empirical evaluation

In this section, we evaluate an implementation of Algorithm SPA-ST-strong empirically. We implemented our algorithm in Python[16], and we performed our experiments on a system with dual Intel Xeon CPU E5-2640 processors with 64GB of RAM, running Ubuntu 17.10. For this experiment, our aim was to examine the proportion of SPA-ST instances that admit strongly stable matchings but no super-stable matchings.

5.5.1 Experimental setup and datasets

When generating random datasets, there are clearly several parameters that can be varied. However, we used the parameters and datasets from the previous chapter (see Section 4.6) in this chapter as well. We recall that we randomly generated a set of SPA-ST instances, involving n_1 students (which we will henceforth refer to as the size of the instance), $0.5n_1$ projects, $0.2n_1$ lecturers and $1.5n_1$ total project capacity which was randomly distributed amongst the projects such that each project has capacity at least 1. The capacity for each lecturer l_k was chosen uniformly at random to lie between the highest capacity of the projects offered by l_k and the sum of the capacities of the projects that l_k offers. In each set, we measured the proportion of instances that admit a strongly stable matching.

5.5.2 Correctness Testing

To test the correctness of our algorithm's implementation, we implemented (i) a brute-force algorithm that finds all the strongly stable matchings (if any) in a given SPA-ST instance, and (ii) our IP model for SPA-ST under strong stability using the Gurobi optimisation solver in Python. We then proceeded as follows.

(i) We randomly generated $100,000$ SPA-ST instances, each consisting of 15 students, 10 projects, 5 lecturers, and total project capacity of 18. The capacity for each project and lecturer was obtained as explained above. Also, the length of each student's preference list was fixed at 3, with tie density of 0.1 in the students' and lecturers' preference lists. For each instance, we verified consistency between the outcomes of our implementation of Algorithm SPA-ST-strong and our brute-force implementation in terms of the existence or otherwise of a strongly stable matching. The outcomes of the two implementations were consistent in 100% of the randomly-generated instances. Further, we observed that 45% of the instances admitted a strongly stable matching.

[16]https://github.com/sofiatolaosebikan/spa-st-strong

(ii) We randomly generated $10,000$ SPA-ST instances, each consisting of 100 students, 50 projects, 20 lecturers, and total project capacity of 120. Again, the capacity of each project and lecturer was as described above. Also, the length of each student's preference list was fixed at 10, with tie density $t_d \in \{0.05, 0.1\}$ in both the students' and lecturers' preference lists. Similar to above, we verified consistency between the outcomes of our implementation of Algorithm SPA-ST-strong and our implementation of the IP-based algorithm in terms of the existence or otherwise of a strongly stable matching. Again, the outcomes of the two implementations were consistent in 100% of the randomly-generated instances. Moreover, with $t_d = 0.1$, we observed that 4% of the instances admitted a strongly stable matching, and with $t_d = 0.05$, approximately 20% of the instances admitted a strongly stable matching.

5.5.3 Experimental results

We note that the setup for the next two experiments is similar to that of Experiments 1 and 2 in the super-stability setting. For completeness, we recall them in what follows.

Experiment 1

We examined the existence of strongly stable matchings as we varied the length of the students' preference lists for various values of x ($5 \leq x \leq 50$) in increments of 5. For each x, we increased the number of students n_1 ($100 \leq n_1 \leq 1000$) in increments of 100 while maintaining a constant ratio of projects, lecturers, project capacities and lecturer capacities as described above. Also, we set $t_{d_s} = t_{d_l} = 0.005$, and with each of these parameters, we randomly generated 1000 instances. With this setup, we observed that the proportion of instances that admitted a strongly stable matching is exactly the same as the proportion of instances that admitted a super-stable matching (see Figure 4.6). We give a potential justification for why we observed this consistency in what follows. As highlighted in the super-stability case, for a given dataset, 1 out of 5 students has a single tie of length 2 in her preference list, and this holds similarly for the lecturers. Thus, the ties in the preference lists are so sparse that any blocking pair for the super-stability type is bound to be a blocking pair for the strong stability type as well.

We also recorded the time taken for our algorithm's implementation to terminate (see Table 5.1). For an instance size of 1000 and preference list length 50, the implementation terminates in less than 1.5 seconds, which is three times slower than what we recorded in the super-stability case. This is not surprising since the complexity of the super-stability algorithm is $O(L)$ time, while that of the strong stability algorithm is $O(m^2)$ time, where L is

the total length of all the preference lists and m is the total length of the students' preference lists.

Table 5.1: Time (in seconds) for our algorithm's implementation to terminate averaged over 1000 for each instance size, with the length of the students' preference lists fixed at 50.

n_1	100	200	300	400	500	600	700	800	900	1000
Time	0.042	0.106	0.187	0.286	0.402	0.543	0.703	0.890	1.094	1.330

Experiment 2

Here, we investigated the existence of strongly stable matchings as we varied the tie density in the students' preference lists t_{d_s} $(0 \leq t_{d_s} \leq 0.05)$ and the tie density in the lecturers' preference lists t_{d_l} $(0 \leq t_{d_l} \leq 0.05)$, both in increments of 0.005. For each pair of tie densities in $t_{d_s} \times t_{d_l}$, we increased the number of students n_1 $(100 \leq n_1 \leq 1000)$ in increments of 100, and we maintained the same ratio of projects, lecturers, project capacities and lecturer capacities as in Experiment 1. Also, we fixed the length of each student's preference list at 50. With each of these parameters, we randomly-generated 1000 SPA-ST instances.

Again, the results we observed with this setup is exactly the same as those observed in the super-stability setting (see Figure 4.7): i.e., with $n_1 = 1000, t_{d_s} = 0$ and $t_{d_l} = 0.05$, 74% of the randomly-generated instances admitted a strongly stable matching; however, with $n_1 = 1000, t_{d_s} = 0.005$ and $t_{d_l} = 0$, only about 5% of the 1000 randomly-generated instances admitted a strongly stable matching. Considering that when ties are on one side of the preference lists only, the definition of super-stability and strong stability coincides, it is not surprising that the experimental results also agree with this. On the other hand, when we varied the tie density on both sides, there was little (0.1% increase) to no difference between the proportion of instances that admitted strongly stable matchings and those that admitted super-stable matchings. Perhaps, a small instance size, a small preference list length, and an increase in tie density in both the students' and lecturers' preference lists will lead to an increase in the proportion of instances that admit a strongly stable matching but no super-stable matching? We explore this further in the next experiment.

Experiment 3

Considering the consistency of the results obtained in Experiments 1 and 2 with those observed in the super-stability experiments, we decided to compare the proportion of instances that admit super-stable and/or strongly stable matchings for various values of n_1 $(10 \leq n_1 \leq 50)$ in increments of 10, with the length of each student's preference list fixed at 5. The

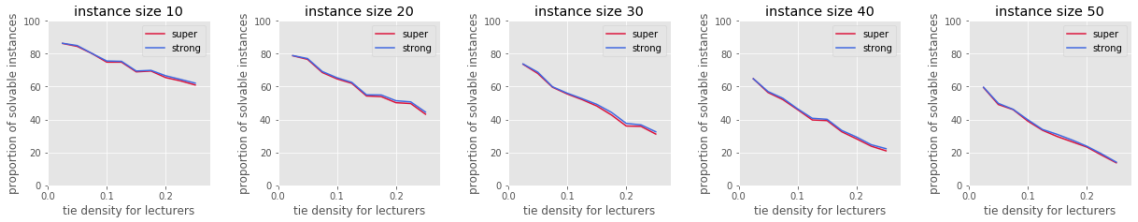
ratio of projects, lecturers, project capacities and lecturer capacities was obtained as in the previous experiments. We varied the tie density in the students' preference lists t_{d_s} $(0.025 \leq t_{d_s} \leq 0.15)$ and the tie density in the lecturers' preference lists t_{d_l} $(0.025 \leq t_{d_l} \leq 0.25)$, both in increments of 0.025. With each of these parameters, we randomly-generated 1000 SPA-ST instances. The result displayed in Figure 5.13 shows some slight increase in the proportion of instances that admit a strongly stable matching but no super-stable matching.

5.6 Conclusion and open problems

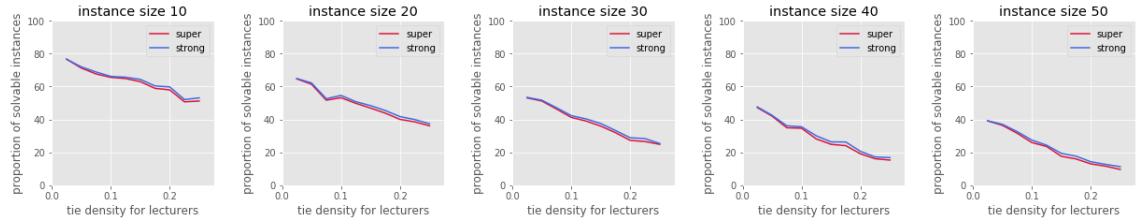
In this chapter, we have described a polynomial-time algorithm to find a strongly stable matching or to report that no such matching exists, given an instance of SPA-ST. We also established that our algorithm produces a student-optimal strongly stable matching for solvable instances, i.e., each student who is assigned in a solution output by our algorithm is assigned to at least as good a project as she could obtain in any strongly stable matching. We leave open the formulation of a lecturer-oriented counterpart to our algorithm.

We also carried out an empirical evaluation of our algorithm's implementation. The aim of our experiment was to examine the likelihood of the existence of strongly stable matchings compared to super-stable matchings in randomly-generated instances of SPA-ST. We observed that as we increased the size of the instance and the length of the preference list, and as we varied the tie density in both the students' and lecturers' preference lists, the results that we obtained from our strong stability experiments are consistent with what we observed in our super-stability experiments, with respect to the instances that we generated randomly. However, for instances of size between 10 and 50, we observed a slight increase in the proportion of instances that admitted strongly stable matchings but no super-stable matching. Further evaluation of our implementation could investigate how other parameters (e.g., the popularity of some projects, or the position of the ties in the preference lists) affect the existence of a strongly stable matching when no super-stable matching exists.

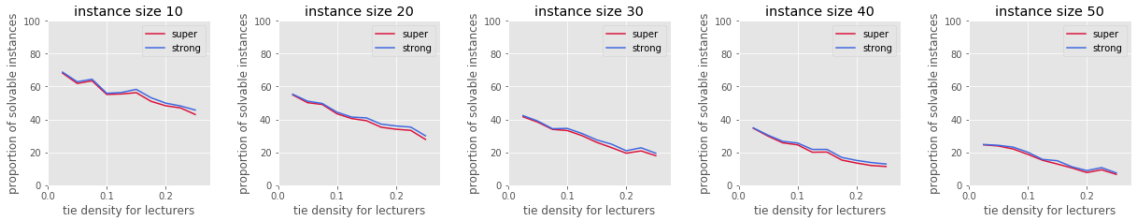
It would also be interesting to explore some of the open problems we suggested in Chapter 4, which include: (i) formalising the results on the probability of a strongly stable matching existing, given an arbitrary instance of SPA-ST (see [102] for some results regarding this for the Stable Roommates problem); and (ii) characterising the strongly stable matchings in an instance of SPA-ST with respect to a dominance relation (see [82] for results regarding this for SMT).



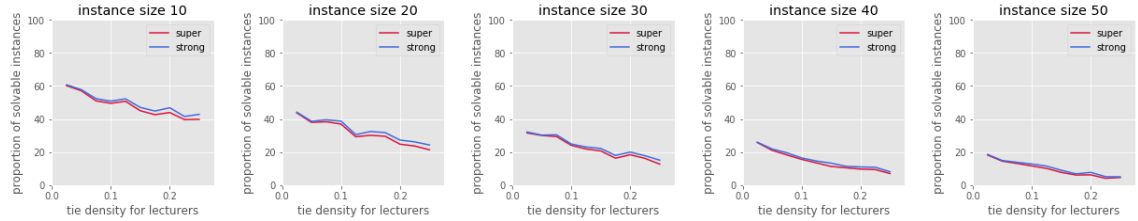
(a) $t_{d_s} = 0.025$.



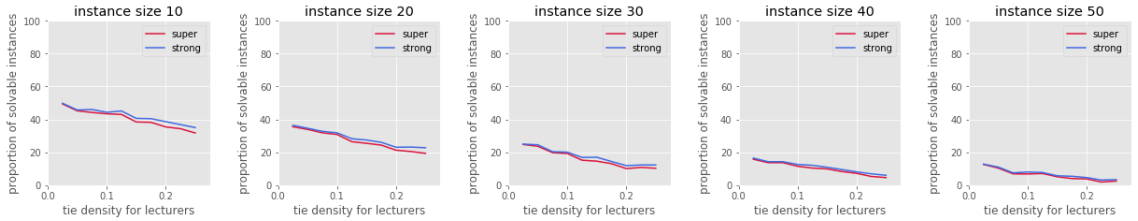
(b) $t_{d_s} = 0.05$.



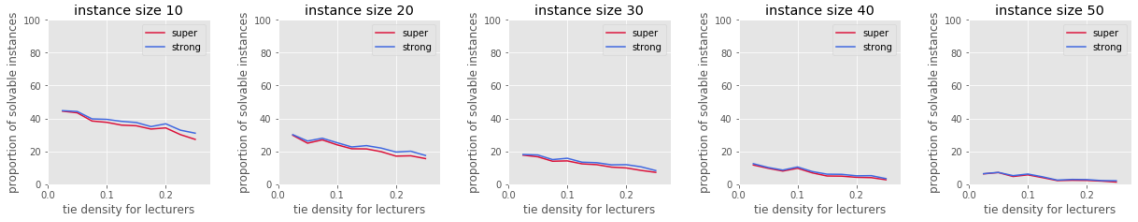
(c) $t_{d_s} = 0.075$.



(d) $t_{d_s} = 0.1$.



(e) $t_{d_s} = 0.125$.



(f) $t_{d_s} = 0.150$.

Figure 5.13: Proportion of instances that admit a strongly stable and/or super-stable matching as the size of the instance increases, with the length of each student's preference list fixed at 5, and with varying tie density in the lecturers' preference lists.

Chapter 6

Algorithmic and Experimental Results for SPA-P

6.1 Introduction

In Section 2.3.4, we considered the SPA-P [87] model, which is a variant of SPA where students have preferences over the available projects that they find acceptable while lecturers have preferences over their proposed projects. As the reader may recall, stable matchings for an instance of SPA-P can have different sizes, and the problem of finding a maximum size stable matching (MAX-SPA-P) is NP-hard, under certain restrictions [87]. Moreover, the best known approximation algorithm for MAX-SPA-P has a performance guarantee of $\frac{3}{2}$ [61].

Our contribution in this chapter is to present new algorithmic and experimental results for SPA-P. On the algorithmic side, first we explore the complexity of MAX-SPA-P under the following restrictions: if (i) the instance involves r lecturers (denoted MAX-SPA-P-Lr), where r is a constant; and (ii) each preference list is of length at most 3 (denoted $(3,3)$-MAX-SPA-P). For the first restriction, if there is only one lecturer involved, we show that MAX-SPA-P is polynomial-time solvable. In contrast to this, if there are two lecturers involved, we show that the problem remains NP-hard and is not approximable within some constant $c > 1$ unless P = NP. For the second restriction, we show that MAX-SPA-P remains NP-hard. We then move on to describe an IP model to enable MAX-SPA-P to be solved optimally, in the general case where there are no restrictions on the problem instance.

On the experimental side, we present results arising from an empirical evaluation that investigates how the solutions produced by the existing approximation algorithms for MAX-SPA-P [61, 87] compare to optimal solutions obtained from our IP model, with respect to the size of the stable matchings constructed, on instances that are both randomly-generated and derived from real datasets. These real datasets are based on actual student preference data and man-

Students' preferences	Lecturers' preferences
s_1: p_3 p_2 p_1	l_1: p_2 p_1
s_2: p_1 p_2	l_2: p_3
s_3: p_3	
	Project capacities: $c_1 = c_2 = c_3 = 1$
	Lecturer capacities: $d_1 = 2,\ d_2 = 1$

Figure 6.1: An instance I_1 of SPA-P.

ufactured lecturer preference data from previous runs of student-project allocation processes at the School of Computing Science, University of Glasgow. We also present results showing the time taken by the IP model to solve the problem instances optimally. Our main finding is that the $\frac{3}{2}$-approximation algorithm finds stable matchings that are very close to having maximum cardinality.

This chapter is organised as follows. We give a formal definition of stability in the context of SPA-P in Section 6.2. We describe a polynomial-time algorithm for MAX-SPA-P-L1 in Section 6.3.1, and we give an inapproximability result for MAX-SPA-P-L2 in Section 6.3.2. In Section 6.4 we show that $(3, 3)$-MAX-SPA-P is NP-hard. In Section 6.5, we describe our IP model for MAX-SPA-P. We present our empirical evaluation in Section 6.6, along with some discussions regarding results obtained from the experiments. Finally, in Section 6.7 we give some conclusions and open problems.

6.2 Preliminary definitions

Formally, an instance I of SPA-P involves a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of *students*, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of *projects* and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of *lecturers*. In contrast to SPA-S (see Section 2.3.2), each lecturer $l_k \in \mathcal{L}$ ranks the projects in P_k in strict order of preference instead of ranking the students in $\mathcal{S}$. An example instance of SPA-P is illustrated in Figure 6.1, which involves the set $\mathcal{S} = \{s_1, s_2, s_3\}$ of students, the set $\mathcal{P} = \{p_1, p_2, p_3\}$ of projects and the set $\mathcal{L} = \{l_1, l_2\}$ of lecturers, with $P_1 = \{p_1, p_2\}$ and $P_2 = \{p_3\}$.

All the notation and terminology defined for SPA-S also holds for SPA-P, except that of stability, which we define as follows.

Definition 6.2.1 ([87]). *Let I be an instance of* SPA-P *and let M be a matching in I. An acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ is a* blocking pair *for M if (i), (ii) and (iii) holds as follows:*

(i) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$;

(ii) p_j is undersubscribed in M;

(iii) 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 in* M*,*
(c) $s_i \notin M(l_k)$ *and* l_k *prefers* p_j *to his worst non-empty project.*

If a pair (s_i, p_j) forms a blocking pair for M, we may also say that (s_i, p_j) *blocks* M. For ease of exposition, throughout this chapter, we will refer to any such pair as a blocking pair of type (a), type (b) or type (c). With respect to the SPA-P instance given in Figure 6.1, $M_1 = \{(s_1, p_3), (s_2, p_1)\}$ is clearly a matching. It is obvious that each of students s_1 and s_2 is assigned to her first ranked project in M_1. Although s_3 is unassigned in M_1, the lecturer offering p_3 (the only project that s_3 finds acceptable) is assumed to be indifferent among those students who find p_3 acceptable. Also p_3 is full in M_1. Thus, we say that M_1 admits no blocking pair.

Another way in which a matching could be undermined is through a group of students acting together, forming a *coalition*. Given a matching M, a *coalition* is a set of students $C = \langle s_{i_0}, \ldots, s_{i_{r-1}} \rangle$, for some $r \geq 2$ such that each student s_{i_j} $(0 \leq j \leq r-1)$ is assigned in M and prefers $M(s_{i_{j+1}})$ to $M(s_{i_j})$, where addition is performed modulo r. Again, considering the SPA-P instance given in Figure 6.1, the matching $M_2 = \{(s_1, p_1), (s_2, p_2), (s_3, p_3)\}$ admits a coalition $\{s_1, s_2\}$, as students s_1 and s_2 would rather permute their assigned projects in M_2 so as to be better off. We note that the number of students assigned to each project and lecturer involved in any such swap remains the same after such a permutation. Moreover, the lecturers involved would have no incentive to prevent the switch from occurring since they are assumed to be indifferent between the students assigned to the projects they are offering. If a matching admits no coalition, we define such matching to be *coalition-free*.

A matching M in I is said to be *stable* if M admits no blocking pair and is coalition-free. Some intuition for the stable matching definition is given in [87]. The reader can easily verify that each of the matchings $M_1 = \{(s_1, p_3), (s_2, p_1)\}$ and $M_3 = \{(s_1, p_2), (s_2, p_1), (s_3, p_3)\}$ is stable in the SPA-P instance shown in Figure 6.1.

6.3 SPA-P with constant number of lecturers

As mentioned in Section 2.3.4, MAX-SPA-P is not approximable within δ_1, for some $\delta_1 > 1$ unless P = NP [87]. Moreover, the result holds even if each project and lecturer has capacity 1, and each preference list is of length at most 4. We wished to answer the following question: what is the complexity of finding a maximum size stable matching if a constant number of lecturers are involved in an arbitrary SPA-P instance I? As it turns out, the problem

is polynomial-time solvable if there is only one lecturer involved in I and hard to approximate if there are two lecturers involved in I. We present the proof of these results in the next two sections.

We denote by SPA-P-Lr an instance of SPA-P involving r lecturers, where r is a constant, and we denote by MAX-SPA-P-Lr the problem of finding a maximum size stable matching given an instance of SPA-P-Lr.

6.3.1 A polynomial-time algorithm for MAX-SPA-P-L1

Let I be an instance of SPA-P-L1 where l is the lecturer involved in I; assume all notation and terminology from the general SPA-P case. Given an acceptable pair (s_i, p_j), we define $\mathrm{rank}(s_i, p_j)$, the *rank* of p_j on s_i's preference list, to be $r + 1$, where r is the number of projects that s_i prefers to p_j. Let R be the maximum rank of a project in any student's preference list. We define the *profile* $\rho(M)$ of a matching M in I as an R-tuple $(x_1, x_2, \ldots, x_R)$, such that for each r $(1 \leq r \leq R)$, x_r is the number of students, say s_i, assigned in M to a project, say p_j, such that $\mathrm{rank}(s_i, p_j) = r$.

For example, the matching $M_3 = \{(s_1, p_2), (s_2, p_1), (s_3, p_3)\}$ in the instance I_1 given in Figure 6.1 has the profile $\rho(M_3) = (2, 1, 0)$, since two students (i.e., s_2 and s_3) are assigned to their first choice project in M_3, one student (i.e., s_1) is assigned to her second choice project in M_3 and no student is assigned to her third choice project in M_3. A *greedy maximum matching* in I is a matching of maximum cardinality that has *lexicographically maximum profile*, i.e., the maximum number of students are assigned to their first choice and subject to this, the maximum number of students are assigned to their second choice, and so on. Clearly, M_3 is a greedy maximum matching in I_1.

In what follows, we give an informal description of our polynomial-time algorithm for finding a maximum size stable matching given an instance I of SPA-P-L1, which we denote by Algorithm MAX-SPA-P-L1. Our algorithm begins by first constructing a greedy maximum matching M in I using the polynomial-time algorithm given in [76]. As we will prove later, at this point in the algorithm, M is coalition-free and does not admit a blocking pair of type (a) or type (b).

If M admits a type (c) blocking pair then the first while loop of the algorithm is executed. In this loop, we identify a student, say s_i, involved in a type (c) blocking pair, say (s_i, p_j), such that (s_i, p_j) is the best blocking pair for s_i. Next, we identify l's worst non-empty project in M, say p_q, we identify any student assigned to p_q in M, say s_r, and we remove the pair (s_r, p_q) from M. Finally, we promote s_i by adding the pair (s_i, p_j) to M. As we will prove later, M does not admit a blocking pair at the termination of this loop.

If M admits a coalition $C = \langle s_{i_0}, s_{i_1}, \ldots, s_{i_{r-1}} \rangle$ at this point, for some $r \geq 2$, then we *satisfy* the coalition in the second while loop, by letting the students in C swap their projects, i.e., for each j ($0 \leq j \leq r-1$), s_{i_j} moves project from $M(s_{i_j})$ to $M(s_{i_{j+1}})$, where addition is taken modulo r. At the termination of this loop, M is output as a maximum size stable matching in I. We describe our algorithm in pseudocode form in Algorithm 3, and we provide a proof of correctness in Theorem 6.3.1.

Algorithm 3 Algorithm MAX-SPA-P-L1

Input: SPA-P-L1 instance I

Output: a maximum size stable matching in I

1: $M \leftarrow$ a greedy maximum matching in I
2: **while** there exists a type (c) blocking pair **do**
3: $\quad s_i \leftarrow$ a student involved in such blocking pair
4: $\quad (s_i, p_j) \leftarrow$ best blocking pair for s_i
5: $\quad p_q \leftarrow$ worst non-empty project offered by l
6: $\quad s_r \leftarrow$ any student assigned to p_q in M
7: $\quad M \leftarrow M \setminus \{(s_r, p_q)\}$
8: $\quad M \leftarrow M \cup \{(s_i, p_j)\}$ /* promote s_i to p_j */
9: **while** M admits a coalition C **do**
10: $\quad$ satisfy C
11: **return** M

Theorem 6.3.1. *Given an instance I of* SPA-P-L1*, Algorithm* MAX-SPA-P-L1 *terminates with a maximum size stable matching in $O(n_1^2 Rm)$ time, where n_1 is the number of students, R is the maximum rank of a project in any student's preference list and m is the total length of the students' preference lists.*

Proof. Let E be an arbitrary execution of Algorithm MAX-SPA-P-L1, and let M be the matching at the termination of E. Let M_0 be the greedy maximum matching in I found on line 1 during E; clearly M_0 is of maximum cardinality. In what follows, we show that just before the first while loop was initiated during E, M_0 admits no blocking pair of type (a) or type (b), and is coalition-free.

Suppose that (s_i, p_j) is a blocking pair of type (a), i.e., $s_i \in M_0(l)$ and s_i prefers p_j to $M_0(s_i)$; p_j is undersubscribed in M_0 and l prefers p_j to $M_0(s_i)$. Let $M_0(s_i) = p_{j'}$. Now, let $M_0' = (M_0 \cup \{(s_i, p_j)\}) \setminus \{(s_i, p_{j'})\}$. Then $|M_0'| = |M_0|$ and $\rho(M_0') > \rho(M_0)$ (according to lexicographic order), contradicting the fact that M_0 is a greedy maximum matching. Suppose that (s_i, p_j) is a blocking pair of type (b), i.e., s_i is unassigned in M_0, and each of p_j and l is undersubscribed in M_0. Now, p_j undersubscribed and l undersubscribed implies that $M_0 \cup \{(s_i, p_j)\}$ is a matching in I, contradicting the maximality of M_0. Next, we show that M_0 is coalition-free. Suppose M_0 admits a coalition of students $C = \langle s_{i_0}, s_{i_1}, \ldots, s_{i_{r-1}} \rangle$ for

some $r \geq 2$. Suppose M_0' is the matching that results from satisfying C. Then $|M_0'| = |M_0|$ and $\rho(M_0') > \rho(M_0)$, since each student in C will prefer her assigned project in M_0' to that in M_0. This is a contradiction to the fact that M_0 has a lexicographically maximum profile.

We note that the first while loop is bound to terminate since (i) the number of students involved in I is finite, and (ii) for each student involved in a type (c) blocking pair that gets promoted, the lecturer obtains one more student assigned to a project that is better than her previous worst non-empty project. Let M_1 be the matching at the termination of the first loop. Clearly, M_1 cannot admit a type (c) blocking pair. Moreover, M_1 cannot admit a type (b) blocking pair either, since for each student that becomes unassigned from a project within the loop, another student becomes promoted to a project in the same while loop iteration. Thus the cardinality of M_1 remains unchanged, and the proof follows as in the previous paragraph.

Next, we show that M_1 cannot admit a blocking pair of type (a). Suppose (s_i, p_j) is a blocking pair of type (a); this implies that $s_i \in M_1(l)$ and l prefers p_j to $M_1(s_i)$. Let $M_1(s_i) = p_k$. First, we note that (s_i, p_k) must have been added to M_1 as a result of satisfying a type (c) blocking pair within the first while loop. For, suppose otherwise, $(s_i, p_k) \in M_0$. We have already established that M_0 admits no blocking pair of type (a). Hence p_j must have been full in M_0 and became undersubscribed during the first while loop. This is only possible if l's worst non-empty project in M_1 is p_j or better. Hence (s_i, p_j) cannot be a type (a) blocking pair for M_1 in this case. Thus (s_i, p_k) must have been added to M_1 within the first while loop.

Moreover, when (s_i, p_k) was added to the matching during the first while loop, p_j must be full, since we satisfy the best blocking pair for each student within a loop iteration. However, for (s_i, p_j) to form a type (a) blocking pair for M_1, p_j must be undersubscribed in M_1. Now, the only way p_j can end up becoming undersubscribed in M_1, having been full in M_0, is if p_j was l's worst non-empty project at a point where a type (c) blocking pair was identified during the first while loop. Thus in M_1, either p_j is l's worst non-empty project, or l's worst non-empty project is better than p_j. Hence (s_i, p_j) cannot be a type (a) blocking pair for M_1.

We have proved in the last two paragraphs that the first while loop terminates with a maximum matching M_1 in I, which admits no blocking pair. The second while loop is also bound to terminate, since (i) each student has a finite number of projects in her preference list, and (ii) for any coalition C of students that exists, each student involved in C obtains a better project after the swap takes place. Clearly M cannot admit any new blocking pair at the end of the second while loop. Hence, at the termination of the algorithm, M is a maximum size stable matching in I.

Analysis of algorithm Algorithm MAX-SPA-P-L1 finds a maximum size stable matching in I in three phases. In the first phase, the algorithm finds a greedy maximum matching

M in I. This can be obtained in $O(n_1^2 Rm)$ time using the GREEDY-MAX-SPA algorithm described in [76], and this complexity dominates the overall runtime of the algorithm. In the second phase, the algorithm eliminates potential type (c) blocking pairs in M. This phase is bounded by the length of the lecturer's preference list, since for any type (c) blocking pair that is eliminated, the lecturer obtains an additional student assigned to a project that she prefers to her previous worst non-empty project. Thus, the complexity of this phase is $O(n_2)$ time, where n_2 is the number of projects.

In the third phase, the algorithm eliminates potential coalitions that might have been introduced in the second phase. We remark that coalitions in M corresponds to cycles in the *envy graph* $G(M)$, which contains a vertex for each student who is assigned in M and a directed edge from student s_{i_j} to s_{i_q} if s_{i_j} prefers $M(s_{i_q})$ to $M(s_{i_j})$. Clearly M is coalition-free if and only if $G(M)$ is acyclic (see page 148 for further discussions on this). Further, we remark that cyclic coalitions can be eliminated in $O(m)$ time (see [83, p.308] for a detailed explanation on how to achieve this). Hence, the overall complexity of Algorithm MAX-SPA-P-L1 is $O(n_1^2 Rm)$. □

6.3.2 Inapproximability of MAX-SPA-P-L2

Let I be an instance of SPA-P-L2, and let $s^+(I)$ denote the maximum size of a stable matching in I. Define MAX-SPA-P-L2 to be the problem of computing $s^+(I)$. In this section, we show that MAX-SPA-P-L2 is NP-hard, and moreover that there exists a constant $\delta_2 > 1$ such that it is NP-hard to approximate MAX-SPA-P-L2 within δ_2. The result holds even if each project has capacity 1.

We prove this result using a reduction from a problem relating to matchings in graphs. A matching M in a graph G is said to be *maximal* if no proper superset of M is a matching in G. Let $\beta_1^-(G)$ denote the minimum size of a maximal matching in G. Define MIN-MM to be the problem of computing $\beta_1^-(G)$, given a graph G. The following result regarding the inapproximability of MIN-MM is proved in [45].

Theorem 6.3.2 ([45]). *Let $G = (V, E)$ be an instance of* MIN-MM*, where $m = |E|$. Then there exist constants $c_0 > 0$ and $\delta_0 > 1$ such that it is* NP*-hard to distinguish between the cases $\beta_1^-(G) \leq c_0 m$ and $\beta_1^-(G) > \delta_0 c_0 m$. Hence it is* NP*-hard to approximate* MIN-MM *within δ_0. The result holds even for subdivision graphs[1] of cubic graphs.*

We will use Theorem 6.3.2 together with the notion of a *gap-preserving reduction* [121], which may be defined as follows.

[1]Given a graph G, the *subdivision graph* of G, denoted by $S(G)$, is obtained by subdividing each edge $\{u, w\}$ of G in order to obtain two edges $\{u, v\}$ and $\{v, w\}$ of $S(G)$, where v is a new vertex.

Definition 6.3.3 ([121]). *Let $\prod_1$ be a minimisation problem and let $\prod_2$ be a maximisation problem. Denote by $OPT_i(x)$ the optimal measure over all feasible solutions for a given instance x of $\prod_i$ ($i \in \{1,2\}$). Let $\alpha \geq 1$ be some constant and let g_1 be a function that maps an instance x of $\prod_1$ to a positive rational number. Then a* gap-preserving reduction *from $\prod_1$ to $\prod_2$ is a tuple $\langle f, \beta, g_2 \rangle$ such that:*

(a) f maps an instance x of $\prod_1$ to an instance $f(x)$ of $\prod_2$ in polynomial time;
(b) $\beta \geq 1$ is a constant;
(c) g_2 maps an instance $f(x)$ of $\prod_2$ to a positive rational number;
(d) for any instance x of $\prod_1$:

(i) if $OPT_1(x) \leq g_1(x)$ then $OPT_2(f(x)) \geq g_2(f(x))$;
(ii) if $OPT_1(x) > \alpha g_1(x)$ then $OPT_2(f(x)) < (1/\beta) g_2(f(x))$.

The following proposition is an immediate consequence of Definition 6.3.3.

Proposition 6.3.4 ([121]). *Let $\prod_1$ be a minimisation problem and let $\prod_2$ be a maximisation problem, and suppose that there is a gap-preserving reduction from $\prod_1$ to $\prod_2$. Assuming the notation of Definition 6.3.3, suppose further that it is* NP*-hard to distinguish between instances x of $\prod_1$ such that $OPT_1(x) \leq g_1(x)$ and $OPT_1(x) > \alpha g_1(x)$. Then it is* NP*-hard to distinguish between instances $f(x)$ of $\prod_2$ such that $OPT_2(f(x)) \geq g_2(f(x))$ and $OPT_2(f(x)) < (1/\beta) g_2(f(x))$. Hence it is* NP*-hard to approximate $\prod_2$ within β.*

We use Proposition 6.3.4, together with Theorem 6.3.2, to prove the NP-hardness and inapproximability result for MAX-SPA-P-L2.

Theorem 6.3.5. MAX-SPA-P-L2 *is* NP*-hard. Moreover, it is* NP*-hard to approximate* MAX-SPA-P-L2 *within δ_2, for some $\delta_2 > 1$. The result holds even if each project has capacity* 1.

Proof. Let G (a subdivision graph of some cubic graph G') be an instance of MIN-MM. Then G is a bipartite graph, where $G = (U, W, E)$ and, without loss of generality, suppose each vertex in U has degree 2 and each vertex in W has degree 3. Let $U = \{u_1, u_2, \ldots, u_{n_1}\}$ and $W = \{w_1, w_2, \ldots, w_{n_2}\}$. For each $u_i \in U$, let w_{j_i} and w_{k_i} be the vertices adjacent to u_i in G, where $j_i < k_i$.

We construct an instance I of SPA-P-L2 as follows. Let $S \cup U^1 \cup U^2$ be the set of students, where $S = \{s_1, s_2, \ldots, s_{n_2}\}$, $U^1 = \{u_i^1 : u_i \in U\}$ and $U^2 = \{u_i^2 : u_i \in U\}$. Let $P \cup Q \cup R \cup T$ be the set of projects, where $P = \{p_1, p_2, \ldots, p_{n_2}\}$, $Q = \{q_1, q_2, \ldots, q_{n_1}\}$, $R = \{r_1, r_2, \ldots, r_{n_1}\}$ and $T = \{t_1, t_2, \ldots, t_{n_2}\}$, and each project has capacity 1. Let $\{x, y\}$ be the set of lecturers, where x has capacity n_2 and offers $P \cup T$, while y has capacity

$2n_1$ and offers $Q \cup R$. The preference lists in I are shown in Figure 6.2. We claim that $s^+(I) = 2n_1 + n_2 - \beta_1^-(G)$.

Students' preferences			Lecturers' preferences	
u_i^1:	$q_i\ p_{j_i}\ p_{k_i}\ r_i$	$(1 \leq i \leq n_1)$	x:	$p_1 \cdots p_{n_2}\ t_1 \ldots t_{n_2}$
u_i^2:	$q_i\ p_{k_i}\ p_{j_i}$	$(1 \leq i \leq n_1)$	y:	$q_1 \ldots q_{n_1}\ r_1 \ldots r_{n_1}$
s_j:	t_j	$(1 \leq j \leq n_2)$		

Figure 6.2: Preference lists for constructed instance of SPA-P-L2

Suppose firstly that G has a maximal matching M, where $|M| = \beta_1^-(G)$. We construct a matching M' in I as follows. For each edge $\{u_i, w_j\} \in M$, if $j = j_i$, add (u_i^1, p_{j_i}) and (u_i^2, q_i) to M'; whilst if $j = k_i$, add (u_i^1, q_i) and (u_i^2, p_{k_i}) to M'. For each $u_i \in U$, if u_i is unassigned in M, add (u_i^1, r_i) and (u_i^2, q_i) to M'. For each $w_j \in W$, if w_j is unassigned in M, add (s_j, t_j) to M' for $1 \leq j \leq n_2 - |M|$. Clearly, M' is a matching in I and $|M'| = 2|M| + 2(n_1 - |M|) + (n_2 - |M|)$.

No project in Q can be involved in a blocking pair of M', since each member of Q is full in M'. Similarly, no project in T can be involved in a blocking pair of M', since lecturer x is full in M', and since x prefers her worst non-empty project in T to all the undersubscribed projects in T. Hence, no student in S can be involved in a blocking pair of M'. Similarly, no $u_i^2 \in U^2$ can be involved in a blocking pair of M', since u_i^2 is assigned in M' to either her first or second choice project. Also no project in R can be involved in a blocking pair of M', since each member of U^1 is assigned in M'. Now suppose the pair (u_i^1, p_j) blocks M'. Then $(u_i^1, r_i) \in M'$, and p_j is undersubscribed in M'. Thus no edge in M is incident to u_i or w_j in G. Hence $M \cup \{\{u_i, w_j\}\}$ is a matching in G, contradicting the maximality of M. Thus, M' admits no blocking pair.

We next verify that M' is coalition-free. Clearly, no student in S can be involved in a coalition, since any such student who is assigned in M has her first choice project. Also, no student who is assigned in M' to a project in Q can be in a coalition, since every such student is assigned to her first choice project. As a consequence, no student who is assigned in M' to her second choice project can be in a coalition, since each such student prefers only a project in Q. Finally, no student in U^1 who is assigned to a project in R can be in a coalition, since no assigned student prefers a project in R to her project in M'. Hence M' is stable. Also $s^+(I) \geq 2n_1 + n_2 - \beta_1^-(G)$.

Conversely, suppose that M' is a stable matching in I such that $|M'| = s^+(I)$. First, we claim that each project in Q is full in M'. For suppose not, then there exists i $(1 \leq i \leq n_1)$ such that $(u_i^1, q_i) \notin M'$ and $(u_i^2, q_i) \notin M'$; and thus (u_i^1, q_i) blocks M', a contradiction.

Hence

$$M = \{\{u_i, w_j\} \in E : (u_i^1, p_j) \in M' \vee (u_i^2, p_j) \in M'\}$$

is a matching in G. Suppose that M is not maximal. Then there is some edge $\{u_i, w_j\} \in E$ such that no edge of M is incident to u_i or w_j. Thus by the construction of M, either (i) $(u_i^1, r_i) \in M'$ or u_i^1 is unassigned in M', or (ii) u_i^2 is unassigned in M'. Also p_j is undersubscribed in M', and x cannot fill all n_2 places with students who are assigned to projects in P. It follows that (u_i^1, p_j) blocks M' in case (i), whilst (u_i^2, p_j) blocks M' in case (ii), a contradiction to the stability of M'. Hence M is indeed maximal in G.

For each $\{u_i, w_j\} \in M$, it follows that $(u_i^z, p_j) \in M'$ for some z $(1 \leq z \leq 2)$; and thus, $(u_i^{3-z}, q_i) \in M'$. Hence at most $n_1 - |M|$ projects in R are full in M'. Moreover, since lecturer x has capacity n_2 and since $|M|$ projects in P are full in M', then at most $n_2 - |M|$ projects in T are full in M'. Also, recall from the previous paragraph that each project in Q is full in M'. Hence $|M'| \leq |M| + n_1 + (n_1 - |M|) + (n_2 - M) = 2n_1 + n_2 - |M|$, and thus $s^+(I) \leq 2n_1 + n_2 - \beta_1^-(G)$.

We have that $s^+(I) = 2n_1 + n_2 - \beta_1^-(G)$, and hence $s^+(I) + \beta_1^-(G) = 2n_1 + n_2$. Now $2n_1 = 3n_2$, as G is the subdivision graph of the cubic graph G'. Also $m = 2n_1$, where m is the number of edges in G. Let n be the number of students in I. Then $n = 2n_1 + n_2$.

Let c_0 and δ_0 be the constants given by Theorem 6.3.2, such that it is NP-hard to distinguish between the cases $\beta_1^-(G) \leq c_0 m$ and $\beta_1^-(G) > \delta_0 c_0 m$. Hence if $\beta_1^-(G) \leq c_0 m$ then $s^+(I) \geq c_2 n$, whilst if $\beta_1^-(G) > \delta_0 c_0 m$ then $s^+(I) < (1/\delta_2) c_2 n$, where $c_2 = \frac{4-3c_0}{4}$ and $\delta_2 = \frac{4-3c_0}{4-3\delta_0 c_0}$. The result then follows by Theorem 6.3.2 and Proposition 6.3.4. □

6.4 NP-hardness of $(3,3)$-MAX-SPA-P

Recall that $(3,3)$-MAX-SPA-P is the restriction of MAX-SPA-P in which each preference list is of length at most 3. In this section, we show that $(3,3)$-MAX-SPA-P is NP-hard. The result holds even if each project and lecturer has capacity 1. To achieve this, we will show that $(3,3)$-COM-SPA-P is NP-complete — this is the problem of deciding, given an instance of SPA-P in which all the preference lists are of length at most 3, whether a *complete* stable matching (i.e., a stable matching in which every student is assigned) exists. Clearly, the NP-completeness of $(3,3)$-COM-SPA-P implies the NP-hardness of $(3,3)$-MAX-SPA-P.

In order to prove this result, we use a reduction from a restricted version of SAT, which we define as follows. Let $(2,2)$-E3-SAT denote the problem of deciding, given a Boolean formula B in CNF in which each clause contains exactly 3 literals and, for each variable v_i, each of literals v_i and $\overline{v_i}$ appears exactly twice in B, whether B is satisfiable. Berman *et al.* [22] showed that $(2,2)$-E3-SAT is NP-complete.

Students' preferences			Lecturers' preferences		
x_{4i}:	$y_{4i}\ c(x_{4i})\ y_{4i+1}$	$(0 \le i \le n-1)$	l_j^s:	$p_j^s\ c_j^s\ q_j^s$	$(1 \le j \le m \wedge 1 \le s \le 3)$
x_{4i+1}:	$y_{4i+1}\ c(x_{4i+1})\ y_{4i+2}$	$(0 \le i \le n-1)$	y'_r:	y_r	$(0 \le r \le 4n-1)$
x_{4i+2}:	$y_{4i+3}\ c(x_{4i+2})\ y_{4i+2}$	$(0 \le i \le n-1)$	z'_j:	z_j	$(1 \le j \le m)$
x_{4i+3}:	$y_{4i}\ c(x_{4i+3})\ y_{4i+3}$	$(0 \le i \le n-1)$			
d_j^s:	$z_j\ p_j^s$	$(1 \le j \le m \wedge 1 \le s \le 3)$			
u_j:	$q_j^1\ q_j^2\ q_j^3$	$(1 \le j \le m)$			

Figure 6.3: Preference lists for constructed instance of $(3,3)$-COM-SPA-P.

Theorem 6.4.1. $(3,3)$-COM-SPA-P *is* NP*-complete. The result holds even if each project and lecturer has capacity* 1.

Proof. Suppose we are given an assignment M in an arbitrary instance of $(3,3)$-COM-SPA-P, clearly we can verify in polynomial-time if M is a complete stable matching. Hence $(3,3)$-COM-SPA-P is in NP. Let B be an instance of $(2,2)$-E3-SAT. Let $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and $C = \{c_1, c_2, \ldots, c_m\}$ be the set of variables and clauses respectively in B. Then for each $v_i \in V$, each of literals v_i and $\overline{v_i}$ appears exactly twice in B. Also, for each $c_j \in C$, $|c_j| = 3$. Hence $m = \frac{4n}{3}$.

We construct an instance I of $(3,3)$-COM-SPA-P as follows. Let $X \cup D \cup U$ be the set of students, where $X = \bigcup_{i=0}^{n-1} X_i$, $X_i = \{x_{4i+r} : 0 \le r \le 3\}$ $(0 \le i \le n-1)$, $D = \bigcup_{j=1}^{m} D_j$, $D_j = \{d_j^1, d_j^2, d_j^3\}$ $(1 \le j \le m)$ and $U = \{u_j : c_j \in C\}$. Let $Y \cup Z \cup P \cup Q \cup C'$ be the set of projects, where $Y = \bigcup_{i=0}^{n-1} Y_i$, $Y_i = \{y_{4i+r} : 0 \le r \le 3\}$ $(0 \le i \le n-1)$, $Z = \{z_j : c_j \in C\}$, $P = \bigcup_{j=1}^{m} P_j$, $P_j = \{p_j^1, p_j^2, p_j^3\}$ $(1 \le j \le m)$, $Q = \{q_j^s : c_j \in C \wedge 1 \le s \le 3\}$ and $C' = \{c_j^s : c_j \in C \wedge 1 \le s \le 3\}$. Let $L \cup Y' \cup Z'$ be the set of lecturers, where $L = \{l_j^s : c_j \in C \wedge 1 \le s \le 3\}$ and offers $P \cup C' \cup Q$; $Y' = \{y'_r : 0 \le r \le 4n-1\}$ and offers Y; and $Z' = \{z'_j : 1 \le j \le m\}$ and offers Z. Finally, each project and lecturer has capacity 1. The preference lists in I are shown in Figure 6.3.

In the preference list of a student $x_{4i+r} \in X$ $(0 \le i \le n-1)$, if $r \in \{0,1\}$ then the symbol $c(x_{4i+r})$ denotes the project $c_j^s \in C'$ such that the $(r+1)$th occurrence of literal v_i appears at position s of c_j. Similarly, if $r \in \{2,3\}$ then the symbol $c(x_{4i+r})$ denotes the project $c_j^s \in C'$ such that the $(r-1)$th occurrence of literal $\overline{v_i}$ appears at position s of c_j. We also use $x(c_j^s)$ to denote x_{4i+r} for $r \in \{0,1,2,3\}$. Clearly each preference list is of length at most 3.

For each i $(0 \le i \le n-1)$, let $T_i = \{(x_{4i+r}, y_{4i+r}) : 0 \le r \le 3\}$ and $F_i = \{(x_{4i+r}, y_{4i+r+1}) : 0 \le r \le 3\}$, where addition is taken modulo 4. We claim that B is satisfiable if and only if I admits a complete stable matching.

Let f be a satisfying truth assignment of B. Define a complete stable matching M in I as follows. For each variable $v_i \in V$, if v_i is true under f, add the pairs in T_i to M; otherwise, add the pairs in F_i to M. Now let $c_j \in C$. As c_j contains a literal that is true under f, let

$s \in \{1,2,3\}$ denote the position of c_j in which this literal occurs. Add the pairs (u_j, q_j^s), (d_j^s, z_j), (d_j^a, p_j^a) and (d_j^b, p_j^b) to M, where $\{a,b\} = \{1,2,3\} \setminus \{s\}$ and $a < b$. Clearly M is a complete matching in I.

No project in $Y \cup Z$ can be involved in a blocking pair of M, since each member of $Y \cup Z$ is full in M. Hence, no student in D can be involved in a blocking pair since any such student can only potentially prefer a project in Z. Similarly, no project in Q can be involved in a blocking pair of M since each lecturer $l_j^s \in L$ is full in M and since l_j^s either prefers her worst non-empty project to q_j^s, or q_j^s is her worst non-empty project. Hence no student in U can be involved in a blocking pair of M. Now suppose that $(x_{4i+r}, c(x_{4i+r}))$ blocks M, where $0 \leq i \leq n-1$ and $0 \leq r \leq 3$. Let $c_j^s = c(x_{4i+r})$, where $1 \leq j \leq m$ and $1 \leq s \leq 3$. Then l_j^s is full in M and $(u_j, q_j^s) \in M$. If $r \in \{0,1\}$ then $(x_{4i+r}, y_{4i+r+1}) \in M$, so that v_i is false under f. But literal v_i occurs (unnegated) in c_j at position s, a contradiction, since literal v_i was supposed to be true under f by construction of M (since $(u_j, q_j^s) \in M$ if and only if c_j^s is true). Hence $r \in \{2,3\}$ and $(x_{4i+r}, y_{4i+r}) \in M$, so that v_i is true under f. But literal $\overline{v_i}$ occurs in c_j at position s, a contradiction, since $\overline{v_i}$ was supposed to be true under f by construction of M. Hence M admits no blocking pair.

We next verify that M is coalition-free. Clearly, no student in U can be in a coalition, since no two assigned students in U find the same project acceptable. Also, no student in D who is assigned in M to a project in Z can be in a coalition, since every such student is assigned to her first choice project. As a consequence, no student in D who is assigned in M to a project in P can be in a coalition, since each such student prefers only a project in Z. For each i $(0 \leq i \leq n-1)$, no student in X_i can be in a coalition; for if $M \cap (X_i \times Y_i) = T_i$ then neither x_{4i} nor x_{4i+1} can be involved in a coalition, since each one of them is assigned in M to her first choice project. As a consequence, x_{4i+3} cannot be in a coalition, since the only student x_{4i+3} can potentially form a coalition with is x_{4_i}; and thus x_{4i+2} cannot be in a coalition, since the only student x_{4i+2} can potentially form a coalition with is x_{4i+3}. A similar argument can be made if $M \cap (X_i \times Y_i) = F_i$. Hence M is stable.

Conversely, suppose that M is a complete stable matching in I. Firstly, we claim that every project in C' is undersubscribed in M. To see this, observe that for each j $(1 \leq j \leq m)$, u_j is assigned in M to some q_j^s $(1 \leq s \leq 3)$. As a consequence, the three members of D_j can only be assigned to the three members of $\{z_j\} \cup (P_j \setminus \{p_j^s\})$, since each lecturer in L has capacity 1. Hence, for each s $(1 \leq s \leq 3)$, c_j^s is not assigned to any student. Next, for each i $(0 \leq i \leq n-1)$, we claim that $M \cap (X_i \times Y_i)$ is a perfect matching of $X_i \cup Y_i$. For suppose otherwise, then either (i) some student $x_{4i+r} \in X$ for $r \in \{0,1,2,3\}$ is unassigned in M, since every project in C' is undersubscribed in M, or (ii) some project $y_{4i+r} \in Y$ for $r \in \{0,1,2,3\}$ is undersubscribed in M. If (i) holds, we arrive at a contradiction, since M is a complete stable matching. If (ii) holds, we reach a contradiction following the argument from (i). We form a truth assignment f in B as follows. If $M \cap (X_i \times Y_i) = T_i$, set v_i to be

true under f. Otherwise $M \cap (X_i \times Y_i) = F_i$, in which case we set v_i to be false under f.

Now let c_j be a clause in C $(1 \leq j \leq m)$. There exists some s $(1 \leq s \leq 3)$ such that $(u_j, q_j^s) \in M$. Let $x_{4i+r} = x(c_j^s)$ for some i $(0 \leq i \leq n-1)$ and r $(0 \leq r \leq 3)$. If $r \in \{0, 1\}$ then $(x_{4i+r}, y_{4i+r}) \in M$, since M is stable. Thus variable v_i is true under f and hence clause c_j is true under f, since literal v_i occurs unnegated in c_j. If $r \in \{2, 3\}$ then $(x_{4i+r}, y_{4i+r+1}) \in M$ (where addition is taken modulo 4), since M is stable. Thus variable v_i is false under f, and hence clause c_j is true under f, since literal $\overline{v_i}$ occurs in c_j. Hence f is a satisfying truth assignment of B. □

6.5 An IP model for MAX-SPA-P

In this section, we describe an IP model to enable MAX-SPA-P to be solved optimally. Let I be an instance of SPA-P involving a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of students, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of projects and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of lecturers. We construct an IP model J of I as follows. Firstly, we create binary variables $x_{i,j} \in \{0, 1\}$ $(1 \leq i \leq n_1, 1 \leq j \leq n_2)$ for each acceptable pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ such that $x_{i,j}$ indicates whether s_i is assigned to p_j in a solution or not. Henceforth, we denote by S a solution in the IP model J, and we denote by M the matching derived from S in the following natural way: if $x_{i,j} = 1$ under S then s_i is assigned to p_j in M, otherwise s_i is not assigned to p_j in M.

6.5.1 Constraints

In this section, we give the set of constraints to ensure that the assignment obtained from a feasible solution in J is a matching, and that the matching admits no blocking pair and is coalition-free.

Matching constraints The feasibility of a matching can be ensured with the following three set of constraints.

$$\sum_{p_j \in A_i} x_{i,j} \leq 1 \qquad (1 \leq i \leq n_1), \tag{6.1}$$

$$\sum_{i=1}^{n_1} x_{i,j} \leq c_j \qquad (1 \leq j \leq n_2), \tag{6.2}$$

$$\sum_{i=1}^{n_1} \sum_{p_j \in P_k} x_{i,j} \leq d_k \qquad (1 \leq k \leq n_3)\ . \tag{6.3}$$

Note that Inequality (6.1) ensures that each student $s_i \in \mathcal{S}$ is not assigned to more than one project, while Inequalities (6.2) and (6.3) ensures that the capacity of each project $p_j \in \mathcal{P}$ and each lecturer $l_k \in \mathcal{L}$ is not exceeded.

Given an acceptable pair (s_i, p_j), we define $\mathrm{rank}(s_i, p_j)$, the *rank* of p_j on s_i's preference list, to be $r+1$ where r is the number of projects that s_i prefers to p_j. Given a lecturer $l_k \in \mathcal{L}$ and a project $p_j \in P_k$, an analogous definition holds for $\mathrm{rank}(l_k, p_j)$, the *rank* of p_j on l_k's preference list. With respect to an acceptable pair (s_i, p_j), we define $S_{i,j} = \{p_{j'} \in A_i : \mathrm{rank}(s_i, p_{j'}) \leq \mathrm{rank}(s_i, p_j)\}$, the set of projects that s_i likes as much as p_j. For a project p_j offered by lecturer $l_k \in \mathcal{L}$, we also define $T_{k,j} = \{p_q \in P_k : \mathrm{rank}(l_k, p_j) < \mathrm{rank}(l_k, p_q)\}$, the set of projects that are worse than p_j on l_k's preference list.

In what follows, we fix an arbitrary acceptable pair (s_i, p_j) and we impose constraints to ensure that (s_i, p_j) is not a blocking pair for the matching M (i.e., (s_i, p_j) is not a type (a), type (b) or type (c) blocking pair for M). Firstly, let l_k be the lecturer who offers p_j.

Blocking pair constraints We define $\theta_{i,j} = 1 - \sum_{p_{j'} \in S_{i,j}} x_{i,j'}$. Intuitively, $\theta_{i,j} = 1$ if and only if s_i is unassigned in M or prefers p_j to $M(s_i)$. Next we create a binary variable α_j in J such that if p_j is undersubscribed in M then $\alpha_j = 1$. We enforce this condition by imposing the following constraint.

$$c_j \alpha_j \geq c_j - \sum_{i'=1}^{n_1} x_{i',j} \ , \tag{6.4}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$. If p_j is undersubscribed in M then the RHS of Inequality (6.4) is at least 1, and this implies that $\alpha_j = 1$; otherwise, α_j is not constrained. Now let $\gamma_{i,j,k} = \sum_{p_{j'} \in T_{k,j}} x_{i,j'}$. Intuitively, if $\gamma_{i,j,k} = 1$ in S then s_i is assigned to a project $p_{j'}$ offered by l_k in M, where l_k prefers p_j to $p_{j'}$. The following constraint ensures that (s_i, p_j) does not form a type (a) blocking pair for M.

$$\boxed{\theta_{i,j} + \alpha_j + \gamma_{i,j,k} \leq 2 \ .} \tag{6.5}$$

Note that if the sum of the binary variables in the LHS of Inequality (6.5) is less than or equal to 2, this implies that at least one of the variables, say $\gamma_{i,j,k}$, is 0. Thus the pair (s_i, p_j) is not a type (a) blocking pair for M.

Next we define $\beta_{i,k} = \sum_{p_{j'} \in P_k} x_{i,j'}$. Clearly, s_i is assigned to a project offered by l_k in M if and only if $\beta_{i,k} = 1$ in S. We define $D_{k,j} = \{p_{j'} \in P_k : \mathrm{rank}(l_k, p_{j'}) \leq \mathrm{rank}(l_k, p_j)\}$, the set of projects that l_k likes as much as p_j. Next, we create a binary variable $\eta_{j,k}$ in J such that $\eta_{j,k} = 1$ if l_k is undersubscribed or prefers p_j to his worst non-empty project in M. We

enforce this by imposing the following constraint.

$$d_k\eta_{j,k} \geq d_k - \sum_{i'=1}^{n_1} \sum_{p_{j'} \in D_{k,j}} x_{i',j'} \ , \tag{6.6}$$

where $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in D_{k,j}} x_{i',j'}$ is the occupancy of l_k involving projects that are at least as good as p_j on l_k's preference list. Intuitively if l_k is undersubscribed in M or if l_k prefers p_j to her worst non-empty project in M, then the RHS of Inequality (6.6) is at least 1. Finally, to avoid type (b) and type (c) blocking pairs, we impose the following constraint.

$$\boxed{\theta_{i,j} + \alpha_j + (1 - \beta_{i,k}) + \eta_{j,k} \leq 3 \ .} \tag{6.7}$$

Next, we give the constraints to ensure that the matching obtained from a feasible solution in J is coalition-free.

Coalition constraints First, we introduce some additional notation. Given an instance I' of SPA-P and a matching M' in I', we define the *envy graph* $G(M') = (\mathcal{S}, A)$, where the vertex set $\mathcal{S}$ is the set of students in I', and the arc set

$$A = \{(s_i, s_{i'}) : s_i \text{ prefers } M'(s_{i'}) \text{ to } M'(s_i)\}.$$

It is clear that the matching $M_2 = \{(s_1, p_1), (s_2, p_2), (s_3, p_3)\}$ admits a coalition $\{s_1, s_2\}$ with respect to the instance given in Figure 6.1. The resulting envy graph $G(M_2)$ is illustrated below.

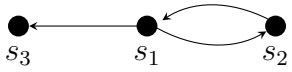


Figure 6.4: The envy graph $G(M_2)$ corresponding to the SPA-P instance in Figure 6.1.

Clearly, $G(M')$ contains a directed cycle if and only if M' admits a coalition. Moreover, $G(M')$ is acyclic if and only if it admits a topological ordering. Now to ensure that the matching M obtained from a feasible solution S under J is coalition-free, we will enforce J to encode the envy graph $G(M)$ and impose the condition that it must admit a topological ordering. In what follows, we build on our IP model J of I.

We create a binary variable $e_{i,i'}$ for each $(s_i, s_{i'}) \in \mathcal{S} \times \mathcal{S}$, $s_i \neq s_{i'}$, such that the $e_{i,i'}$ variables will correspond to the adjacency matrix of $G(M)$. For each i and i' ($1 \leq i \leq n_1,\ 1 \leq i' \leq n_1,\ i \neq i'$) and for each j and j' ($1 \leq j \leq n_2,\ 1 \leq j' \leq n_2$) such that s_i prefers $p_{j'}$ to p_j, we

impose the following constraint:

$$e_{i,i'} + 1 \geq x_{i,j} + x_{i',j'} \ . \tag{6.8}$$

If $(s_i, p_j) \in M$ and $(s_{i'}, p_{j'}) \in M$ and s_i prefers $p_{j'}$ to p_j, then $e_{i,i'} = 1$ and we say s_i *envies* $s_{i'}$; otherwise, $e_{i,i'}$ is not constrained. Next we enforce the condition that $G(M)$ must have a topological ordering. To hold the label of each vertex in a topological ordering, we create an integer-valued variable v_i corresponding to each student $s_i \in \mathcal{S}$ (and intuitively to each vertex in $G(M)$). We wish to enforce the constraint that if $e_{i,i'} = 1$ (i.e., $(s_i, s_{i'}) \in A$), then $v_i < v_{i'}$ (i.e., the label of vertex s_i is smaller than the label of vertex $s_{i'}$). This is achieved by imposing the following constraint for all i and i' ($1 \leq i \leq n_1,\ 1 \leq i' \leq n_1,\ i \neq i'$).

$$\boxed{v_i < v_{i'} + n_1(1 - e_{i,i'}) \ .} \tag{6.9}$$

Note that the LHS of Inequality (6.9) is strictly less than the RHS of Inequality (6.9) for all i, i ($i \neq i$) if and only if $G(M)$ does not admit a directed cycle, and this implies that M is coalition-free.

6.5.2 Variables

We define a collective notation for each variable involved in J as follows.

$$\begin{aligned} X &= \{x_{i,j} : s_i \in \mathcal{S} \wedge p_j \in A_i\}, & \Lambda &= \{\alpha_j : 1 \leq j \leq n_2\}, \\ H &= \{\eta_{j,k} : 1 \leq j \leq n_2, 1 \leq k \leq n_3\}, & V &= \{v_i : 1 \leq i \leq n_1\} \ . \\ E &= \{e_{i,i'} : 1 \leq i \leq n_1, 1 \leq i' \leq n_1\}, \end{aligned}$$

6.5.3 Objective function

The objective function given below is a summation of all the $x_{i,j}$ binary variables. It seeks to maximize the number of students assigned (i.e., the size of the matching).

$$\boxed{\max \sum_{i=1}^{n_1} \sum_{p_j \in A_i} x_{i,j} \ .} \tag{6.10}$$

Finally, we have constructed an IP model J of I comprising the set of integer-valued variables X, Λ, H, E and V, the set of Inequalities (6.1) - (6.9) and an objective function (6.10). Note that J can then be used to solve MAX-SPA-P optimally. Moreover, if J admits a feasible solution S, then the objective value denoted $obj(S)$ is equivalent to the number of students that are assigned in M, i.e., $obj(S) = |M|$.

6.5.4 Correctness of the IP model

Given an instance I of SPA-P formulated as an IP model J using the above transformation, we establish the correctness of J via the following lemmas.

Lemma 6.5.1. *A feasible solution S to J corresponds to a stable matching M in I, where $obj(S) = |M|$.*

Proof. Assume firstly that J has a feasible solution S. Let $M = \{(s_i, p_j) \in \mathcal{S} \times \mathcal{P} : x_{i,j} = 1\}$ be the assignment in I generated from S. Clearly $obj(S) = |M|$. We note that Inequality (6.1) ensures that each student is assigned in M to at most one project. Moreover, Inequalities (6.2) and (6.3) ensures that the capacity of each project and lecturer is not exceeded in M. Thus M is a matching. We will prove that Inequalities (6.4) - (6.7) guarantees that M admits no blocking pair.

Suppose for a contradiction that there exists some acceptable pair (s_i, p_j) that forms a blocking pair for M, where l_k is the lecturer who offers p_j. This implies that s_i is either unassigned in M or prefers p_j to $M(s_i)$. In either of these cases, $\sum_{p_{j'} \in S_{i,j}} x_{i,j'} = 0$, and thus $\theta_{i,j} = 1$. Moreover, as (s_i, p_j) is a blocking pair for M, p_j has to be undersubscribed in M, and thus $\sum_{i'=1}^{n_1} x_{i',j} < c_j$. This implies that the RHS of Inequality (6.4) is strictly greater than 0, and since S is a feasible solution to J, $\alpha_j = 1$.

Now suppose (s_i, p_j) is a type (a) blocking pair, and suppose $M(s_i) = p_{j''}$ for some $p_{j''} \in P_k$. We have that l_k prefers p_j to $p_{j''}$, thus $\gamma_{i,j,k} = \sum_{p_{j'} \in T_{k,j}} x_{i,j'} = 1$. Now, $\theta_{i,j} = \alpha_j = \gamma_{i,j,k} = 1$ implies that the LHS of Inequality (6.5) is strictly greater than 2. Thus S is not a feasible solution, a contradiction.

Next suppose (s_i, p_j) is a type (b) or type (c) blocking pair for M. This implies that $s_i \notin M(l_k)$ and thus $1 - \beta_{i,k} = 1 - \sum_{p'_j \in P_k} x_{i,j'} = 1$. Also, either l_k is undersubscribed in M or l_k prefers p_j to p_z, where p_z is l_k's worst non-empty project in M. This implies that the RHS of Inequality (6.6) is strictly greater than 0, and thus $\eta_{j,k} = 1$. Hence the LHS of Inequality (6.7) is strictly greater than 3. Thus S is not a feasible solution, a contradiction.

Finally, we show that Inequalities (6.8) and (6.9) ensure that M is coalition-free. Suppose for a contradiction that M admits a coalition $\langle s_{i_0}, \ldots, s_{i_{r-1}} \rangle$, for some $r \geq 2$. This implies that for each t $(0 \leq t \leq r-1)$, s_{i_t} prefers $M(s_{i_{t+1}})$ to $M(s_{i_t})$, where addition is taken modulo r, and hence $e_{i_t, i_{t+1}} = 1$, by Inequality (6.8). It follows from Inequality (6.9) that $v_{i_0} < v_{i_1} < \cdots < v_{i_{r-2}} < v_{i_{r-1}} < v_{i_r} = v_{i_0}$, a contradiction. Hence M is coalition-free, and thus M is a stable matching. □

Lemma 6.5.2. *A stable matching M in I corresponds to a feasible solution S to J, where $|M| = obj(S)$.*

Proof. Let M be a stable matching in I. First we set all the binary variables involved in J to 0. For all $(s_i, p_j) \in M$, we set $x_{i,j} = 1$. Now, since M is a matching, it is clear that Inequalities (6.1) - (6.3) is satisfied. For any acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that s_i is unassigned in M or prefers p_j to $M(s_i)$, we set $\theta_{i,j} = 1$. For any project $p_j \in \mathcal{P}$ that is undersubscribed in M, we set $\alpha_j = 1$ and thus Inequality (6.4) is satisfied. For Inequality (6.5) not to be satisfied, its LHS must be strictly greater than 2. This would only happen if there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$ and $\gamma_{i,j,k} = 1$. This implies that s_i is assigned in M to a project $p_{j'}$ offered by l_k such that s_i prefers p_j to $p_{j'}$, p_j is undersubscribed in M, and l_k prefers p_j to $p_{j'}$. Thus (s_i, p_j) is a type (a) blocking pair for M, a contradiction to the stability of M. Hence Inequality (6.5) is satisfied.

Suppose l_k is a lecturer in $\mathcal{L}$ and p_j is any project on l_k's preference list. Let p_z be l_k's worst non-empty project in M. If l_k is undersubscribed in M or l_k prefers p_j to p_z, we set $\eta_{j,k} = 1$. Then Inequality (6.6) is satisfied. Now suppose Inequality (6.7) is not satisfied. This would only happen if there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1, \alpha_j = 1, \beta_{i,k} = 0$ and $\eta_{j,k} = 1$. This implies that either s_i is unassigned in M or prefers p_j to $M(s_i)$, $s_i \notin M(l_k)$, p_j is undersubscribed in M and either l_k is undersubscribed in M or l_k prefers p_j to his worst non-empty project in M. Thus (s_i, p_j) is either a type (b) or type (c) blocking pair for M, a contradiction to the stability of M. Hence Inequality (6.7) is satisfied.

We denote by $G(M) = (\mathcal{S}, A)$ the envy graph of M. Suppose s_i and $s_{i'}$ are any two distinct students in $\mathcal{S}$ such that $(s_i, p_j) \in M$, $(s_{i'}, p_{j'}) \in M$ and s_i prefers $p_{j'}$ to p_j (i.e., $(s_i, s_{i'}) \in A$), we set $e_{i,i'} = 1$. Thus Inequality (6.8) is satisfied. Since M is a stable matching, M is coalition-free. This implies that $G(M)$ is acyclic and has a topological ordering

$$\sigma : \mathcal{S} \to \{1, 2, \ldots, n_1\} \ .$$

For each i ($1 \leq i \leq n_1$), let $v_i = \sigma(s_i)$. Now suppose Inequality (6.9) is not satisfied. This implies that there exists vertices s_i and $s_{i'}$ in $G(M)$ such that $v_i \geq v_{i'} + n_1(1 - e_{i,i'})$. This is only possible if $e_{i,i'} = 1$ since $1 \leq v_i \leq n_1$ and $1 \leq v_{i'} \leq n_1$. Hence $v_i \geq v_{i'}$, a contradiction to the fact that σ is a topological ordering of $G(M)$ (since $(s_i, s_{i'}) \in A$ implies that $v_i < v_{i'}$). Hence S, comprising the above assignment of values to the variables in $X \cup \Lambda \cup H \cup \Delta \cup E \cup V$, is a feasible solution to J; and clearly $|M| = obj(S)$. □

Lemmas 6.5.1 and 6.5.2 immediately give rise to the following theorem regarding the correctness of J.

Theorem 6.5.3. *A feasible solution to J is optimal if and only if the corresponding stable matching in I is of maximum cardinality.*

Proof. Let S be an optimal solution to J. Then by Lemma 6.5.1, S corresponds to a stable matching M in I such that $obj(S) = |M|$. Suppose M is not of maximum cardinality. Then there exists a stable matching M' in I such that $|M'| > |M|$. By Lemma 6.5.2, M' corresponds to a feasible solution S' to J such that $obj(S') = |M'| > |M| = obj(S)$. This is a contradiction, since S is an optimal solution to J. Hence M is a maximum size stable matching in I. Similarly, if M is a maximum size stable matching in I then M corresponds to an optimal solution S to J. □

6.6 Empirical evaluation

In this section we present results from an empirical evaluation that investigates how the sizes of the stable matchings produced by the approximation algorithms compares to those of the optimal solution obtained from our IP model, on SPA-P instances that are both randomly generated and derived from real datasets.

6.6.1 Experimental setup

When generating SPA-P instances, there are clearly several parameters that can be varied, such as the number of students, projects and lecturers; the length of the students' preference lists; as well as the total capacities of the projects and lecturers. For each range of values for the first two parameters, we generated a set of random SPA-P instances. In each set, we recorded the average size of a stable matching obtained from running the approximation algorithms and the IP model. Further, we considered the average time taken for the IP model to find an optimal solution.

Very broadly, the approximation algorithms involve a sequence of applications and deletions. The students applies to projects that they find acceptable, and when a project and/or lecturer becomes full, certain (student, project) pairs are deleted. By design, the approximation algorithms were randomised with respect to the sequence in which students apply to projects, and the choice of students to reject when projects and/or lecturers become full. In the light of this, for each dataset, we also ran the approximation algorithms 100 times and record the size of the largest stable matching obtained over these runs. Our experiments therefore involved five algorithms: the optimal IP-based algorithm, the two approximation algorithms run once, and the two approximation algorithms run 100 times.

We performed our experiments on a machine with dual Intel Xeon CPU E5-2640 processors with 64GB of RAM, running Ubuntu 17.10. Each of the approximation algorithms was implemented in Java[2]. For our IP model, we carried out the implementation using the

[2]https://github.com/sofiatolaosebikan/spa-p-isco-2018

Gurobi optimisation solver in Java (see footnote 2). For correctness testing on these implementations, we designed a stability checker which verifies that the matching returned by the approximation algorithms and the IP model does not admit a blocking pair or a coalition.

6.6.2 Randomly-generated datasets

All the SPA-P instances that we randomly generated involved n_1 students (n_1 is henceforth referred to as the size of the instance), $0.5n_1$ projects, $0.2n_1$ lecturers and $1.1n_1$ total project capacity which was randomly distributed amongst the projects such that each project has capacity at least 1. The capacity for each lecturer l_k was chosen uniformly at random to lie between the highest capacity of the projects offered by l_k and the sum of the capacities of the projects that l_k offers. In the first experiment, we present results obtained from comparing the performance of the IP model, with and without the coalition constraints in place.

Experiment 0

We increased the number of students n_1 while maintaining a ratio of projects, lecturers, project capacities and lecturer capacities as described above. For various values of n_1 ($100 \leq n_1 \leq 1000$) in increments of 100, we created 100 randomly-generated instances. Each student's preference list contained a minimum of 2 and a maximum of 5 projects. With respect to each value of n_1, we obtained the average time taken for the IP solver to output a solution, both with and without the coalition constraints being enforced. The results, displayed in Table 6.1 show that when we removed the coalition constraints, the average time for the IP solver to output a solution is significantly faster than when we enforced the coalition constraints.

Table 6.1: Results for Experiment 0. Average time (in seconds) for the IP solver to output a solution, both with and without the coalition constraints being enforced.

Size of instance	100	200	300	400	500	600	700	800	900	1000
Av. time without coalition	0.12	0.27	0.46	0.69	0.89	1.17	1.50	1.86	2.20	2.61
Av. time with coalition	0.71	2.43	4.84	9.15	13.15	19.34	28.36	38.18	48.48	63.50

In the remaining experiments, we thus remove the constraints that enforce the absence of a coalition in the solution. We are able to do this for the purposes of these experiments because the largest size of a stable matching is equal to the largest size of a matching that potentially admits a coalition but admits no blocking pair[3], and we were primarily concerned with mea-

[3]This holds because the number of students assigned to each project and lecturer in the matching remains the same even after the students involved in such coalition permute their assigned projects.

suring stable matching cardinalities. However the absence of the coalition constraints should be borne in mind when interpreting the IP solver runtime data in what follows.

In the next two experiments, we discuss results obtained from running the five algorithms on randomly-generated datasets.

Experiment 1

As in the previous experiment, we maintained the ratio of the number of students to projects, lecturers and total project capacity, as well as the length of the students' preference lists. For various values of n_1 $(100 \leq n_1 \leq 2500)$ in increments of 100, we created 1000 randomly-generated instances. With respect to each value of n_1, we obtained the average sizes of stable matchings constructed by the five algorithms run over the 1000 instances. The result displayed in Figure 6.5(a) shows the ratio of the average size of the stable matching produced by the approximation algorithms with respect to the average size of the maximum cardinality stable matchings produced by the IP solver.

Figure 6.5(a) shows that each of the approximation algorithms produces stable matchings with a much higher cardinality from multiple runs, compared to running them only once. Also, the average time taken for the IP solver to find a maximum cardinality matching increases as the size of the instance increases, with a running time of less than one second for instance size 100, increasing roughly linearly to 13 seconds for instance size 2500 (see Figure 6.5(b)). Perhaps not surprising, each of the approximation algorithms terminates in less than one second for all the datasets.

Experiment 2

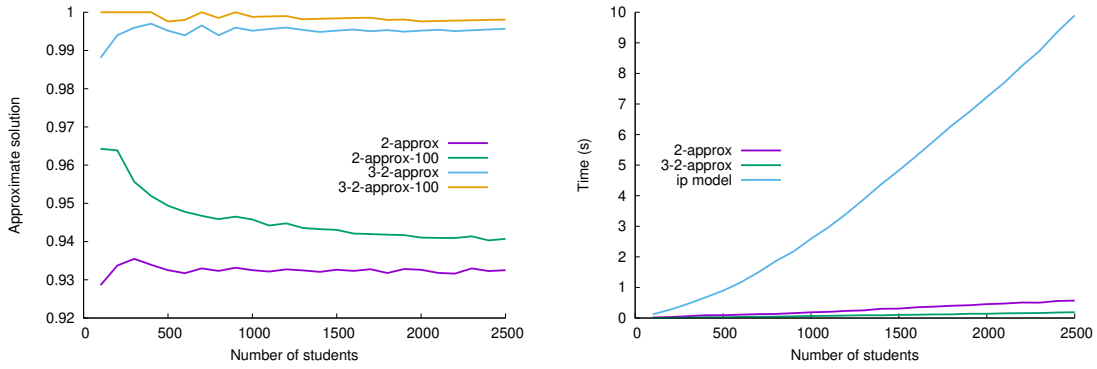
In this experiment, we varied the length of each student's preference list while maintaining a fixed number of students, projects, lecturers and total project capacity. For various values of x $(2 \leq x \leq 10)$, we generated 1000 instances, each involving 1000 students, with each student's preference list containing exactly x projects. The result for all values of x is displayed in Figure 6.6(a), which shows that as we increase the preference list length, the sizes of the stable matchings produced by each of the approximation algorithms approaches optimality. Figure 6.6(a) also shows that with a preference list length greater than 5, the $\frac{3}{2}$-approximation algorithm produces an optimal solution, even on a single run. Moreover, the average time taken for the IP solver to find the size of a maximum size stable matching increases as the length of the students' preference lists increases, with a running time of two seconds when each student's preference list is of length 2, increasing roughly linearly to 17 seconds when each student's preference list is of length 10 (see Figure 6.6(b)).

6.6.3 Real datasets

The real datasets in this chapter are based on actual student preference data and manufactured lecturer data from previous runs of student-project allocation processes at the School of Computing Science, University of Glasgow. Table 6.2 shows the properties of the real datasets, where n_1, n_2 and n_3 denotes the number of students, projects and lecturers respectively; and l denotes the length of each student's preference list. For all these datasets, each project has a capacity of 1 and the capacity of each lecturer was provided as part of the datasets. In the next experiment, we discuss how the lecturer preferences over their proposed projects were generated (which is the only information we manufactured). We also discuss the results obtained from running the five algorithms on the corresponding SPA-P instances.

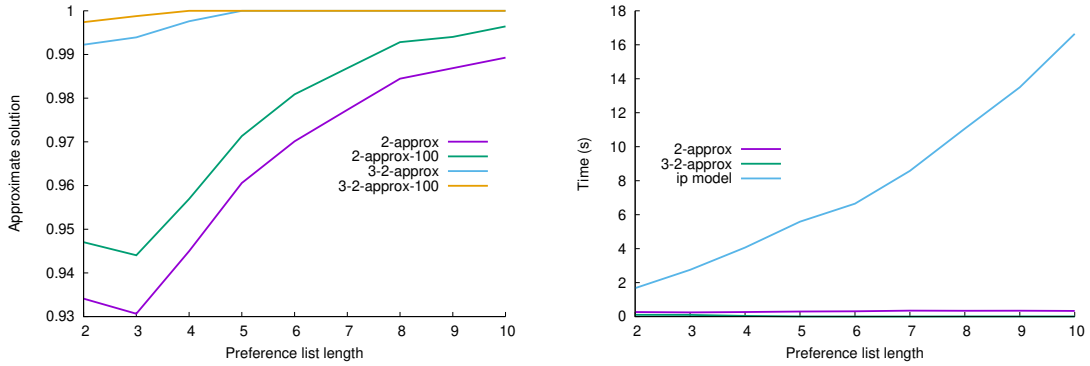
Experiment 3

We derived the lecturer preference data from the real datasets as follows. For each lecturer l_k, and for each project p_j offered by l_k, we obtained the number a_j of students that find p_j acceptable. Next, we generated a strictly-ordered preference list for l_k by arranging l_k's proposed projects in (i) a random manner, (ii) ascending order of a_j, and (iii) descending order of a_j, where (ii) and (iii) are taken over all projects that l_k offers. Table 6.2 shows the size of stable matchings obtained from the five algorithms, and the results are essentially consistent with the findings in the previous experiments: i.e., the $\frac{3}{2}$-approximation algorithm produces stable matchings whose sizes are close to optimal.



(a) (b)

Figure 6.5: Result for Experiment 1.



(a) (b)

Figure 6.6: Result for Experiment 2.

Table 6.2: Properties of the real datasets and the size of stable matchings obtained from the five algorithms, with respect to Experiment 3, where A, B, C, D and E denotes the solution obtained from the IP model, 100 runs of the $\frac{3}{2}$-approximation algorithm, single run of the $\frac{3}{2}$-approximation algorithm, 100 runs of the 2-approximation algorithm, and single run of the 2-approximation algorithm, respectively.

					Random					Most popular					Least popular				
Year	n_1	n_2	n_3	l	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
2014	55	149	38	6	55	55	55	54	53	55	55	55	54	50	55	55	55	54	52
2015	76	197	46	6	76	76	76	76	72	76	76	76	76	72	76	76	76	76	75
2016	92	214	44	6	84	82	83	77	75	85	85	83	79	76	82	80	77	76	74
2017	90	289	59	4	89	87	85	80	76	90	89	86	81	79	88	85	84	80	77

6.6.4 Discussions

The results presented in this section suggest that even as we increase the number of students, projects, lecturers, and the length of the students' preference lists, each of the approximation algorithms finds stable matchings that are close to having maximum cardinality, outperforming their approximation factor. Perhaps most interesting is the $\frac{3}{2}$-approximation algorithm, which finds stable matchings that are very close in size to optimal, even on a single run. These results also holds analogously for the instances derived from real datasets.

We remark that when we removed the coalition constraints, we were able to run the IP model on an instance size of 10000, with the solver returning the size of a maximum matching in an average time of 100 seconds, over 100 randomly-generated instances. This shows that

the IP model (without enforcing the coalition constraints), can be run on large instances that could potentially appear in practical applications of the SPA-P model, to find maximum size stable matchings that potentially admits coalition of students. These coalitions should then be eliminated in polynomial time by repeatedly constructing an *envy graph*, similar to the one described in [83, p.290], finding a directed cycle and letting the students in the cycle swap projects.

6.7 Conclusions and open problems

In this chapter, we have presented algorithmic and experimental results for finding maximum size stable matchings in instances of SPA-P. From an algorithmic perspective, we have shown that MAX-SPA-P becomes polynomial-time solvable if there is only one lecturer, whilst the problem remains NP-hard to approximate even if there are two lecturers involved. We also proved that it is NP-hard to find a maximum size stable matching if each preference list is of length at most 3. It would be interesting to consider other polynomial-time solvable special cases, for example, what if each student's preference list is of length at most 2 and each lecturer's preference lists is of unbounded length?

Moving away from hardness results, a particularly interesting direction would be to explore parameterisations that lead to FPT algorithms for MAX-SPA-P. Whilst our NP-hardness result shows that parameterising on the number of lecturers or the maximum length of a preference list is not a good choice, other suitable parameterisations that could be explored include the maximum capacity of a project or lecturer, which we might expect to be small in practice. On the other hand, a different direction is to establish W[1]-hardness results under various parameterisations.

To enable MAX-SPA-P to be solved optimally in practice, we went on to describe an IP model for the problem. From our experimental results, we were able to deduce that the $\frac{3}{2}$-approximation algorithm of Iwama *et al.* [61] constructs stable matchings whose size is very close to optimal. Nevertheless, the question remains as to whether there exists an approximation algorithm for MAX-SPA-P that has a performance guarantee better than $\frac{3}{2}$?

Bibliography

[1] `http://www.nrmp.org` (National Resident Matching Program website). Accessed 25 March 2020.

[2] `http://www.carms.ca` (Canadian Resident Matching Service website). Accessed 25 March 2020.

[3] `http://www.jrmp.jp` (Japan Residency Matching Program website). Accessed 25 March 2020.

[4] `http://www.paireddonation.org`. (Alliance for Paired Donation website. Accessed 25 March 2020).

[5] `http://www.gurobi.com` (Gurobi Optimization website). Accessed 25 March 2020.

[6] `https://www.gnu.org/software/glpk` (GNU Linear Programming Kit). Accessed 25 March 2020.

[7] `http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/` (CPLEX Optimization Studio). Accessed 25 March 2020.

[8] `https://eli.thegreenplace.net/2008/08/23/initializing-an-array-in-constant-time#`. (Initialising an array in constant time: Eli Bendersky's website. Accessed 05 July 2020).

[9] A. Abdulkadiroğlu, P.A. Pathak, and A.E. Roth. The Boston public school match. *American Economic Review*, 95(2):368–371, 2005.

[10] A. Abdulkadiroğlu, P.A. Pathak, and A.E. Roth. The New York City high school match. *American Economic Review*, 95(2):364–367, 2005.

[11] A. Abdulkadiroğlu, P.A. Pathak, A.E. Roth, and T. Sönmez. Changing the Boston school-choice mechanism. NBER working paper 11965, 2006.

[12] A. Abdulkadiroğlu and T. Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.

[13] 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.

[14] A.H. Abu El-Atta and M.I. Moussa. Student project allocation with preference lists over (student,project) pairs. In *Proceedings of ICCEE 09: the 2nd International Conference on Computer and Electrical Engineering*, pages 375–379. IEEE, 2009.

[15] K.C. Ágoston, P. Biró, and I. McBride. Integer programming methods for special college admissions problems. *Journal of Combinatorial Optimization*, 32(4):1371–1399, 2016.

[16] B. Aldershof and O.M. Carducci. Stable matching with couples. *Discrete Applied Mathematics*, 68:203–207, 1996.

[17] A.A. Anwar and A.S. Bahaj. Student project allocation using integer programming. *IEEE Transactions on Education*, 46(3):359–367, 2003.

[18] M. Baidas, M. Bahbahani, E. Alsusa, K. Hamdi, and Z. Ding. D2d group association and channel assignment in uplink multi-cell noma networks: A matching theoretic approach. *To appear in the IEEE Transactions on Communications*, page 220, 2019.

[19] M. Baidas, Z. Bahbahani, and E. Alsusa. User association and channel assignment in downlink multi-cell noma networks: A matching-theoretic approach. *EURASIP Journal on Wireless Communications and Networking*, 2019(2):220, 2019.

[20] M. Baidas, Z. Bahbahani, N. El-Sharkawi, H. Shehada, and E. Alsusa. Joint relay selection and max-min energy-efficient power allocation in downlink multicell noma networks: A matching-theoretic approach. *Transactions on Emerging Telecommunications Technologies*, 30(5):3564, 2019.

[21] 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.

[22] P. Berman, M. Karpinski, and Alexander D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. Electronic Colloquium on Computational Complexity Report, number 49, 2003.

[23] P. Biro. Matching practices in secondary school - Hungary, MiP country profile 6. Web document available at `http://www.matching-in-practice.eu/secondary-schools-in-hungary/`. (Accessed 25 March 2020).

[24] P. Biró. Applications of matching models under preferences. In U. Endriss, editor, *Trends in Computational Social Choice*, chapter 18. AI Access, 2017. Supported by COST Action IC1205 on Computational Social Choice.

[25] P. Biró and F. Klijn. Matching with couples: a multidisciplinary survey. *International Game Theory Review*, 15(2), 2013. article number 1340008.

[26] P. Biró, D.F. Manlove, and I. McBride. The Hospitals / Residents problem with Couples: Complexity and integer programming models. In *Proceedings of SEA 2014: the 13th International Symposium on Experimental Algorithms*, volume 8504 of *Lecture Notes in Computer Science*, pages 10–21. Springer, 2014.

[27] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.

[28] K. Cechlárová, Á. Cseh, and D. Manlove. Selected open problems in matching under preferences. *Bulletin of the EATCS*, 128:14 – 28, 2019.

[29] K. Cechlárová, T. Fleiner, D. Manlove, I. McBride, and E. Potpinková. Modelling practical placement of trainee teachers to schools. *Central European Journal of Operations Research*, 23(3):547–562, 2015.

[30] C. Cheng, E. McDermid, and I. Suzuki. A unified approach to finding good stable matchings in the hospitals/residents setting. *Theoretical Computer Science*, 400(1-3):84–99, 2008.

[31] M. Chiarandini, R. Fagerberg, and S. Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research*, 275(1):39 – 78, 2019.

[32] J. Combe, O.Tercieux, and C. Terrier. The design of teacher assignment: Theory and evidence. Technical report, London School of Economics (LSE), 2015.

[33] Frances Cooper and David Manlove. A 3/2-Approximation Algorithm for the Student-Project Allocation Problem. In *Proceedings of SEA '18: the 17th International Symposium on Experimental Algorithms*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1 – 8:13, 2018.

[34] Frances Cooper and David Manlove. A 3/2-Approximation Algorithm for the Student-Project Allocation Problem. *CoRR*, abs/1804.02731, 2018. Available from `http://arxiv.org/abs/1804.02731`.

[35] D. Dimakopoulos and C. Heller. Matching with waiting times: The German entry-level labour market for lawyers. In *Proceedings of the German Economic Association Annual Conference 2015: Economic Development - Theory and Policy*, 2015.

[36] J. Dye. A constraint logic programming approach to the stable marriage problem and its application to student-project allocation. BSc Honours project report, University of York, Department of Computer Science, 2001.

[37] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[38] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[39] M.R. Garey and D.S. Johnson. *Computers and intractability*. WH Freeman New York, 1979.

[40] N. Garg, T. Kavitha, A. Kumar, K. Mehlhorn, and J. Mestre. Assigning papers to referees. *Algorithmica*, 58(1):119–136, 2010.

[41] M.J. Geiger and W. Wenger. On the assignment of students to topics: A variable neighborhood search approach. *Socio-Economic Planning Sciences*, 44(1):25 – 34, 2010.

[42] Y.A. Gonczarowski, L. Kovalio, N. Nisan, and A. Romm. Matching for the israeli "mechinot" gap-year programs: Handling rich diversity requirements. Technical Report 1905.00364, Computing Research Repository, Cornell University Library, 2019. Available from `http://arxiv.org/abs/1905.00364` (Accessed 25 March 2020).

[43] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16(1):111–128, 1987.

[44] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[45] M.M. Halldórsson, R.W. Irving, K. Iwama, D.F. Manlove, S. Miyazaki, Y. Morita, and S. Scott. Approximability results for stable marriage problems with ties. *Theoretical Computer Science*, 306(1-3):431–447, 2003.

[46] M.M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Randomized approximation of the stable marriage problem. In *Proceedings of COCOON '03: the 9th Annual International Computing and Combinatorics Conference*, volume 2697 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 2003.

[47] M.M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Randomized approximation of the stable marriage problem. *Theoretical Computer Science*, 325(3):439–465, 2004. Preliminary version appeared as [46].

[48] P.R. Harper, V. de Senna, I.T. Vieira, and A.K. Shahani. A genetic algorithm for the project assignment problem. *Computers and Operations Research*, 32:1255–1265, 2005.

[49] J.E. Hopcroft and R.M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[50] R.W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

[51] R.W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.

[52] R.W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98: the 6th European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 1998.

[53] R.W. Irving and P. Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15(3):655–667, 1986.

[54] 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.

[55] R.W. Irving and D.F. Manlove. Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *Journal of Combinatorial Optimization*, 16(3):279–292, 2008.

[56] R.W. Irving, D.F. Manlove, and G. O'Malley. Stable marriage with ties and bounded length preference lists. *Journal of Discrete Algorithms*, 7(2):213–219, 2009.

[57] 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.

[58] R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. Technical Report TR-2002-123, University of Glasgow, Department of Computing Science, 2002. Revised May 2005.

[59] R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS '03: the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 2003.

[60] K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Proceedings of ICALP '99: the 26th International Colloquium on Automata, Languages, and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 1999.

[61] 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.

[62] K. Iwama, S. Miyazaki, and H. Yanagisawa. A 25/17-approximation algorithm for the stable marriage problem with one-sided ties. *Algorithmica*, 68:758–775, 2014.

[63] Y. Kamada and F. Kojima. Stability and strategy-proofness for matching with constraints: A problem in the Japanese medical match and its solution. *American Economic Review: Papers and Proceedings*, 102(3):366–370, 2012.

[64] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373 – 395, 1984.

[65] B.A. Kassa. A linear programming approach for placement of applicants to academic programs. *SpringerPlus*, 2(1):682, 2013.

[66] 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.

[67] D. Kazakov. Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science. Available from `http://www-users.cs.york.ac.uk/kazakov/papers/proj.pdf` (Accessed 25 March 2020), 2001.

[68] K.M. Keizer, M. de Klerk, B.J.J.M. Haase-Kromwijk, and W. Weimar. The Dutch algorithm for allocation in living donor kidney exchange. *Transplantation Proceedings*, 37:589–591, 2005.

[69] L.G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Academii Nauk SSSR*, 20:191 – 194, 1979.

[70] Z. Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60:3–20, 2011.

[71] Z. Király. Linear time local approximation algorithm for maximum stable marriage. *Algorithms*, 6(3):471–484, 2013.

[72] M. de Klerk, K.M. Keizer, F.H.J. Claas, M. Witvliet, B.J.J.M. Haase-Kromwijk, and W. Weimar. The Dutch national living donor kidney exchange program. *American Journal of Transplantation*, 5:2302–2305, 2005.

[73] D.E. Knuth. *Mariages Stables et leurs relations avec d'autres problèmes combinatoires*. Les Presses de L'Université de Montréal, 1976. English translation in *Stable Marriage and its Relation to Other Combinatorial Problems*, volume 10 of CRM Proceedings and Lecture Notes, American Mathematical Society, 1997.

[74] E. Kujansuu, T. Lindberg, and E. Mäkinen. The stable roommates problem and chess tournament pairings. *Divulgaciones Matemáticas*, 7(1):19–28, 1999.

[75] A. Kwanashie. *Efficient algorithms for optimal matching problems under preferences*. PhD thesis, University of Glasgow, School of Computing Science, 2015.

[76] A. Kwanashie, R.W. Irving, D.F. Manlove, and C.T.S. Sng. Profile-based optimal matchings in the Student–Project Allocation problem. In *Proceedings of IWOCA 2014: the 25th International Workshop on Combinatorial Algorithms,* to appear, Lecture Notes in Computer Science. Springer, 2015.

[77] A. Kwanashie and D.F. Manlove. An Integer Programming approach to the Hospitals/Residents problem with Ties. In *Operations Research Proceedings 2013*, pages 263–269. Springer, 2014.

[78] T. Lauri, K.Poder, and A. Veski. Matching practices in elementary schools - Estonia, MiP country profile 18. Web document available at `http://www.matching-in-practice.eu/elementary-schools-estonia`. (Accessed 25 March 2020).

[79] D. Lebedev, F. Mathieu, L. Viennot, A.-T. Gai, J. Reynier, and F. de Montgolfier. On using matching theory to understand P2P network design. In *Proceedings of INOC '07: International Network Optimization Conference*, 2007.

[80] C.L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, New York, 1968.

[81] D.F. Manlove. Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, January 1999.

[82] D.F. Manlove. The structure of stable marriage with indifference. *Discrete Applied Mathematics*, 122(1-3):167–181, 2002.

[83] D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

[84] 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.

[85] D.F. Manlove, I. McBride, and J. Trimble. "Almost-stable" matchings in the Hospitals/Residents problem with Couples. *Constraints*, pages 1–23, 2016.

[86] D.F. Manlove, D. Milne, and S. Olaosebikan. An Integer Programming Approach to the Student-Project Allocation Problem with Preferences over Projects. In *Proceedings of ISCO '18: the 5th International Symposium on Combinatorial Optimization*, volume 10856 of *Lecture Notes in Computer Science*, pages 313 – 325. Springer, 2018.

[87] D.F. Manlove and G. O'Malley. Student project allocation with preferences over projects. *Journal of Discrete Algorithms*, 6:553–560, 2008.

[88] D.F. Manlove and G. O'Malley. Paired and altruistic kidney donation in the UK: algorithms and experimentation. *ACM Journal of Experimental Algorithmics*, 19(1), 2014.

[89] D. Marx and I. Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010.

[90] D. Marx and I. Schlotter. Stable assignment with couples: parameterized complexity and local search. *Discrete Optimization*, 8:25–40, 2011.

[91] I. McBride. *Complexity and integer programming models for generalisations of the hospitals /residents problem*. PhD thesis, University of Glasgow, School of Computing Science, 2015.

[92] I. McBride and D.F. Manlove. An integer programming model for the Hospitals/Residents problem with Couples. In *Operations Research Proceedings*, pages 293–299. Springer, 2014.

[93] 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.

[94] 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.

[95] D.G. McVitie and L.B. Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

[96] National Resident Matching Program. Main residency match data and reports. Web document available at `http://www.nrmp.org/main-residency-match-data` (Accessed 25 March 2020).

[97] S. Olaosebikan and D. Manlove. Super-Stability in the Student-Project Allocation Problem with Ties. In *Proceedings of COCOA '18: 12th Annual International Conference on Combinatorial Optimization and Applications*, volume 11346 of *Lecture Notes in Computer Science*, pages 357 – 371. Springer, 2018.

[98] G. O'Malley. *Algorithmic Aspects of Stable Matching Problems*. PhD thesis, University of Glasgow, Department of Computing Science, 2007.

[99] Organ Donation and Transplantation Directorate, (NHS Blood and Transplant website). `http://www.organdonation.nhs.uk`. (Accessed 25 March 2020).

[100] K. Paluch. Faster and simpler approximation of stable matchings. *Algorithms*, 7(2):189–202, 2014.

[101] L. Pan, S.C. Chu, G. Han, and J.Z. Huang. Multi-criteria student project allocation: A case study of goal programming formulation with dss implementation. In *Proceedings of ISORA 2009: The Eight International Symposium on Operations Research and Its Applications*, pages 75 – 82. Zhangjiajie, China, 2009.

[102] B.G. Pittel and R.W. Irving. An upper bound for the solvability probability of a random stable roommates instance. *Random Structures and Algorithms*, 5:465–486, 1994.

[103] L.G. Proll. A simple method of assigning projects to students. *Operational Research Quarterly*, 23(2):195–201, 1972.

[104] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[105] 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.

[106] A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.

[107] A.E. Roth. New physicians: A natural experiment in market organization. *Science*, 250:1524–1528, 1990.

[108] A.E. Roth. A natural experiment in the organization of entry level labor markets: Regional markets for new physicians and surgeons in the U.K. *American Economic Review*, 81:415–440, 1991.

[109] A.E. Roth, T. Sönmez, and M.U. Ünver. Efficient kidney exchange: Coincidence of wants in a market with compatibility-based preferences. *American Economic Review*, 97(3):828–851, 2007.

[110] 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.

[111] 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.

[112] H.M. Saber and J.B. Ghosh. Assigning students to academic majors. *Omega*, 29(6):513 – 523, 2001.

[113] S. Scott. *A study of stable marriage problems with ties*. PhD thesis, University of Glasgow, Department of Computing Science, 2005.

[114] B. Spieker. The set of super-stable marriages forms a distributive lattice. *Discrete Applied Mathematics*, 58:79–84, 1995.

[115] D. Srinivasan and L. Rachmawati. Efficient fuzzy evolutionary algorithm-based approach for solving the student project allocation problem. *IEEE Transactions on Education*, 51(4):439 – 447, 2008.

[116] C.Y. Teo and D.J. Ho. A systematic approach to the implementation of final year project in an electrical engineering undergraduate course. *IEEE Transactions on Education*, 41(1):25–30, 1998.

[117] C. Terrier. Matching practices for secondary public school teachers - France, MiP country profile 20. Web document available at `http://www.matching-in-practice.eu/`

matching-practices-of-teachers-to-schools-france/. (Accessed 25 March 2020).

[118] M. Thorn. A constraint programming approach to the student-project allocation problem. BSc Honours project report, University of York, Department of Computer Science, 2003.

[119] S. Varone and D. Schindl. Course opening, assignment and timetabling with student preferences. In *Proceedings of ICORES: International Conference on Operations Research and Enterprise Systems*, 2013.

[120] J.E. Vande Vate. Linear programming brings marital bliss. *Operations Research Letters*, 8(3):147–153, 1989.

[121] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[122] H. Zhang. *An analysis of the Chinese college admission system*. PhD thesis, University of Edinburgh, School of Economics, 2009.

[123] L. Zhou. On a conjecture by Gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990.