Algorithmic Aspects of Stable Matching Problems

by

Gregg O'Malley

A thesis submitted to the
Faculty of Information and Mathematical Sciences
at the University of Glasgow
for the degree of
Doctor of Philosophy

August 2007

Abstract

The *Stable Marriage* problem (SM), the *Hospitals/Residents* problem (HR) and the *Stable Roommates* problem (SR) are three classical stable matching problems that were first studied by Gale and Shapley in 1962. These problems have widespread practical application in centralised automated matching schemes, which assign applicants to posts based on preference lists and capacity constraints in both the UK and internationally. Within such schemes it is often the case that an agent's preference list may be incomplete, and agents may also be allowed to express indifference in the form of ties. In the presence of ties, three stability criteria can be defined, namely weak stability, strong stability and super-stability. In this thesis we consider stable matching problems from an algorithmic point of view. Some of the problems that we consider are derived from new stable matching models, whilst others are obtained from existing stable matching models involving ties and incomplete lists, with additional natural restrictions on the problem instance. Furthermore, we also explore the use of constraint programming with both SM and HR.

We first study a new variant of the *Student-Project Allocation* problem in which each student ranks a set of acceptable projects in preference order and similarly each lecturer ranks his available projects in preference order. In this context, two stability definitions can be identified, namely weak stability and strong stability. We show that the problem of finding a maximum weakly stable matching is NP-hard. However, we describe two 2-approximation algorithms for this problem. Regarding strong stability, we describe a polynomial-time algorithm for finding such a matching or reporting that none exists.

Next we investigate SM with ties and incomplete lists (SMTI), and HR with ties (HRT), where the length of each agent's list is subject to an upper bound. We present both polynomial-time algorithms and NP-hardness results for a range of problems that are derived from imposing upper bounds on the length of the lists on one or both sides.

We also consider HRT, and SR with ties and incomplete lists (SRTI), where the preference lists of one or both sets of agents (as applicable) are derived from one or two master lists in which agents are ranked. For super-stability, in the case of each of HRT and SRTI with a master list, we describe a linear-time algorithm that simplifies the algorithm used in the general case. In the case of strong stability, for each of HRT and SRTI with a master list, we describe an algorithm that is faster than that for the general case. We also show that, given an instance I of SRTI with a master list, the problem of finding a weakly stable matching is polynomial-time solvable. However, we show that given such an I, the problem

of finding a maximum weakly stable matching is NP-hard.

Other new stable matching models that we study are the variants of SMTI and SRTI with symmetric preferences. In this context we consider two models that are derived from alternative ways of interpreting the rank of an agent in the presence of ties. For both models we show that deciding if a complete weakly stable matching exists is NP-complete. Then for one of the models we show that each of the problem of finding a minimum regret and an egalitarian weakly stable matching is NP-hard and that the problem of determining if a (man,woman) pair belongs to a weakly stable matching is NP-complete. We then describe algorithms for each of the problems of finding a super-stable matching and a strongly stable matching, or reporting that none exists, given instances of SRTI and HRT with symmetric preferences (regardless of how the ranks are interpreted).

Finally, we use constraint programming techniques to model instances of SM and HR. We describe two encodings of SM in terms of a constraint satisfaction problem. The first model for SM is then extended to the case of HR. This encoding for HR is then extended to create a model for HRT under weak stability. Using this encoding we can obtain, with the aid of search, all the weakly stable matchings, given an instance of HRT.

Contents

List of Figures

List of Algorithms

Acknowledgements

First I would like to thank David Manlove for four years of help, guidance and encouragement. There is no doubt that this thesis benefited greatly from his rigorous approach and attention to detail. David's constant stream of ideas is inspiring, and I can only thank him for the time he has given up over the years to help me. I would also like to thank Rob Irving, my second supervisor, for providing great advice relating to both the work contained in this thesis and for his general guidance. Thanks also to my examination committee, Paul Dunne and Patrick Prosser, who provided some excellent feedback.

The Ph.D. was supported for the first two years by EPSRC (Engineering and Physical Sciences Research Council) and third year by the Department of Computer Science, University of Glasgow.

Thanks also goes to the "team" of friends who proof read my thesis. Both Scott Pennock and Stuart Cotterell did a fantastic job covering several chapters of this thesis in great detail. I would like to give a special thanks to Alastair Donaldson who carried out the phenomenal task of reading every chapter from beginning to end in fine detail. All the comments passed on were invaluable.

I would also like to dedicate this to both my Dad and my Gran who sadly passed away while I was undertaking this study. There is no doubt I would not be the person I am today without them, and their encouragement is everything I could have ever hoped for. My girlfriend Liz McLeod also deserves a round of applause for sticking with me through this. She has encouraged me endlessly and supported me throughout. My brother has also been great at passing ideas by along the way. Finally, and most importantly, I would like to thank my Mum. She is the one who made me believe all this could happen, supported me, and never had any doubts. A third supervisor if ever there was one.

Declaration

This thesis is submitted in accordance with the rules for the degree of Doctor of Philosophy at the University of Glasgow in the Faculty of Information and Mathematical Sciences. None of the material contained herein has been submitted for any other degree. The material that appears in Section 2.2.2 and the proof of Theorem 2.2.8 is due to David Manlove. Additionally the proof of Lemma 2.2.5 was undertaken jointly with him. Finally the constraints found in Tables 7.3, 7.4, and 8.1 were developed jointly with David Manlove, however all related proofs are original. Otherwise the results contained herein are claimed

to be original.

Publications and papers submitted for publication

1. D.F. Manlove and G. O'Malley. Student-project allocation with preferences over projects. In *Proceedings* of *ACiD 2005: the 1st Algorithms and Complexity in Durham workshop*, volume 4 of Texts in Algorithmics, pages 69-80, *KCL Publications*, 2005. (This paper is based on Chapter 2.)

2. D.F. Manlove and G. O'Malley. Student-project allocation with preferences over projects. To appear in *Journal of Discrete Algorithms*, 2007. (This paper is based on Chapter 2, and is a fully revised and extended paper of Paper 1.)

3. R.W. Irving, D.F. Manlove, and G. O'Malley. Stable marriage with ties and bounded length preference lists. In *Proceedings* of *ACiD 2006: the 2nd Algorithms and Complexity in Durham workshop*, volume 7 of Texts in Algorithmics, pages 95-106, *College Publications*, 2006. (This paper is based on Chapter 3.)

4. R.W. Irving, D.F. Manlove, and G. O'Malley. Stable marriage with ties and bounded length preference lists. Submitted to *Journal of Discrete Algorithms*, 2006. (This paper is based on Chapter 3, and is a fully revised and extended paper of Paper 3.)

5. D.J. Abraham, A. Levavi, D.F. Manlove and G. O'Malley. The Stable Roommates Problem with Globally-Ranked Pairs. To appear in Proceedings of *WINE 2007: the 3rd International Workshop On Internet and Network Economics*, Lecture Notes in Computer Science, *Springer*, 2007. (This paper is based on Chapter 6.)

6. D.F. Manlove and G. O'Malley. Modelling and solving the stable marriage problem using constraint programming. In *Proceedings of the Fifth Workshop on Modelling and Solving Problems with Constraints*, held at IJCAI '05, pages 10-17, 2005. (This paper is based on Chapter 7.)

7. D.F. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A Constraint Programming approach to the Hospitals/Residents problem. In *Proceeding of the Fourth Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, held at CP '05, pages 28-43, 2005. (This paper is based on Chapter 8.)

8. D.F. Manlove, G. O'Malley, P. Prosser, and C. Unsworth. A Constraint Programming approach to the Hospitals/Residents problem. In *Proceedings of CP-AI-OR 2007: the Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 4510 of *Lecture Notes in Computer Science*, pages 155-170, Springer, 2007. (This paper is based on Chapter 8.)

Chapter 1

Review of Stable Matching Problems

Matching problems can be found at the heart of a wide variety of important large-scale practical applications. In a matching problem we seek to assign a set of agents to one another, typically subject to constraints involving preference lists and capacities. The preference list of an agent contains the set of agents, listed in order of preference, that he is prepared to become assigned to. The capacity of an agent is the maximum number of agents that he can become assigned to. Stability of a matching is widely accepted to be a desirable property [63], and ensures that no two agents would rather be assigned together than remain with their current assignees. A matching that satisfies a stability criterion is said to be a *stable matching*. Example real-world instances of stable matching problems include assigning medical students to hospital posts, assigning children to schools and in kidney exchange schemes.

In this chapter we discuss several stable matching problems. We define in Section 1.1 the *Stable Marriage* problem, the first stable matching problem to be formally studied in the literature. The Stable Marriage problem involves two sets of agents, namely men and women, where the men rank the women in strict order of preference, and similarly the women rank the men in strict order of preference. We then describe in Section 1.2 the many-to-one generalisation of the Stable Marriage problem known as the *Hospitals/Residents* problem. Here we have a set of residents, each of whom seeks to be assigned to a hospital post, where a hospital may have multiple posts. In this case we require a matching (i.e. each resident is assigned to at most one hospital and no hospital is over-

subscribed) that is stable. Next we introduce in Section 1.3 the *Student-Project Allocation* problem, a generalisation of the Hospitals/Residents problem. An instance of the Student-Project Allocation problem consists of a set of students, a set of projects, and a set of lecturers. In the model that we describe in this chapter, each student ranks a subset of the projects offered by the lecturers in strict order of preference, and each lecturer ranks in strict preference order the appropriate set of students. Additionally, each project has a capacity indicating the maximum number of students who can undertake the project, and each lecturer has a capacity indicating an upper bound on the number of students he wishes to supervise. Finally, we review in Section 1.4 the *Stable Roommates* problem, the non-bipartite generalisation of the Stable Marriage problem. An instance of the Stable Roommates problem consists of a set of agents, each of whom rank one another in strict order of preference.

For each of the stable matching problems that we discuss, we define the problem formally, give an example instance, and then state the key algorithmic results concerned. The discussion of stable matching problems as presented in Sections 1.1-1.4 then leads to a review in Section 1.5 of the problems that we consider in subsequent chapters, including a summary of the main contributions of this thesis.

1.1 Stable Marriage Problem

1.1.1 The Gale-Shapley algorithm

In 1962 David Gale and Lloyd Shapley published their paper 'College admissions and the stability of marriage' [18]. This paper was the first to formally define the Stable Marriage problem (SM), and provide an algorithm for its solution. An instance I of SM involves n men and n women, each of whom ranks all n members of the opposite sex in strict order of preference. In I we denote the set of men by $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ and the set of women by $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$. In SM the preference lists are said to be *complete*, that is each member of I ranks every member of the opposite sex.

We seek to find a *matching* M (a bijection from $\mathcal{M}$ to $\mathcal{W}$) from the men to the women. If $(m, w) \in M$, we say that a man m is *matched to* a woman w in M and w is *matched to* m in M. Also if $(m, w) \in M$, we say that w is m's *partner* in M and m is w's *partner* in M. An *assignment* A is a set of (man, woman) pairs $(m, w) \in \mathcal{M} \times \mathcal{W}$; we note that an assignment need not be a matching. Similar to the notation used with a matching, if

$(m, w) \in A$ we say that m is *assigned to* a woman w in A and w is *assigned to* m in A. Again if $(m, w) \in A$ we say that m is w's *partner* in A and w is m's *partner* in A. Let $A(p)$ denote p's partners in A, where $p \in \mathcal{M} \cup \mathcal{W}$. If $A(p) \neq \emptyset$, then we say that p is *assigned* in A, otherwise p is *unassigned* in A.

A pair $(m, w) \in \mathcal{M} \times \mathcal{W}$ *blocks* a matching M, or is a *blocking pair*, if m prefers w to $M(m)$ and w prefers m to $M(w)$. A matching is *stable* if there exists no blocking pair. Stability as a criterion for a matching ensures that no party can seek to improve outside of the matching scheme, as there is no incentive for any one agent to improve. We also say that (m, w) are a *stable pair* if m and w are matched to one another in some stable matching. An example instance of SM is shown in Figure 1.1 (we use the convention that preference lists are ordered from left to right in decreasing preference order throughout this thesis). One possible stable matching in instance I_1 of Figure 1.1 is $M = \{(m_1, w_1), (m_2, w_3), (m_3, w_2)\}$.

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

Figure 1.1: Instance I_1 of SM.

The algorithm presented by Gale and Shapley for finding a stable matching uses a simple "deferred acceptance" strategy, comprising proposals and rejections. There are two possible 'orientations', depending on who makes the proposals, namely the *man-oriented* algorithm and the *woman-oriented* algorithm. In the man-oriented algorithm, each man m proposes in turn to the first woman w on his list to whom he has not previously proposed. If w is free, then she becomes engaged to m. Otherwise, if w prefers m to her current fiancé m', she rejects m', who becomes free, and w becomes engaged to m. Otherwise w prefers her current fiancé to m, in which case w rejects m, and m remains free. This process is repeated while some man remains free. For the woman-oriented algorithm the process is similar, only here the proposals are made by the women.

The man-oriented and woman-oriented algorithms return the *man-optimal* and *woman-optimal* stable matchings respectively. The man-optimal stable matching has the property that each man obtains his best possible partner in any stable matching. However, while each man obtains his best possible partner, each woman simultaneously obtains her worst

possible partner in any stable matching. Correspondingly, when the woman-oriented algorithm is applied, each woman gets her best possible partner while each man get his worst possible partner in any stable matching.

Consider the SM instance I_2 of size 4 shown in Figure 1.2. Two possible matchings for instance I_2 are:

$$M_0 = \{(m_1, w_4), (m_2, w_1), (m_3, w_2), (m_4, w_3)\}$$

and,

$$M_z = \{(m_1, w_2), (m_2, w_1), (m_3, w_4), (m_4, w_3)\}$$

Matchings M_0 and M_z denote the man-optimal and woman-optimal stable matchings for instance I_2 respectively. For instance I_1 shown in Figure 1.1, the man-optimal stable matching and the woman-optimal stable matchings are the same, namely:

$$M_0 = M_z = \{(m_1, w_1), (m_2, w_3), (m_3, w_2)\}$$

and hence $M_0 = M_z$ is the unique stable matching in I_1.

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

Figure 1.2: Instance I_2 of SM.

In [18] Gale and Shapley indicated that their algorithm involved at most $n^2 - 2n + 2$ stages. However, it was not until 14 years later that Knuth [48] showed that the time complexity of the Gale-Shapley algorithm is indeed $O(n^2)$. It is natural to consider the possibility of obtaining an improved lower bound. However, Ng and Hirschberg [59] showed that $\Omega(n^2)$ is, in fact, a lower bound for SM. This is shown in the following theorem.

Theorem 1.1.1. *Any algorithm to find a stable matching or to check if a given matching is stable or to determine whether a given pair is stable requires $\Omega(n^2)$ time in the worst case, even when both the preference lists and ranking arrays are given as input.*

1.1.2 Extended Gale-Shapley algorithm

To exploit the many structural properties of SM, an extended version of the Gale-Shapley algorithm (EGS algorithm) was developed that 'reduces' the preference lists of each

man/woman by making deletions from them [26, Section 1.2.4] (entries are only ever deleted from the preference list if they cannot be involved in a stable matching). The extended version of the Gale-Shapley algorithm is shown in Algorithm 1. Again the algorithm has two orientations, the man-oriented EGS (MEGS) algorithm and the woman-oriented EGS (WEGS) algorithm (the MEGS algorithm is shown in Algorithm 1). In Algorithm 1 "delete the pair (m, w)" is the operation of deleting both m from w's list and w from m's list.

Algorithm 1 Extended-GS

```
1: assign each person to be free;
2: while some man m is free do
3:     w := first woman on m's list;
4:     if some man p is assigned to w then
5:         assign p to be free;
6:     assign w to m;
7:     for each successor m' of m on w's list do
8:         delete the pair (m', w);
```

During the EGS algorithm all proposals are accepted. To understand why this can be done, suppose that some man m proposes to a woman w, with w currently being engaged to m'. Then she must prefer m to m', for otherwise the pair (m, w) would previously have been deleted.

When the algorithm terminates, i.e. when a matching has been found for a given instance, the reduced preference lists form what are known as the *man-oriented Gale-Shapley* lists, abbreviated to *MGS-lists* (all executions of the algorithm give the same MGS-lists). Similarly, the reduced preference lists obtained using the woman-oriented Gale-Shapley algorithm are known as the *woman-oriented Gale-Shapley* lists, or *WGS-lists*. Taking the intersection (i.e. the entries that are common to both lists) of the MGS-lists and the WGS-lists yields the *GS-lists*. The theorem below, taken from [26], provides a summary of the properties of the GS-lists.

Theorem 1.1.2. *For a given instance of the stable marriage problem:*

(i) all stable matchings are contained in the GS-lists;

(ii) no matching contained in the GS-lists can be blocked by a pair that is not in the GS-lists;

(iii) in the man-optimal (respectively woman-optimal) stable matching, each man is partnered by the first (respectively last) woman on his GS-list, and each woman by the last (respectively first) man on hers.

1.1.3 Optimal Stable Marriage problems

In this section we define two variants of SM in which we seek to find a stable matching that is "optimal" in a precise sense.

We define the *rank* of an agent p on an agent q's list, denoted by $rank(p, q)$, to be the position of q on p's list. Let I be an instance of SM, where $\mathcal{M}$ is the set of men and $\mathcal{W}$ is the set of woman in I. Let M be a stable matching in I, and let p be some agent in I. We define the *cost* of p with respect to M, denoted by $cost_M(p)$, to be $rank(p, M(p))$. Furthermore we define the *regret* of M by:

$$r(M) = \max_{p \in \mathcal{M} \cup \mathcal{W}} cost_M(p).$$

We say that M has *minimum regret* if $r(M)$ is minimised over all stable matchings in I. Gusfield [25] described an $O(n^2)$ algorithm that finds a minimum regret stable matching given an instance of SM.

We now define the *cost* of a matching M by:

$$c(M) = \sum_{p \in \mathcal{M} \cup \mathcal{W}} cost_M(p).$$

An *egalitarian* stable matching M is a stable matching such that $c(M)$ is minimised over all stable matchings in I. The problem of finding an egalitarian stable matching was first posed by Knuth [47], with Irving et al. [37] describing an $O(n^4)$ algorithm for the problem. Feder [14] later described the fastest current algorithm for finding an egalitarian stable matching, which runs in time $O(n^3)$.

1.1.4 Stable Marriage with Incomplete Lists

In the context of SM, it is possible that an agent may find a member of the opposite sex *unacceptable*, and hence this person does not appear on their preference list. If a woman w_j appears on a man m_i's list, then we say that m_i finds w_j *acceptable* (and vice-versa). This gives rise to the *Stable Marriage problem with Incomplete lists*, or SMI for short. Consider the instance of SMI shown in Figure 1.3.

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

Figure 1.3: SMI instance I_1.

In Figure 1.3 man m_1 finds w_2 unacceptable, and as a result w_2 does not appear on the preference list of m_1. However m_2 finds all the women acceptable, while m_3 finds only w_3 acceptable. In general, preference lists are *consistent* if q is deleted from p's preference list implies that p is also deleted from q's preference list. In an SMI instance I we assume that the preference lists in I are consistent.

A matching M is a partial injective function from $\mathcal{M}$ to $\mathcal{W}$ such that $(m, w) \in M$ only if m and w find each other acceptable. Let $M(p)$ denote p's partner in M, where $p \in \mathcal{M} \cup \mathcal{W}$. If $M(p) \neq \emptyset$, then we say that p is *matched* in M, otherwise p is *unmatched* in M.

Allowing unacceptable partners means that the definition of stability for SMI has to be altered slightly from the case of SM. A pair (m, w) *blocks* a matching M, or is a *blocking pair* of M, if:

(i) m and w are not matched in M, but m and w find each other acceptable.

(ii) m is either unmatched in M, or prefers w to his partner in M.

(iii) w is either unmatched in M, or prefers m to her partner in M.

The EGS algorithm described Section 1.1.2 can easily be adapted to handle an instance of SMI. However it is possible that in a stable matching with respect to a given instance of SMI, a man or woman may be unmatched. This leads to an interesting result due to Gale and Sotomayor [19], shown below.

Theorem 1.1.3. *In an instance of* SMI*, the same set of men and women are matched in all stable matchings.*

1.1.5 Stable Marriage: a constraint programming approach

Appendix A gives an overview of constraint programming (CP) and indicates that CP can be useful when dealing with problems that are known to be computationally hard. As

we have seen, finding a stable matching for an instance of SM is polynomial-time solvable using the Gale-Shapley algorithm. However, there has also been interest in obtaining a similar bound to that achieved by the Gale-Shapley algorithm using arc consistency (AC) applied to a CSP encoding of SM. Furthermore, there are many variants of SM that are NP-hard [54, 60, 62], and the encodings described here could potentially be extended to solve such variants.

Two encodings for an instance I of SMI are presented in [20]. The first encoding uses a set of 'conflict' matrices to represent the constraints. This model produces $O(n^2)$ conflict matrices, each having size $O(n^2)$, giving the encoding an overall size of $O(n^4)$, where n is the number of men and women. The authors also show that after AC propagation the variables' domains correspond, in a precise way, to the GS-lists of I. However, the size of the encoding means that forcing AC propagation is achieved in $O(n^4)$ time, resulting in a poorer time complexity than running the Gale-Shapley algorithm on the original SM instance.

The second encoding presented in [20] takes the form of a Boolean encoding (an encoding where the domains of the variables are 0 and 1). The encoding itself is more complex than the conflict matrices approach, however this results in a more compact model using $O(n^2)$ space, and AC is established in $O(n^2)$ time. Thus the encoding is asymptotically optimal. In contrast to the conflict matrices encoding, the variables' domains after AC propagation do not, in general, correspond to the GS-lists. Instead, the domains correspond to a weaker structure called the *Extended GS-lists*, or *XGS-lists*. The XGS-list of person p contains all persons on p's preference list between his partners in M_0 and M_z (inclusive), i.e. it yields the bounds on the GS-lists. In general the XGS-lists are supersets of the GS-lists and need not be consistent.

1.1.6 Stable Marriage with Ties

In this section we consider the effects of allowing an agent to be *indifferent* between two or more agents; indifference here takes the form of preference lists with ties. When the lists are complete we denote this variant of SM by SMT. Consider the instance of SMT shown in Figure 1.4. Here m_1 is indifferent between w_1, w_2 and w_3, whilst m_3 strictly prefers w_3 to each of w_1 and w_2, whom he is indifferent between.

The introduction of ties in a participant's preference list gives rise to three definitions of stability, namely *weak stability*, *strong stability*, and *super-stability*. The different ver-

Men's preferences	Women's preferences
$m_1:\ (w_1\ \ w_2\ \ w_3)$	$w_1:\ (m_1\ \ m_2)\ \ m_3$
$m_2:\ w_1\ \ w_3\ \ w_2$	$w_2:\ m_2\ \ (m_3\ \ m_1)$
$m_3:\ w_3\ \ (w_1\ \ w_2)$	$w_3:\ m_1\ \ m_2\ \ m_3$

Figure 1.4: SMT instance I_2.

sions of stability describe, as their names suggest, how resistant a matching is to being undermined by pairs of participants. Definitions are now given for a blocking pair for each form of stability. A pair (m, w) is said to *block* a matching M, and is called a *blocking pair* if:

- weak stability – both m and w strictly prefer each other to their partners in M.
- strong stability – Either:
  - (i) m strictly prefers w to his partner in M, and w either strictly prefers m to her partner in M or is indifferent between them, or
  - (ii) w strictly prefers m to her partner in M, and m either strictly prefers w to his partner in M or is indifferent between them.
- super-stability – each of m and w either strictly prefers the other to their partner in M or is indifferent between them.

Let M be a matching. If there exists no blocking pair with respect to M, then M is said to be *weakly stable*, *strongly stable* and *super-stable* with respect to the above definitions. We also observe that a super-stable matching is strongly stable and a strongly stable matching is weakly stable.

Allowing an agent to be indifferent between a set of agents brings with it many new and interesting problems. With the classical Gale-Shapley algorithm, a stable matching can always be found given an instance of SM and SMI. However, with regards to strong stability and super-stability, it is possible that a strongly stable matching or a super-stable matching need not exist, given an instance of SMT. For example, Figure 1.5 shows an instance of SMT where no strongly stable or super-stable matching exists. A weakly stable matching can always be found for an instance of SMT simply by breaking the ties arbitrarily and applying the Gale-Shapley algorithm to this derived instance I' of SM. This produces a matching that is weakly stable in the original instance with ties.

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

Figure 1.5: SMT instance I.

An alternative way of describing the above stability criteria is given below. Let M be a matching for an instance I of SMT. Then:

- weak stability – M is weakly stable if and only if M is stable in some instance of SM obtained from I by breaking the ties.
- strong stability – M is strongly stable if and only if:
  (i) There is some instance I' of SMT obtained from I by breaking the ties on the men's side, such that for every instance of SM obtained from I' by breaking the ties (on the women's side), M is stable, and
  (ii) There is some instance I' of SMT obtained from I by breaking the ties on the women's side, such that for every instance of SM obtained from I' by breaking the ties (on the men's side), M is stable.
- super-stability – M is super-stable if and only if M is stable in every instance of SM obtained from I by breaking the ties.

Irving [35] describes an $O(n^4)$ algorithm that finds a strongly stable matching, or reports that no such matching exists, and an $O(n^2)$ algorithm that finds a super-stable matching, or reports that no such matching exists. Both algorithms use a similar "deferred acceptance" strategy to that used in the extended Gale-Shapley algorithm. The super-stability algorithm is a straightforward extension of the Gale-Shapley algorithm, whilst the strong stability algorithm is more elaborate than its super-stable counterpart. Detailed discussions of these algorithms are deferred until Section 1.2.5, where more general versions of these algorithms are presented, with respect to the so-called as the Hospitals/Residents problem.

1.1.7 Stable Marriage with Ties and Incomplete Lists

By combining the extensions SMI and SMT of the classical stable marriage problem, we obtain the Stable Marriage problem with Ties and Incomplete lists, or SMTI for short.

A blocking pair in the context of SMTI is defined by combining the definitions given in Sections 1.1.4 and 1.1.6. Again for strong stability and super-stability, a matching satisfying either of these criteria still need not exist. Manlove [53], however, describes two polynomial-time algorithms to determine if a strongly stable or a super-stable matching exist, and to find such a matching if one does.

Theorem 1.1.3 shows that, given an instance of SMI, all stable matchings have the same size. Similarly, for any instance I of SMT, all weakly, strongly and super-stable matchings have the same size. However, in the case of SMTI, this is no longer true for weak stability. For example, consider the instance I_1 shown in Figure 1.6. Here two possible weakly stable matchings $M = \{(m_1, w_1), (m_2, w_2)\}$ and $M' = \{(m_2, w_1)\}$ have different sizes. It is worth noting that all strongly stable matchings for an SMTI instance have the same size, as is the case with all super-stable matchings [53], should such a matching exist.

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

Figure 1.6: Instance I_1 of SMTI with weakly stable matchings of different cardinality.

With the possibility of weakly stable matchings having different sizes for a given SMTI instance, it is natural to consider the problem of finding a weakly stable matching with maximum cardinality; we denote this problem by MAX-SMTI. The decision problem for MAX-SMTI is defined below.

Name:	MAX-SMTI-D
Instance:	An SMTI instance I, and integer K.
Question:	Does I have a weakly stable matching of size $\geq K$?

It is known that MAX-SMTI-D is NP-complete, even if the ties are at the tails of the lists and on one side only, there is at most one tie per list, and each tie is of length two. This result, proved by Manlove et al. [54], gives a strong indication that the existence of an efficient algorithm for finding a maximum cardinality weakly stable matching for an instance of SMTI is unlikely.

It is also useful to consider the problem of finding a *complete* weakly stable matching. That is, finding a weakly stable matching in which every man and every woman is matched – we assume that the number of men and women are equal in this case. We denote this

problem by COM-SMTI, and note the fact that the NP-completeness proof for MAX-SMTI holds even if $K = n$, and therefore also shows-that COM-SMTI is NP-complete. The problem is now defined formally below.

Name:	COM-SMTI
Instance:	An SMTI instance I.
Question:	Does I have a complete weakly stable matching?

As a result of the NP-hardness of MAX-SMTI it is natural to consider the use of approximation algorithms. We use the following notation when discussing approximation algorithms: let $OPT(I)$ denote the value of an optimal solution for some instance I of an optimisation problem X, and let A be an approximation algorithm for X. We denote by $A(I)$ the value of a feasible solution returned by A for instance I. Then A has a performance guarantee of c, for some $c \geq 1$, if:

- X is a minimisation problem, and $A(I) \leq c \times OPT(I)$ for all instances I, or
- X is a maximisation problem, and $A(I) \geq (1/c) \times OPT(I)$ for all instances I.

In each of these cases A is said to be a c-approximation algorithm.

For MAX-SMTI, a 2-approximation algorithm was given by Manlove et al. [54] for the general case. Recently improved performance guarantees have been presented for various cases of MAX-SMTI [28, 29, 43], with the best currently standing at 1.875.

The inclusion of both ties and incomplete lists has added some interesting behaviour to the stable marriage problem. Firstly, an instance of SMT/SMTI may not admit a strongly stable or super-stable stable matching. Secondly, although a weakly stable matching can always be found for an instance I of SMTI, two weakly stable matchings may be of different sizes and it is NP-hard to find the largest weakly stable matching. In later sections we will see that this is also the case for many other stable matching problems.

1.2 Hospitals/Residents Problem

In their seminal paper [18], Gale and Shapley introduced a many-to-one generalisation of SM called the Hospitals/Residents problem (HR). At the time this was known as the College Admissions problem, but latterly has become known as the Hospitals/Residents problem. This section discusses HR in further detail.

1.2.1 Introduction

As mentioned above, HR is a many-to-one generalisation of SM. The problem takes its name from the application of assigning graduating medical students (residents) to hospital posts. In a number of countries this task is carried out using a centralised matching scheme that has at its heart an algorithm for HR. The US has one of the largest such automated schemes, namely the National Resident Matching Program (NRMP) [61]. This allocates around 31,000 residents to hospital posts every year. In Scotland, the Scottish Foundation Allocation Scheme (SFAS) is also used to allocate graduating medical students to hospital posts [36, 68].

The algorithm employed at the heart of the NRMP is essentially an extension of the Gale-Shapley algorithm. Given that the NRMP has been in existence since 1952, its algorithm therefore pre-dates the Gale-Shapley algorithm for SM by 10 years. This was noted by Roth in [63].

To further understand why stability and centralised matching schemes are important, we consider an alternative system whereby we have an informal "free-for-all", where students must approach a hospital and negotiate undertaking an available post at the hospital. It is known that a "free-for-all" approach may not be to the benefit of all students and hospitals, as this strategy typically involves a race for hospital posts. As such, this approach quickly descends into chaos: many hospitals are faced with the problem of a student accepting a post it offers, which the student later rejects if they discover a post at another hospital that they prefer. This leads to an undesirable process whereby students are continually accepting and rejecting post until no further switches arise, and such a process is clearly undesirable.

In HR, each hospital has one or more posts that it requires to fill, and a preference list ranking a subset of the residents. Similarly, each resident has a preference list ranking a subset of the hospitals. The capacity of a hospital is its number of available posts. We require to match each resident to at most one hospital such that no hospital exceeds its capacity, whilst observing the appropriate stability criterion to be defined. An instance of HR is defined formally as follows:

- set of residents $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$.
- set of hospitals $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$.
- preference list for all $r_i \in \mathcal{R}$, each of whom ranks a subset of $\mathcal{H}$ in strict order.

Residents' preferences

r_1 : h_1 h_3

r_2 : h_1 h_5 h_4 h_3

r_3 : h_1 h_2 h_5

r_4 : h_1 h_2 h_4

r_5 : h_3 h_1 h_2

r_6 : h_3 h_2 h_1 h_5

r_7 : h_3 h_4 h_5 h_1

r_8 : h_5 h_4

Hospitals' preferences

h_1 : (2) : r_3 r_7 r_5 r_2 r_4 r_6 r_1

h_2 : (3) : r_5 r_6 r_3 r_4

h_3 : (1) : r_2 r_5 r_6 r_1 r_7

h_4 : (1) : r_8 r_2 r_4 r_7

h_5 : (1) : r_3 r_7 r_6 r_8 r_2

Figure 1.7: HR instance I_1.

- preference list for all $h_j \in \mathcal{H}$, each of whom ranks its *applicants*, the residents who find that particular hospital acceptable, in strict order.
- list of capacities c_j $(1 \leq j \leq m)$ for each hospital.

We say that a resident r_i finds a hospital h_j *acceptable* if r_i's preference list contains h_j, and h_j finds r_i *acceptable* if h_j's preference list contains r_i. An *assignment* M for an instance I of HR is a set of (resident,hospital) pairs $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ such that $(r_i, h_j) \in M$ only if r_i and h_j find each other acceptable. If $(r_i, h_j) \in M$, we say that r_i is *assigned to* h_j, and h_j is *assigned* r_i. For any $p \in \mathcal{R} \cup \mathcal{H}$, we denote by $M(p)$ the set of assignees of p in M. If $M(p) \neq \emptyset$ we say that p is *assigned* in M, otherwise r is *unassigned* in M. Where there is no ambiguity we use $M(r_i)$ to denote the single hospital assigned to r_i in M.

Let M be an assignment for an instance I of HR. We say that a hospital $h_j \in \mathcal{H}$ is *under-subscribed*, *over-subscribed* or *full* in M when $|M(h_j)| < c_j$, $|M(h_j)| > c_j$, or $|M(h_j)| = c_j$ respectively.

A *matching* M in the context of HR, is a set of (resident, hospital) pairs such that no resident is assigned to more than one hospital and no hospital is over-subscribed.

In HR instance I_1 shown in Figure 1.2.1, $\mathcal{R} = \{r_1, r_2, \ldots, r_8\}$, $\mathcal{H} = \{h_1, h_2, \ldots, h_5\}$, and the capacity of each hospital is indicated in parenthesis.

A pair $(r, h) \in \mathcal{R} \times \mathcal{H}$ is said to *block* a matching M for an instance of HR, and is called a *blocking pair*, when all the following conditions are satisfied:

- h and r find each other acceptable;

- either r is unmatched, or prefers h to $M(r)$;
- either h is under-subscribed, or h prefers r to at least one of $M(h)$.

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

$M = \{(r_1, h_3), (r_2, h_1), (r_3, h_1), (r_4, h_2), (r_5, h_2), (r_6, h_2), (r_7, h_4), (r_8, h_5)\}$ is a matching for instance I_1. In M it can be seen that (r_5, h_3) blocks M as h_3 prefers r_5 to its worst assignee, namely r_1. A stable matching for instance I_1 is $M' = \{(r_2, h_1), (r_3, h_1), (r_4, h_2), (r_5, h_3), (r_6, h_2), (r_7, h_5), (r_8, h_4)\}$. The stability of M' can be easily verified.

1.2.2 The resident-oriented algorithm

Algorithm 2 Resident-oriented algorithm for HR

```
1:  M = ∅
2:  while (some resident r is free) and (r has a non-empty list) do
3:      h := first hospital on r's list; /** r 'proposes' to h */
4:      M = M ∪ {(r, h)};
5:      if h is over-subscribed then
6:          r' := worst resident assigned to h;
7:          M = M\{(r', h)};
8:      if h is full then
9:          r' := worst resident assigned to h;
10:         for each successor s of r' on the list of h' do
11:             delete the pair (s, h);
```

In this section we present an algorithm similar to that used by the NRMP. The algorithm, originally presented by Dubins and Freedman [11], is shown in Algorithm 2 and is called the *resident-oriented* (RGS) algorithm for HR. The algorithm uses an *apply* operation similar to a proposal step in the Gale-Shapley algorithm. While some resident r is free, he applies to the first hospital h on his list, and becomes provisionally assigned to h. If h becomes over-subscribed as a result of this assignment, the worst resident r' assigned to h is identified, and r' is assigned to be free. If hospital h is full, we again identify h's worst assigned resident r'. Then for each successor s of r' on h's list the pair (s, h) is deleted. Here "delete the pair (s, h)" means that s is deleted from h's list and h is deleted from s's list. The resident-oriented algorithm always terminates with a matching.

The RGS algorithm finds the *resident-optimal* stable matching M_0. In M_0 each resident is matched to its best possible hospital in any stable matching. In addition to this, the

deletions that occur as part of the RGS algorithm result in a reduced set of preference lists called the *RGS-lists*.

The resident-oriented algorithm, if implemented with suitable data structures, runs in time linear in the input size, i.e. $O(\lambda)$, where λ is the total length of the preference lists.

Applying Algorithm 2 to instance I_1 shown in Figure 1.2.1, generates the matching $M_0 = \{(r_2, h_1), (r_3, h_1), (r_4, h_2), (r_5, h_3), (r_6, h_2), (r_7, h_4), (r_8, h_5)\}$. The stability of M_0 can be easily verified.

1.2.3 The hospital-oriented algorithm

The *hospital-oriented* (HGS) algorithm (shown in Algorithm 3) again uses a similar strategy to the EGS algorithm. While some hospital h is under-subscribed and there exists a resident on h's list who is not already assigned to h, we choose the first such resident r on h's list. If r is already assigned, we break this assignment, and provisionally assign r to h. At this point r cannot obtain a hospital h' worse than h on his list, hence we can delete the pair (r, h') for any such hospital h'. Here "delete the pair (r, h')" means delete h' from r's list and r from h''s list. The reduced preference lists after termination of Algorithm 3 are known as the *HGS-lists* (Hospital Gale-Shapley lists). As in the case of SM, the intersection of the RGS-lists and the HGS-lists yields the GS-lists.

Algorithm 3 Hospital-oriented algorithm for HR

```
1: M = ∅;
2: while (some hospital h is under-subscribed) and
   (h's list contains a resident r not provisionally assigned to h) do
3:    r := first such resident on h's list;
4:    if r is already assigned, say to h' then
5:       M = M\{(r, h')};
6:    M = M ∪ {(r, h)};
7:    for each successor h' of h on r's list do
8:       delete the pair (r, h');
```

The matching generated by Algorithm 3 is simultaneously the best possible stable matching for all the hospitals [63]. It is known as the *hospital-optimal* stable matching, denoted by M_z. In the hospital-optimal stable matching, each under-subscribed hospital is assigned to all the residents on its reduced list, and each fully subscribed hospital with q places is assigned to the first q residents on its reduced list. Once again, as is the case

for the Gale-Shapley algorithm, if implemented with suitable data structures the hospital-oriented algorithm runs in time linear in the input size, i.e. $O(\lambda)$, where λ is the total length of the preference lists.

For HR instance I_1 shown in Figure 1.2.1. The hospital-optimal stable matching is $M_z = \{(r_2, h_3), (r_3, h_1), (r_4, h_2), (r_5, h_1), (r_6, h_2), (r_7, h_5), (r_8, h_4)\}$.

1.2.4 The Rural Hospitals Theorem

An interesting and important result with significant practical consequences is the *Rural Hospitals Theorem* for HR. The name arises from a pattern that developed when the NRMP matching scheme was run: hospitals in rural areas were more likely to be under-subscribed. This was due to residents finding hospitals in rural areas unattractive compared to those in cities, hence these hospitals typically appear at the tail of a resident's preference list or are even considered unacceptable. As a result of this the administrators of the NRMP wondered if changing the algorithm used, to find a different stable matching, could push more people into rural hospitals. However, it was shown that no matter what stable matching algorithm the NRMP had chosen, each under-subscribed hospital would end up with the same set of residents. This result is known as the Rural Hospitals Theorem, and is shown in Theorem 1.2.1.

Theorem 1.2.1. *For a given hospital's/resident's instance:*
(i) each hospital is assigned the same number of residents in all stable matchings;
(ii) exactly the same residents are unassigned in all stable matchings;
(iii) any hospital that is under-subscribed in one stable matching is assigned precisely the same set of residents in all stable matchings.

The results that make up the Rural Hospitals Theorem were proved by Gale and Sotomayor [19] (parts (i) and (ii)) and Roth [64] (part (iii)).

1.2.5 Hospitals/Residents with Ties

Just as ties were introduced to an instance of SM, we also consider an instance of HR in which agent's preference lists may contain ties, obtaining the Hospitals/Residents problem with Ties (HRT). Again a matching is *stable* if it admits no blocking pair, and we can define a blocking pair for an instance of HRT with respect to the three levels of stability introduced in Section 1.1.6.

A pair $(r, h) \in \mathcal{R} \times \mathcal{H}$ is said to *block* a matching M for an instance of HRT, and is called a *blocking pair* when:

- weak stability
  - (i) r, h find each other acceptable;
  - (ii) r is either unassigned or strictly prefers h to his assigned hospital in M;
  - (iii) h either is under-subscribed or strictly prefers r to its worst assigned resident in M.
- strong stability
  - (i) r, h find each other acceptable;
  - (ii) either,
    - (a) r is either unassigned or strictly prefers h to his assigned hospital in M, and h is either under-subscribed or strictly prefers r to its worst assigned resident in M or is indifferent between them; or
    - (b) r is either unassigned or strictly prefers h to his assigned hospital in M or is indifferent between them, and h is either under-subscribed or strictly prefers r to its worst assigned resident in M.
- super-stability
  - (i) r, h find each other acceptable;
  - (ii) r is either unassigned or strictly prefers h to his assigned hospital in M or is indifferent between them;
  - (iii) h is either under-subscribed or strictly prefers r to its worst assigned resident in M or is indifferent between them.

A matching is said to be be *weakly stable*, *strongly stable* or *super-stable* if it admits no blocking pair with respect to the relevant definitions above.

As HRT is a generalisation of SMTI, the NP-hardness result for finding a maximum weakly stable matching generalises to the case of finding a maximum weakly stable matching for an instance of HRT. However, once again finding a super-stable matching and a strongly stable matching, if such a matching exists, is polynomial-time solvable, as discussed in detail in the following sections.

HRT **under super-stability**

Algorithm 4 HRT-`super-res`

```
1: M = ∅;
2: for each h ∈ H do
3:    full(h) := false;
4: while some resident r is free and has a non-empty list do
5:    for each hospital h at the head of r's list do
6:       M = M ∪ {(r, h)};
7:       if h is over-subscribed then
8:          for each resident s' at the tail of h's list do
9:             if s' is provisionally assigned to h then
10:               M = M\{(s', h)};
11:            delete the pair (s', h);
12:      if h is full then
13:         full(h) := true;
14:         s := worst resident provisionally assigned to h;
15:         for each strict successor s' of s on h's list do
16:            delete the pair (s', h);
17: if some resident is multiply assigned or
    (some hospital h is under-subscribed and full(h)) then
18:    no super-stable matching exists;
19: else
20:    M is a super-stable matching;
```

The algorithm used for SMT under super-stability, discussed in Section 1.1.7, has been superseded by an algorithm for HRT under super-stability. Algorithm 4 shows algorithm HRT-`super-res` presented in [39] for finding a super-stable matching, or reporting that none exists, given an instance of HRT.

The algorithm proceeds as following: while some resident r is free and has a non-empty list, he becomes provisionally assigned to the set of hospitals H at the head of his list. Let $h \in H$. Then if, as a result of these provisional assignments, h becomes over-subscribed, then each resident s' tied at the tail of h's list is identified, and we break any provisional assignment between s' and h and delete the pair (s', h). Here "delete the pair (s', h)" means to delete s' from h's list and vice-versa. Furthermore, if h becomes full as a result of these provisional assignments, the worst resident s assigned to h is identified, and we delete the pair (s', h) for each strict successors s' of s on h's list. If a hospital becomes full during

the execution of the main while loop, and this hospital is subsequently under-subscribed on termination of this loop, then no super-stable matching exists. Also on termination of the main while loop, a check is made to ensure that there are no residents who are multiply assigned. If this is the case no super-stable matching exists for this instance of HRT. Otherwise the assignment relation output is a super-stable matching.

Resident's preferences	Hospital's preferences
$r_1: \ h_1 \ (h_2 \ h_3)$	$h_1: \ r_1 \ r_2 \ r_3 \ (r_4 \ r_5)$
$r_2: \ (h_2 \ h_3) \ h_1$	$h_2: \ (r_4 \ r_5) \ (r_1 \ r_2) \ r_3$
$r_3: \ h_3 \ (h_1 \ h_2)$	$h_3: \ r_2 \ (r_1 \ r_3) \ (r_4 \ r_5)$
$r_4: \ h_2 \ h_3 \ h_1$	
$r_5: \ h_2 \ (h_1 \ h_3)$	

Hospital capacities: $c_i = 2$, for $1 \leq i \leq 3$

Figure 1.8: HRT Instance I_3.

Consider the HRT instance I_3 shown in Figure 1.8. When algorithm HRT-super-res is applied to instance I_3, the following matching is generated:

$$M = \{(r_1, h_1), (r_2, h_3), (r_3, h_3), (r_4, h_2), (r_5, h_2)\}.$$

If implemented with suitable data structures (as described in [39]), algorithm HRT-`super-res` runs in time linear in the input size, i.e. $O(\lambda)$, where λ is the total length of the preference lists for an HRT instance.

Strong stability in HRT

In [40] a polynomial-time algorithm is presented for strong stability. This algorithm finds a strongly stable matching, or reports that none exists, in time $O(\lambda^2)$, given an instance of HRT, where λ is the total length of the preference lists.

An improvement on the algorithm presented in [40] was given by Kavitha et al. [45]. The running time of the algorithm is $O(C\lambda)$, where C is the sum of the hospitals' capacities.

1.3 Student Project Allocation Problem

As part of the senior level of many undergraduate degree courses, students are required to undertake project work. A lecturer typically publishes a set of projects that he is willing

to supervise, and each student lists the projects that he finds acceptable. Often a lecturer may be constrained to supervising a certain number of students, and as such, a lecturer may have an associated capacity. A project may also be available to be undertaken by more than one student, but may have an upper bound on the number of students who can be assigned to it. We refer to the problem of assigning students to projects, subject to these preference lists and capacity constraints, as the *Student-Project Allocation problem* (SPA) .

Currently there is a growing trend for centralised matching schemes in the context of student-project allocation. Examples of such schemes can be found in Department of Computer Science at the University of York [12, 46, 70], the University of Southampton [7, 31] and elsewhere [69]. The rising number of such allocation schemes motivates the search for efficient algorithms for SPA.

This section discusses the SPA model presented by Abraham et al. [3].

1.3.1 Introduction

An instance I of SPA consists of:

- set of students $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$;
- set of projects $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$;
- set of lecturers $\mathcal{L} = \{l_1, l_2, \ldots, l_q\}$;
- list of project capacities c_j, for $1 \leq j \leq m$;
- list of lecturer capacities d_k, for $1 \leq k \leq q$.

Let $\mathcal{A}_i$ denote the set of projects that a student s_i finds acceptable, and let $\mathcal{P}_k$ denote the set of projects *offered by* lecturer l_k, where $\mathcal{P}_1, \ldots, \mathcal{P}_q$ partitions $\mathcal{P}$ (i.e. each project is offered by exactly one lecturer). Each student s_i ranks a subset of $\mathcal{P}$ (namely $\mathcal{A}_i$) in strict preference order, and each lecturer has a preference list $\mathcal{L}_k$ ranking in strict order the students that find a project offered by that lecturer acceptable. We also denote by $\mathcal{L}_k^j$ the *projected preference list* of l_k for p_j, where $l_k \in \mathcal{L}$ and $p_j \in \mathcal{P}_k$ – this is obtained by removing all students from $\mathcal{L}_k$ that do not find p_j acceptable. Each project p_j offered has a capacity c_j, which indicates the maximum number of students allowed to undertake that project. In addition to this, each lecturer l_k also has a capacity d_k, which indicates the maximum number of students he is willing to supervise. It is assumed that $\max\{c_j : p_j \in \mathcal{P}_k\} \leq d_k$.

Student preferences	Lecturer preferences	
s_1 : p_1 p_3 p_5	l_1 : s_1 s_3 s_2 s_4	l_1 offers p_1, p_2, p_3
s_2 : p_1 p_4	l_2 : s_1 s_2 s_4	l_2 offers p_4, p_5
s_3 : p_1 p_2		
s_4 : p_3 p_1 p_5		
	Project capacities: $c_1 = 2$, $c_i = 1$ $(2 \leq i \leq 5)$	
	Lecturer capacities: $d_1 = 2$, $d_2 = 2$	

Figure 1.9: Instance I_1 of SPA.

An *assignment* M for an instance I of SPA is a set of (student,project) pairs $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ such that $(s_i, p_j) \in M$ only if $p_j \in \mathcal{A}_i$. If $(s_i, p_j) \in M$, and l_k is the lecturer who offers p_j, we say that s_i is *assigned to* p_j and l_k, and each of p_j and l_k is *assigned to* s_i. For any $r \in \mathcal{S} \cup \mathcal{P} \cup \mathcal{L}$, we denote by $M(r)$ the set of assignees of r in M. If $M(r) \neq \emptyset$ we say that r is *assigned* in M, otherwise r is *unassigned* in M. Where there is no ambiguity we use $M(s_i)$ to denote the project that s_i is assigned to, for an assigned student s_i.

Let M be an assignment for an instance I of SPA-P. We say that a project $p_j \in \mathcal{P}$ is *under-subscribed*, *over-subscribed* or *full* in M when $|M(p_j)| < c_k$, $|M(p_j)| > c_k$, or $|M(p_j)| = c_k$ respectively. Similarly a lecturer $l_k \in \mathcal{L}$ is *under-subscribed*, *over-subscribed* or *full* in M when $|M(l_k)| < d_k$, $|M(l_k)| > d_k$, or $|M(l_k)| = d_k$ respectively.

Also, M is a *matching* if $|M(s_i)| \leq 1$ for all $s_i \in \mathcal{S}$, $|M(p_j)| \leq c_j$ for all $p_j \in \mathcal{P}$, and $|M(l_k)| \leq d_k$ for all $l_k \in \mathcal{L}$. That is, each student is assigned to at most one project in M, and no project or lecturer is over-subscribed in M.

An example SPA instance I_1 is shown in Figure 1.9. This instance has the set of students $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, set of projects $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$, and the set of lecturers $\mathcal{L} = \{l_1, l_2\}$.

A *blocking pair* in the context of SPA is now defined. A (student,project) pair (s_i, p_j) *blocks* a matching M, or is a *blocking pair* of M, if:

(i) s_i finds p_j acceptable;

(ii) Either s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$;

(iii) Either

(a) p_j is under-subscribed and l_k is under-subscribed, or

(b) p_j is under-subscribed, l_k is full, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$, or

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

where l_k is the lecturer who offers p_j.

A matching is said to be *stable* if it admits no blocking pair.

HR as presented in Section 1.2 is a special case of SPA in which $m = q$, $c_j = d_j$ $(1 \leq j \leq m)$ and $\mathcal{P}_k = \{p_k\}$, for each $l_k \in \mathcal{L}$, and hence projects and lecturers are essentially indistinguishable.

1.3.2 The student-oriented algorithm

Algorithm SPA-`student` shown in Algorithm 5 is known as the *student-oriented* algorithm for SPA [3]. For an instance I the algorithm returns the *student-optimal* stable matching, in which each student is simultaneously assigned to the best possible project he could obtain in any stable matching [3].

In Algorithm SPA-`student` we use an *apply* operation similar to the proposal step in the Gale-Shapley algorithm. While some student s_i is free and has a non-empty list, s_i applies to the project p_j at the head of his list, and becomes provisionally assigned to p_j. If some project p_j becomes over-subscribed, then the worst student s_r assigned to p_j is identified, and the provisional assignment between s_r and p_j is broken. Similarly if l_k (the lecturer who offers p_j) becomes over-subscribed, then the worst student s_r assigned to l_k and his associated project p_t is identified, and the assignment between s_r and p_t is broken. If a project p_j is full, then the worst student s_t assigned to p_j is identified, and p_j is deleted from the list of each student who l_k ranks below s_t; in addition to this, s_t can also be deleted from $\mathcal{L}_k^j$. Similarly if l_k becomes full, each student s_t whom l_k ranks lower than his worst assigned student is identified, and each project p_u offered by l_k is removed from s_t's list; the relevant students are also deleted from $\mathcal{L}_k^u$. In SPA-`student`, the operation "delete the pair (s_t, p_u)" is used to denote the deletion of p_u from s_t's list and the deletion of s_t from $\mathcal{L}_k^u$.

The student-optimal stable matching for instance I_1 (shown in Figure 1.9) is $M = \{(s_1, p_1), (s_2, p_4), (s_3, p_1), (s_4, p_5)\}$. We verify the stability of this matching by inspecting the preference list of each student. Both s_1 and s_3 have their first choice project, so there is no blocking pair involving s_1 or s_3. Student s_2 prefers p_1 to his assigned project. However

Algorithm 5 SPA-`student`

assign all students to be free;
assign all projects and lecturers to be totally unsubscribed;
while (some student s_i is free) **and** (s_i has a non-empty list) **do**
  p_j := first project on s_i's list; /** s_i apples to p_j */
  l_k := lecturer who offers p_j; /** and to l_k */
  provisionally assign s_i to p_j;
  if p_j is over-subscribed **then**
    s_r := worst student assigned to p_j;
    break provisional assignment between s_r and p_j;
  else if l_k is over-subscribed **then**
    s_r := worst student assigned to l_k;
    p_t := project assigned to s_r;
    break provisional assignment between s_r and p_t;
  if p_j is full **then**
    s_r := worst student assigned to p_j;
    for each successor s_t of s_r on $\mathcal{L}_k^j$ **do**
      delete the pair (s_t, p_j);
  if l_k is full **then**
    s_r := worst student assigned to l_k;
    for each successor s_t of s_r on $\mathcal{L}_k$ **do**
      for each project $p_u \in \mathcal{P}_k \cap \mathcal{A}_t$ **do**
        delete the pair (s_t, p_u);

Student preferences	Lecturer preferences	
s_1 : p_2 p_1	l_1 : s_1 s_2	l_1 offers p_1
s_2 : p_1 p_2	l_2 : s_2 s_1	l_2 offers p_2

Project capacities: $c_1 = 1$, $c_2 = 1$

Lecturer capacities: $d_1 = 2$, $d_2 = 2$

Figure 1.10: Instance I_2 of SPA.

l_1 (who offers p_1) is full and prefers all his assignees to s_2. Lastly, s_4 has his worst project. Once again l_1 (who offers s_4's two preferred projects) is full and prefers all his assignees to s_4. Hence M is indeed a stable matching for instance I_1.

It can be shown that, when implemented with suitable data structures (as described in [3]), the algorithm SPA-student runs in time linear in the input size, i.e. $O(\lambda)$ where λ is the total length of the preference lists [3].

1.3.3 Properties of SPA

The Rural Hospitals Theorem (Theorem 1.2.1) provides some useful structural properties in the context of HR. Its counterpart in the context of SPA, proved in [3], is as follows:

Theorem 1.3.1. *For a given* SPA *instance:*
(i) each lecturer has the same number of students in all stable matchings;
(ii) exactly the same students are unassigned in all stable matchings;
(iii) a project offered by an under-subscribed lecturer has the same number of students in all stable matchings.

It should be noted that Theorem 1.3.1 is not an exact counterpart to the Rural Hospitals Theorem (Theorem 1.2.1). More details of this can be found in [3].

1.3.4 The lecturer-oriented algorithm

In addition to the student-oriented algorithm, Abraham et al. [3] present a *lecturer-oriented* algorithm. As expected, this algorithm generates (in linear time) a stable matching that is simultaneously the best possible for all lecturers. The algorithm, correctness proof and optimality criteria can be found in [3].

1.4 Stable Roommates Problem

1.4.1 Introduction

In their seminal paper [18] Gale and Shapley presented the *Stable Roommates* Problem (SR), a non-bipartite generalisation of SM. An instance of SR involves set of agents $\mathcal{P} = \{p_1, p_2, \ldots, p_{2n}\}$, each of whom ranks the others in strict order of preference. In this context a *matching* is a set of n disjoint pairs of agents. A pair $\{p_i, p_j\}$ *blocks* a matching M, or is a *blocking pair*, if p_i and p_j prefer each other to their actual partners in M. A matching is said to be *stable* if there are no blocking pairs.

p_1 : p_3 p_2 p_4

p_2 : p_1 p_3 p_4

p_3 : p_2 p_1 p_4

p_4 : *arbitrary*

Figure 1.11: SR instance I_1.

The most notable difference between SM and SR is that an instance of SR need not admit a stable matching. An example to show this was presented by Gale and Shapley in [18], and is shown in Figure 1.11. In instance I_1, if any of $\{p_1, p_4\}, \{p_2, p_4\}, \{p_3, p_4\}$ are involved in a matching M, then M will be blocked by the pairs $\{p_1, p_2\}$, $\{p_2, p_3\}$, $\{p_3, p_1\}$ respectively.

Knuth conjectured [47] that the problem of deciding whether a stable matching exists, given an instance of SR, is NP complete. However, Irving provided a linear-time algorithm that finds a stable matching or reports that none exists [34]. Irving's algorithm operates in two distinct phases, each of which we now describe in the following sections.

1.4.2 Stable Roommates algorithm phase 1

Algorithm 6 shows phase 1 of the Stable Roommates algorithm. While some free agent p_i has a non-empty list, we find the first agent p_j on p_i's list. If p_j is already *semi-assigned* to some agent p_k, then the semi-assignment between p_j and p_k is broken, and p_i becomes semi-assigned to p_j. We note that the semi-assignment relation is not symmetric, that is, if p_i is semi-assigned to p_j, it need not be the case that p_j is semi-assigned to p_i; p_j may still be free or semi-assigned to someone else. For each successor p_l of p_i on p_j's list we

Algorithm 6 Phase 1 Stable Roommates

```
1: assign each agent to be free;
2: while some free agent p_i has a non-empty list do
3:    p_j := first agent on p_i's list;
4:    if some agent p_k is semi-assigned to p_j then
5:       assign p_k to be free;
6:    assign p_i to be semi-assigned to p_j;
7:    for each successor p_l of p_i on p_j's list do
8:       delete the pair {p_l, p_j};
```

then delete the pair $\{p_l, p_j\}$, which entails deleting p_l from p_j's list and deleting p_j from that of p_l.

On termination of phase 1 the reduced preference lists generated by the algorithm are known as the *phase 1 table*. In order to define this term, we require some additional notation and terminology. Firstly, to describe the set of preference lists before, during, and after deletions have taken place, the term *preference table* is used. Let T be a preference table. Then:

- $f_T(p_i)$ denotes the first entry on p_i's list in T;
- $l_T(p_i)$ denotes the last entry on p_i's list in T;
- $s_T(p_i)$ denotes the second entry on p_i's list in T, undefined if $f_T(p_i) = l_T(p_i)$;
- $n_T(p_i)$ denotes $l_T(s_T(p_i))$.

A *stable preference table* (often shortened to *stable table*) satisfies the following properties.

1. $p_j = f_T(p_i)$ if and only if $p_i = l_T(p_j)$;
2. the pair $\{p_i, p_j\}$ is absent if and only if p_i prefers $l_T(p_i)$ to p_j or p_j prefers $l_T(p_j)$ to p_i;
3. no person's list is empty.

If some agent's preference list after phase 1 is empty, then no stable matching exists. Otherwise, the preference table after phase 1 is a stable table, which is known as the phase 1 table [26, Lemma 4.2.2].

Figure 1.13 shows the phase 1 table for instance I_2 shown in Figure 1.12.

p_1 : p_3 p_2 p_4

p_2 : p_4 p_1 p_3

p_3 : p_2 p_4 p_1

p_4 : p_1 p_2 p_3

Figure 1.12: SR instance I_2.

If each agent's list is a single entry after phase 1, then we have found a stable matching. However, in general, it may be the case that no agent has an empty list and not all agents' lists consist of a single entry. In this case we run phase 2 of the algorithm.

1.4.3 Stable Roommates algorithm phase 2

In phase 2 of the algorithm for SR, the preference lists are continually reduced by eliminating *rotations*. Informally a rotation is a cycle of ordered pairs. A rotation ρ has the form:

$$\rho = (p_0, q_0), (p_1, q_1), \ldots, (p_{r-1}, q_{r-1}),$$

where p_i and q_i, for $0 \leq i \leq r-1$, are agents in the SR instance, with $q_i = f_T(p_i)$ and $q_{i+1} = s_T(p_i)$. A rotation ρ exposed in a table T can be *eliminated*. The table arising from the elimination of ρ is denoted by T/ρ and is formed by deleting, for $0 \leq i \leq r-1$, all pairs $\{q_i, z\}$ such that q_i prefers p_{i-1} to z.

The algorithm for phase 2 of SR is shown in Algorithm 7. The algorithm continually eliminates rotations from the phase 1 table until either all lists have one entry, indicating that a stable matching has been found, or an agent's list becomes empty, in which case no stable matching exists for this instance.

From the phase 1 table T_0 shown in Figure 1.13, we identify the following rotation $\rho = (p_3, p_2), (p_4, p_1)$ in T_0 – details of how we find such a rotation can be found in [26, Section 4.2.3]. The elimination of ρ moves p_3 down to p_1 and p_1 up to p_3, as a result of

p_1 : p_3 p_2 p_4

p_2 : p_4 p_1 p_3

p_3 : p_2 p_1

p_4 : p_1 p_2

Figure 1.13: Phase 1 table T_0 of SR instance I_2.

Algorithm 7 Phase 2 Stable Roommates

```
1: T := T_0;
2: while (some list in T has more than one entry) and (no list in T is empty) do
3:     find a rotation ρ exposed in T;
4:     T := T/ρ;
5: if some list in T is empty then
6:     output no stable matching exists;
7: else
8:     output T, which is a stable matching;
```

which $\{p_1, p_2\}$ and $\{p_1, p_4\}$ are deleted. Similarly, the rotation also moves p_4 down to p_2 and p_2 up to p_4, which results in the deletion of $\{p_2, p_3\}$. After the above deletions, each person has only one element left on their reduced list. This signals the termination of the algorithm and also indicates that we have found a stable matching for SR instance I_2.

Irving proved that the algorithm has $O(n^2)$ worst-case complexity for an instance involving $2n$ people. As in the case of SM we denote an instance of SR with incomplete preference lists by SRI, and note that the algorithm by Irving can easily be adapted to handle this case [26, Section 4.5.2].

The following theorem, also presented in [26], states that, given an SM instance, we can construct an SR instance such that the stable matchings are in one-to-one correspondence.

Theorem 1.4.1. *Given an instance I of* SM *involving n men and n women, there is an instance J (in fact there are many instances) of the* SR *involving those $2n$ agents such that the stable matchings in J are precisely the stable matchings in I.*

Theorem 1.4.1 allows us to apply the $\Omega(n^2)$ lower bound result described in Section 1.1.1 (Theorem 1.1.1) to SR. Therefore the algorithm given by Irving for SR is asymptotically optimal.

1.4.4 Stable Roommates Problem with Ties and Incomplete Lists

As in the case of SM and HR, SR can be generalised to include the possibility of ties in the preference lists. This extension of SR is denoted by SRT. The three stability criteria introduced in Section 1.1.7 can be easily adapted to the Stable Roommates case and are given below.

A pair $\{x, y\}$ is said to *block* a matching M for an instance of SRT, and is called a *blocking pair* when:

- weak stability – both x and y strictly prefer each other to their partners in M.
- strong stability – x strictly prefers y to his partner in M and y either strictly prefers x to his partner in M or is indifferent between them.
- super-stability – each of x and y either strictly prefers the other to his partner or is indifferent between them.

A matching M is said to be *weakly stable*, *strongly stable* or *super-stable* respectively, if there exists no blocking pair in M with respect to the above definitions. An instance of SR with both ties and incomplete lists is denoted by SRTI, and the stability definitions above can easily be extended to the case of SRTI.

Weak stability

For instances of SMT and HRT, it is easy to find a matching that is weakly stable by simply breaking the ties arbitrarily, and running the appropriate algorithm for finding a stable matching, which is also stable in the original instance. However, as already discussed in Section 1.4.1, there is no guarantee that an SR instance admits a stable matching even without the presence of ties. Hence breaking the ties arbitrarily in an SRT instance gives no guarantee that a stable matching will be found. To complicate matters further, there are exponentially many ways of breaking the ties. Ronn [62] showed that the problem of deciding whether an instance of SRT admits a weakly stable matching is NP-complete.

$$
\begin{aligned}
p_1 &: \quad p_4 \quad p_3 \\
p_2 &: \quad p_4 \\
p_3 &: \quad p_1 \\
p_4 &: \quad (p_1 \quad p_2)
\end{aligned}
$$

Figure 1.14: SRTI Instance I_3.

Irving and Manlove [38] further explored weak stability in SRT and SRTI. In addition to providing a simpler NP-completeness proof compared to that found in [62] for the problem of deciding if a weakly stable matching exists for an instance of SRT, they showed that weakly stable matchings for an instance of SRTI may have different sizes. Figure 1.14 shows an instance I_3 of SRTI that admits weakly stable matchings of sizes 1 ($\{\{p_1, p_4\}\}$) and 2 ($\{\{p_1, p_3\}, \{p_2, p_4\}\}$). Given that weakly stable matchings may be of different cardinalities, it is natural to pose the question as to whether we can efficiently find a maximum

cardinality weakly stable matching. As described in Section 1.1.7, the problem of deciding if an instance of SMTI admits a weakly stable matching M, with $|M| \geq K$ for some K, is NP-complete. This result carries over to SRTI, using an analogue of Theorem 1.4.1 for SMTI under weak stability. The NP-hardness of finding a maximum cardinality weakly stable matching in an instance of SRTI also holds by restriction to the case that $K = n$ and all preference lists are complete, using Ronn's result stated earlier.

Due to these NP-hardness results, it is natural to consider the approximability of the problems concerned. Irving and Manlove [38] proved that a maximum cardinality weakly stable matching is approximable within a constant factor of 2.

Super and strong stability

An algorithm for finding a super-stable matching, if it exists, was presented by Irving and Manlove [38]. The algorithm works in two phases which are similar to those found in Irving's algorithm for SR. The algorithm and correctness proofs can be found in [38]. There is also a detailed analysis section showing that the algorithm has complexity $O(n^2)$ for a given SRTI instance with $2n$ people.

Scott [67] presents an $O(\lambda^2)$ algorithm for the problem of finding a strongly stable matching, or reporting that none exists, for an instance of SRTI, where λ is the total length of the preference lists. Once again the algorithm involves a two-phase process, and is slightly more complex than the algorithm given for super-stability in [38].

1.5 Contribution of this thesis

The main results obtained in this thesis relate to the problems described in the preceding sections of this chapter. In this section we describe the thesis contribution in greater detail and we also outline the structure of the following chapters.

In Chapter 2 we present an alternative to the SPA model introduced in Section 1.3. In our model students have preferences over projects, however, in contrast to the model described in Section 1.3, lecturers also have preferences over projects. The new model gives rise to two stability definitions, namely weak stability and strong stability. For weak stability we show that weakly stable matchings may be of different sizes, and we present an NP-completeness result for the problem of finding a complete weakly stable matching. Given that the problem of finding a maximum weakly stable matching is NP-hard in

this setting, we present two 2-approximation algorithms (each of which satisfy different additional criteria) for the problem of finding a maximum weakly stable matching. In contrast to weak stability, we prove that all strongly stable matchings have the same size, but also show that a strongly stable matching need not exist. However, we give a linear time algorithm that finds a strongly stable matching, or reports that none exists.

In Chapter 3, we consider restrictions on instances of both SMTI and HRT, where the preference lists on one or both sides are bounded in length (here we assume the length of an agent p_i's preference list is the number of agents who appear on p_i's list). We show that, in contrast to the general case, the problem of finding a maximum weakly stable matching is polynomial-time solvable, given an instance of SMTI, where the men's lists are of length 2 and the women's lists are of unbounded length. A faster algorithm is then given for the special case of finding a complete weakly stable matching, or reporting that none exists, given an instance of SMTI, where the men's lists are of length 2 and the women's lists are of unbounded length. Next we show that if the men's lists are of length 3 and the women's lists are of length 4, the problem of finding a maximum weakly stable matching is NP-hard. We then consider the problem of deciding whether a complete weakly stable matching exists, given an instance of HRT where the residents' lists are of length 3 and the hospitals' lists are of unbounded in length, and show that this problem is NP-complete.

In Chapter 4 we consider the restrictions of HRT where an agent's list is derived from a *master list* in which a set of agents are ranked according to some (possibly objective) criteria such as academic merit. Given an instance of HRT where the residents' lists are of length 3 and the hospitals' lists are of unbounded length, we show that, even in the presence of a master lists on both sides, the problem of deciding whether a complete weakly stable matching exists is NP-complete. We then describe a simpler algorithm, compared to that shown in Algorithm 4 for the general case of HRT, for the problem of finding a super-stable matching, or reporting that none exists, given an instance of HRT where the hospitals' lists are derived from a master list of residents. Furthermore, we show that if a super-stable matching exists, then this matching is in fact the unique super-stable matching. We then present a faster algorithm, compared to that for the general HRT case [45], for the problem of finding a strongly stable matching, or reporting that none exists, given an instance of HRT where the hospitals' preference lists are derived from a master list of residents.

In Chapter 5 we consider both SRI and SRTI with a master list of agents. For the case of SRI, we describe an algorithm that finds a stable matching without the need

for the two-phase approach described in Section 1.4. Then, in contrast to the general case, we show that finding a weakly stable matching is polynomial-time solvable and describe an algorithm to find such a matching. However, we also show that weakly stable matchings may have different sizes, and that the problem of finding a maximum weakly stable matching is NP-hard. As in the case of HRT, we present an algorithm that is simpler than that for the general SRTI case [38], which finds a super-stable matching, or reports that none exists, given an instance of SRTI with a master list of agents. Again we prove that a matching returned is in fact the unique super-stable matching. We then describe a faster algorithm, compared to that for the general SRTI case [67], for the problem of finding a strongly stable matching or reporting that none exists, given an instance of SRTI with a master list of agents.

We then consider, in Chapter 6, stable matching problems where the agents' preference lists are *symmetric* (that is p_i ranks p_j in k^{th} place if and only if p_j ranks p_i in k^{th} place). In this setting we identify two models, based on the interpretation of an agent's rank. We show that, regardless of the model under consideration, the problem of finding a weakly stable matching, given an instance of SRTI with symmetric preferences, is polynomial-time solvable. An example that illustrates weakly stable matchings may have different sizes, given an instance of SRTI with symmetric preferences, is then presented, and it is shown that for both models the problem of determining if a complete weakly stable matching exists is NP-complete. For one of the models we also show that each of the problems of finding an egalitarian weakly stable matching and a minimum regret weakly stable matching, given an instance of SMTI with symmetric preferences, is NP-hard. For the same model we also show that the problem of determining if a (man,woman) pair belongs to a weakly stable matching is NP-complete. We then describe two polynomial-time algorithms that simplify the algorithms given for the general case [38, 39], for each of the problems of finding a super-stable matching, or reporting that none exists, given an instance of HRT and SRTI with symmetric preferences. Finally, two faster algorithms, compared to the best-known algorithms for the general case [45, 67], for each of the problems of finding a strongly stable matching, or reporting that none exists, given an instance of HRT and SRTI, are presented.

In Chapter 7 we focus on modelling SMI using constraint programming. We extend the models discussed in Section 1.1.5 to obtain two new CSP encodings for SMI. The first encoding is both simple and elegant and returns the GS-lists after forcing AC propagation.

In this model AC can be established in time $O(n^3)$ (here, for simplicity, we assume the number of men = the number of women = n). The second encoding improves on the performance of the first encoding, and the GS-lists are obtained after forcing AC propagation. AC propagation in the second encoding can be established in $O(n^2)$. For both encodings we show that all the stable matchings can be enumerated without failure during search.

The first CSP encoding for SMI is then extended to the case of HR, and subsequently HRT under weak stability, in Chapter 8. In our HR encoding we show that AC can be established in time $O(n^3)$ and that the variables' domains correspond to the GS-lists of the original HR instance. We then extend this encoding to HRT under weak stability, and prove that if a matching is output by this encoding, then the matching is weakly stable. However, in contrast to the other encodings presented, the weakly stable matchings cannot be enumerated in a failure-free manner during search.

Chapter 2

Student-Project Allocation with Preferences over Projects

2.1 Introduction

We recall from Section 1.3 that stability has previously been considered in the context of SPA by Abraham et al. [3]. In this model lecturers rank the students in order of preference, a task that is often difficult, and as a result rankings tend to be based on academic merit. This strategy for ranking the students often results in students who appear lower down the merit list being less likely to be matched with the more popular projects, should they rank them higher on their preference list. Additionally a lecturer may not have specific preferences over students who find a particular project acceptable, but instead they may have preferences over projects that relate to their current research interests. It is therefore natural to investigate a model whereby both the students and lecturers have preferences over projects.

In this chapter we consider this variant of the SPA model, denoted by SPA-P. With respect to an instance of SPA-P, two stability definitions naturally arise, namely weak stability and strong stability. We denote these two versions of SPA-P by SPA-PW and SPA-PS respectively. The terminology arises from similarities between the weak stability and strong stability concepts in instances of HRT and SPA-P.

The remaining sections are structured as follows. In Section 2.2.1 we give a formal definition of SPA-PW, and show that for an instance of SPA-P, weakly stable matchings can have different sizes. However, in practical situations, we aim to match as many students as possible, and in Section 2.2.2 we show that the problem of finding a maximum

cardinality weakly stable matching is NP-hard. In Sections 2.2.3 and 2.2.4, we give two approximation algorithms with performance guarantee 2. Both algorithms return a weakly stable matching for an instance of SPA-P, however the matchings constructed have different additional properties. Finally, in Section 2.3, we present a linear-time algorithm for the problem of finding a strongly stable matching, if one exists, given an instance of SPA-P.

2.2 Weak Stability

In this section we formally define the SPA-PW model introduced above, and also define the concept of weak stability in this context.

2.2.1 Definition of SPA-PW

An instance I of SPA-P involves a set of students $\mathcal{S}$, a set of projects $\mathcal{P}$, and a set of lecturers $\mathcal{L}$. Each student $s_i \in \mathcal{S}$ ranks, in strict order of preference, an *acceptable* set of projects $\mathcal{A}_i \subseteq \mathcal{P}$. In addition, each lecturer $l_k \in \mathcal{L}$ supplies a set of projects P_k that they are willing to supervise. Lecturer l_k ranks P_k in strict order of preference. Implicitly each lecturer is indifferent amongst all students who find a given project acceptable. Each project $p_j \in \mathcal{P}$ has an associated capacity c_j, which indicates an upper bound on the number of students that may undertake p_j. Similarly each lecturer l_k has a capacity d_k, which indicates an upper bound on the number of students that l_k is willing to supervise.

Figure 2.1 shows an example SPA-P instance I_1, with $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, $\mathcal{P} = \{ p_1, p_2, p_3, p_4 \}$, and $\mathcal{L} = \{l_1, l_2\}$.

Student preferences	Lecturer preferences
s_1 : p_1 p_3 p_2	l_1 : p_1 p_2
s_2 : p_1 p_4 p_3	l_2 : p_4 p_3
s_3 : p_2 p_4	
s_4 : p_3 p_1 p_2 p_4	

Project capacities: $c_1 = 2, c_2 = 1, c_3 = 1, c_4 = 1$
Lecturer capacities: $d_1 = 2, d_2 = 2$

Figure 2.1: An instance I_1 of SPA-P.

We use the notation and terminology defined for an instance of SPA as described in

Section 1.3 throughout this section. That is, notation and terminology defined previously in terms of instances and matchings in the SPA context are also valid in the SPA-P context. However blocking pair and stability concepts will be specifically defined for an instance of SPA-P in this section.

We now define a *blocking pair* with respect to M. The pair (s_i, p_j) is said to *block* M, or to be a *blocking pair* of M, if each of conditions 1, 2 and 3 are satisfied as follows:

1. $p_j \in A_i$ (i.e. s_i finds p_j acceptable).
2. Either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$.
3. p_j is under-subscribed and one of
   (a) $s_i \in M(l_k)$ and l_k prefers p_j to $M(s_i)$, or
   (b) $s_i \notin M(l_k)$ and l_k is under-subscribed, or
   (c) $s_i \notin M(l_k)$ and l_k is full and l_k prefers p_j to his worst non-empty project,

   holds, where l_k is the lecturer who offers project p_j.

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

The motivation behind Conditions 2 and 3 above is now given (Condition 1 is straightforward). Let (s_i, p_j) be a blocking pair, and let l_k be the lecturer who offers p_j. Condition 2 states that s_i would rather be matched than unmatched, and if s_i is matched and prefers p_j to $M(s_i)$, then s_i would rather reject $M(s_i)$ and become assigned to p_j. Condition 3 models the conditions under which a lecturer can improve with respect to M. Here l_k would not strictly improve by rejecting a student from a project that was already full, and hence p_j must be under-subscribed. Then Condition 3(a) states that if s_i is already assigned to a project p_z offered by l_k, but l_k prefers p_j to p_z, then l_k would improve by allowing s_i to move from p_z to p_j. Condition 3(b) states that l_k would "improve" by taking on an additional student in an under-subscribed project, if l_k is also under-subscribed. Finally, Condition 3(c) states that if l_k prefers p_j to his worst non-empty project p_z, then l_k would improve by rejecting a student from p_z and taking on s_i to do p_j.

In addition to being weakly stable, we may also seek to find a matching that is *coalition-free*. A matching M is said to be *coalition-free* if there exists no *exchange-blocking coalition* $\langle s_{i_0}, s_{i_1}, \ldots, s_{i_{r-1}} \rangle$, where s_i prefers $M(s_{i+1})$ to $M(s_i)$ $(0 \leq i \leq r-1)$, with addition taken modulo r. If such a coalition exists in a matching M, and each student s_i switches from

Student preferences	Lecturer preferences
s_1 : p_1 p_2	l_1 : p_1 p_2
s_2 : p_2 p_1	

Project capacities: $c_1 = 1, c_2 = 1$

Lecturer capacities: $d_1 = 2$

Figure 2.2: An instance I_2 of SPA-P.

Student preferences	Lecturer preferences
s_1 : p_1 p_2	l_1 : p_1
s_2 : p_1	l_2 : p_2

Project capacities: $c_1 = 1$, $c_2 = 1$

Lecturer capacities: $d_1 = 1$, $d_2 = 1$

Figure 2.3: An instance I_3 of SPA-P.

$M(s_i)$ to $M(s_{i+1})$, it can be seen that no lecturer becomes worse off, since a lecturer is implicitly indifferent among the students, and moreover the same number of students remain assigned to each project and lecturer following such a switch. For example, in instance I_2, shown in Figure 2.2, one possible weakly stable matching is $M_1 = \{(s_1, p_2), (s_2, p_1)\}$. However, we can identify a matching $M_2 = \{(s_1, p_1), (s_2, p_2)\}$ that is weakly stable but also coalition-free.

It turns out that weakly stable matchings may have different sizes for an instance of SPA-P. To see this, consider instance I_3 shown in Figure 2.3. Two possible weakly stable matchings for I_3 are $M_1 = \{(s_1, p_2), (s_2, p_1)\}$ and $M_2 = \{(s_1, p_1)\}$. As weakly stable matchings may have different sizes for the same SPA-P instance, it is natural to investigate the problem of finding a weakly stable matching that matches the largest number of students. In the following sections we present some algorithmic results for this problem.

2.2.2 NP-hardness of finding a maximum weakly stable matching

Denote by MAX-SPA-PW the problem of finding a maximum weakly stable matching, given an instance of SPA-P. In this section we show that MAX-SPA-PW is NP-hard. This follows immediately from the NP-completeness of COM-SPA-PW, which is the problem of deciding, given an instance of SPA-P, whether a complete weakly stable matching exists (i.e. a

matching in which all students are assigned).

In order to prove that COM-SPA-PW is NP-complete we reduce from a problem related to matchings in graphs. A matching M is said to be *maximal* for a graph $G = (V, E)$ if, for every edge $e \in E \setminus M$, $M \cup \{e\}$ is not a matching. Let $\beta^-(G)$ denote the minimum cardinality of a maximal matching in G. Then MIN-MM is the problem of computing $\beta^-(G)$, given a graph G. The decision version of MIN-MM is shown below:

Name:	MIN-MM-D
Instance:	A graph G and an integer K.
Question:	Does G have a maximal matching M with $\|M\| \leq K$?

The NP-hardness of MIN-MM was established by Yannakakis and Gavril [74][1]. Horton and Kilakos [33] showed that MIN-MM-D is NP-complete for cubic graphs, furthermore they showed that the same is true for subdivision graphs[2] [33], and NP-completeness also holds for subdivision graphs of cubic graphs [27]. In the following lemma (due to Abraham et al. [1]) we show that, even for subdivision graphs of cubic graphs, the problem EXACT-MM of finding a maximal matching of size K in G, for a given integer K and a graph G, is NP-complete. First we formally define EXACT-MM.

Name:	EXACT-MM
Instance:	A graph G and an integer K.
Question:	Does G have a maximal matching M with $\|M\| = K$?

Lemma 2.2.1. EXACT-MM *is NP-complete, even for subdivision graphs of cubic graphs.*

Proof. Clearly EXACT-MM is in NP. To prove that EXACT-MM is NP-hard we reduce from MIN-MM-D. Let G be a subdivision graph of some cubic graph and let K be a positive integer, forming an instance of MIN-MM-D. We claim that G has a maximal matching M with $|M| \leq K$ if and only if G has a maximal matching M' with $|M'| = K'$.

Suppose that G has a maximal matching M with $|M| = k \leq K$. If $k = K$ then we are done. Therefore suppose $k < K$. Let $\beta(G)$ denote the size of a maximum matching in G. Then without loss of generality $k < K \leq \beta(G)$. Since maximal matchings satisfy

[1]In fact Yannakakis and Gavril showed that the minimum edge dominating set problem (MIN-EDS) is NP-hard. However it is known that MIN-EDS and MIN-MM are polynomially equivalent.

[2]The subdivision graph of a graph G is a graph G' in which we replace every edge in G by a path of length two.

the interpolation property [30] (i.e. G has a maximal matching of size j, for each j such that $k \leq j \leq \beta(G)$), G has a maximal matching of size K. The converse can be easily verified. □

We note that in this chapter we only require that EXACT-MM is NP-complete for subdivision graphs. However, in later chapters we require the NP-completeness of EXACT-MM restricted to subdivision graphs of cubic graphs.

We now use the NP-completeness of EXACT-MM to establish the NP-completeness of COM-SPA-PW.

Theorem 2.2.2. COM-SPA-PW *is NP-complete.*

Proof. Clearly COM-SPA-PW belongs to NP. To show NP-hardness, we transform from EXACT-MM restricted to subdivision graphs, which is NP-complete by Lemma 2.2.1. Hence let G (a subdivision graph of some graph G') and K (a positive integer) be an instance of EXACT-MM. Then G is a bipartite graph (since G is a subdivision graph, and therefore cannot have an odd cycle), so that $G = (U, W, E)$, where without loss of generality all vertices in U have degree 2. Suppose that $n_1 = |U|$ and $n_2 = |W|$. Again, without loss of generality assume that $K \leq \min\{n_1, n_2\}$. 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 two neighbours of u_i in G, where $j_i < k_i$.

We construct an instance I of COM-SPA-PW as follows: let $U \cup U' \cup V$ be the set of students, where $U' = \{u'_1, u'_2, \ldots, u'_{n_1}\}$ and $V = \{v_1, v_2, \ldots, v_{n_2-K}\}$; let $P \cup Q \cup R \cup S$ be the set of projects, where $P = \{p_1, p_2, \ldots, p_{n_2}\}$, $Q = \{q_1, q_2, \ldots, q_{n_2}\}$, $R = \{r_1, r_2, \ldots, r_{n_1}\}$ and $S = \{s_1, s_2, \ldots, s_{n_1-K}\}$; and let $W \cup X \cup Y$ be the set of lecturers, where $X = \{x_1, x_2, \ldots, x_{n_1}\}$, and $Y = \{y_1, y_2, \ldots, y_{n_1-K}\}$. Each project and lecturer has capacity 1. The preference lists in I are shown in Figure 2.4. These preference lists also indicate the acceptable projects for each student, and the projects offered by each lecturer. In a given preference list, projects within square brackets are listed in arbitrary strict order at the point where the symbol appears. We claim that G has a maximal matching of size K if and only if I admits a weakly stable matching in which all students are assigned.

For, suppose that G has a maximal matching M, where $|M| = K$. We construct a matching M' in I as follows. For each edge $\{u_i, w_j\}$ in M, if $j = j_i$, then we add (u_i, p_{j_i}) and (u'_i, r_i) to M'. If $j = k_i$, then we add (u'_i, p_{k_i}) and (u_i, r_i) to M'. There remain $n_2 - K$ lecturers in W who are under-subscribed in M'. Denote these lecturers

$$\text{Student preferences:} \quad \begin{cases} u_i: & r_i \;\; p_{j_i} \;\; p_{k_i} \;\; [S] & (1 \le i \le n_1) \\ u_i': & r_i \;\; p_{k_i} & (1 \le i \le n_1) \\ v_i: & [Q] & (1 \le i \le n_2 - K) \end{cases}$$

$$\text{Lecturer preferences:} \quad \begin{cases} w_j: & p_j \;\; q_j & (1 \le j \le n_2) \\ x_j: & r_j & (1 \le j \le n_1) \\ y_j: & s_j & (1 \le j \le n_1 - K) \end{cases}$$

Figure 2.4: Preference lists for the constructed instance of COM-SPA-PW.

by w_{t_j} $(1 \le j \le n_2 - K)$. Add (v_j, q_{t_j}) to M' $(1 \le j \le n_2 - K)$. Similarly there remain $2(n_1 - K)$ students in $U \cup U'$ who are unassigned in M'. Denote these students by u_{z_i}, u'_{z_i} $(1 \le i \le n_1 - K)$. Add (u_{z_i}, s_i) and (u'_{z_i}, r_{z_i}) to M' $(1 \le i \le n_1 - K)$. Clearly M' is a matching in I in which all students are assigned.

No project in $Q \cup R \cup S$ can be involved in a blocking pair of M', since each member of $W \cup R \cup S$ is full in M'. Hence no student in $U' \cup V$ can be involved in a blocking pair of M', since every student is assigned in M'. Finally, no pair $(u_i, p_j) \notin M'$ blocks M', where $u_i \in U$ and $p_j \in P$. For if this occurs, then $(u_i, s_l) \in M'$ for some $s_l \in S$, and p_j is under-subscribed. Thus no edge of 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' is weakly stable.

Conversely, suppose that M' is a weakly stable matching in I in which all students are assigned. For each $r_j \in R$, it follows that r_j is assigned either u_j or u'_j, for otherwise (u_j, r_j) blocks M', a contradiction. Hence

$$M = \left\{\{u_i, w_j\} \in E : (u_i, p_j) \in M' \;\vee\; (u'_i, p_j) \in M'\right\}$$

is a matching in G. Now each student in V is assigned in M' to a project in Q, so $n_2 - K$ projects in Q are full in M'. Hence at most K projects in P are full in M', since each lecturer in W has capacity 1. Now in M', at most $n_1 - K$ students in U are assigned to projects in S. As already observed, exactly n_1 students in $U \cup U'$ are assigned in M' to projects in R. Hence at least K students in $U \cup U'$ are assigned in M' to projects in P, so that $|M| = K$.

Suppose that M is not maximal. Then there is some edge $\{u_i, w_j\}$ in G such that no edge of M is incident to u_i or w_j. Thus $(u'_i, r_i) \in M'$, so that $(u_i, s_l) \in M'$ for some $s_l \in S$. Also either w_j is under-subscribed, or $(v_k, q_j) \in M'$ for some $v_k \in V$. Hence (u_i, p_j) blocks

M', for p_j is under-subscribed. This contradiction to the stability of M' implies that M is indeed maximal. □

The following corollary is an immediate consequence of Theorem 2.2.2.

Corollary 2.2.3. MAX-SPA-PW *is NP-hard, even if each project and lecturer has capacity 1.*

2.2.3 Coalition-free approximation algorithm for SPA-PW

Overview of the algorithm

Due to the NP-hardness of MAX-SPA-PW, we consider the problem of finding a weakly stable matching that is close to optimal. In this section we present a 2-approximation algorithm for MAX-SPA-PW. In addition the weakly stable matching produced by the algorithm is coalition-free.

Consider the algorithm SPA-PW-approx1 shown in Algorithm 8. The algorithm uses a series of *apply* and *delete* operations to obtain a weakly stable matching that is at least half the size of an optimal weakly stable matching. At each iteration of the algorithm, some free student s_i with a non-empty preference list applies to the first project p_j on his list. If p_j is full, then p_j is removed from s_i's list. If l_k is full, and l_k's worst non-empty project p_z is the same as p_j, then p_j is also removed from s_i's list. Otherwise s_i becomes assigned to p_j. If l_k becomes over-subscribed as a result of this assignment, the algorithm identifies an arbitrary student s_r assigned to p_z, assigns s_r to be free, and deletes p_z from s_r's list. At this point if l_k is full, each project p_t that l_k finds less desirable than his worst non-empty project is deleted from the preference list of each student that finds p_t acceptable.

We will now prove that on termination of SPA-PW-approx1 the algorithm always outputs a matching M (Lemma 2.2.4), and that the matching is weakly stable (Lemma 2.2.5 and 2.2.6) and coalition-free (Lemma 2.2.7).

Correctness and performance guarantee of the algorithm

Lemma 2.2.4. SPA-PW-approx1 *returns a matching.*

Proof. Clearly the while loop terminates. For, at the beginning of some loop iteration, let s_i be a student who is free and has a non-empty list, and let p_j be the first project on

Algorithm 8 SPA-PW-approx1

```
1:  M := ∅;
2:  while some student s_i is unassigned and s_i has a non-empty list do
3:      p_j := first project on s_i's list;
4:      l_k := lecturer who offers p_j;
5:      p_z := l_k's worst project;
6:      if l_k is assigned at least one student then
7:          p_z := l_k's worst non-empty project;
8:      /** s_i applies to p_j */
9:      if p_j is full or (l_k is full and p_j = p_z) then
10:         delete p_j from s_i's list;
11:     else
12:         M := M ∪ {(s_i, p_j)};
13:         /** s_i provisionally assigned to p_j and l_k */
14:         if l_k is over-subscribed then
15:             s_r := arbitrary student assigned to p_z;
16:             M := M\{(s_r, p_z)};
17:             delete p_z from s_r's list;
18:
19:         if l_k is full then
20:             p_z := l_k's worst non-empty project;
21:             for each p_t ∈ {successors_{l_k}(p_z)} do
22:                 for each student s_r who finds p_t acceptable do
23:                     delete p_t from s_r's list;
24: return M;
```

s_i's list. If s_i does not become provisionally assigned to p_j during the same loop iteration, then p_j is removed from s_i's list. If s_i becomes provisionally assigned to p_j then some student s_r may become free – if this is the case p_j is always deleted from s_r's list. Hence, eventually, we are guaranteed that each student is either assigned to some project or has an empty list. Let M be the assignment relation upon termination of SPA-P-approx1. It is immediate that each student is assigned to at most one project in M, whilst no project or lecturer is over-subscribed in M. □

Lemma 2.2.5. *Suppose that some project p_t is deleted from a student s_r's list during an execution of* SPA-PW-approx1. *Then (s_r, p_t) cannot block a matching output by* SPA-P-approx1.

Proof. Let E be an execution of the algorithm during which p_t is deleted from s_r's list. By Lemma 2.2.4, let M be the matching output at the termination of E. Suppose for a contradiction that (s_r, p_t) blocks M. We consider four cases.

Case (i): p_t was deleted from s_r's list as a result of p_t being full during E. Since (s_r, p_t) blocks M, p_t is under-subscribed in M. Hence p_t changed from being full during E to being under-subscribed, which can only occur as a result of some lecturer l_k being over-subscribed during E, where p_t was l_k's worst non-empty project at that point. Thus l_k is full in M, and l_k's worst non-empty project is either p_t or better. Hence (s_r, p_t) does not block M in this case.

Case (ii): p_t was deleted from s_r's list as a result of l_k being full during E, and p_t was l_k's worst non-empty project. Clearly on termination of E, l_k is full, and l_k's worst non-empty project is p_t or better. Hence (s_r, p_t) does not block M in this case.

Case (iii): p_t was deleted from s_r's list as a result of l_k being over-subscribed during E. Then just before the deletion occurred, p_t was l_k's worst non-empty project. Now l_k is full in M, and l_k's worst non-empty project is either p_t or better. Hence (s_r, p_t) does not block M in this case.

Case (iv): p_t was deleted from s_r's list as a result of l_k being full during E. Then l_k is full in M, and l_k prefers his worst non-empty project to p_t. Hence (s_r, p_t) does not block M in this case.

□

Lemma 2.2.6. SPA-PW-approx1 *returns a weakly stable matching.*

Proof. Let E be an execution of the algorithm, and by Lemma 2.2.4, let M be the matching output upon termination of E. Suppose that (s_i, p_j) blocks M. By Lemma 2.2.5, p_j is not deleted from s_i's list during E. Hence s_i's list is non-empty upon termination of E. If s_i is unassigned in M then the while loop would not have terminated, a contradiction. Hence s_i is assigned in M and prefers p_j to $p_r = M(s_i)$. But when s_i applied to p_r, it follows that p_r was the first project on s_i's list, a contradiction to the fact that (s_i, p_j) has not been deleted. Hence M is weakly stable. □

Lemma 2.2.7. SPA-PW-approx1 *returns a matching that is coalition-free.*

Proof. By Lemma 2.2.4, let M be the matching output by an execution E of SPA-PW-approx1. Suppose, for a contradiction, that there exists an exchange-blocking coalition $\langle s_{i_0}, s_{i_1}, \ldots, s_{i_r-1} \rangle$ with respect to M. Then $(s_{i_t}, M(s_{i_{t+1}}))$ is deleted during E for each t $(0 \leq t \leq r-1)$, where addition is taken modulo r. Let $(s_{i_j}, M(s_{i_{j+1}}))$ be the first such pair to be deleted during E. Let $p_z = M(s_{i_{j+1}})$ and let l_k be the lecturer who offers p_z. We consider the following four cases.

Case (i): p_z was deleted from s_{i_j}'s list when p_z became full during E. Then $s_{i_{j+1}}$ must have applied to p_z after s_{i_j} did. Suppose that this is not the case. Then $s_{i_{j+1}}$ was already assigned to p_z when s_{i_j} applied to p_z. Hence $M(s_{i_{j+2}})$ must have already been deleted from $s_{i_{j+1}}$'s list, a contradiction to the fact that (s_{i_j}, p_z) is the first such deletion of the form $(s_{i_t}, M(s_{i_{t+1}}))$ $(0 \leq t \leq r-1)$ to take place. Therefore p_z must have gone from being full to being under-subscribed during E. This can only happen if l_k became over-subscribed during E, and p_z was l_k's worst non-empty project at that point. Thus when $s_{i_{j+1}}$ applies to p_z, it follows that p_z is still l_k's worst non-empty project, and l_k is full. Therefore l_k rejects $s_{i_{j+1}}$ from p_z, a contradiction.

Case (ii): p_z was deleted from s_{i_j}'s list at line 10, when l_k became full during E and p_z was l_k's worst non-empty project. As in Case (i), $s_{i_{j+1}}$ must apply to p_z after s_{i_j} does. Furthermore, l_k remains full at every subsequent iteration of E. Therefore at the iteration where $s_{i_{j+1}}$ becomes assigned to p_z, it follows that l_k's worst non-empty project is p_z or better. In either case the pair $(s_{i_{j+1}}, p_z)$ is deleted, a contradiction.

Case (iii): p_z was deleted from s_{i_j}'s list when l_k became over-subscribed during E, and p_z was l_k's worst non-empty project. As in Case (i), $s_{i_{j+1}}$ must have applied to

p_z after s_{i_j} did. The rest of the proof for this case is identical to Case (ii).

Case iv: p_z was deleted from s_{i_j}'s list when l_k became full during E, and l_k's worst non-empty project was better than p_z. At this point p_z is removed from $s_{i_{j+1}}$'s list as well, a contradiction.

□

The next result shows that SPA-P-approx1 has a performance guarantee of 2.

Theorem 2.2.8. SPA-PW-approx1 *is an approximation algorithm for* MAX-SPA-PW *with a performance guarantee of 2.*

Proof. Let I be an instance of SPA-P and let M be a weakly stable matching of maximum size in I. By Lemma 2.2.6, let M' be a weakly stable matching output by SPA-P-approx1 as applied to I, and suppose for a contradiction that $|M'| < |M|/2$. Let X (respectively Y) be those students who are assigned in M but not M' (respectively M' but not M), and let Z be those students who are assigned in both M and M'. Then

$$|X| = |M| - |Z| > 2|M'| - |Z| = 2|Y| + |Z| \geq |M'|. \tag{2.1}$$

Now suppose that the students in X are collectively assigned in M to projects $P' = \{p_1, \ldots, p_s\}$ offered by lecturers $l_1, \ldots l_t$. Suppose that $P'_1, \ldots, P'_t$ is a partition of P' such that lecturer l_k $(1 \leq k \leq t)$ offers the projects in P'_k. Similarly let $S_1, \ldots, S_t$ be a partition of X such that each student in S_k is assigned in M to a project in P'_k $(1 \leq k \leq t)$.

Now let k be given $(1 \leq k \leq t)$ and let p_j be any project in P'_k. Then there is some student $s_i \in S_k$ who is assigned to p_j in M but unassigned in M'. Hence in M', either (i) p_j is full, or (ii) l_k is full (or both), for otherwise (s_i, p_j) blocks M'. It follows that, in M', either (a) all projects in P'_k are full, or (b) l_k is full (or both). Hence

$$|M'| \geq \sum_{k=1}^{t} \min \left(d_k, \sum_{p_j \in P'_k} c_j \right). \tag{2.2}$$

Since no project or lecturer is over-subscribed in M, it follows that, for each k $(1 \leq k \leq t)$, $\sum_{p_j \in P'_k} c_j \geq |S_k|$ and $d_k \geq |S_k|$. Hence Inequality 2.2 implies that $|M'| \geq \sum_{k=1}^{t} |S_k| = |X|$, which is a contradiction to Inequality 2.1. Thus $|M'| \geq |M|/2$ as required. □

Additional properties of SPA-PW-approx1

This section contains a series of properties of SPA-PW-approx1. The first property shows that there exists an instance I of SPA-P and a weakly stable matching of maximum size M_{max} in I, such that every execution of SPA-PW-approx1 applied to I will output a weakly stable matching M such that $|M| < |M_{max}|$. Thus for this instance, no execution of SPA-PW-approx1 finds a maximum weakly stable matching. We also show that, the performance guarantee cannot be improved by comparing the larger of the matchings output when the students apply in increasing indicial order and when the students apply in decreasing indicial order.

Proposition 2.2.9. *There exists an instance I of* SPA-P *and a weakly stable matching M_{max} of maximum size in I such that every execution of* SPA-PW-approx1 *applied to I will output a weakly stable matching M such that $|M| < |M_{max}|$.*

Proof. Consider instance I with $\mathcal{S} = \{s_1, s_2, s_3\}$, $\mathcal{P} = \{p_1, p_2, p_3\}$, and $\mathcal{L} = \{l_1, l_2\}$. Then we define the student and lecturer's preference lists, and capacities, as follows:

Student preferences	Lecturer preferences
s_1 : p_2 p_1	l_1 : p_1 p_2
s_2 : p_1 p_3	l_2 : p_3
s_3 : p_1	

Project capacities: $c_1 = 2$, $c_2 = 1$, $c_3 = 1$

Lecturer capacities: $d_1 = 2$, $d_2 = 1$

In I there exists a maximum weakly stable matching $M_{max} = \{(s_1, p_1), (s_2, p_3), (s_3, p_1)\}$. However regardless of the order in which the students apply, it is easy to verify that algorithm SPA-PW-approx1 always returns a matching of size two. □

Now consider a variant of algorithm SPA-PW-approx1 as shown in Algorithm 9.

Proposition 2.2.10. SPA-PW-approx1-reverse *performs no better than* SPA-PW-approx1 *in general.*

Proof. Consider instance I with $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$, $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$, and $\mathcal{L} = \{l_1, l_2\}$.

Algorithm 9 SPA-PW-approx1-reverse

```
1: M1 := SPA-PW-approx1(); /** Students apply in increasing indicial order */
2: M2 := SPA-PW-approx1(); /** Students apply in decreasing indicial order */
3:
4: if |M1| > |M2| then
5:     output M1;
6: else
7:     output M2;
```

For each student and lecturer the preference lists and capacities are defined as follows:

Student preferences	Lecturer preferences
$s_1: \ p_2 \ \ p_1$	$l_1: \ p_1 \ \ p_2$
$s_2: \ p_1 \ \ p_3$	$l_2: \ p_3 \ \ p_4$
$s_3: \ p_1 \ \ p_4$	
$s_4: \ p_2 \ \ p_1$	

Project capacities: $c_1 = 2$, $c_2 = 1$, $c_3 = 1$, $c_4 = 1$

Lecturer capacities: $d_1 = 2$, $d_2 = 2$

First, it can be easily verified that $M_{max} = \{(s_1, p_1), (s_2, p_3), (s_3, p_4), (s_4, p_1)\}$ is a maximum weakly stable matching in I. Now consider the application sequence $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$; this yields the weakly stable matching $M_1 = \{(s_2, p_1), (s_3, p_1)\}$ of size two. Reversing this application sequence, we obtain the weakly stable matching M_2 such that $M_1 = M_2$. Therefore both M_1 and M_2 are half the size of the maximum weakly stable matching M_{max}. Hence, in general, algorithm SPA-PW-approx1-reverse performs no better than SPA-PW-approx1. □

2.2.4 A Generalised Approximation Algorithm for SPA-PW

Overview of the algorithm

In Section 2.2.3, approximation algorithm SPA-PW-approx1 was presented that finds a weakly stable matching for an instance of SPA-P. As illustrated by Proposition 2.2.9, there exists an instance of SPA-P for which all executions of SPA-PW-approx1 fail to find a weakly stable matching of maximum size, regardless of the order in which the students apply. In this section we present a second 2-approximation algorithm, SPA-PW-approx2, that finds a weakly stable matching for an instance of SPA-P. We show that, for a given instance I of

SPA-P, there is an execution of SPA-PW-approx2 that will find a maximum weakly stable matching in I. The algorithm consists of two phases: the first phase, SPA-PW-approx2-phase1, as shown in Algorithm 10, finds a weakly stable matching M. However, M may admit exchange-blocking coalitions. A second phase of the algorithm, SPA-PW-approx2-phase2, is then used to eliminate exchange-blocking coalitions, without affecting the weak stability or cardinality of M.

Phase 1

The algorithm uses a similar apply and delete strategy to SPA-PW-approx1. However in SPA-PW-approx2-phase1, if a student s_i applies to a project p_j then s_i immediately becomes assigned to p_j. If as a result of this assignment p_j becomes over-subscribed, p_j is removed from the list of some arbitrary student s_r assigned to p_j. Otherwise p_j must be full or under-subscribed. If l_k is now over-subscribed, l_k's worst non-empty project p_z is identified, and p_z is deleted from the list of some arbitrary student s_r assigned to p_z. Finally if l_k is full, l_k's worst non-empty project p_z is once again identified, and each project p_t that appears below p_z on l_k's list is deleted from the list of each student who finds p_t acceptable.

It can be shown that the matching output by SPA-PW-approx2-phase1 is weakly stable using a similar argument to that in Section 2.2.3. Furthermore, SPA-PW-approx2-phase1 has a performance guarantee of 2, which follows directly from the proof of Theorem 2.2.8. In the following lemma we show that for a given instance I of SPA-P there always exists an execution of SPA-PW-approx2-phase1 that finds a maximum weakly stable matching in I.

Lemma 2.2.11. *If an instance I of* SPA-P *admits a stable matching of size K, then there exists an execution of* SPA-PW-approx2-phase1 *that finds a weakly stable matching of size $\geq K$.*

Proof. Let M_K be a weakly stable matching of size K. Without loss of generality, we assume $M_K(s_i) = p_{r_i}$ $(1 \leq i \leq K)$ for some sequence $r_1, \ldots, r_k$, and that students $s_{K+1}, \ldots, s_n$ are unmatched in M_K. We construct an execution E of algorithm SPA-PW-approx2-phase1 by deleting preference list entries using the following strategy. If a project p_y becomes over-subscribed, student s_x is rejected from p_y, where $s_x \notin M_K(p_y)$. Similarly if a lecturer l_k becomes over-subscribed, and p_y denotes l_k's worst non-empty project, student s_x is rejected from p_y, where $s_x \notin M_K(p_y)$. We claim that such a student

Algorithm 10 SPA-PW-approx2-phase1

```
1: M := ∅;
2: while some student s_i is unassigned and s_i has a non-empty list do
3:     p_j := first project on s_i's list;
4:     l_k := lecturer who offers p_j;
5:     /** s_i provisionally assigned to p_j and l_k */
6:     M := M ∪ {(s_i, p_j)};
7:     if p_j is over-subscribed then
8:         s_r := arbitrary student assigned to p_j;
9:         M := M\{(s_r, p_j)};
10:        delete p_j from s_t's list;
11:    else
12:        if l_k is over-subscribed then
13:            p_z := l_k's worst non-empty project;
14:            s_r := arbitrary student assigned to p_z;
15:            M := M\{(s_r, p_z)};
16:            delete p_z from s_r's list;
17:        if l_k is full then
18:            p_z := l_k's worst non-empty project;
19:            for each p_t ∈ {successors_{l_k}(p_z)} do
20:                for each student s_r who finds p_t acceptable do
21:                    delete p_t from s_r's list;
22: return M;
```

can always be found. Suppose that this is not the case. Let z' denote the first iteration during E in which l_k is over-subscribed and l_k's worst non-empty project p_y satisfies $M'(p_y) \subseteq M_K(p_y)$, where M' is the assignment at this iteration just before any deletions occur. Hence there exists a project p_z on l_k's list such that $|M'(p_z)| > |M_K(p_z)|$, otherwise l_k is over-subscribed in M_K. Then p_z is under-subscribed in M_K, and moreover l_k prefers p_z to p_y, since p_y is l_k's worst non-empty project at this point and $|M'(p_y)| \leq |M_K(p_y)|$. Now let s_t be a student assigned to p_z in M', but not in M_K. We firstly note that if s_t is unassigned in M_K, then (s_t, p_z) blocks M_K, a contradiction. Hence $(s_t, p_{r_t}) \in M_K$. We now prove that at every iteration prior to z' where deletions are made, the algorithm never deletes a pair of the form (s_i, p_{r_i}) $(1 \leq i \leq K)$, and as such s_t strictly prefers p_z to p_{r_t}. Suppose that this is not the case. Let $z'' < z'$ be the first iteration during E in which a pair of the form (s_i, p_{r_i}) $(1 \leq i \leq K)$ is deleted. Consider the following three cases.

> *Case (i):* p_{r_i} is deleted from s_i's list when p_{r_i} becomes over-subscribed. This cannot happen as s_t is matched to p_{r_i} in M_K, and hence the deletion strategy does not allow this.
>
> *Case (ii):* p_{r_i} is deleted from s_i's list when l_w, the lecturer who offers p_{r_i}, becomes over-subscribed. Once again this cannot happen using the above deletion strategy, by choice of z'.
>
> *Case (iii):* p_{r_i} is deleted from s_i's list when l_w, the lecturer who offers p_{r_i}, becomes full with projects he prefers to p_{r_i}. Let M'' be the assignment just before this deletion takes place during iteration z''. At this point l_w is full in M'', and there exists a project p_b with fewer assignees in M_K than in M, for otherwise l_w is over-subscribed in M_K, since $(s_i, p_{r_i}) \in M_K \setminus M$. Hence choose $s_a \in M(p_b) \backslash M_K(p_b)$. Now p_b is under-subscribed in M_K – moreover l_w prefers p_b to p_{r_i}. Hence if s_a is unmatched in M_K, (s_a, p_b) blocks M_K, therefore s_a must be matched in M_K. Since M_K is weakly stable, s_a prefers p_{r_a} to p_b. Therefore $M_K(s_a)$ must have been deleted before the z''^{th} iteration, a contradiction to the choice of (s_i, p_{r_i}) as the first pair to be deleted prior to iteration z''. Hence p_{r_i} is not deleted from s_i's list in this case.

Therefore either s_t is unassigned in M_K or s_t prefers p_z to p_{r_t}. Also l_k prefers p_z to p_y, hence as p_z is under-subscribed in M_K, it follows that (s_t, p_z) blocks M_K, a contradiction. This completes the construction of E; we note the order in which the students apply is unimportant.

Let M be the matching output by SPA-PW-approx2-phase1 on termination of E. We claim that if $(s_i, p_{r_i}) \in M_K$, for each i $(1 \leq i \leq K)$, then during E, the algorithm never deletes p_{r_i} from s_i's list. To prove this result, the steps required are identical to those used in the three cases above (in Case (ii), we no longer need to refer to z', since it has already been established that the deletion strategy is well-defined; as before, z'' is the first iteration during E in which a pair (s_i, p_{r_i}) is deleted). Thus $|M| \geq |M_K|$. Hence if I admits a weakly stable matching of size K, the algorithm is capable of finding a matching of at least this size. □

Phase 2

In contrast to SPA-PW-approx1, the matching output by SPA-PW-approx2-phase1 may not be coalition-free. This is illustrated by the proof of the following proposition.

Proposition 2.2.12. *A matching output by* SPA-PW-approx2-phase1 *need not be coalition-free.*

Proof. Consider instance I with $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5\}$, $\mathcal{P} = \{p_1, p_2, p_3\}$, and $\mathcal{L} = \{l_1, l_2\}$. For each student and lecturer the preference lists and capacities are defined as follows:

Student preferences	Lecturer preferences
s_1 : p_1	l_1 : p_2 p_1
s_2 : p_1 p_3	l_2 : p_3
s_3 : p_2	
s_4 : p_2	
s_5 : p_3 p_1	

Project capacities: $c_1 = 2$, $c_2 = 2$, $c_3 = 1$
Lecturer capacities: $d_1 = 3$, $d_2 = 1$

Consider the following sequence of events that can occur during an execution of SPA-PW-approx2 applied to instance I (we use $\rightarrow$ to indicate a student applying to a particular project).

Step	1	2	3	4	5	6	7
	$s_1 \rightarrow p_1$	$s_2 \rightarrow p_1$	$s_3 \rightarrow p_2$	$s_4 \rightarrow p_2$	$s_2 \rightarrow p_3$	$s_5 \rightarrow p_3$	$s_5 \rightarrow p_1$

At steps 4 and 7, lecturer l_1 became over-subscribed and rejected students s_2 and s_1 respectively. At step 5, p_3 became over-subscribed and rejected s_5. Therefore on termination of SPA-PW-approx2, matching $M = \{(s_2, p_3), (s_3, p_2), (s_4, p_2), (s_5, p_1)\}$ is output. From this we can see that s_2 prefers $M(s_5)$ to $M(s_2)$, and that s_5 prefers $M(s_2)$ to $M(s_5)$. Hence the algorithm outputs a matching containing an exchange-blocking-coalition $\langle s_2, s_5 \rangle$. □

The second phase of the approximation algorithm, SPA-PW-approx2-phase2, is shown in Algorithm 12. Algorithm SPA-PW-approx2-phase2 first creates a network $G = \langle V, E, c, w \rangle$ (constructed using Algorithm 11) corresponding to the matching M output by SPA-PW-approx2-phase1, where V is the set of vertices, E is the set of directed edges, $c : E \to \mathbb{N}$ is a capacity function, and $w : E \to \mathbb{N}$ is a cost function. The algorithm then finds a minimum cost maximum flow f in G, and outputs a coalition-free weakly stable matching corresponding to this flow.

Algorithm 11 ConstructNetwork(M)

Require: Matching M output by SPA-PW-approx2-phase1.

1: add vertices u,t to G; /** the source and the sink */
2: **for each** $(s_i, p_j) \in M$ **do**
3: add vertex s_i and edge (u, s_i) of cost 0 and capacity 1;
4: add vertex p_j and edge (p_j, t) of cost 0 and capacity $|M(p_j)|$;
5: **for each** $p_j \in \mathcal{P}$ such that s_i prefers p_j to $M(s_i)$ or $p_j = M(s_i)$ **do**
6: /** $rank(s_i, p_j)$ denotes the rank of p_j on s_i's list */
7: add edge (s_i, p_j) to G with cost $rank(s_i, p_j)$ and capacity 1;
8: **return** G;

Algorithm 12 SPA-PW-approx2-phase2

Require: Matching M output by SPA-PW-approx-phase1.

1: $G :=$ ConstructNetwork(M);
2: $f :=$ Find minimum cost maximum flow in G;
3: $M' := \{(s_i, p_j) : s_i \in \mathcal{S} \wedge p_j \in \mathcal{P} \wedge (s_i, p_j) \in E \wedge f(s_i, p_j) = 1\}$
4: **return** M';

Lemma 2.2.13. *Let M be a matching output by* SPA-PW-approx2-phase1*, and let f be a minimum cost maximum flow returned by* SPA-PW-approx2-phase2*. Then $val(f) = |M|$.*

Proof. For any flow in the network G constructed using Algorithm 11, it follows by construction of the network that $val(f) \leq \sum_{p_j \in \mathcal{P}} |M(p_j)|$. Conversely we can achieve a flow

of value $|M|$ by pushing a flow of 1 along edge (u, s_i), where $(s_i, p_j) \in M$. □

Remark 2.2.14. Let M be the matching returned by SPA-PW-approx2-phase1, and M' be the assignment returned by SPA-PW-approx2-phase2. Then each project and lecturer has the same number of assignees in M as in M'. Hence M' is a matching.

Lemma 2.2.15. *The matching returned by* SPA-PW-approx2-phase2 *is coalition-free.*

Proof. Let M' be the matching returned by SPA-PW-approx2-phase2, and f be the flow corresponding to M' with respect to the network G constructed by Algorithm 11. Suppose that M' admits a coalition $\langle s_{i_0}, s_{i_1}, \ldots, s_{i_{r-1}} \rangle$, for some i_j $(0 \leq j \leq r-1)$. Then s_{i_j} prefers $M'(s_{i_{j+1}})$ to $M'(s_{i_j})$, for each j $(0 \leq j \leq r-1)$, where addition is taken modulo r. Let M'' be the matching obtained by each student s_{i_j} switching to student $s_{i_{j+1}}$'s project, i.e.

$$M'' = \big(M' \backslash \{(s_{i_j}, M'(s_{i_j})) : 0 \leq j \leq r-1\}\big) \cup \{(s_{i_j}, M'(s_{i_{j+1}})) : 0 \leq j \leq r-1\}.$$

We show that the size of the flow f' corresponding to M'' in G is the same as flow f. We then show that the cost of f' is smaller than f, contradicting the fact that f is a minimum cost maximum flow. First let $s_i \in \mathcal{S}$. Clearly the same set of students are matched in M' as in M''. Therefore $f'(u, s_i) = f(u, s_i)$. Hence $val(f') = val(f)$. However $cost(f') < cost(f)$, as at least two students obtained a better project in M'' to that in M', whilst no student is worse off in M'' as compared to M', a contradiction. Therefore M' is exchange-coalition-free. □

Lemma 2.2.16. SPA-PW-approx2-phase2 *returns a weakly stable matching.*

Proof. Let M' be the matching returned by algorithm SPA-PW-approx2-phase2. Now suppose (s_i, p_j) blocks M'. Either s_i is unmatched in M', and so unmatched in M (where M is the matching returned by phase 1 of the algorithm), or s_i prefers p_j to $M'(s_i)$ (and hence to $M(s_i)$). Also p_j is under-subscribed in M', and hence in M by Remark 2.2.14. Since (s_i, p_j) does not block M, l_k must be full in M, and l_k's worst non-empty project in M is p_j or better. By Remark 2.2.14, l_k is assigned the same number of students in M as in M', and has the same number of assignees to all of his projects in M and M'. Hence l_k must be full in M', and l_k's worst non-empty project in M' is p_j or better. Thus (s_i, p_j) cannot block M', and so M' is weakly stable. □

We bring together the lemmas from above into the following theorem.

Theorem 2.2.17. *For a given* SPA-P *instance* I*:*

(i) SPA-PW-approx2 *returns a coalition-free weakly stable matching in* I*;*

(ii) SPA-PW-approx2 *has a performance guarantee of 2;*

(iii) there exists an execution of SPA-PW-approx2 *that returns a coalition-free maximum weakly stable matching.*

2.3 Strong Stability

The NP-completeness of MAX-SPA-PW naturally leads us to consider an alternative notion of stability (an analogue of strong stability) that is obtained by altering the blocking pair definition. This model is denoted by SPA-PS. We note that strong stability is a more robust form of stability. To understand why, we observe that under weak stability if a lecturer has a fully-subscribed project then a student not assigned to it could attempt to convince the lecturer to reject a student from the project in his favour.

2.3.1 Definition of SPA-PS

An instance of SPA-P is identical to that described in Section 2.2.1 for SPA-PW, with the definition of stability changing as follows.

A *blocking pair* relative to a matching M is defined to be a (student,project) pair (s_i,p_j) such that each of conditions (i), (ii) and (iii) are satisfied:

(i) $p_j \in \mathcal{A}_i$ (i.e. s_i finds p_j acceptable),

(ii) Either s_i is unmatched in M or s_i prefers p_j to $M(s_i)$,

(iii) Either

(a) p_j is under-subscribed and l_k is under-subscribed, or

(b) $|M(p_r)| > 0$ for some project p_r, where l_k prefers p_j to p_r, or $p_j = p_r$,

where l_k is the lecturer who offers p_j.

Again with this blocking pair definition the choice of terminology is by analogy with the corresponding term in the context of HRT.

A matching is *strongly stable* if it admits no blocking pair. Informally, the motivation for considering this blocking pair definition is that a student who wants to become assigned

to a full project p_j could try to tempt lecturer l_k to switch s_i for some student already assigned to p_j. Again this allows for the possibility that s_i was already assigned to some project offered by l_k.

We consider the motivation for Condition 3 in greater detail. Suppose that s_i prefers a project p_j offered by l_k. If p_j has some assignee s_t then l_k would be no worse off by rejecting s_t and taking on s_i instead – Condition 3(b). Otherwise suppose that p_j is empty. If l_k is under-subscribed then both p_j and l_k have a free place for s_i – Condition 3(a). Otherwise l_k is full. If l_k prefers p_j to some project p_r with at least one assignee s_t, then l_k could improve by rejecting s_t from p_r and taking on s_i to do p_j instead – Condition 3(b).

2.3.2 Student-oriented Algorithm for SPA-PS

The student-oriented algorithm SPA-PS-`student`, is shown in Algorithm 13. First we set M to be the empty matching, and $over(l_k)$ to be false for every lecturer $l_k \in \mathcal{L}$. While some student s_i is free and has a non-empty list, s_i applies to the first project p_j on his list, with all applications being provisionally accepted. During the execution of the algorithm, entries are progressively deleted from the students' lists until the termination condition is met. If at some iteration of the main while loop a lecturer becomes over-subscribed, then his worst project p_r with at least one assignee is identified. Then for p_r and each successor of p_r on l_k's list, a set T is constructed which contains, for each such project p_v, the set of (student,project) pairs of the form (s_t, p_v), where (s_t, p_v) is in M. We then remove the set of the pairs in T from M. Finally we "delete p_v" for each such p_v, i.e. we delete p_v from the list of every student who finds p_v acceptable. Similarly if a project p_r becomes over-subscribed at some iteration of the main while loop, then for p_r and each successor of p_r on l_k's list we carry out the process described above. Finally, if on termination of the main while loop a lecturer l_k is under-subscribed and $over(l_k)$ is true, then (as we will show) no strongly stable matching exists, otherwise M is a strongly stable matching.

2.3.3 Correctness of Algorithm SPA-PS-student

The following lemmas are used to prove the correctness of the algorithm SPA-PS-`student`.

Lemma 2.3.1. *Algorithm* SPA-PS-`student` *terminates with a matching. Further, if the algorithm reports that the assignment relation M is a strongly stable matching then M is indeed such a matching.*

Algorithm 13 SPA-PS-`student`

1: $M := \emptyset$;
2: **for each** lecturer l_k **do**
3: $over(l_k) :=$ false;
4:
5: **while** some student s_i is free **and** has a non-empty list **do**
6: $p_j :=$ first project on s_i's list;
7: $l_k :=$ lecturer who offers p_j;
8: $M := M \cup \{(s_i, p_j)\}$; /** s_i becomes provisionally assigned to p_j */
9:
10: **if** l_k is over-subscribed *or* p_j is over-subscribed **then**
11: $over(l_k) :=$ true;
12: **if** l_k is over-subscribed **then**
13: $p_r := l_k$'s worst project with ≥ 1 assignee;
14: **else**
15: $p_r := p_j$;
16:
17: **for each** $p_v \in \{p_r\} \cup \{successors_{l_k}(p_r)\}$ **do**
18: $T := \{(s_t, p_v) : s_t \in S \wedge (s_t, p_v) \in M\}$;
19: $M := M \setminus T$; /** break assignments */
20: delete p_v;
21:
22: **if** some lecturer l_k is under-subscribed **and** $over(l_k)$ **then**
23: no strongly stable matching exists;
24: **else**
25: M is a strongly stable matching;

Proof. Each iteration of the main while loop involves a free student s_i applying to the next project on his list. No student can apply to the same project twice, as when the assignment between (s_i,p_j) is broken, p_j is removed from s_i's preference list. The number of iterations is therefore bounded by the length of a student's preference list. Thus the main while loop terminates. Also, each student is assigned to at most one project, as we only ever consider free students. Furthermore, it is clear that no project or lecturer is over-subscribed, hence M is a matching. □

Lemma 2.3.2. *If a pair (s_i,p_j) is deleted during an execution E of* SPA-PS-`student`*, then the pair cannot block a matching generated by the algorithm.*

Proof. Let M be a matching output by SPA-PS-`student`. Now suppose the pair (s_i,p_j) is deleted and that (s_i, p_j) blocks M. Let l_k be the lecturer who offers p_j. Then (s_i,p_j) is deleted at some iteration z when either (i) some project p_r becomes over-subscribed where p_j is a successor of p_r or $p_j = p_r$, or (ii) l_k becomes over-subscribed.

Case (i): Since a matching was output, it follows that l_k must be full in M, as $over(l_k)$ was set to true at iteration z. Therefore (s_i,p_j) does not satisfy Condition 3(a).

Also l_k must prefer p_r to p_j or $p_r = p_j$. In either case p_r, and all its successors, are deleted from every student's preference list and all relevant (student,project) pairs are removed from M. Hence in M there exists no (student,project) pair (s_t, p_v), such that l_k prefers p_r (and therefore p_j) to p_v, and so (s_i,p_j) does not satisfy Condition 3(b).

Case (ii): A similar argument to Case (i) can be used to prove that (s_i,p_j) does not satisfy Condition 3(a).

For Condition 3(b) it can be seen that when l_k becomes over-subscribed, the worst non-empty project p_r is identified from l_k's current list of assignees. Then for each successor p_v of p_r on l_k's list (including p_r itself), p_v is removed from the list of each student that finds p_v acceptable and all relevant (student,project) pairs are removed from M. Therefore in M it follows that l_k cannot supervise p_r or a project he ranks lower then p_r, hence (s_i, p_j) does not satisfy Condition 3(b).

□

Lemma 2.3.3. *If on termination of* SPA-PS-`student` *a matching M is output, then M is a strongly stable matching.*

Proof. Suppose for a contradiction that M is not strongly stable. Hence there exists a pair (s_i, p_j) that blocks M. By Lemma 2.3.2 the pair has not been deleted. Furthermore s_i must be assigned in M to a project p_r, for otherwise the algorithm would not have terminated, contradicting Lemma 2.3.1. Therefore as (s_i, p_j) blocks M, it follows that s_i prefers p_j to p_r. However as s_i always applies to the first non-empty project on his list, it also follows that (s_i, p_j) was deleted, a contradiction. □

Lemma 2.3.4. *Algorithm* SPA-PS-`student` *never deletes a pair that belongs to a strongly stable matching.*

Proof. Let E be an execution of SPA-PS-`student` and let M be a strongly stable matching. Now suppose for a contradiction that $(s_i, p_j) \in M$ and that (s_i, p_j) is deleted at some iteration z of E. Let l_k be the lecturer who offers p_j and let M' denote the matching at line 10 during iteration z. Without loss of generality suppose that (s_i, p_j) is the first strongly stable pair to be deleted during E. Then (s_i, p_j) is deleted when either (i) some project $p_r \in P_k$ becomes over-subscribed, where p_r appears before p_j, or $p_r = p_j$, on l_k's list, or (ii) l_k becomes over-subscribed.

Case (i): There exists a student s_u assigned to p_r in M' but not in M, as p_r cannot be over-subscribed in M. First suppose either s_u is unassigned in M or assigned in M to a project p_x worse than p_r. Then as l_k prefers p_r to p_j, or $p_r = p_j$, and since $|M(p_j)| > 0$ (as $(s_i, p_j) \in M$), it follows that (s_u, p_r) blocks M, a contradiction. Hence s_u is assigned in M to a project p_v that he prefers to p_r. However when s_u became assigned to p_r during E, p_r must have been at the head of s_u's list. Therefore (s_u, p_v) was deleted at an iteration prior to z, contradicting the fact that (s_i, p_j) was the first strongly stable pair to be deleted during E.

Case (ii): Since l_k is not over-subscribed in M, there exists a pair $(s_u, p_v) \in M'$ with $(s_u, p_v) \notin M$. As in Case (i) s_u cannot obtain a project better than p_v in M. Hence either s_u is unassigned in M or obtains a project worse than p_v in M. However as (s_i, p_j) is deleted at iteration z, either $p_j = p_v$ or l_k prefers p_v to p_j. In either case (s_u, p_v) blocks M, a contradiction.

□

Lemma 2.3.5. *Let M be the assignment relation on termination of the main while loop and let M' be a strongly stable matching in I (assuming that one exists). Then $|M(l_k)| = |M'(l_k)|$ for each lecturer $l_k \in \mathcal{L}$.*

Proof. Let $l_k \in \mathcal{L}$. We first claim that if l_k is full in M then l_k is full in M'. For, if not then $|M(l_k)| = d_k > |M'(l_k)|$, i.e. l_k is under-subscribed in M'. For every $p_j \in P_k$ and every $(s_i, p_j) \in M$, it follows that $(s_i, p_j) \in M'$. Suppose that this is not the case. Then there exists some $p_j \in P_k$, and some $(s_i, p_j) \in M \setminus M'$. Then s_i prefers $M'(s_i)$ to p_j, as (s_i, p_j) blocks M' otherwise. However when s_i became assigned to p_j during the while loop, p_j must have been the first project at the head of s_i's list. Hence the pair $(s_i, M'(s_i))$ was deleted, contradicting Lemma 2.3.4. Hence l_k is full in M' after all. By a similar argument it follows that if l_k is under-subscribed in M then l_k fills at least as many places in M'. Hence

$$|M(l_k)| \leq |M'(l_k)| \tag{2.3}$$

for each $l_k \in \mathcal{L}$.

Now let S_1 denote the set of students provisionally assigned to a project in M and let S_2 denote the set of students assigned to a project in M', i.e. $|S_1| = |M|$ and $|S_2| = |M'|$. By Lemma 2.3.4 we have that every student who is unassigned in M is also unassigned in M', and so $|S_1| \geq |S_2|$. Considering the lecturers $l_k \in \mathcal{L}$ we have,

$$\sum_{l_k \in \mathcal{L}} |M(l_k)| = |S_1| \geq |S_2| = \sum_{l_k \in \mathcal{L}} |M'(l_k)|$$

Thus by inequality 2.3 above, $|M(l_k)| = |M'(l_k)|$, for each $l_k \in \mathcal{L}$.

□

Lemma 2.3.6. *If on termination of the while loop in algorithm* SPA-PS-`student` *some lecturer l_k is under-subscribed and over(l_k) is set to true, then I admits no strongly stable matching.*

Proof. Let M' be a strongly stable matching in I and suppose that on termination of the main while loop some lecturer l_k is under-subscribed and *over*(l_k) is set to true. Let M be the assignment relation on termination of the main while loop.

Now as l_k is under-subscribed in M, but *over*(l_k) is true, if follows that either (i) l_k became over-subscribed, or (ii) some project p_v offered by l_k became over-subscribed.

Case (i): There must exist a (student,project) pair (s_u, p_v) that was deleted from M during the while loop and $(s_u, p_v) \notin M'$, for otherwise l_k is over-subscribed in M'. Now using an argument similar to that in the proof of Case (i) of Lemma 2.3.4, we have that s_u cannot obtain a partner better than p_v in M'. By Lemma 2.3.5, l_k is under-subscribed in M', and so it follows that (s_u, p_v) blocks M', a contradiction.

Case (ii): Similarly suppose that p_v is a project that became over-subscribed during an iteration of the while loop. Then there exists a (student,project) pair (s_u, p_v) that was deleted during this iteration such that $(s_u, p_v) \notin M'$. Again using a similar argument to Case (i) above, it is easy to verify that (s_u, p_v) blocks M'.

□

Corollary 2.3.7. *All strongly stable matchings have the same size.*

Collectively, Lemmas 2.3.2 to 2.3.6 to prove the following theorem.

Theorem 2.3.8. *For a given instance of* SPA-P*, algorithm* SPA-PS-`student` *returns the student-optimal strongly stable matching or reports that none exists.*

2.3.4 Properties of SPA-PS

We now describe some properties of strongly stable matchings with respect to an instance of SPA-P.

Theorem 2.3.9. *For a given* SPA-P *instance I:*

(i) each lecturer is assigned the same number of students in every strongly stable matching.

(ii) the same set of students are matched and unmatched in every strongly stable matching.

(iii) each lecturer l_k who is under-subscribed in one strongly stable matching obtains exactly the same set of students in every strongly stable matching.

(iv) if a lecturer l_k is under-subscribed in some strongly stable matching in I (and hence in every strongly stable matching in I by Part (iii)) and $p_j \in P_k$, then p_j obtains the same number of students in every strongly stable matching.

Proof. Let M' be the student-optimal strongly stable matching and let M be any other strongly stable matching in I.

(i) Follows directly from Lemma 2.3.5.

(ii) By Lemma 2.3.4 we have that each student who is unassigned in M' is also unassigned in M, and by Part (i) it follows that in every strongly stable matching the same number of students are matched. Thus, the set of students assigned in M and M' is the same.

(iii) Suppose that l_k is assigned a different set of students in M' to that in M. By Part (i), l_k is assigned the same number of students in M' as in M, hence there exists a student $s_i \in M'(l_k) \setminus M(l_k)$. By Part (ii) s_i must be assigned in M. However, since M' is the student-optimal strongly stable matching, it follows that s_i prefers $M'(s_i)$ to $M(s_i)$. Therefore as l_k is under-subscribed in M, the pair $(s_i, M'(s_i))$ blocks M.

(iv) Suppose $|M'(p_j)| > |M(p_j)|$. Therefore p_j is under-subscribed in M and there exists a pair $(s_i, p_j) \in M' \setminus M$. Since M' is the student-optimal strongly stable matching, either s_i is unmatched in M or prefers p_j to $M(s_i)$. Thus as l_k is under-subscribed in M, it follows that (s_i,p_j) blocks M. Hence, $|M(p_j)| \geq |M'(p_j)|$. Now suppose that $|M(p_j)| > |M'(p_j)|$. By Part (i), l_k is assigned the same number of students in M' as in M, hence there exist a project $p_l \neq p_j$ such that $|M'(p_l)| > |M(p_l)|$. Using a similar argument to earlier, it follows that (s_i, p_l) blocks M, a contradiction. Hence $|M(p_j)| = |M'(p_j)|$.

□

The theorem above is not an exact counterpart to the Rural Hospitals Theorem. In particular not every project obtains the same number of students in all strongly stable matchings.

To see why this is true consider Figure 2.5. Here two possible matchings are:

$$M = \{(s_1, p_1)\} \qquad M' = \{(s_1, p_2)\}.$$

Clearly each project does not obtain the same number of students in both M and M'.

2.3.5 Analysis of algorithm SPA-PS-student

To show that the SPA-PS-student algorithm can be implemented with worst case time-complexity $O(\lambda)$, where λ is the total length of all the preference lists, it is necessary to show how certain operations can be implemented efficiently. The non trivial steps in the algorithm are:

Student preferences	Lecturer preferences
$s_1:\ p_1\ \ p_2$	$l_1:\ p_2\ \ p_1$

Project capacities: $c_1 = 1, c_2 = 1$

Lecturer capacity: $d_1 = 1$

Figure 2.5: A SPA-P instance I_2.

(i) obtaining all successors of a specific project in a preference list;

(ii) checking in constant time if a student finds a project acceptable;

(iii) deleting projects from lecturers' and students' lists;

(iv) finding the worst project with ≥ 1 assignee.

The representation of the lecturer and student preference lists must allow for the lookup of an element in constant time. With the student preference lists matters are further complicated by the fact that deletions take place at unpredictable locations. To represent the lecturer preference lists a simple array-based approach is all that is required. However, student preference lists require a more elaborate structure. An effective way to represent a student's preference list is by means of a doubly-linked list embedded inside an array. This allows for the lookup of project in constant time, but also provides an efficient means of deleting a project. It is also required that we are able to lookup a project's position in both the student and lecturer preference lists in constant time. For example, this is required when deleting an element in a student's preference list or finding the successors of a particular project on a lecturer's preference list. To allow us to look up these details, the students and lecturer preference lists are used to construct *rank arrays*. A rank array contains an entry for each project p_j that holds an integer value indicating the position of p_j in the appropriate list; the value may be zero in a student's rank array if he finds p_j unacceptable and zero in a lecturer's rank array if $p_j \notin \mathcal{L}_k$. These two structures allow for the efficient handling of (i), (ii), and (iii) above.

To address item (iv), a lecturer l_k has to be able to identify his worst project p_r with at least one assignee in constant time. This is required in order to delete each successor of p_r when l_k becomes over-subscribed. However, rather than keep track of p_r explicitly, a pointer is maintained to the last project p_z on l_k's list. The list can then be traversed in

reverse from p_z to p_r when the appropriate deletions are to be carried out. To identify p_r the assignment relation is used. This is an array that maps a project to a set of students assigned to that project; the set of students can be implemented as a linked list with an element count. So for each project found during the traversal, a check is made against the assignment relation to see if any students are assigned to that particular project, with the traversal ending when the first project with at least one assignee is found.

2.4 Conclusion and Open Problems

Below we reflect on some of the issues with the strong stability model presented and give a selection of open problems for SPA. We describe open problems both for the SPA model presented in this chapter and also for the model described by Abraham et al. [3].

2.4.1 Strong stability

With respect to the blocking pair definition given in Section 2.3.1 for strong stability, we can identify issues with regards to a student switching projects supervised by the same lecturer. For example, consider instance I_3 shown in Figure 2.6.

Student preferences	Lecturer preferences
$s_1:\ p_1\ \ p_2$	$l_1:\ p_2\ \ p_1$
Project capacities: $c_1 = 1, c_2 = 1$	
Lecturer capacity: $d_1 = 2$	

Figure 2.6: A SPA-P instance I_3.

Here $M = \{(s_1, p_2\}\}$ is not a strongly stable matching as (s_1, p_1) blocks M. However if s_1 switches to p_1, the matching is strongly stable, but l_1 becomes worse off. On the other hand if such a situation is seen to be undesirable, we observe the following alternative definition of a blocking pair for strong stability.

A *blocking pair* relative to a matching M is a (student,project) pair $(s_i, p_j) \in (S \times P) \backslash M$ such that conditions 1, 2 and 3 are satisfied:

1. $p_j \in A_i$ (i.e. s_i finds p_j acceptable).
2. Either s_i is unmatched in M or s_i prefers p_j to $M(s_i)$.
3. One of,

(a) $s_i \in M(l_k)$ and p_j is under-subscribed and l_k prefers p_j to $M(s_i)$, or

(b) $s_i \notin M(l_k)$ and p_j is under-subscribed and l_k is under-subscribed, or

(c) $s_i \notin M(l_k)$ and l_k prefers p_j to his worst non-empty project p_r, or $p_j = p_r$,

holds, where l_k is the lecturer who offers p_j.

The intuition behind Conditions 1 and 2 are as before. We now consider Condition 3. Firstly suppose that s_i was already assigned to a project p_r offered by l_k. If p_j was full then l_k must reject a student from p_j before taking on s_i. In this case the number of students assigned to l_k would decrease by 1, so we assume that l_k would not agree to the switch. Hence p_j is under-subscribed. Moreover l_k would only let s_i change projects from p_r to p_j if he prefers p_j to p_r – Condition 3(a). Secondly suppose that s_i was not already assigned to a project offered by l_k. If p_j has some assignee s_t then l_k would be no worse off by rejecting s_t and taking on s_i to do p_j instead – Condition 3(c). Otherwise suppose that p_j is empty. If l_k is under-subscribed then both p_j and l_k have a free place for s_i – Condition 3(b). Otherwise l_k is full. If l_k prefers p_j to some project p_r with at least one assignee s_t, then l_k could improve by rejecting s_t from p_r and taking on s_i to do p_j instead – Condition 3(c).

Consider the instance I_4 of SPA-PS as shown in Figure 2.7.

With respect to the revised definition of strong stability, each of the following matchings is strongly stable: $M_1 = \{(s_1, p_1), (s_2, p_2)\}$, $M_2 = \{(s_1, p_2), (s_2, p_1)\}$. Since p_1 has capacity 1, there is no student-optimal strongly stable matching in I_4.

From the example above, it can be seen that preventing a student from switching projects supervised by the same lecturer brings with it problems. Thus it motivates our consideration of the original blocking pair definition for strong stability, as the problem can be solved in linear time and several structural results exist.

We leave the following open problems for the revised definition of strong stability.

Student preferences	Lecturer preferences
s_1 : p_1 p_2	l_1 : p_2 p_1
s_2 : p_1 p_2	

Project capacities: $c_1 = 1, c_2 = 1$

Lecturer capacity: $d_1 = 2$

Figure 2.7: An instance I_4 of SPA-P.

- Is it possible that strongly stable matchings could be of different sizes with respect to the above definition?
- Are there any other properties of the "Rural Hospitals Theorem" that do not hold under this revised definition?
- Can the algorithm for finding a strongly stable matching be adapted, or does it become NP-hard to determine whether a strongly stable matching exists?

2.4.2 Improved approximation algorithm for SPA-PW

In Section 2.2.3 and Section 2.2.4 we presented two approximation algorithms with performance guarantee 2. Also in [55] it is shown that the problem of finding a maximum weakly stable matching, given an instance of SPA-PW, is not approximable with δ, for some $\delta > 1$, unless P=NP. Is it possible to find an approximation algorithm with improved performance guarantee or to establish a stronger lower bound on the inapproximability of this problem?

2.4.3 SPA with preference over (student,project) pairs

A model that generalises both SPA-P, and the model given in [3], is one in which lecturers have preferences over (student,project) pairs. For such a model, can we construct an appropriate stability criterion, and is there an efficient algorithm to find a stable matching under this criterion? For further discussion of this problem see Abraham et al. [4].

2.4.4 SPA with ties

Much of the focus in the literature in recent years has been concerned with finding stable matchings where indifference is allowed in the preference lists. It is therefore natural to investigate the effects of introducing ties in the preference lists of the various SPA models.

2.4.5 SPA with lower bounds

For both SPA-P and the model presented in [4], we can introduce the idea of a lower bound z_j on a project p_j, where $z_j \leq c_j$. We can identify two models to consider. In the first model a matching M must satisfy the condition that $z_j \leq |M(p_j)| \leq c_j$, for each such p_j. In the second model we allow the possibility that a project can be empty, i.e. $|M(p_j)| \in \{0\} \cup \{k : z_j \leq k \leq c_j\}$. The value of 0 indicates that a project need not run,

otherwise it requires at least z_j participants in order to be viable. Clearly in both cases a stable matching that satisfies all the upper and lower bounds need not exist. Is the problem of finding a stable matching, if one exists, polynomial-time solvable under these conditions?

Chapter 3

Stable Matching Problems with Bounded Length Preference Lists

3.1 Introduction

In practice, there are many applications in which we seek to find a maximum weakly stable matching. In particular, in the Scottish Foundation Allocation Scheme (SFAS), each hospital ranks the medical students in order of preference, where a hospital's preference list may include ties. Additionally, each student ranks at most six hospitals, where students' preference lists do not contain ties. With such a scheme it is desirable to match the largest number of students possible subject to weak stability.

However, the existence of NP-hardness results indicates that efficient solutions for solving this problem are unlikely. This leads us to consider the complexity of the problem when imposing certain restrictions on the instance. For example, we may know that the length of an agent's preference list is bounded (as in the case of SFAS). In this chapter we look at such restrictions of the problem.

We recall from Section 1.1.7 that COM-SMTI, the problem of determining if a complete weakly stable matching exists, given an instance of SMTI, is known to be NP-complete and therefore that MAX-SMTI is NP-hard. Counterparts of both problems can also be defined for HRT. We denote the problem of finding a complete weakly stable matching for an instance of HRT by COM-HRT; a matching is said to be complete if all residents are matched and each hospital is full. Similarly, we denote the problem of finding a maximum weakly stable matching for an instance of HRT by MAX-HRT.

This chapter is structured as follows. Section 3.2 recalls some definitions and introduces

new terminology. Section 3.3 presents a polynomial-time algorithm for MAX-SMTI, where the preference lists of both men and women contain ties, the men's lists are of length at most 2 and the women's lists are of unbounded length. We then present a faster algorithm than that given in Section 3.3 for COM-SMTI, where the men's preference lists are of length at most 2, and the women's preference lists are of unbounded length. In Section 3.5 we present an NP-hardness result for MAX-SMTI, where each man's list has length at most 3 and each woman's list has length at most 4. Finally, in Section 3.6, we present an NP-completeness result for COM-HRT where each resident's list has length at most 3, and each hospital's list is unbounded in length.

3.2 Definitions

We recall that an instance of SMTI consists of a set of men $\mathcal{M} = \{m_1, m_2, \ldots, m_{n_1}\}$ and a set of women $\mathcal{W} = \{w_1, w_2, \ldots, w_{n_2}\}$. Additionally, each man in $\mathcal{M}$ ranks in order of preference a subset of the women in $\mathcal{W}$ and each women in $\mathcal{W}$ ranks in order of preference a subset of the men in $\mathcal{M}$, where both the men's and women's lists may contain ties. Similarly, an instance of HRT consists of a set of residents $\mathcal{R} = \{r_1, r_2, \ldots, r_{n_1}\}$ and a set of hospitals $\mathcal{H} = \{h_1, h_2, \ldots, h_{n_2}\}$. Each resident in $\mathcal{R}$ ranks in order of preference a subset of the hospitals in $\mathcal{H}$ and each hospital in $\mathcal{H}$ ranks in order of preference a subset of the residents, where both the resident's and the hospital's lists may contain ties.

In this chapter we consider restrictions on the preference lists and, in particular, upper bounds on their length. To represent this restriction on an instance of SMTI we use notation of the form (a,b)-SMTI, where $a \in \mathbb{N}$ indicates the upper bound on the men's lists and $b \in \mathbb{N}$ indicates the upper bound on the women's lists. When the men or women's lists are unbounded in length we use the value ∞ to represent this, e.g. $(2,\infty)$-SMTI. This notation may also be used for describing restrictions on an HRT instance (with the first parameter referring to the residents' lists and the second to the hospitals' lists). An example instance of (3,4)-SMTI is shown in Figure 3.1.

We may also wish to restrict MAX-SMTI to an instance of (a, b)-SMTI. In this case we denote the problem of finding a maximum weakly stable matching given an instance of (a, b)-SMTI by (a, b)-MAX-SMTI. The problems (a, b)-COM-SMTI, (a, b)-MAX-HRT, and (a, b)-COM-HRT are similarly defined.

Men's preferences	Women's preferences
m_1 : $(w_1 \;\; w_2) \;\; w_3$	w_1 : $(m_1 \;\; m_2)$
m_2 : $w_2 \;\; (w_1 \;\; w_3)$	w_2 : $m_2 \;\; m_1$
m_3 : w_3	w_3 : $(m_1 \;\; m_2) \;\; (m_3 \;\; m_4)$
m_4 : $(w_3 \;\; w_4)$	w_4 : m_4

Figure 3.1: (3,4)-SMTI instance I.

3.3 (2,∞)-MAX-SMTI

In this section we present a polynomial-time algorithm for MAX-SMTI where the preference lists of both men and women contain ties, the men's lists are of length at most 2 and the women's lists are of unbounded length.

Consider the algorithm $(2, \infty)$-MAX-SMTI-`alg` shown in Figure 14. The algorithm consists of three phases, where each phase is highlighted in the figure. We use the term *reduced lists* to refer to participants' lists after any deletions made by the algorithm. Phase 1 of $(2, \infty)$-MAX-SMTI-`alg` is a simple extension of the Gale-Shapley algorithm, and is used to delete certain (man,woman) pairs that can never be part of a weakly stable matching. Here to "delete the pair (m_i, w_j)", we delete m_i from w_j's list and delete w_j from m_i's list. Phase 1 proceeds as follows. All men are initially unmarked. While some man m_i remains unmarked and m_i has a non-empty reduced list, we set m_i to be marked – it is possible that m_i may again become unmarked at a later stage of the execution. If m_i's reduced list is not a tie of length 2, we let w_j be the woman in first position in m_i's reduced list. Then, for each strict successor m_k of m_i on w_j's list, we delete the pair (m_k, w_j) and set m_k to be unmarked (regardless of whether or not he was already marked). Here a person being marked is analogous to that of a proposal in the Gale-Shapley algorithm.

We remark that the following situation may occur during phase 1. Suppose that some man m_i is indifferent between two women w_j and w_k on his original preference list, and suppose that during some iteration of the while loop he becomes marked. We note that the algorithm does not delete the strict successors of m_i on w_j's list at this stage. Now suppose that, during a subsequent loop iteration, the pair (m_i, w_k) is deleted. Then m_i becomes unmarked, only to be re-marked during a subsequent loop iteration. This re-marking results in the deletions of all pairs (m_r, w_j), where m_r is a strict successor of m_i on w_j's list, as will be required.

Algorithm 14 Algorithm $(2, \infty)$-MAX-SMTI-`alg`.

```
1: /** Phase 1 */
2: set all men to be unmarked;
3: while some man m_i is unmarked and m_i has a non-empty reduced list do
4:    set m_i to be marked;
5:    if m_i's reduced list is not a tie of length 2 then
6:       w_j := woman in first position on m_i's reduced list;
7:       for each strict successor m_k of m_i on w_j's list do
8:          set m_k to be unmarked;
9:          delete the pair (m_k, w_j);
10:
11: /** Phase 2 */
12: G := BuildGraph();
13: M_G := minimum cost maximum matching in G;
14:
15: /** Phase 3 */
16: M := M_G;
17: while there exists a man m_i who is assigned to his second-choice woman w_k in M and
    his first-choice woman w_j is unassigned in M do
18:    M := M \ {(m_i, w_k)};
19:    M := M ∪ {(m_i, w_j)};
20: return M;
```

Algorithm 15 Algorithm `BuildGraph`.

```
1: V := M ∪ W;
2: E := ∅;
3: for each man m_i ∈ M do
4:    for each woman w_j on m_i's reduced list do
5:       E := E ∪ {(m_i, w_j)}
6:       cost(m_i, w_j) := rank(w_j, m_i);
7: G := (V, E);
8: return G;
```

In phase 2 we construct a weighted bipartite graph G and find a minimum cost maximum matching in G using the algorithm in [17]. The graph G is constructed using algorithm `BuildGraph` shown in Algorithm 15. That is, each man and woman is represented by a vertex in G, and for each man m_i on woman w_j's reduced list, we add an edge from m_i to w_j with cost $rank(w_j, m_i)$, where $rank(w_j, m_i)$ is the rank of m_i on w_j's reduced list (i.e. 1 plus the number of strict predecessors of m_i on w_j's reduced list). We then find a minimum cost maximum matching M_G in G.

In general, after phase 2, M_G need not be weakly stable in I. In particular, some man m_i who has a reduced list of length 2 that is strictly ordered may be assigned to his second-choice woman w_k in M_G, while his first-choice woman w_j may be unassigned in M_G. Clearly (m_i, w_j) blocks such a matching. To obtain a weakly stable matching M from M_G we execute phase 3. Initially, M is set to be equal to M_G. Next, we move each such m_i to his first-choice woman. We note that m_i must be in the *tail* of w_j's reduced list (this is the set of one or more entries tied in last place on w_j's reduced list) since m_i must have been marked during phase 1, causing all strict successors of m_i on w_j's list to be deleted. Further, we note that there may exist more than one such man in w_j's tail who satisfies the above criterion. Moreover when m_i moves to w_j, w_k becomes unassigned in M. As a result, there may be some other man m_r (who strictly ranks w_k in first place) who now satisfies the loop condition. This process is repeated until no such man exists. Upon termination of phase 3 we will show that the matching M returned is a maximum weakly stable matching.

We begin by showing that the algorithm $(2, \infty)$-MAX-SMTI-`alg` terminates. It is easy to see that phase 2 terminates. For phase 1, we observe that during an iteration of phase 1 some man becomes marked and if a man becomes unmarked, then at least one entry is deleted from his list, so the termination condition of the loop is bound to be satisfied. Hence phase 1 also terminates. The following lemma shows that the same is true for phase 3.

Lemma 3.3.1. *Phase 3 of* $(2, \infty)$-MAX-SMTI-`alg` *terminates.*

Proof. We show that the while loop terminates during an execution E of phase 3. For, at a given iteration of the while loop of phase 3, let m_i be some man assigned to his second-choice woman w_k in M and suppose that his first-choice woman w_j is unassigned in M, where m_i's reduced list is of length 2 and is strictly ordered. Then during E, m_i switches from w_k to w_j. Hence each such m_i must strictly improve (in fact m_i can only

improve at most once). Therefore since the number of men is finite, phase 3 is bound to terminate. □

We next show that phase 1 of $(2,\infty)$-MAX-SMTI-`alg` never deletes a *weakly stable pair*, which is a (man,woman) pair that belongs to some weakly stable matching in I.

Lemma 3.3.2. *The algorithm* $(2,\infty)$-MAX-SMTI-`alg` *never deletes a weakly stable pair.*

Proof. Let (m_i, w_j) be a pair deleted during an execution E of $(2,\infty)$-MAX-SMTI-`alg` such that $(m_i, w_j) \in M$, where M is a weakly stable matching in I. Without loss of generality suppose this is the first weakly stable pair to be deleted during E. Then m_i was deleted from w_j's list during some iteration z of the while loop in phase 1 during E. This deletion was made as a result of w_j being in first position on the reduced list of some man m_r, where m_r's reduced list was not a tie of length 2, and w_j prefers m_r to m_i. Then in M, m_r must obtain a woman w_s whom he prefers to w_j, otherwise (m_r, w_j) blocks M. Therefore during E, (m_r, w_s) must have already been deleted before iteration z, a contradiction. □

We prove that the matching returned by $(2,\infty)$-MAX-SMTI-`alg` is weakly stable in I.

Lemma 3.3.3. *The matching returned by algorithm* $(2,\infty)$-MAX-SMTI-`alg` *is weakly stable in* I.

Proof. Suppose for a contradiction that the matching M output by the algorithm $(2,\infty)$-MAX-SMTI-`alg` is not weakly stable. Then there exists a pair (m_i, w_j) that blocks M. We consider the following four cases corresponding to a blocking pair.

> *Case (i)*: both m_i and w_j are unassigned in M. Then m_i is unassigned in M_G, and either w_j is unassigned in M_G or becomes unassigned during phase 3. First suppose that w_j is unassigned in M_G. Then the size of the matching M_G could be increased by adding the edge (m_i, w_j) to M_G, contradicting the fact that M_G is a maximum matching. Now suppose that w_j became unassigned as a result of phase 3. Let m_{p_1} denote w_j's partner in M_G. Then during phase 3, m_{p_1} must have become assigned to his first-choice woman w_{q_1}. Suppose w_{q_1} was unassigned in M_G. Then we can find a larger matching by augmenting along the path $(m_i, w_j), (w_j, m_{p_1}), (m_{p_1}, w_{q_1})$, contradicting the fact that M_G is a maximum matching. Therefore w_{q_1} must have been assigned in M_G and became unassigned as a result of phase 3. Hence the man m_{p_2}, to whom w_{q_1} was assigned in M_G, switched to his first-choice woman

w_{q_2}. Using an argument similar to that above for w_{q_1}, we can show that w_{q_2} must be assigned in M_G. Therefore some man switched from w_{q_2} during phase 3 to his first-choice woman. If we continue this process, since each man must strictly improve and the number of men is finite, there exists a finite number of women that can become unassigned as a result of phase 3. Hence at some point there exists a man m_{p_r} who switches to his first-choice woman w_{q_r} and w_{q_r} was already unassigned in M_G. We can then construct an augmenting path in G of the form $(m_i, w_j), (w_j, m_{p_1}), (m_{p_1}, w_{q_1}), (w_{q_1}, m_{p_2}), (m_{p_2}, w_{q_2}), \ldots, (m_{p_r}, w_{q_r})$, which contradicts the fact that M_G is a maximum matching.

Case (ii): m_i is unassigned in M and w_j prefers m_i to her assignee m_k in M. Then m_i is unassigned in M_G. Suppose that w_j is assigned to m_k in M_G. As w_j prefers m_i to m_k, we could obtain a matching with a smaller cost, but with the same size, by removing (m_k, w_j) and adding (m_i, w_j) to M_G, a contradiction. Now suppose that w_j is not assigned to m_k in M_G. Then w_j is either unassigned in M_G or w_j is assigned in M_G to m_r, where $m_r \neq m_k$ and $m_r \neq m_i$. If w_j is unassigned in M_G, we contradict the fact that M_G is a maximum matching. Now suppose w_j is assigned to m_r in M_G. Then since w_j is no longer assigned to m_r in M, m_r must have switched to his first-choice woman w_s during phase 3. Therefore either w_s is unassigned in M_G or w_s became unassigned as a result of some man switching from w_s to his first-choice woman. Again, using a similar argument to that in Case (i), we obtain an augmenting path that contradicts the fact that M_G is a maximum matching.

Case (iii): m_i is assigned to w_s in M and m_i prefers w_j to w_s and w_j is unassigned in M. Thus clearly m_i's list is of length 2 and does not contain a tie, and w_j is m_i's first-choice woman. In this situation the loop condition of phase 3 is satisfied. Therefore since the algorithm terminates (Lemma 3.3.1) this situation can never arise.

Case (iv): m_i is assigned to w_s in M and m_i prefers w_j to w_s, and w_j is assigned to m_r in M and w_j prefers m_i to m_r. Thus again m_i's list cannot contain a tie, and w_j is his first-choice woman. Therefore either m_i proposed to w_j during phase 1 or w_j was deleted from m_i's list. Hence in either case, m_r would have been deleted from w_j's list during phase 1, so it is then impossible that $(m_r, w_j) \in M$.

□

Since phase 1 of the algorithm never deletes a weakly stable pair (by Lemma 3.3.2), a maximum weakly stable matching must consist of (man,woman) pairs that belong to the reduced lists. Furthermore we note that G is constructed from the reduced lists, and since we find a maximum matching in G, the matching output by the algorithm must indeed be a maximum weakly stable matching (by Lemma 3.3.3, and since phase 3 does not change the size of the matching output by the algorithm: every man matched in M_G is also matched in M).

The time complexity of the algorithm is dominated by finding a minimum cost maximum matching in $G = (V, E)$. The required matching in G can be constructed in $O(\sqrt{|E|}|V|\log|V|)$ time [17]. Let $n = |V| = n_1 + n_2$. Since $|E| \leq 2n_1 = O(n)$, it follows that $(2,\infty)$-MAX-SMTI-`alg` has time complexity $O(n^{\frac{3}{2}}\log n)$.

We summarise the results of this section in the following theorem.

Theorem 3.3.4. *Given an instance I of* $(2,\infty)$-SMTI*, algorithm* $(2,\infty)$-MAX-SMTI-`alg` *returns a weakly stable matching of maximum size in $O(n^{\frac{3}{2}}\log n)$ time, where n is the total number of men and women in I.*

3.4 (2,∞)-COM-SMTI

In this section we present a polynomial-time algorithm for $(2,\infty)$-COM-SMTI. The time complexity of the algorithm is an improvement over that of the algorithm described in Section 3.3.

Algorithm (2,∞)-COM-SMTI-`alg` shown in Algorithm 16 determines if a complete weakly stable matching exists for an instance I of (2,∞)-SMTI and outputs such a matching should one exist. The algorithm iterates over each man $m_i \in \mathcal{M}$: if m_i's preference list does not contain a tie, we identify m_i's first-choice woman w_j, and delete each strict successor m_r of m_i on w_j's list. In the algorithm "delete the pair (m_r, w_j)" refers to deleting m_r from w_j's list and w_j from m_r's list. Next we build the underlying graph $G = (V, E)$, where $V = \mathcal{M} \cup \mathcal{W}$ and E is constructed by adding an edge from each man in $m_i \in \mathcal{M}$ to each woman that remains on m_i's list after any deletions are made. Finally G admits a perfect matching M if and only if there is a complete weakly stable matching in I.

We prove, using the following lemmas, that if (2,∞)-COM-SMTI-`alg` returns a matching, for an instance I of (2,∞)-SMTI, then the matching is complete, and weakly stable. In

Algorithm 16 (2,∞)-COM-SMTI-alg

```
1: for each man m_i ∈ M do
2:    if m_i's preference list does not contain a tie then
3:       w_j := m_i's first-choice woman;
4:       for each strict successor m_r of m_i on w_j's list do
5:          delete the pair (m_r, w_j);
6: build underlying graph G;
7: if G admits a perfect matching M then
8:    output M;
9: else
10:   output "no complete weakly stable matching exists";
```

addition we also show that if no matching is output, then no complete weakly stable matching exists for I.

Lemma 3.4.1. *If a matching M is output on termination of* (2,∞)-COM-SMTI-alg*, then M is weakly stable.*

Proof. Let E be an execution of the algorithm that outputs a matching M on termination of E. Suppose that M is not weakly stable, and that (m_i, w_j) blocks M. Then m_i's list cannot contain a tie. Let m_k denote w_j's assignee in M; hence w_j prefers m_i to m_k. Therefore (m_i, w_j) cannot have been deleted during E, for otherwise (m_k, w_j) would also have been deleted. As M is complete, m_i must be assigned in M to his second-choice partner. However, w_j must be matched in M to a man no worse than m_i in M, as all successors of m_i on w_j's list were deleted prior to finding the matching. Hence (m_i, w_j) cannot block M and therefore M is weakly stable. □

Lemma 3.4.2. *If on termination of* (2,∞)-COM-SMTI-alg *the algorithm reports that no complete weakly stable matching exists, then indeed no complete weakly stable matching exists.*

Proof. Let E be an execution of the algorithm that outputs the message that no complete weakly stable matching exists for an instance I of (2,∞)-SMTI. Now suppose that M is a complete weakly stable matching in I. Let G be the graph constructed from the reduced preference lists. Then G admits no perfect matching. We observe that no weakly stable pair is ever deleted by the algorithm. For, suppose that a pair $(m_i, w_j) \in M'$ for some weakly stable matching M' and the pair is deleted by the algorithm. Then some man m_r,

whose preference list is not a tie of length 2, has w_j as his first-choice and w_j strictly prefers m_r to m_i. Hence (m_r, w_j) blocks M', a contradiction. Since no weakly stable pair is ever deleted, we can form a perfect matching in G by matching together each (man,woman) pair in M to form a perfect matching in G, contradicting the fact that G admits no perfect matching. Hence no weakly stable matching exists in I. □

We now analyse the running time of (2,∞)-COM-SMTI-`alg`. We note that the overall complexity of the nested for loops is $O(|E|) = O(n_1)$ since each man's preference list is of length ≤ 2, thus it follows that the complexity of the algorithm is bounded by the time taken to find a maximum cardinality matching in the underlying graph. The Hopcroft-Karp algorithm for finding a maximum cardinality matching in a graph $G = (V, E)$ has time complexity $O(\sqrt{|V|}|E|)$ [32]. Since each man's preference lists is of length ≤ 2, $|E| = O(n_1)$. Hence in this case we can find a maximum cardinality matching in the graph in $O(\sqrt{n_1 + n_1}n_1) = O(n^{\frac{3}{2}})$ time. We summarise this result and the others in this section in the following theorem.

Theorem 3.4.3. *Given an instance I of* (2,∞)-SMTI*, algorithm* (2,∞)-COM-SMTI-`alg` *returns a complete weakly stable matching, or else reports that no such matching exists, in $O(n^{\frac{3}{2}})$ time.*

3.5 (3,4)-MAX-SMTI

In this section we show that, in contrast to (2,∞)-MAX-SMTI, (3,4)-MAX-SMTI is NP-hard, and hence that (k,l)-MAX-SMTI is NP-hard for any $k \geq 3$ and $l \geq 4$. The decision version of (3,4)-MAX-SMTI is shown below.

Name :	(3,4)-MAX-SMTI-D
Instance:	An SMTI instance I where each man's list has length at most 3 and each woman's list has length at most 4, and an integer K.
Question:	Does I have a weakly stable matching of size $\geq K$?

We prove, in the following theorem, that (3,4)-MAX-SMTI-D is NP-complete by reducing from MIN-MM-D, which as noted in Section 2.2.2 of Chapter 2 is NP-complete, even for subdivision graphs of cubic graphs.

Theorem 3.5.1. (3,4)-MAX-SMTI-D *is NP-complete*

Proof. Clearly (3,4)-MAX-SMTI-D is in NP. To prove that (3,4)-MAX-SMTI-D NP-hard we reduce from MIN-MM-D, restricted to subdivision graphs of cubic graphs. Let G be the subdivision graph of some cubic graph G' and let K (a positive integer) be an instance of MIN-MM-D. Then G is a bipartite graph $G = (U, W, E)$, where without loss of generality each vertex in U has degree 2 and each vertex in W has degree 3. Let $U = \{m_1, m_2, \ldots, m_{n_1}\}$ and $W = \{w_1, w_2, \ldots, w_{n_2}\}$. Without loss of generality we may assume that $K \leq \min\{n_1, n_2\}$. For each $m_i \in U$, let U_i denote the two vertices that are adjacent to m_i in G. Similarly for each $w_j \in W$, let W_j denote the three vertices that are adjacent to w_j in G.

We construct an instance I of (3,4)-MAX-SMTI-D as follows: let $U \cup X$ be the set of men and $W \cup Y$ be the set of women, where $X = \{x_1, x_2, \ldots, x_{n_2}\}$ and $Y = \{y_1, y_2, \ldots, y_{n_1}\}$. The preference lists of the men and women in I are shown in Figure 3.2. In a given preference list, entries that appear within round brackets are tied. Let $K' = n - K$, where $n = n_1 + n_2$. We claim that G has a maximal matching of size $\leq K$ if and only if I admits a weakly stable matching of size $\geq K'$.

Men's preferences				Women's preferences			
m_i :	(U_i)	y_i	$(1 \leq i \leq n_1)$	w_j :	(W_j)	x_j	$(1 \leq j \leq n_2)$
x_i :	w_i		$(1 \leq i \leq n_2)$	y_j :	m_j		$(1 \leq j \leq n_1)$

Figure 3.2: Preference lists for the constructed instance of (3,4)-MAX-SMTI-D.

Suppose that G has a maximal matching M, where $|M| = t_1 \leq K$. We construct a matching M' in I as follows. Initially let $M' = M$. There remain $n_1 - t_1$ men in U that are unmatched in M'; denote these men by m_{k_i} $(1 \leq i \leq n_1 - t_1)$, and add (m_{k_i}, y_{k_i}) to M'. Finally there remain $n_2 - t_1$ women in W that are unmatched in M'; denote these women by w_{l_j} $(1 \leq j \leq n_2 - t_1)$, and add (x_{l_j}, w_{l_j}) to M'. Clearly each man in M' is only matched to a single woman, and vice-versa. Hence M' is a matching in I, and $|M'| = t_1 + (n_1 - t_1) + (n_2 - t_1) = n - t_1 \geq n - K = K'$.

We now prove that M' is weakly stable. No man in X and no woman in Y can be involved in a blocking pair of M', since every person in $U \cup W$ is matched in M'. Now suppose that (m_i, w_j) blocks M', where $m_i \in U$ and $w_j \in W$. Therefore $(m_i, y_i) \in M'$, and $(x_j, w_j) \in M'$. Hence each of m_i and w_j is unmatched in M. Moreover, the edge

$\{m_i, w_j\}$ belongs to E, so that $M \cup \{\{m_i, w_j\}\}$ is a matching in M, contradicting the maximality of M. Hence M' is stable in I.

Conversely suppose that M' is a weakly stable matching in I, where $|M'| \geq K'$. Let $M = M' \cap E$, and $t_2 = |M|$. Then in M', at most $n_1 - t_2$ men in U are matched to woman in Y, and at most $n_2 - t_2$ women in W are matched to men in X. Hence $|M'| \leq t_2 + (n_1 - t_2) + (n_2 - t_2) = n - t_2$, therefore $|M| \leq n - K' = K$.

Now suppose that M is not a maximal matching in G. Then there exists an edge $\{m_i, w_j\}$ such that no edge in M is incident to m_i or w_j. Therefore in M', either m_i is unmatched or $(m_i, y_i) \in M'$, and either w_j is unmatched or $(x_j, w_j) \in M'$. Thus (m_i, w_j) blocks M' in I. This is a contradiction to the weak stability of M', and hence M is indeed maximal in G. □

3.6 (3,∞)-COM-HRT

In this section we consider HRT, the many-to-one generalisation of SMTI. We present an NP-completeness result for the problem of finding a complete weakly stable matching given an instance of (3,∞)-HRT. Here a "complete" weakly stable matching M means that each resident is matched in M and all hospitals are full in M. The problem is defined formally below.

Name:	(3,∞)-COM-HRT
Instance:	An HRT instance I where each resident has a preference list of length at most 3 and each hospital has a preference list of unbounded length.
Question:	Does I have a complete weakly stable matching M?

We use the NP-completeness of EXACT-MM in subdivision graphs (as established by Lemma 2.2.1 in Chapter 2) to show that (3,∞)-COM-HRT is also NP-complete.

Theorem 3.6.1. (3,∞)-COM-HRT *is NP-complete.*

Proof. Clearly (3,∞)-COM-HRT is in NP. To show NP-hardness we reduce from EXACT-MM restricted to subdivision graphs. Hence let G (the subdivision graph of some graph G') and K (a positive integer) be an instance of EXACT-MM. Then G is a bipartite graph, so that $G = (R, H, E)$, where, without loss of generality, all vertices in R have degree 2. Suppose that $n_1 = |R|$ and $n_2 = |H|$. Again without loss of generality suppose that

$K \leq \min\{n_1, n_2\}$. Let $R = \{r_1, r_2, \ldots, r_{n_1}\}$ and $H = \{h_1, h_2, \ldots, h_{n_2}\}$. For each $r_i \in R$, let R_i denote the two vertices that are adjacent to r_i in G. Similarly for each $h_j \in H$, let H_j denote the vertices that are adjacent to h_j in G.

We construct an instance I of (3,∞)-COM-HRT as follows: let $R \cup X$ be the set of residents, where $X = \{x_1, x_2, \ldots, x_{n_2}\}$ and let $H \cup \{y, y'\}$ be the set of hospitals. Each hospital $h_j \in H$ has capacity 1, y has capacity $n_1 - K$, and y' has capacity K. The preference lists of I are shown in Figure 3.3. In these preference lists, residents who appear within the square brackets are listed in arbitrary strict order. Also those participants that appear within round brackets are tied with each other. We claim that G has a maximal matching of size K if and only if I admits a complete weakly stable matching.

Residents' preferences		Hospitals' preferences			
r_i :	$(R_i)\ \ y$ $(1 \leq i \leq n_1)$	h_j :	(1) :	$(H_j)\ \ x_j$	$(1 \leq j \leq n_2)$
x_i :	$(h_i\ \ y')$ $(1 \leq i \leq n_2)$	y :	$(n_1 - K)$:	[residents in R]	
		y' :	(K) :	[residents in X]	

Figure 3.3: Preference lists for the constructed instance of (3,∞)-COM-HRT.

Suppose that G has a maximal matching M, where $|M| = K$. We construct a matching M' in I as follows. Initially let $M = M'$. There remain $n_1 - K$ residents in R who are unmatched in M'; denote these residents by r_{k_i} $(1 \leq i \leq n_1 - K)$, and add (r_{k_i}, y) to M' for each i. There also remain $n_2 - K$ hospitals in H that are under-subscribed in M'; denote these hospitals by h_{l_i} $(1 \leq i \leq n_2 - K)$, and add (x_{l_i}, h_{l_i}) to M' for each i. This leaves K residents in X who are unmatched in M'; denote these residents by x_{p_i} $(1 \leq i \leq K)$, and add (x_{p_i}, y') to M'. Clearly each hospital is full in M', and each resident is assigned to one hospital in M'. Hence it follows that M' is a complete matching. We now show that M' is weakly stable.

No resident in X can be involved in a blocking pair. Similarly y cannot be involved in a blocking pair as every resident in R is matched. Hence a blocking pair must be of the form (r_i, h_j), where $r_i \in R$ and $h_j \in H$. Were such a blocking pair to exist, r_i must be assigned to y in M', and h_j must be assigned to some resident $x_k \in X$ in M'. Hence neither r_i nor h_j is matched in M, but r_i and h_j are adjacent in G, therefore $M \cup \{\{r_i, h_j\}\}$ is a matching in G, contradicting the maximality of M. Thus M' is weakly stable.

Conversely suppose that M' is a complete weakly stable matching in I. Let $M =$

$M' \cap E$. We now prove that $|M| = K$. Suppose $|M| < K$. This means that at least $n_1 - K + 1$ residents in R are assigned to y in M', hence y is over-subscribed, contradicting the fact that M' is a matching. Now suppose $|M| > K$. If this is the case then there are at most $n_2 - K - 1$ hospitals in H assigned to residents in X in M'. Therefore at least $K + 1$ residents in X must be assigned to y' in M', hence y' is over-subscribed, contradicting the fact that M' is a matching. Hence $|M| = K$.

Finally, suppose that M is not a maximal matching in G. Then there exists an edge $\{r_i, h_j\}$ in G such that no edge of M is incident to r_i or h_j. Therefore in M', r_i must be assigned to y, and h_j must be assigned to a resident in X. Thus (r_i, h_j) blocks M'. This contradiction to the weak stability of M' implies that M is indeed maximal. □

We remark that the above proof shows NP-completeness of (3,∞)-COM-HRT even if drop the requirement that a hospital should be full in a complete weakly stable matching.

3.7 Open Problems

In this section we provide an overview of the problems with bounded length preference lists that remain open. The table below shows the details of these problems.

Problem	Complexity	Reference
SMTI		
(3,4)-MAX-SMTI	NP-hard	Section 3.5.
(3,3)-COM-SMTI	NP-complete	See [56].
(2,∞)-MAX-SMTI	P	Section 3.3.
HRT		
(2,∞)-MAX-HRT	Open	Section 3.7.1.
(2,∞)-COM-HRT	Open	
(3,∞)-COM-HRT	NP-complete	Section 3.6 and [56].

3.7.1 (2,∞)-MAX-HRT

It seems natural to consider a straightforward extension of the algorithm given for (2,∞)-MAX-SMTI to the case of (2,∞)-MAX-HRT. However such an extension does not appear to

be obvious and the algorithm given in Section 3.3 does not appear to generalise directly to the case of (2,∞)-MAX-HRT. Is the problem of finding a maximum weakly stable matching given an instance of (2,∞)-HRT NP-hard or polynomial-time solvable?

Chapter 4

The Hospitals/Residents Problem with Master Lists

4.1 Introduction

In the context of large-scale centralised matching schemes based on HR, it is often difficult for a hospital to individually rank a large number of applicants in order of preference. For example, in the NRMP, a hospital may typically rank hundreds of medical students in preference order. Ranking a large number of agents in such a way can be a laborious and error-prone process. However in the context of such schemes it is often the case that hospitals have access to a single uniform ranking of all their applicants according to some objective criteria such as academic merit. As such, it is useful to consider the problems that arise when each hospital's list is derived from a single *master list*. In this chapter we consider the algorithmic complexity of variants of HRT where the preference list on one or both sides are derived from one or two master lists involving the residents and/or hospitals.

Stable matching problems involving master list have been considered previously. Scott [67] describes two variants of the master lists problem for SMTI. The first involves an instance of SMTI where the preference lists are derived from a master list on one side only; we denote this by SMTI-1ML. For example, consider an instance I of SMTI-1ML with a master list of men. Then in I, each woman's preference list is derived from this single master list of men. That is, each woman w_j's list consists of the master list with her unacceptable partners deleted; therefore w_j's ranking is inherited from that of the master list. An example instance of SMTI-1ML is shown in Figure 4.1. In general throughout this

section we assume that an instance of SMTI-1ML contains a master list of men and that the derived lists are consistent. Also Scott describes an instance of SMTI in which preference lists on both sides are derived from two master lists; we denote this by SMTI-2ML. We similarly define both HRT-1ML and HRT-2ML with respect to HRT. Here we assume that, for an instance of HRT-1ML, the master list comprises of residents. An example instance of HRT-1ML is shown in Figure 4.2.

Men's preferences	Women's preferences
	Master list: $(m_1 \;\; m_2) \;\; (m_3 \;\; m_4)$
$m_1 : \;\; w_2 \;\; w_3 \;\; w_4$	$w_1 : \;\; (m_3 \;\; m_4)$
$m_2 : \;\; (w_2 \;\; w_3)$	$w_2 : \;\; (m_1 \;\; m_2) \;\; m_4$
$m_3 : \;\; w_1 \;\; (w_3 \;\; w_4)$	$w_3 : \;\; (m_1 \;\; m_2) \;\; (m_3 \;\; m_4)$
$m_4 : \;\; (w_2 \;\; w_3) \;\; (w_1 \;\; w_4)$	$w_4 : \;\; m_1 \;\; (m_3 \;\; m_4)$

Figure 4.1: Example instance of SMTI-1ML.

It is known that the problem of finding a maximum weakly stable matching for an instance of SMTI, and therefore for an instance of HRT, is NP-hard [54]. Scott [67] shows that for an instance of SMTI-2ML (and therefore also SMTI-1ML) the problem remains NP-hard. However, Irving et al. [41] describe a polynomial-time algorithm that finds a super-stable matching for an instance of SMTI-1ML if such a matching exists, and they also show that the matching is in fact unique if it exists. In the same paper, Irving et al. also describe a faster polynomial-time algorithm (over the general case [53]) which obtains a strongly stable matching, or reports that none exists, given an instance of SMTI-1ML.

We first describe in Section 4.2 a simple algorithm for finding a stable matching given an instance of HR-1ML (where HR-1ML is an instance of HR with a master list of residents).

Residents' preferences	Hospitals' preferences
	Master list: $r_1 \;\; r_2 \;\; (r_3 \;\; r_4)$
$r_1 : \;\; (h_1 \;\; h_2) \;\; h_3$	$h_1 : (1) : \;\; r_1 \;\; r_2 \;\; (r_3 \;\; r_4)$
$r_2 : \;\; h_1 \;\; h_3$	$h_2 : (2) : \;\; r_1 \;\; r_3$
$r_3 : \;\; (h_1 \;\; h_2)$	$h_3 : (2) : \;\; (r_1 \;\; r_2) \;\; r_4$
$r_4 : \;\; h_1 \;\; h_3$	

Figure 4.2: Example instance of HRT-1ML.

We then adapt the result in Section 3.6 of Chapter 3, to prove that, even in the presence of a master list of residents and a master list of hospitals, and even if each resident has a list of length at most 3 and each hospital's list is of unbounded length, the problem of determining if a complete weakly stable matching exists remains NP-complete. We then extend the results for super-stability and strong stability in SMTI-1ML to HRT-1ML. In Section 4.4 we present a polynomial-time algorithm that finds a super-stable matching, or reports that none exists, for an instance of HRT-1ML. We also prove that if a super-stable matching is returned then it is in fact unique. In Section 4.5 we describe a polynomial-time algorithm that finds a strongly stable matching, or reports that none exists, for an instance of HRT-1ML in time $O(\sqrt{C}\lambda)$. The time complexity of our algorithm is an improvement over that for the general case, namely $(O(C\lambda))$ [45].

4.2 HR-1ML

We first consider the problem of finding a stable matching for an instance of HR-1ML. It is known that this problem is polynomial-time solvable using the algorithm described by Gale and Shapley [18]. However in this section we describe a simplified version of the algorithm for an instance of HR-1ML and prove that the matching returned by the algorithm is in fact unique.

Algorithm 17 shows algorithm HR-1ML-`alg` for finding a stable matching M for an instance I of HR-1ML. First we set M to be empty. We then process each resident r_i on the master list in turn, where r_i's list contains an under-subscribed hospital. Let h_j be the first under-subscribed hospital on r_i's list. We now simply add (r_i, h_j) to M and repeat the process for each such resident on the master list in turn .

Algorithm 17 HR-ML-`alg`

```
1: M := ∅;
2: for each resident r_i in the master list in turn and
   r_i's list contains an under-subscribed hospital do
3:     h_j := first under-subscribed hospital on r_i's list;
4:     M := M ∪ {(r_i, h_j)};
5: return M;
```

We now prove in the following lemma that the matching returned by HR-ML-`alg` is stable.

Lemma 4.2.1. *The matching returned by* HR-ML-`alg` *is stable.*

Proof. Suppose, for a contradiction, that the algorithm returns a matching M that is not stable. Hence there exists a pair (r_i, h_j) that blocks M. As r_i is unassigned in M or assigned to a hospital worse than h_j in M, then at the iteration of the for loop where resident r_i is processed, h_j must have been full. Also since the residents are processed in preference order, h_j must be full with residents it prefers to r_i in M. Hence (r_i, h_j) cannot block M, a contradiction. Therefore M is indeed a stable matching. □

The lemma below proves that the matching returned by HR-ML-`alg` for an instance I of HR-1ML is the unique stable matching for I.

Lemma 4.2.2. *The matching M returned by* HR-ML-`alg` *for an instance I of* HR-1ML *is the unique stable matching in I.*

Proof. Suppose, for a contradiction, that there exists a stable matching M' in I, such that $M' \neq M$. Hence there exists a pair $(r_i, h_j) \in M$ such that $(r_i, h_j) \notin M'$. Without loss of generality let r_i be the first resident on the master list with this property. Since each resident before r_i on the master list must obtain the same hospital in M as in M' and since h_j was r_i's most preferred under-subscribed hospital at this point, r_i must obtain a hospital worse than h_j in M' – we note that, by the Rural Hospitals Theorem (Theorem 1.2.1), r_i cannot be unassigned in M'. However since every resident better than r_i on the master list has the same hospital in M as in M', it follows that h_j must either be under-subscribed in M' or be full with at least one resident worse than r_i in M'. Therefore (r_i, h_j) blocks M', a contradiction. Hence M is the unique stable matching in I. □

We now consider the time complexity of the algorithm. We firstly note that each preference list entry is examined at most once. Therefore the overall complexity of the algorithm is $O(\lambda)$ time, where λ is the total length of the preference lists. We bring together the above results in the following theorem.

Theorem 4.2.3. *For a given instance I of* HR-1ML*, algorithm* HR-ML-`alg` *outputs the unique stable matching for I in time $O(\lambda)$, where λ is the total length of the preference lists.*

4.3 HRT-2ML under weak stability

As shown in Section 3.6, the problem of deciding if a complete weakly stable matching exists for an instance of HRT is NP-complete, even if the resident's lists are of size 3 and the hospital's lists are of unbounded length. In this section we show that in addition, even if we have both a master list of residents and a master list of hospitals, the problem remains NP-complete. We define the problem (3,∞)-COM-HRT-2ML as follows.

Name:	(3,∞)-COM-HRT-2ML
Instance:	An HRT instance I where each resident has a preference list of length at most 3 which is derived from a master list of hospitals, and each hospital has a preference list of of unbounded length which is derived from a master list of residents.
Question:	Does I have a complete weakly stable matching M?

Theorem 4.3.1. (3,∞)-COM-HRT-2ML *is NP-complete.*

Proof. Clearly (3,∞)-COM-HRT-2ML is in NP. To prove NP-hardness we use a reduction from EXACT-MM similar to that in Theorem 3.6.1. We make a slight modification in order to construct an instance of (3,∞)-COM-HRT with master lists on both sides. An instance I of (3,∞)-COM-HRT-2ML is constructed as follows: let $R \cup X$ be the set of residents, where $X = \{x_1, x_2, \ldots, x_{n_2}\}$ and let $H \cup \{y, y'\}$ be the set of hospitals. Each hospital $h_j \in H$ has capacity 1, y has capacity $n_1 - K$, and y' has capacity K. The preference lists of I are shown in Figure 4.3. Additionally, also shown in Figure 4.3, are the master lists of hospitals and residents. In these preference lists, residents who appear within the square brackets are listed in arbitrary strict order. Also those participants that appear within round brackets are tied with each other. We claim that G has a maximal matching of size K if and only if I admits a complete weakly stable matching.

The remainder of the proof is identical to that of Theorem 3.6.1. □

We note that in this case we require arbitrary length ties in a hospital y's preference lists. There does not appear to be an obvious way to extended the NP-completeness reduction to have ties that are shorter in length.

Residents' preferences	Hospitals' preferences
Master list: (hospitals in H y') y	Master list: (residents in R) [residents in X]

r_i : (R_i) y $(1 \leq i \leq n_1)$

x_i : $(h_i$ $y')$ $(1 \leq i \leq n_2)$

h_j : (1) : (H_j) x_j $(1 \leq j \leq n_2)$

y : $(n_1 - K)$: (residents in R)

y' : (K) : [residents in X]

Figure 4.3: Preference lists for the constructed instance of (3,∞)-COM-HRT-2ML.

4.4 HRT-1ML under super-stability

In this section we present algorithm HRT-ML-`super`, which finds a super-stable matching, or reports that none exists, given an instance of HRT-1ML. The complexity of the algorithm is comparable to that given in [39] for the general HRT case, however the algorithm is simpler and helps to identify key structural properties of a super-stable matching for an instance of HRT-1ML.

Algorithm HRT-ML-`super` is shown in Algorithm 18. The pseudocode uses the following notation. Suppose that master list tie T containing resident r_i is processed during an iteration z of the main loop of HRT-ML-`super`. We refer to the (possibly empty) set of hospitals tied at the head of r_i's list as r_i's *key hospitals*, denoted by H_i. If the algorithm returns null before the master list tie T containing resident r_i is processed then H_i is undefined. Note that the definition of H_i for a given resident r_i is well defined as the algorithm is deterministic.

The algorithm proceeds as follows. For an instance I of HRT-1ML, we first set the matching M to be empty. Then for each tie T in the master list in turn we identify the set of residents P contained in T. If at least one such resident has more than one key hospital, then (as we will show) no super-stable matching exists for I. Otherwise we construct the set Q from the union of the sets H_i for each resident $r_i \in P$. Then for each hospital $h_j \in Q$, if the number of residents that have h_j as a key hospital exceeds the number of posts that h_j has remaining, then we will show that no super-stable matching exists for I. Otherwise we can assign each resident r_i his (unique) key hospital. If during this process any hospital becomes full we identify each strict successor r_k of r_i on the master list and "delete the pair (r_k, h_j)", which comprises deleting h_j from r_k's list and vice-versa

(however no deletion is made from the master list).

Algorithm 18 HRT-ML-super

```
1: M := ∅;
2: for each tie T in the master list in turn do
3:    P := set of residents in T;
4:    for each r_i ∈ P do
5:       if |H_i| ≥ 2 then
6:          return null;
7:
8:    Q := ⋃_{r_i ∈ P} H_i;
9:    for each h_j ∈ Q do
10:      A_j := {r_i : H_i = h_j};
11:      if |A_j| + |M(h_j)| > c_j then
12:         return null;
13:
14:   for each r_i ∈ P where r_i has a non-empty list do
15:      h_j := r_i's key hsopital; /** H_i = {h_j} */
16:      M := M ∪ {(r_i, h_j)};
17:      if h_j is full then
18:         for each strict successor r_k of r_i on h_j's list do
19:            delete the pair (r_k, h_j);
20: return M;
```

To establish the correctness of the algorithm, we begin by showing that a pair that belongs to a super-stable matching is never deleted.

Lemma 4.4.1. *Algorithm* HRT-ML-super *never deletes a pair* (r_i, h_j) *that belongs to a super-stable matching.*

Proof. Let M be a super-stable matching for an instance I of HRT-1ML. Suppose for a contradiction that $(r_i, h_j) \in M$ and that (r_i, h_j) is deleted during an execution E of HRT-ML-super. Without loss of generality suppose that this is the first super-stable pair to be deleted during E. Then (r_i, h_j) is deleted when h_j becomes full during E with a set of assignees S that it strictly prefers to r_i. Therefore in M at least one resident $r_s \in S$ must obtain a hospital h_t that he strictly prefers to h_j, for otherwise (r_s, h_j) blocks M. However when the algorithm processes the master list tie containing r_s, it matches r_s with h_j at the head of r_s's list, so the super-stable pair (r_s, h_t) was already deleted, a contradiction. □

We next show using Lemma 4.4.1 that if HRT-ML-`super` outputs a matching then the matching is a super-stable matching.

Lemma 4.4.2. *If a matching M is output by* HRT-ML-`super`*, then M is a super-stable matching.*

Proof. Suppose for a contradiction that there exists a pair (r_i, h_j) that blocks M. If (r_i, h_j) had been deleted then h_j is full with assignees it prefers to r_i, contradicting the fact that (r_i, h_j) blocks M. Hence the pair (r_i, h_j) has not been deleted. In M, r_i cannot be unassigned, for otherwise r_i's list is empty by the fourth for loop condition, a contradiction to the fact that (r_i, h_j) has not been deleted. Hence $(r_i, h_k) \in M$, where $H_i = \{h_k\}$, for otherwise the algorithm would have returned null rather than outputting M. Since (r_i, h_j) blocks M, it follows that r_i strictly prefers h_j to h_k or is indifferent between them. Hence (r_i, h_j) was deleted during the algorithm's execution, a contradiction. □

In the following lemma we show that in every super-stable matching M for an instance of HRT-1ML, if $H_i = \emptyset$ then r_i is unassigned in M, otherwise r_i obtains a partner in H_i.

Lemma 4.4.3. *Let r_i be a resident whose set of key hospitals H_i is defined. Let M be a super-stable matching. If $H_i = \emptyset$ then r_i is unassigned in M, otherwise $(r_i, h_j) \in M$, for some $h_j \in H_i$.*

Proof. Let E be an execution of HRT-ML-`super` for an instance I of HRT-1ML. At some iteration z of the main loop during E, suppose that the tie T containing resident r_i is processed. We firstly note that if $H_i = \emptyset$ then r_i's preference list is empty at the beginning of iteration z, and hence r_i is unassigned in M by Lemma 4.4.1. Now suppose that H_i is non-empty. We consider four cases.

> *Case (i)*: r_i obtains a hospital h_j in M that he strictly prefers to the hospitals in H_i. Therefore at the beginning of iteration z the pair (r_i, h_j) must have already been deleted. Hence by Lemma 4.4.1 the pair (r_i, h_j) can never be super-stable, a contradiction.
>
> *Case (ii)* r_i obtains a hospital h_j in M such that $h_j \notin H_i$ and r_i is indifferent between h_j and the hospitals in H_i. Then (r_i, h_j) was deleted prior to iteration z, so that (r_i, h_j) cannot be a super-stable pair by Lemma 4.4.1, a contradiction.

Case (iii): r_i obtains a hospital in M that he finds inferior to the hospitals in H_i. Let r_i be the most-preferred resident (according to the master list) with this property. Let $h_j \in H_i$. Then in M, h_j must be full with residents it strictly prefers to r_i, for otherwise (r_i, h_j) blocks M. Let M' be the matching constructed so far at the beginning of iteration z during E.

We first note that h_j is under-subscribed in M'. For otherwise, h_j could not be in H_i. Hence there exists a resident r_s assigned to h_j in M but not in M'. Since h_j prefers r_s to r_i, resident r_s strictly precedes resident r_i on the master list. Clearly H_s is defined (or else both H_s and H_i are undefined), and it follows by Lemma 4.4.1 that H_s is non-empty. If $|H_s| > 1$ then the algorithm would have returned null before iteration z, a contradiction since H_i is defined. Hence $H_s = \{h_k\}$ for some $h_k \in \mathcal{H}$.

By Case (i), r_s does not prefer h_j to h_k. Also if r_s prefers h_k to h_j it contradicts our choice of r_i. Hence r_s is indifferent between h_j and h_k. If $h_k \neq h_j$, it follows that (r_s, h_k) was deleted, contradicting Lemma 4.4.1 since $(r_s, h_j) \in M$. Therefore $h_k = h_j$. Thus r_s becomes assigned to h_j during E before iteration z. However $(r_s, h_j) \notin M'$, therefore (r_s, h_j) was deleted during E, a contradiction by Lemma 4.4.1.

Case (iv) r_i is unassigned in M. The argument is similar to that used in Case (iii). □

We note that in the case of super-stability if H_i is defined and $H_i \neq \emptyset$ then $H_i = \{M(r_i)\}$ but the lemma above is stated more generally for re-use in the strong stability section to follow. We now prove that no super-stable matching exists for an instance I of HRT-1ML, if for any resident $r_i \in \mathcal{R}$, H_i contains more than one hospital.

Lemma 4.4.4. *If for some resident r_i, H_i is defined and $|H_i| > 1$, then no super-stable matching exists.*

Proof. Suppose for a contradiction that there exists a super-stable matching M and at iteration z of an execution E of HRT-ML-`super` there exists some resident r_i such that $|H_i| > 1$. Let M' be the matching constructed so far at the beginning of iteration z during E. Then r_i must be the first resident on the master list with more than one key hospital, as otherwise the algorithm would have return null prior to iteration z. By Lemma 4.4.3, r_i is assigned in M to a hospital $h_j \in H_i$. Consider hospital $h_k \in H_i$, where $h_k \neq h_j$. If h_k

is either under-subscribed in M or full with at least one assignee no better than r_i in M, then (r_i, h_k) blocks M. Hence h_k must be full with assignees that it strictly prefers to r_i in M. By Lemma 4.4.3 each resident r_s (where r_s is strictly better than r_i on the master list) is assigned to a hospital in H_s, where $|H_s| = 1$ by above observation regarding r_i. However since h_k is under-subscribed in M' (this follows by a similar argument used in Case (iii) of Lemma 4.4.3 when establishing h_j is under-subscribed in M'), there exist less than c_k residents that h_k strictly prefers to r_i that have key set $H_s = \{h_k\}$, for otherwise h_k would be full at iteration z, a contradiction. Hence it is impossible that h_k is full in M with residents it prefers to r_i, a contradiction. □

Finally we show that if, during an execution of HRT-ML-`super`, the algorithm returns null then no super-stable matching exists.

Lemma 4.4.5. *If the algorithm* HRT-ML-`super` *returns null then no super-stable matching exists.*

Proof. Suppose for a contradiction that, during an execution E of HRT-ML-`super` for an instance I of HRT-1ML, the algorithm returns null yet there exists a super-stable matching M. Consider the following two cases.

> *Case (i)*: null is returned when H_i is defined and $|H_i| > 1$ for some resident r_i. By Lemma 4.4.4 no super-stable matching exists.
>
> *Case (ii)*: null is returned when the set of residents P whose key hospital h_j at iteration z of E exceeds the number of posts h_j has available. Let Z_j denote the set of assignees to h_j at the beginning of iteration z. Then by Lemma 4.4.4, for each resident $r_i \in Z_j$, h_j must be r_i's key hospital, and therefore matched to h_j in M. Similarly each resident in P must also obtain h_j in M by Lemma 4.4.3. Therefore in M, h_j has $|Z_j| + |P| > c_j$ hospitals, and hence h_j is over-subscribed in M, a contradiction.

□

We calculate the time complexity of the algorithm by noting that the outer for loop in the worst case may iterate n_1 times when the master list contains no ties. We note that the first inner for loop runs in time $O(n_1)$, although this will not be required for the complexity analysis. For the second inner for loop we note that $|Q| \leq |P|$ as each resident in P has at most one hospital at the head of their list. We represent Q as an array where

each element in the array contains two values. The first value represents the hospital $h_j \in Q$ and the second value the number of residents that have h_j as a key hospital. The position of each hospital in the set Q is held in a rank list which contains all the hospitals and when a hospital h_j is added to Q, h_j's position in the array is stored in the rank list. We note that each element in the rank list can be initialised to 0 before the outer for loop. This allows us to reset the array back to its initial state in $O(n_1)$ overall time after lines 9-12. Therefore in the second inner for loop we can identify the number of residents a that have a hospital $h_j \in Q$ as a key hospital by first finding h_j's position in Q by inspecting the rank list, then by retrieving a from the array. This operation can therefore be achieved in constant time, therefore the overall time taken for the second inner for loop is $O(n_1)$. Finally in the third for loop, we observe that the operation "delete the pair (r_i, h_j)" is executed at most once for each such pair. Therefore the overall time complexity of the algorithm is dominated by this last for loop and as such the overall complexity of the algorithm is $O(\lambda)$, where λ is the total length of the preference lists. We use this result and the lemmas above to obtain the following theorem.

Theorem 4.4.6. *For a given instance of* HRT-1ML *algorithm* HRT-ML-`super` *finds the unique super-stable matching or reports that none exists in time $O(\lambda)$, where λ is the total length of the preference lists.*

4.5 HRT-1ML under strong stability

In this section we present an algorithm HRT-ML-`strong` that determines if a strongly stable matching exists for an instance of HRT-1ML and outputs such a matching if one does. We show that the complexity of HRT-ML-`strong` improves on that of the best-known algorithm for the general HRT case, namely $O(C\lambda)$ [45], where C is the sum of the hospital capacities and λ is the total length of the preference lists.

Algorithm HRT-ML-`strong` is shown in Algorithm 19. The algorithm determines if a strongly stable matching exists by attempting to construct such a matching. We first note that a resident r_i's set of key hospitals H_i is as defined in Section 4.4. Initially the algorithm starts with an empty matching. Then for each tie T on the master list in turn, we identify the set of residents P in T whose lists are non-empty, and the set of hospitals Q, which is the union of the sets H_i, for each $r_i \in P$. A graph G is then built using the algorithm shown in Algorithm 20. The vertex set V of G consists of both the residents in

P and the hospitals in Q, and the edge set E of G is initially set to be empty. We then set the upper bound for each hospital in V to be 0. Next, for each resident $r_i \in P$, we identify each hospital h_j in H_i, and add the edge (r_i, h_j) to E. Additionally we increment the upper bound $u(h_j)$. At this point a hospital's upper bound may be larger than the number of available posts, as more residents may have found the hospital acceptable than there are available posts. Hence for each hospital $h_j \in Q$, we set the upper bound on h_j's vertex to be the minimum of the number of residents that find h_j acceptable (i.e. the current value of $u(h_j)$) and the number of available posts in h_j. We then check if the sum of the upper bounds on each vertex representing a hospital in G equals the number of residents in P. If this is not the case, then we will prove that no strongly stable matching exists. Otherwise we find a maximum degree-constrained subgraph D with respect to the capacity function u of G using Gabow's algorithm [16] (that is for G we find a subgraph D with the greatest possible number of edges incident to each vertex v such that $d_v \leq u(v)$, where d_v is the degree of v in D). If the number of edges in D is less than the number of residents in P, then we have failed to match all the residents in P, and we will prove in this case that no strongly stable matching exists. Finally we add the pairs in D to the matching M. If, as a result of this assignment, a hospital $h_j \in Q$ becomes full, then we identify the worst assigned resident r_i in M. Then for each strict successor r_k of r_i on the master list we "delete the pair (r_k, h_j)", which comprises deleting h_j from r_k's list and vice-versa. The matching M is then returned.

We now define additional notation used in this section. Let E be an execution of HRT-ML-`strong` for an instance I of HRT-1ML and z be some iteration of the main for loop in E. Then we denote the set of residents in P at iteration z by P_z and the set of hospitals in Q at iteration z by Q_z. We also denote the set of residents that have h_j as a key hospital at iteration z by $R_{z,j}$. That is, $R_{z,j} = \{r_i \in P_z : h_j \in H_i\}$. We note that $R_{z,j}$ is well defined as the same set of deletions are made regardless of which degree-constrained subgraph we choose. Now let M' be the matching at the beginning of iteration z. We define the following partition of Q_z:

$$Q_z^1 = \{h_j \in Q_z : u(h_j) = c_j - |M'(h_j)|\},$$

$$Q_z^2 = \{h_j \in Q_z : u(h_j) = |R_{z,j}|\},$$

To prove the correctness of the algorithm, we begin by showing that if the algorithm deletes a pair then that pair does not belong to any strongly stable matching.

Algorithm 19 HRT-ML-`strong`

```
1: M := ∅;
2: for each tie T in the master list in turn do
3:    P := set of residents in T that have a non-empty list;
4:    Q := ⋃_{r_i ∈ P} H_i;
5:
6:    ⟨G, u⟩ := BuildGraph(P, Q, M); /** Algorithm 20 */
7:
8:    if Σ_{h_j ∈ Q} u(h_j) ≠ |P| then
9:       return null;
10:
11:   D := maximum degree-constrained subgraph of G;
12:   E_D := edge set of D;
13:   if |E_D| < |P| then
14:      return null;
15:   M_D := E_D;
16:   M := M ∪ M_D;
17:   for each h_j ∈ Q do
18:      if h_j is full in M then
19:         r_i := h_j's worst assigned resident in M;
20:         for each strict successor r_k of r_i on the master list do
21:            delete the pair (r_k, h_j);
22: return M;
```

Algorithm 20 `BuildGraph`(P, Q, M)

1: $V := P \cup Q$;
2: $E := \emptyset$
3: **for each** $h_j \in Q$ **do**
4: $u(h_j) := 0$;
5:
6: **for each** $r_i \in P$ **do**
7: $u(r_i) :=1$;
8: **for each** $h_j \in H_i$ **do**
9: $E := E \cup \{(r_i, h_j)\}$;
10: $u(h_j) := u(h_j) + 1$;
11: /* $u(h_j) = |R_{z,j}|$ */
12: **for each** $h_j \in Q$ **do**
13: $u(h_j) := \min\{u(h_j), c_j - |M(h_j)|\}$;
14:
15: $G := (V, E)$;
16: **return** $\langle G, u \rangle$;

Lemma 4.5.1. *Algorithm* HRT-ML-`strong` *never deletes a pair* (r_i, h_j) *that belongs to a strongly stable matching.*

Proof. The proof is identical to that of Lemma 4.4.1. □

Lemma 4.5.2. *Let* E *be an execution of* HRT-ML-`strong` *for an instance* I *of* HRT-1ML. *At every iteration* z *of the main loop during* E, *each resident* $r_i \in P_z$ *is assigned in* M_D *and each hospital* $h_j \in Q_z$ *is assigned* $u(h_j)$ *residents in* M_D, *or else the algorithm returns null.*

Proof. Clearly each resident $r_i \in P_z$ is assigned in M_D for otherwise the algorithm would have returned null at line 14. Suppose that the algorithm did not return null at line 9. Then $|P_z| = \sum_{h_k \in Q_z} u(h_k)$. Now suppose that some hospital h_j is assigned fewer than $u(h_j)$ residents in M_D. Let $M_D(h_k)$ denote the set of residents assigned to h_k in M_D, for

$h_k \in \mathcal{H}$. Then $|M_D(h_j)| < u(h_j)$ thus:

$$\begin{aligned}
|M_D| &= \sum_{h_k \in Q_z} |M_D(h_k)| \\
&= \sum_{h_k \in Q_z \setminus \{h_j\}} |M_D(h_k)| + |M_D(h_j)| \\
&\leq \sum_{h_k \in Q_z \setminus \{h_j\}} u(h_k) + |M_D(h_j)| \\
&< \sum_{h_k \in Q_z \setminus \{h_j\}} u(h_k) + u(h_j) \\
&= \sum_{h_k \in Q_z} u(h_k) \\
&= |P_z|.
\end{aligned}$$

Hence the algorithm would have returned null at line 14. □

The following lemma shows that in every strongly stable matching each resident is assigned to a key hospital.

Lemma 4.5.3. *Let r_i be a resident whose set of key hospitals H_i is defined. Let M be any strongly stable matching. If $H_i = \emptyset$ then r_i is unassigned in M, otherwise $(r_i, h_j) \in M$, for some $h_j \in H_i$.*

Proof. Let E be an execution of HRT-ML-`strong`. Now let z be the iteration of E where r_i is considered. If $H_i = \emptyset$, then all hospitals have been deleted from r_i's list and hence by Lemma 4.5.1, it follows that r_i is unassigned in every stable matching. Therefore H_i is non-empty. We consider the following four cases.

Case (i): r_i obtains a hospital h_j in M that he strictly prefers to the hospitals in H_i. The proof of this case is identical to that of Case (i) in Lemma 4.4.3.

Case (ii) r_i obtains a hospital h_j in M such that $h_j \notin H_i$ and r_i is indifferent between h_j and the hospitals in H_i. The proof of this case is identical to Case (ii) of Lemma 4.4.3.

Case (iii): r_i obtains a hospital in M that he finds inferior to the hospitals in H_i. Let r_i be the first resident on the master list with this property. Then in M, each hospital $h_j \in H_i$ must be full with residents strictly better than r_i, for otherwise (r_i, h_j) blocks M.

Let M' denote the matching at the beginning of iteration z. Let $h_j \in H_i$, then h_j is under-subscribed in M', for otherwise h_j would have been deleted from the list of each such r_i. Hence there exists a resident r_k who is assigned to h_j in M but not in M'.

Let y be the iteration of E for which $r_k \in P_y$. Then since $y < z$, it follows that r_k appears strictly before r_i on the master list. Now H_k must be defined, for otherwise the algorithm would have returned null prior to iteration z, a contradiction. If r_k strictly prefers h_j to the hospitals in H_k, we contradict Case (i). Also if r_k strictly prefers the hospitals in H_k to h_j then we contradict the choice of r_i. Hence r_k is indifferent between h_j and the hospitals in H_k. Moreover (r_k, h_j) is not deleted during E by Lemma 4.5.1. Hence $h_j \in H_k$. Let M'' denote the matching at the beginning of iteration y. Then at iteration y, it follows that $u(h_j) = c_j - |M''(h_j)|$, for if $u(h_j) = |R_{y,j}|$, then by Lemma 4.5.2, $R_{y,j} \subseteq M'(h_j)$, contradicting the fact that r_k is not assigned to h_j in M'. It follows that h_j must have become full at iteration y, a contradiction.

Case (iv) r_i is unassigned in M. The argument is similar to that used in Case (iii). □

Lemma 4.5.4. *Let E be an execution of* HRT-ML-`strong` *for an instance I of* HRT-1ML. *If, at some iteration z of the main loop during E, $u(h_j) = |R_{z,j}|$ for some $h_j \in Q_z$, then in every strongly stable matching M, h_j is assigned the set of residents $R_{z,j}$ in M.*

Proof. Suppose that, at iteration z, $u(h_j) = |R_{z,j}|$ and suppose for a contradiction that M is a strongly stable matching such that $r_i \notin M(h_j)$, where $h_j \in Q_z$ and $r_i \in R_{z,j}$. By Lemma 4.5.3, it follows that r_i is assigned in M to a hospital at least as good as h_j. Hence h_j must be full with residents at least as good as r_i in M, for otherwise (r_i, h_j) blocks M. Then at each iteration y prior to z, where h_j is a key hospital of a resident in P_y, assuming such an iteration exists, $u(h_j) = |R_{y,j}|$. For if this was not the case the algorithm would have returned null prior to iteration z (which is clearly did not happen) or h_j would have become full at iteration y. In the latter case all strict successors of h_j's worst assigned resident would have been deleted, and so h_j could not be a key hospital of any resident at iteration z. Since $u(h_j) = |R_{z,j}|$ and $u(h_j) = |R_{y,j}|$ for all iterations $y < z$, $\sum |R_{x,j}| \leq c_j$, for each iteration x ($x \leq z$), where $h_j \in Q_x$. Hence by Lemma 4.5.2, h_j is assigned $u(h_j)$ residents at each such iteration, i.e. h_j is assigned all the residents that have h_j as a key

hospital at this iteration. Thus h_j cannot be full with residents at least as good as r_i in M, a contradiction. □

Lemma 4.5.5. *Let E be an execution of* HRT-ML-`strong` *for an instance I of* HRT-1ML*. Let z be an iteration of the main loop during E and let M' be the matching at the beginning of iteration z. For each $h_j \in Q_z$, if at iteration z, $u(h_j) = c_j - |M'(h_j)|$, then h_j is full in every strongly stable matching M.*

Proof. Suppose, for a contradiction, that $u(h_j) = c_j - |M'(h_j)|$ at iteration z of the main loop during E and that h_j is under-subscribed in M. Then in each iteration y prior to z such that $h_j \in Q_y$, $u(h_j) = |R_{y,j}|$. For otherwise h_j would have become full during E at iteration y and h_j would have been deleted from the list of each resident strictly worse than those in $R_{y,j}$. Hence h_j cannot be in Q_z, a contradiction. If $c_j - |M'(h_j)| = |R_{z,j}|$, by Lemma 4.5.4, h_j is full in M, therefore $c_j - |M'(h_j)| < |R_{z,j}|$. Hence there exists a resident $r_i \in R_{z,j}$ such that $r_i \notin M(h_j)$, for otherwise h_j is over-subscribed in M. Then by Lemma 4.5.3, r_i is assigned in M to a hospital h_k in Q_z, where r_i is indifferent between h_k and h_j. Therefore if h_j is under-subscribed in M, (r_i, h_j) blocks M. □

We now show that if the algorithm HRT-ML-`strong` outputs a matching M then M is strongly stable.

Lemma 4.5.6. *If a matching M is output by* HRT-ML-`strong`*, then M is strongly stable.*

Proof. Suppose, for a contradiction, that the matching M output by the algorithm is not strongly stable. Hence there exists a pair (r_i, h_j) that blocks M. If (r_i, h_j) was deleted during an execution E of HRT-ML-`strong`, then h_j must be full with assignees it prefers to r_i (as by inspection of algorithm, once a hospital becomes full, its set of assignees is fixed and does not subsequently change) contradicting the fact that (r_i, h_j) blocks M. Hence the pair (r_i, h_j) has not been deleted. We note that r_i must be assigned in M, as r_i's list must be non-empty (it contains h_j), thus HRT-ML-`strong` would have returned null if r_i was unassigned. Let r_i be assigned to h_k in M, where $h_j \neq h_k$. Then r_i is indifferent between h_j and h_k. For, if r_i strictly prefers h_j to h_k, (r_i, h_j) has already been deleted as h_k must have been at the head of r_i's list when he became assigned to h_k, contradicting Lemma 4.5.1. Also, if r_i strictly prefers h_k to h_j, (r_i, h_j) does not block M. Let z denote the iteration of the main loop during E in which the tie that contains r_i is processed. Then $h_j, h_k \in Q_z$. Let M' be the matching at the beginning of iteration z. Clearly $u(h_j)$ cannot

equal the number of residents in P_z that have h_j as a key hospital (i.e $u(h_j) \neq |R_{z,j}|$), as by Lemma 4.5.4, h_j would be assigned to r_i in M. It follows that $c_j - |M'(h_j)| < |R_{z,j}|$ and so $u(h_j)$ equals the number of unfilled posts in h_j at iteration z. By Lemma 4.5.2, h_j must obtain $u(h_j)$ residents from P_z in M. Therefore h_j must be assigned a set of residents at least as good as r_i in M, hence (r_i, h_j) does not block M. □

Lemma 4.5.7. *Let E be an execution of* HRT-ML-`strong` *for an instance I of* HRT-1ML. *If at an iteration z of the main loop during E, $\sum_{h_j \in Q_z} u(h_j) \neq |P_z|$, then no strongly stable matching exists.*

Proof. Let M' be the matching at the beginning of iteration z during E. Let M be a strongly stable matching and let M'' be those pairs of M involving residents in P_z. Then by Lemma 4.5.3, $|M''| = |P_z|$. Now by Lemma 4.5.5 and Lemma 4.5.4, and the fact that h_j could not have become full before iteration z, $\sum_{h_j \in Q_z^1} c_j = \sum_{h_j \in Q_z^1} |M(h_j)| = \sum_{h_j \in Q_z^1} |M'(h_j)| + \sum_{h_j \in Q_z^1} |M''(h_j)|$, i.e.

$$\sum_{h_j \in Q_z^1} |M''(h_j)| = \sum_{h_j \in Q_z^1} (c_j - |M'(h_j)|). \tag{4.1}$$

Also

$$\begin{aligned}
|M''| &= \sum_{h_j \in Q_z} |M''(h_j)| \\
&= \sum_{h_j \in Q_z^1} |M''(h_j)| + \sum_{h_j \in Q_z^2} |M''(h_j)| \\
&= \sum_{h_j \in Q_z^1} |M''(h_j)| + \sum_{h_j \in Q_z^2} |R_{z,j}| \text{ , by Lemma 4.5.3 and Lemma 4.5.4} \\
&= \sum_{h_j \in Q_z^1} (c_j - |M'(h_j)|) + \sum_{h_j \in Q_z^2} |R_{z,j}| \text{ , by Equation 4.1} \\
&= \sum_{h_j \in Q_z} |u(h_j)|, \\
&\neq |P_z| \text{ by assumption, a contradiction.}
\end{aligned}$$

□

Lemma 4.5.8. *Let E be an execution of* HRT-ML-`strong` *for an instance I of* HRT-1ML. *If during an iteration z of the main loop during E, the number of residents assigned in E_D is less than the number of residents in P_z, then no strongly stable matching exists.*

Proof. Suppose, for a contradiction, that there exists a strongly stable matching M and that at iteration z of E, the number of residents assigned in E_D is less than the number

of residents in P_z. Let M' denote the matching at the beginning of iteration z. Hence there exists a resident $r_i \in P_z$ who is unassigned in E_D but by Lemma 4.5.3 is assigned to a hospital $h_j \in Q_z$ in M. Therefore h_j is represented by a vertex in G. Now since r_i is unassigned in E_D and D is a maximum degree-constrained subgraph of G, $u(h_j)$ does not equal the number of residents that have h_j as a key hospital at iteration z, i.e. $u(h_j) \neq |R_{z,j}|$. For, if $u(h_j)$ equals $|R_{z,j}|$, we can increase the number of edges in D by adding the edge (r_i, h_j), contradicting the maximality of D. Therefore $u(h_j) = c_j - |M'(h_j)| < |R_{z,j}|$.

Thus in each iteration y prior to z, $u(h_j) = |R_{y,j}|$, therefore the same residents assigned to h_j in M' are assigned to h_j in M, by Lemma 4.5.4. Also, h_j must be assigned $(c_j - |M'(h_j)|)$ residents in E_D at iteration z, for otherwise we contradict the maximality of D by adding the edge (r_i, h_j) to D. However as $u(h_j) = c_j - |M'(h_j)|$, by Lemma 4.5.5, h_j is full in M. Hence there exists a resident r_s assigned to h_j in E_D but $r_s \notin M(h_j)$, for otherwise h_j is over-subscribed in M. Therefore r_s must have become assigned to h_j at iteration z (if he had become assigned to h_j prior to iteration z, say at iteration x, then at this iteration $u(h_j) = |R_{x,j}|$, thus by Lemma 4.5.4 r_s is assigned to h_j in every strongly stable matching, a contradiction). Hence h_j is indifferent between r_i and r_s.

By Lemma 4.5.3, r_s is assigned to a hospital $h_t \in Q_z$ in M $(h_t \neq h_j)$. However $u(h_t)$ cannot equal $|R_{z,t}|$, for otherwise we can augment along the path $(r_i, h_j), (h_j, r_s), (r_s, h_t)$, increasing the number of edges in D, contradicting the maximality of D. Therefore $u(h_t)$ equals the number of unfilled posts in h_t at iteration z. Hence using a similar argument to that for h_j, there exists a resident r_x assigned to h_t in E_D but $r_x \notin M(h_t)$, for otherwise h_t is over-subscribed in M. Therefore r_x must have become assigned to h_t at iteration z and so h_t is indifferent between r_s and r_x. By Lemma 4.5.3, r_x is assigned to a hospital $h_u \in Q_z$ in M. Clearly continuing in this manner we either obtain an augmenting path or we increase the length of the path and since the number of residents is finite and each such resident is distinct, we must reach a stage where no such resident exists, a contradiction. □

We use Lemma 4.5.8 and Lemma 4.5.7 to obtain the following result.

Lemma 4.5.9. *If the algorithm returns null then no strongly stable matching exists.*

The complexity of the algorithm is dominated by finding a maximum degree-constrained subgraph of the graph G. All other operations take $O(\lambda)$ time over the algorithm's entire execution, where λ is the total length of the preference lists. We can obtain a maximum degree-constrained subgraph in time $O(|E|\sqrt{S})$ using Gabow's algorithm [16], where E is

the set of edges in G and S is the sum of the upper degree-constraints on each vertex. Since the graphs at any two iterations of the main loop of the algorithm involve disjoint sets of edges, HRT-ML-`strong` has time complexity $O(\lambda\sqrt{C})$, where C is the sum of the hospitals' capacities. Finally we bring together the result above and the results given in Lemma 4.5.6 and Lemma 4.5.9 to obtain the following theorem.

Theorem 4.5.10. *For a given instance of* HRT-1ML*, algorithm* HRT-ML-`strong` *finds a strongly stable matching or reports that none exists in time* $O(\lambda\sqrt{C})$*, where* λ *is the total length of the preference lists and* C *is the sum of the hospitals' capacities.*

4.6 Open Problems

We finish this chapter by giving some open problems.

4.6.1 Finding an egalitarian strongly stable matching

An *egalitarian stable matching* M in an instance of HR is a stable matching in which the sum of the ranks of the agents' assignees in M is minimised. It is known that the problem of finding an egalitarian stable matching, given an instance of HR, is polynomial-time solvable [9]. It was also shown by Scott [67] that the problem of finding an egalitarian weakly stable matching is NP-hard, given an instance of HRT-ML (in fact Scott showed this is true for SMT with a master list on one side that contains no ties). We conjecture that the problem of finding an egalitarian strongly stable matching is polynomial-time solvable for the general case of HRT but so far no known proof exists. We also suggest that finding an algorithm for the case of HRT-ML should be easier, and in turn this should allow us to disprove Feder's conjecture [13, p.148], which states that it is NP-hard to find a matching other than the man-optimal and woman-optimal strongly stable matching, should these two matchings exist.

4.6.2 Finding a minimum regret strongly stable matching

A minimum regret stable matching M is a stable matching in which we minimise the maximum rank of an agent's assignee in M. It is known that the problem of finding a minimum regret stable matching, given an instance of SM, is solvable in $O(\lambda)$ [26, Section 4.4.3], where λ is the total length of the preference lists. However Scott showed that the problem of finding a minimum regret weakly stable matching is NP-hard given an instance

of SMT-ML. We conjecture that the problem of finding a minimum regret strongly stable matching is polynomial-time solvable for the case of HRT, but so far no known proof exists. As in the case of Section 4.6.1, finding a minimum regret strongly stable matching may be easier to derive in the case of HRT-ML and would also allow us to disprove Feder's conjecture.

Chapter 5

The Stable Roommates Problem with Master Lists

5.1 Introduction

This chapter concerns the Stable Roommates problem (SR). As mentioned in Chapter 1, it is known that a stable matching need not exist for an instance of SRI [18]. However, it is known that the problem of finding a stable matching or reporting that none exists, given an instance I of SRI can be solved in time $O(\lambda)$ using Irving's algorithm [34], where λ is the total length of the preference lists in I. Additionally deciding whether a weakly stable matching exists for an instance of SRT is known to be NP-complete [38, 62]. By contrast there is an $O(\lambda)$ algorithm that finds a super-stable matching or reports that none exists, given an instance I of SRTI [38]. An $O(\lambda^2)$ algorithm performs the corresponding task in the case of strong stability [67]. We also note that weakly stable matchings may be of different sizes, whereas, in contrast all super-stable matchings have the same size and all strongly stable matchings have the same size [38, 67].

In this chapter we consider instances of SR with and without ties, where the preference lists are derived from a single master list. We obtain a range of algorithmic results for these problem variants under different stability criteria. In cases where polynomial-time algorithms already exist in the general case, we give algorithms that simplify, and in certain cases, speed up the more general algorithm. We also show that one NP-hard problem in the general case becomes polynomial time solvable in the presence of a master list.

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ denote the set of agents for an instance of SRI. Then we denote an instance of SRI in which each agent's list is derived from a single master list by SRI-ML.

Here the concept of a master list is similar to that described in Chapter 4. That is, each agent p_i's preference list is derived from the master list by deleting p_i and additionally all those agents that p_i finds unacceptable. It is assumed that p_i finds p_j unacceptable if and only if p_j finds p_i unacceptable. We also consider the extension SRTI of SRI where ties are allowed in the preference lists of the agents. In this case we denote an instance of SRTI in which each agent's list is derived from a master list by SRTI-ML. Figure 5.1 shows an instance of SRTI-ML.

Master list: $p_1 \quad p_2 \quad (p_3 \quad p_4)$

Agents' individual preference lists

$p_1 : \quad p_1 \quad p_2 \quad p_4$

$p_2 : \quad (p_3 \quad p_4)$

$p_3 : \quad p_1 \quad p_3$

$p_4 : \quad p_2 \quad (p_3 \quad p_4)$

Figure 5.1: An instance of SRTI-ML.

The main results and organisation of this chapter are: in Section 5.2 we provide a simple algorithm, requiring only one phase, which finds the unique stable matching for an instance of SRI-ML. We prove in Section 5.3 that despite the existence of a master list, the problem of finding a maximum weakly stable matching for an instance of SRTI-ML is NP-hard. However, we also show in Section 5.3 that, in contrast to the general case, we can find a weakly stable matching in polynomial time. In Section 5.4, we describe a $O(\lambda)$ algorithm that finds a super-stable matching, or reports that none exists, for an instance I of SRTI-ML, where λ is the total length of the preference lists. This algorithm simplifies its counterpart for the general case [38]. Furthermore we show that if a matching is returned by the algorithm then this is in fact the unique super-stable matching in I. Finally in Section 5.5 we describe an algorithm that finds a strongly stable matching, or reports that none exists, for an instance of SRTI-ML. The algorithm improves the time complexity of the general case, and runs in time $O(\sqrt{n}\lambda)$, where λ is the total length of the preference lists, and n is the number of agents.

5.2 SRI-ML

In this section we describe an algorithm SRI-ML-`alg` that can be used to find a stable matching for an instance I of SRI-ML. The algorithm also indicates that I is bound to admit at least one stable matching, which need not be true in general, given an instance of SR (and therefore of SRI), as already observed. The worst-case running time of the algorithm is identical to that of the algorithm due to Irving [34]. However the algorithm is straightforward and does not require the two-phase approach used in the general case. We show that the matching output by the algorithm is in fact the unique stable matching in I.

Algorithm SR-ML-`alg` is shown in Algorithm 21. Initially a matching M is set to be empty. Then for each agent p_i on the master list in turn, we check to see if p_i is unassigned and p_i's list contains at least one unassigned agent. If this is the case then we identify the first unassigned agent p_j on p_i's list, and add the pair $\{p_i, p_j\}$ to M.

Algorithm 21 SRI-ML-`alg`

```
1: M := ∅;
2: for each agent p_i on the master list in turn do
3:    if p_i is unassigned and p_i's list contains an unassigned agent then
4:       p_j := first unassigned agent on p_i's list;
5:       M := M ∪ {{p_i, p_j}};
6: return M;
```

Let z denote an iteration during an execution E of SRI-ML-`alg` for an instance I of SRI-ML. Then we say that an agent p_i is *processed* if p_i is the agent being considered in the for loop at line 2 during iteration z. We show in the lemma below that the assignment returned by the algorithm is a stable matching.

Lemma 5.2.1. *The matching M returned by algorithm* SRI-ML-`alg` *is a stable matching.*

Proof. Let E be the execution of SR-ML-`alg` for an instance I of SRI-ML. Suppose, for a contradiction, that the matching M returned at the end of E is not stable. Hence there exists a pair $\{p_i, p_j\}$ that blocks M. At iteration z of the for loop during E where p_i is processed (there can only be one such iteration), either (i) p_i is already assigned or (ii) p_i is unassigned.

Case (i): Suppose p_i is assigned to p_k at the beginning of iteration z. Then p_k became assigned to p_i at some iteration $y < z$, and so p_k precedes p_i in the master

list. Also p_j precedes p_k on the master list since $\{p_i, p_j\}$ blocks M. Hence at iteration $x < y$ when p_j was processed, either (a) p_j was already assigned to some agent p_l, or (b) p_j was unassigned. In Case (a), p_l precedes p_j on the master list and hence $\{p_i, p_j\}$ does not block M, a contradiction. In Case (b), p_i could not have been the first unassigned agent on p_j's list, for otherwise $\{p_j, p_i\}$ would have been added to M. Therefore in M, p_j must obtain an agent he prefers to p_i, and so $\{p_i, p_j\}$ does not block M, a contradiction.

Case (ii): Then p_j must already be assigned at iteration z, for otherwise the algorithm would have assigned p_i to p_j in M. In particular p_j must have become assigned, to p_k say, in an iteration prior to z. Since agents are processed in preference order, either p_j became assigned when p_k was processed or when p_j itself was processed. In either case p_j becomes assigned to p_k, and p_k must appear before p_i on the master list. Hence $\{p_i, p_j\}$ cannot block M.

□

Lemma 5.2.2. *The matching M returned by algorithm* SRI-ML-`alg` *for an instance I of* SRI-ML *is the unique stable matching in I.*

Proof. Let E be an execution of algorithm SRI-ML-`alg` for I, producing a matching M. Suppose, for a contradiction, that there exists a matching M' that is stable in I, where $M \neq M'$. Hence there exists a pair $\{p_i, p_j\} \in M$ but $\{p_i, p_j\} \notin M'$. Without loss of generality suppose that p_i is the first agent on the master list with this property. Then since $\{p_i, p_j\} \in M$, when p_i became assigned to p_j during E, p_j was the first unassigned agent on p_i's list. Since each agent better than p_i on the master list obtains the same assignee in M as in M', p_i must obtain an agent worse than p_j in M' or be unassigned in M'. However by the stability of M', p_j must obtain an agent better than p_i in M', which is impossible as again each resident better than p_i on the master list obtains the same assignee in M as in M'. □

The complexity analysis of algorithm SRI-ML-`alg` is straightforward and can be shown to be $O(\lambda)$, where λ is the total length of the preference lists. We now use Lemma 5.2.1 and Lemma 5.2.2 to obtain the following theorem.

Theorem 5.2.3. *For a given instance I of* SRI-ML*, algorithm* SRI-ML-`alg` *outputs the unique stable matching for I in time $O(\lambda)$, where λ is the total length of the preference*

lists.

5.3 SRTI-ML under weak stability

Ronn [62] showed that the problem of deciding whether a weakly stable matching exists, given an instance of SRT, is NP-complete. In contrast, we can always find a weakly stable matching for an instance I of SRTI-ML by simply breaking the ties in the master list arbitrarily and running the algorithm given in Section 5.2. However, weakly stable matchings in I may be of different sizes. For example consider instance I in Figure 5.2. Here two possible weakly stable matching are $M = \{\{p_1, p_4\}, \{p_2, p_3\}\}$ and $M' = \{\{p_1, p_3\}\}$. We now prove that MAX-SRTI-ML, the problem of finding a maximum weakly stable matching given an instance of SRTI-ML, is in fact NP-hard. We firstly define the decision version of this problem.

Name :	MAX-SRTI-ML-D
Instance:	An SRTI instance I in which each agent's list is derived from a master list of agents and a positive integer K.
Question:	Does I have a weakly stable matching of size $\geq K$?

Master list: p_1 p_2 $(p_3$ $p_4)$

Agents' individual preference lists

p_1 : $(p_3$ $p_4)$

p_2 : p_3

p_3 : p_1 p_2

p_4 : p_1

Figure 5.2: An instance I of SRTI-ML where weakly stable matchings can have different sizes.

Theorem 5.3.1. MAX-SRTI-ML-D *is NP-complete, even if the agents' individual preference lists are of length at most 4, where each agent's individual list contains at most one tie of length 3.*

Proof. Clearly MAX-SRTI-ML-D is in NP. To prove that MAX-SRTI-ML-D is NP-hard we reduce from MIN-MM-D restricted to cubic graphs, which as noted in Section 2.2.2 of

Chapter 2 is NP-complete. Let G, a cubic graph and K, a positive integer, be an instance of MIN-MM-D. Furthermore let $P = \{p_1, p_2, \ldots, p_n\}$ denote the vertices of G and let E denote the edge set of G. We denote the set of vertices adjacent to a vertex p_i in G by P_i.

We construct an instance I of MAX-SRTI-ML as follows: let $P \cup X$ be the set of agents, where $X = \{x_1, x_2, \ldots, x_n\}$. The preference lists and master list of the agents in I are shown in Figure 5.3. In the master list, agents who appear within square brackets are listed in arbitrary strict order. Also those agents who appear within round brackets are tied with each other. Let $K' = n - K$. We claim that G has a maximal matching of size $\leq K$ if and only if I admits a weakly stable matching of size $\geq K'$.

Master list: (agents in P) [agents in X]
Agents' individual preference lists
p_i : (P_i) x_i $(1 \leq i \leq n)$
x_i : p_i $(1 \leq i \leq n)$

Figure 5.3: Preference lists for the constructed instance of MAX-SRTI-ML-D.

Suppose G has a maximal matching M, where $|M| = t \leq K$. We construct a matching M' in I as follows. Initially let $M' = M$. There remain $n - 2t$ agents in P that are unassigned in M': denote these agents by p_{k_i} $(1 \leq i \leq n - 2t)$, and add $\{p_{k_i}, x_{k_i}\}$ to M'. Then $|M'| = t + (n - 2t) = n - t \geq n - K = K'$.

Now suppose for a contradiction that M' is not weakly stable. Hence there exists a pair that blocks M'. We first note that, by construction of the matching, no agent $p_i \in P$ can be unassigned in M' as either p_i is assigned to an agent $p_j \in P$ or an agent $x_i \in X$. Hence any pair that blocks M' must have the form $\{p_i, p_j\}$, where $p_i, p_j \in P$. Now as p_i and p_j cannot be unassigned in M', it follows that p_i is assigned to x_i in M' and p_j is assigned to x_j in M'. Therefore $\{p_i, p_j\} \notin M$. Moreover $\{p_i, p_j\}$ belongs to E, so that $\{p_i, p_j\}$ is a matching in M, contradicting the maximality of M. Hence M' is stable in I.

Conversely suppose that M' is a weakly stable matching in I and that $|M'| \geq K'$. Let $M = M' \cap E$ and $t = |M|$. Then in M', there are exactly $2t$ agents in P who are assigned to agents also in P, and at most $n - 2t$ agents in X who are matched to agents in P. Therefore $|M'| \leq t + (n - 2t) = n - t$ and since $|M'| \geq K'$, we have that $n - t \geq K'$, and so $|M| \leq n - K' = K$.

Finally suppose that the matching M is not maximal in G. Hence there exists an edge $\{p_i, p_j\}$ in G, such that no edge in M is incident to p_i or p_j. Therefore p_i must be

unassigned or assigned to x_i in M' and p_j must be unassigned or assigned to x_j in M'. Hence $\{p_i, p_j\}$ blocks M' in I, contradicting the weak stability of M'. Therefore M is indeed maximal in G. □

5.4 SRTI-ML under super-stability

In this section we present an algorithm SRTI-ML-`super` which finds a super-stable matching or reports that none exists, given an instance of SRTI-ML. The running time of the algorithm is $O(\lambda)$, which comparable to that of the algorithm by Irving and Manlove [38], where λ is the total length of the preference lists. However, our algorithm is much simpler and in particular does not require two distinct phases. Additionally, we prove that, given an instance I of SRTI-ML, if a matching M is returned by the algorithm, then M is the unique super-stable matching for I.

Algorithm SRTI-ML-`super` is shown in Algorithm 22. Before explaining the algorithm's operation we first introduce some notation similar to that used in Chapter 4. Let E be the execution of SRTI-ML-`super` and let z be an iteration of the main loop of E. Then we denote the set of agents in U, S and T (as defined in Algorithm 22) at iteration z by U_z, S_z and T_z respectively. Additionally, if $p_i \in S_z \cup T_z$ we say that p_i is *considered* at iteration z and if $p_i \in U_z$ we say that p_i is *processed* at iteration z. Let p_i be an agent who is considered at iteration z. Then we denote the set of agents at the head of p_i's list by P_i, and refer to them as p_i's *key agents*. If the algorithm returns null before iteration z, or if p_i is not considered during E, then P_i is undefined. We will prove that P_i is *well-defined* for each agent $p_i \in \mathcal{P}$, by which we mean that if P_i is defined at some iteration z of the main loop during E, then z is the unique iteration of the main loop in which p_i is considered.

The algorithm SRTI-ML-`super` proceeds as follows: we create a matching M which is initially empty. Then for each tie U on the master list in preference order, we identify the set of agents S in U whose lists are non-empty and who are currently unassigned. Then for each agent $p_i \in S$, if P_i contains more than one agent, the algorithm returns null and (as we will show) no super-stable matching exists. Next we identify the set of agents T consisting of the union of P_i for each agent in S. Then for each agent $p_i \in T$, we ensure that exactly one agent in S has p_i as a key agent. If this is not the case, the algorithm returns null and again (as we will show) no super-stable matching exists. Now for each agent $p_i \in S$ we assign p_i to the single agent p_j at the head of p_i's list by adding $\{p_i, p_j\}$

to M – in the algorithm $head(p_i)$ is used to denote the single agent in P_i. Finally for each successor p_k of p_j on p_i's list we "delete the pair $\{p_i, p_k\}$", and also for each successor p_l of p_i on p_j's list we delete the pair $\{p_j, p_l\}$ (this in effect deletes p_i and p_j from all other lists); here to "delete the pair $\{p_i, p_j\}$" means deleting p_i from p_j's list and p_j from p_i's list.

Algorithm 22 SRTI-ML-`super`

```
1: M := ∅;
2: for each tie U in the master list in preference order do
3:    S := set of agents in U whose lists are non-empty and who are currently unassigned;
4:
5:    for each p_i ∈ S do
6:       /** P_i denotes the head of p_i's list at this iteration **/
7:       if |P_i| > 1 then
8:          return null;
9:
10:   T := ⋃_{p_i ∈ S} P_i;
11:   for each p_i ∈ T do
12:      A_i := {p_j : p_i ∈ P_j};
13:      if |A_i| > 1 then
14:         return null;
15:
16:   for each p_i ∈ S do
17:      p_j := head(p_i);
18:      M := M ∪ {{p_i, p_j}};
19:      for each strict successor p_k of p_j on p_i's list do
20:         delete the pair {p_i, p_k};
21:      for each strict successor p_k of p_i on p_j's list do
22:         delete the pair {p_j, p_k};
23:
24: return M;
```

The following sequence of lemmas establish the correctness of the algorithm, starting with the following result.

Lemma 5.4.1. *Let E be the execution of* SRTI-ML-`super` *for an instance I of* SRTI-ML *during which the algorithm does not return null, and let z be an iteration of the main loop during E. If at iteration z, $p_i \in S_z \cup T_z$, then p_i is assigned at the end of iteration z.*

Proof. Clearly each agent in S_z is assigned at the end of iteration z. Now suppose $p_i \in T_z$. Then p_i is a key agent of exactly one agent in S_z, for otherwise the algorithm would have returned null at line 14. Furthermore each agent in S_z has exactly one key agent, or else the algorithm would have returned null at line 8. Hence p_i must become assigned during E. □

We now show that if p_i is considered at iteration z, and P_i is the head of p_i's list at this iteration, then each agent p_j in P_i is unassigned.

Lemma 5.4.2. *Let E be the execution of* SRTI-ML-`super` *for an instance I of* SRTI-ML *and let $p_i \in S_z \cup T_z$ at some iteration z during E. Let P_i denote the head of p_i's list at iteration z and let $p_j \in P_i$. Then p_j is unassigned at the beginning of iteration z.*

Proof. Suppose $p_i \in S_z \cup T_z$ and that $p_j \in P_i$. Now suppose for a contradiction that p_j is already assigned at the beginning of iteration z. Then p_j must have become assigned to an agent p_k at an iteration y prior to z. Consider the following two cases.

> *Case (i): $p_i \in S_z$.* If $p_i \in S_z$, then since the pair $\{p_i, p_j\}$ has not been deleted prior to iteration z and since p_j became assigned to p_k at iteration y, either p_i is strictly better than p_k on the master list, or p_i and p_k appear in the same tie on the master list. Hence as $p_i \in S_z$, it follows that $p_k \notin S_y$, by construction of the master list. Thus when p_j became assigned to p_k at iteration y, it follows that $p_j \in S_y$. Therefore p_k must have been at the head of p_j's list when p_j and p_k became assigned at iteration y. Thus if p_i is strictly better than p_k on the master list, the pair $\{p_i, p_j\}$ has already been deleted prior to iteration z, a contradiction. Hence p_i and p_k must appear in the same tie on the master list. Now since $p_i \in S_z$, it follows that p_i is unassigned at the beginning of iteration z. Furthermore, p_i must also be unassigned at the end of iteration y. However as $\{p_i, p_j\}$ has not been deleted, and since $p_k \in P_j$ (where P_j is the head of p_j's list at iteration y), it follows that $p_i \in P_j$ and thus $p_i \in T_y$. Now by Lemma 5.4.1, it follows that p_i becomes assigned during iteration y, a contradiction to the fact that p_i is unassigned at the end of this iteration.
>
> *Case (ii): $p_i \in T_z \setminus S_z$.* If $p_i \in T_z \setminus S_z$, then there exists at least one agent $p_l \in S_z$ such that $p_i \in P_l$ (where P_l is the head of p_l's list at iteration z). Therefore by Case (i), as $p_l \in S_z$ and $p_i \in P_l$, we have that p_i is unassigned at the beginning of iteration z. Also, by Lemma 5.4.1, it follows that the master list tie that contains

p_i has not been processed prior to iteration z. Since the pair $\{p_i, p_j\}$ has not been deleted, either p_i is strictly better than p_k on the master list, or p_i and p_k appear as a tie on the master list. Thus the master list tie that contains p_k has also not been processed prior to iteration z. Therefore when p_j became assigned to p_k at iteration y, it follows that $p_j \in S_y$. Also, since p_k became assigned to p_j at iteration y, we have that p_k must have been at the head of p_j's list. Hence if p_i is strictly better than p_k on the master list, the pair $\{p_i, p_j\}$ has already been deleted prior to iteration z, a contradiction. Thus both p_i and p_k appear in the same tie on the master list, and so $p_i \in P_j$ (where P_j is the head of p_j's list at iteration y). Now using Lemma 5.4.1, we have that p_i becomes assigned during iteration y, contradicting the fact that p_i is unassigned at the beginning of iteration z.

□

We can now use Lemma 5.4.1 and Lemma 5.4.2 to obtain the following lemma. In this lemma we show that if P_i is p_i's set of key agents, then P_i is well defined in the sense described at the top of page 110.

Lemma 5.4.3. *For each agent $p_i \in \mathcal{P}$, if p_i's set of key agents P_i is defined then P_i is well-defined.*

Proof. Let E be the execution of SRTI-ML-`super` for an instance I of SRTI-ML and let z be an iteration of the main loop during E. Then by construction of S_z, each agent in S_z is unassigned, and by Lemma 5.4.2, each agent in T_z is also unassigned. Therefore at each iteration of the main loop we only consider agents that are unassigned. Furthermore, if the algorithm does not return null during iteration z, by Lemma 5.4.1, each agent $p_i \in S_z \cup T_z$ becomes assigned during iteration z (clearly if the algorithm does return null then p_i is never considered at any subsequent iteration). Hence each agent is considered only once during E, therefore P_i is well-defined. □

In the following proof we show that if $p_i \in T_z$, for some iteration z during the execution E of SRTI-ML-`super`, and an agent $p_j \in S_z$ has p_i as a key agent, then p_j is a key agent of p_i.

Lemma 5.4.4. *Let E be the execution of* SRTI-ML-`super` *for an instance I of* SRTI-ML. *Then at each iteration z during E, if $p_i \in T_z$ and $p_i \in P_j$, where $p_j \in S_z$, then $p_j \in P_i$.*

Proof. Suppose, for a contradiction, that $p_i \in T_z$ and $p_i \in P_j$, where $p_j \in S_z$, and that $p_j \notin P_i$. Then either (i) p_i strictly prefers p_j to the agents in P_i or (ii) p_i strictly prefers the agents in P_i to p_j.

Case (i): In this case the pair $\{p_i, p_j\}$ has been deleted, but $p_i \in P_j$, a contradiction.

Case (ii): Let $p_k \in P_i$. Then since $p_j \in S_z$, and p_k appears strictly before p_j on the master list, it follows that p_k was considered at an iteration y prior to z. Hence by Lemma 5.4.1, p_k became assigned to an agent p_l at iteration y. By Lemma 5.4.2, p_i is unassigned at the beginning of iteration z, and so p_i is unassigned at the end of iteration y. Now by Lemma 5.4.1, it follows that $p_i \notin S_x \cup T_x$ for each iteration $x \leq y$. Now consider the following two subcases:

Subcase (a): $p_k \in S_y$. Then p_l must have been at the head of p_k's list at iteration y, as p_k and p_l became assigned at this iteration. Additionally as p_i is unassigned at the beginning of iteration z and so $p_i \notin P_k$ (as p_i would otherwise have become assigned at iteration y contradicting by Lemma 5.4.2), it follows that p_l appears strictly before p_i on the master list. Hence when p_k became assigned to p_l at iteration y the pair $\{p_i, p_k\}$ would have been deleted, contradicting the fact that $p_k \in P_i$.

Subcase (b): $p_k \in T_y \setminus S_y$. Since p_l and p_k become assigned at iteration y, it follows that $p_l \in S_y$. However since the pair $\{p_i, p_k\}$ has not been deleted (as $p_k \in P_i$), either p_i and p_l appear in the same tie on the master list or p_i appears strictly before p_l on the master list. Hence as $p_l \in S_y$, it follows that p_i must have been considered at an iteration $x \leq y$, a contradiction.

□

Using the above lemma we obtain the following result.

Lemma 5.4.5. *Let E be the execution of* SRTI-ML-`super` *for an instance I of* SRTI-ML. *If at some iteration z during E, p_i becomes assigned to p_j, then $p_j \in P_i$.*

Proof. At iteration z either (i) $p_i \in S_z$, or (ii) $p_i \in T_z \setminus S_z$.

Case (i): Since p_i and p_j become assigned to one another at line 18 (and hence the algorithm did not return null at iteration z) it follows that p_j is the single agent at the head of p_i's list. Therefore $p_j \in P_i$.

Case (ii): Since p_i and p_j become assigned at line 18 (and hence the algorithm did not return null at iteration z), it follows that $p_j \in S_z$ and that $p_i \in P_j$. Thus by Lemma 5.4.4, we have that $p_j \in P_i$.

□

We now show that if $p_i \in T_z$, for some iteration z during an execution E of SRTI-ML-`super`, then all p_i's key agents are contained in S_z.

Lemma 5.4.6. *Let E be an execution of* SRTI-ML-`super` *for an instance I of* SRTI-ML. *Then at each iteration z during E, if $p_i \in T_z$ then $P_i \subseteq S_z$.*

Proof. Suppose, for a contradiction, that $p_i \in T_z$ and $P_i \not\subseteq S_z$. Hence there exists an agent $p_j \in P_i$ such that $p_j \notin S_z$. Then since $p_i \in T_z$, there exists at least one agent $p_k \in S_z$ such that $p_i \in P_k$ by definition of T_z. Thus by Lemma 5.4.4, we have that $p_k \in P_i$. As such, p_k and p_j must appear as a tie in the master list. However since $p_j \notin S_z$, either p_j's list is empty at the beginning of iteration z or p_j became assigned at an iteration y prior to iteration z. We first note that p_j's list cannot be empty at the beginning of iteration z, as it contains (at least) p_i. Therefore p_j must have become assigned to an agent p_l at iteration y. However since $p_j \in P_i$, by Lemma 5.4.2, p_j is unassigned at iteration z, a contradiction. □

In the following lemma we prove that no pair deleted by the algorithm belongs to a super-stable matching.

Lemma 5.4.7. *Algorithm* SRTI-ML-`super` *never deletes a pair $\{p_i, p_j\}$ that belongs to a super-stable matching.*

Proof. Let M be a super-stable matching for a given instance I of SRTI-ML. Suppose for a contradiction that $\{p_i, p_j\} \in M$ and that $\{p_i, p_j\}$ is deleted at iteration z during the execution E of SRTI-ML-`super`. Now suppose that this is the first super-stable pair to be deleted. Then without loss of generality that $\{p_i, p_j\}$ was deleted when p_i became assigned to p_k at iteration z. Hence p_i strictly prefers p_k to p_j. Therefore in M, p_k must obtain an agent p_l such that p_k strictly prefers p_l to p_i, for otherwise $\{p_i, p_k\}$ blocks M. Then either (i) $p_k \in S_z$, or (ii) $p_k \in T_z \setminus S_z$.

Case (i): Then since p_i and p_k become assigned at iteration z, p_i must have been at the head of p_k's list. Hence the pair $\{p_k, p_l\}$ has already been deleted, contradicting the fact that $\{p_i, p_j\}$ is the first super-stable pair to be deleted.

Case (ii): If $p_k \in T_z \setminus S_z$, then as p_i and p_k became assigned at iteration z, we have that $p_i \in S_z$, and so $p_k \in P_i$. Hence by Lemma 5.4.4, it follows that $p_i \in P_k$. Therefore since p_l appears before p_i on the master list, the pair $\{p_k, p_l\}$ has already been deleted, contradicting the fact that $\{p_i, p_j\}$ is the first super-stable pair deleted. □

We now show that in every super-stable matching, an agent is assigned to a key agent.

Lemma 5.4.8. *Let p_i be an agent whose set of key agents P_i is defined. Let M be a super-stable matching. Then $\{p_i, p_j\} \in M$, for some $p_j \in P_i$.*

Proof. Let E be the execution of SRTI-ML-`super` and let z be the iteration of E where the agent p_i is considered. Then since P_i is defined, it follows that $P_i \neq \emptyset$. We consider the following three cases.

Case (i): p_i obtains an agent p_k in M strictly better than those in P_i. Then at iteration z, the pair $\{p_i, p_k\}$ has already been deleted. Hence by Lemma 5.4.7, the pair $\{p_i, p_k\}$ does not belong to a super-stable matching, a contradiction.

Case (ii): p_i obtains an agent p_k in M and p_k is indifferent between p_k and those agents in P_i. Then since $p_k \notin P_i$, the pair $\{p_i, p_k\}$ must have been deleted prior to iteration z. Thus by Lemma 5.4.7, it follows that $\{p_i, p_k\}$ does not belong to any stable matching, a contradiction.

Case (iii): p_i obtains an agent in M inferior to those in P_i or is unassigned in M. Let p_i be the most-preferred agent on the master list with this property. Let $p_j \in P_i$. Then p_j must obtain an agent p_k in M that he strictly prefers to p_i, for otherwise $\{p_i, p_j\}$ blocks M. Let M' denote the matching at the beginning of iteration z. Then Lemma 5.4.2 implies that p_j is unassigned in M'. Hence $\{p_j, p_k\} \notin M'$. Additionally, P_k must be defined, as the algorithm did not return null at an iteration prior to z, and non-empty, as p_k's list contains at least p_j, for otherwise $\{p_k, p_j\}$ has been deleted, contradicting Lemma 5.4.7.

Let $y < z$ be the iteration of E where p_k is considered. By Cases (i) and (ii) above, and the fact that p_i is the most-preferred agent on the master list to obtain an agent inferior to a key agent (noting that p_k appears strictly before p_i on the master list), it follows that $p_j \in P_k$. If, at iteration y, $p_k \in S_y$, then $p_j \in T_y$, therefore by Lemma 5.4.1, p_j is assigned in M', a contradiction. Hence $p_k \in T_y \setminus S_z$. Thus by Lemma

5.4.6, it follows that $p_j \in S_y$, and again by Lemma 5.4.1, p_j is assigned in M', a contradiction.

□

We now show that if the algorithm returns an assignment then this assignment is in fact a matching, i.e. no agent is multiply assigned.

Lemma 5.4.9. *If* SRTI-ML-`super` *does not return null then the algorithm outputs a matching* M*.*

Proof. Let E be the execution of SRTI-ML-`super` for an instance I of SRTI-ML and suppose that the algorithm does not return null during E. Now let M be the assignment output on termination of E and suppose for a contradiction that M is not a matching. Hence there exist two pairs $\{p_i, p_j\} \in M$ and $\{p_k, p_j\} \in M$, where $i \neq k$. We first claim that p_i and p_k must be tied in the master list. For, suppose not. Without loss of generality suppose that p_i precedes p_k on the master list. If p_i is considered before p_k, then $\{p_j, p_k\}$ is deleted before p_j becomes assigned to p_k, a contradiction. Hence p_k is considered before p_i. This can only happen if $p_j \in S_y$ for some iteration y, where p_j either precedes p_i or is tied with p_i in the master list. Since the algorithm did not return null during E, at iteration y it follows that $P_j = \{p_k\}$. Hence by Lemma 5.4.8, the only agent p_j can become assigned to in M is p_k, contradicting the fact that $\{p_i, p_j\} \in M$. Hence the claim is established. Let z be the iteration of E where p_i and p_k become assigned to p_j - both must become assigned at the same iteration as by Lemma 5.4.2 we only consider unassigned agents at each iteration of E. Then either (i) $p_j \in S_z$ or (ii) $p_j \in T_z \setminus S_z$.

Case (i): Since p_i and p_k appear in the same tie, and both become assigned to p_j at iteration z, it follows that $\{p_i, p_k\} \subseteq P_j$. Hence the algorithm would have returned null at line 8, a contradiction.

Case (ii): Since p_i and p_k both become assigned to p_j at iteration z, it follows that $p_i, p_k \in S_z$. Therefore $P_i = P_k = \{p_j\}$. Hence the algorithm would have returned null at line 14, a contradiction.

□

The following result establishes that if the algorithm returns a matching then that matching is indeed super-stable.

Lemma 5.4.10. *If a matching* M *is output by* SRTI-ML-`super`*, then* M *is super-stable.*

Proof. Let E be the execution of SRTI-ML-`super` for an instance I of SRTI-ML. Now suppose for a contradiction that there exists a pair $\{p_i, p_j\}$ that blocks M. If $\{p_i, p_j\}$ has been deleted, then, without loss of generality, p_i became assigned to an agent p_k that he prefers to p_j, and so $\{p_i, p_j\}$ cannot block M. Hence $\{p_i, p_j\}$ has not been deleted. Also, at least one of p_i and p_j must be assigned in M, for if not the algorithm would have either returned null or assigned p_i and p_j to one another. Now suppose, without loss of generality, that p_i became assigned to an agent p_k at some iteration z. In this case Lemma 5.4.5 implies that $p_k \in P_i$ and, since $\{p_i, p_j\}$ blocks M, either (i) p_i strictly prefers p_j to p_k, or (ii) p_i is indifferent between p_j and p_k.

Case (i): In this case the pair $\{p_i, p_j\}$ is already deleted at iteration z, a contradiction.

Case (ii): As p_i becomes assigned to p_k at iteration z, Lemma 5.4.5 implies that $p_k \in P_i$. Then as p_k and p_j appear in the same tie on the master list, and since $\{p_i, p_j\}$ has not been deleted, it follows that $p_j \in P_i$. Therefore if $p_i \in S_z$, the algorithm would have returned null as $|P_i| > 1$. Therefore $p_i \in T_z \setminus S_z$. Thus by Lemma 5.4.6, it follows that $p_j \in S_z$. Hence by Lemma 5.4.1, p_j becomes assigned to an agent p_l at iteration z, and so is assigned to p_l in M. Furthermore as the algorithm did not return null, it follows that $P_j = \{p_l\}$. Thus if p_i appears before p_l on the master list then $\{p_i, p_j\}$ has been deleted, a contradiction. Similarly if p_i is strictly worse than p_l on the master list, then $\{p_i, p_j\}$ is deleted when p_j and p_l become assigned at iteration z, a contradiction. Therefore p_i and p_l must appear in the same tie on the master list, and since $\{p_i, p_j\}$ has not been deleted, it follows that $P_j = \{p_i, p_l\}$. Hence the algorithm would have returned null at line 8, a contradiction.

□

We now show that if the algorithm returns null as a result of some agent in S having more than one key agent, then no super-stable matching exists.

Lemma 5.4.11. *If the algorithm* SRTI-ML-`super` *returns null at line 8, then no super-stable matching exists.*

Proof. Let E be the execution of SRTI-ML-`super` for an instance I of SRTI-ML. Now suppose, for a contradiction, that at some iteration z of E, there exists an agent $p_i \in S_z$ such that $|P_i| > 1$. Let M be a super-stable matching in I. By Lemma 5.4.8, p_i must

obtain an agent $p_j \in P_i$ in M. However since $|P_i| > 1$, there exists an agent $p_k \in P_i$ such that $p_k \neq p_j$ and $\{p_i, p_k\} \notin M$, as M is a matching. Hence in M, p_k must obtain an agent p_l, where p_l appears strictly before p_i on the master list, for otherwise $\{p_i, p_k\}$ blocks M. However, since $p_i \in S_z$, it follows that p_l must have been considered at an iteration y prior to z. Then either (i) $p_l \in S_y$, or (ii) $p_l \in T_y \setminus S_y$.

Case (i): Since the algorithm did not return null prior to iteration z, it follows that $P_l = \{p_x\}$, for some agent p_x. However, if $p_k = p_x$, $\{p_l, p_k\} \in M$ and so $\{p_i, p_k\}$ was deleted prior to iteration z, a contradiction. Hence $p_k \neq p_x$, and so by Lemma 5.4.8, p_x is assigned to p_l in M. Hence p_k cannot be matched to p_l in M, a contradiction.

Case (ii): As $p_l \in T_y \setminus S_y$, there exists an agent $p_x \in S_y$ such that $p_l \in P_x$. Now since the algorithm did not return null prior to iteration z, it follows that $P_x = \{p_l\}$. Again $p_x \neq p_k$, for otherwise $\{p_i, p_k\}$ would have already been deleted prior to iteration z, contradicting the fact that $p_k \in P_i$. Hence by Lemma 5.4.8, p_x is assigned to p_l in M. Hence p_k cannot be matched to p_l in M, a contradiction.

□

We now prove in the following lemma that if any two agents in S_z have the same key agent, for some iteration z during the execution of SRTI-ML-`super`, then no super-stable matching exists.

Lemma 5.4.12. *If the algorithm* SRTI-ML-`super` *returns null at line 14, then no super-stable matching exists.*

Proof. Let E be the execution of SRTI-ML-`super` for an instance I of SRTI-ML. Now suppose for a contradiction that at iteration z of E, there exist at least two agents $p_i, p_j \in S_z$ such that $P_i \cap P_j \neq \emptyset$, and that M is a super-stable matching in I. Since the algorithm did not return null at line 8, it follows that $|P_i| = |P_j| = 1$, and since $P_i \cap P_j \neq \emptyset$, we have $P_i = P_j = \{p_k\}$. Hence by Lemma 5.4.8, both p_i and p_j are matched in M to p_k, a contradiction to the fact that M is a matching. □

We now show that the matching returned by SRTI-ML-`super` given an instance I of SRTI-ML is the unique super-stable matching in I.

Lemma 5.4.13. *If algorithm* SRTI-ML-`super` *returns a super-stable matching M during the execution E for an instance I of* SMTI-ML*, then M is the unique super-stable matching in I.*

Proof. Let $\{p_i, p_j\} \in M$ and let z denote the iteration of E where p_i and p_j became assigned. Then without loss of generality $p_i \in S_z$. Hence $p_j \in P_i$ and $|P_i| = 1$, so $P_i = \{p_j\}$. Thus Lemma 5.4.8 implies that $\{p_i, p_j\} \in M'$, for any stable matching M'. Furthermore if p_i is unmatched in M, then p_i is unmatched in M' by Lemma 5.4.7. □

Using Lemma 5.4.11 and Lemma 5.4.12 we obtain the following lemma.

Lemma 5.4.14. *If algorithm* SRTI-ML-`super` *returns null, then no super-stable matching exists.*

Using a similar complexity argument to that in Section 4.4 for algorithm HRT-ML-`super`, we can show that SRTI-ML-`super` runs in time $O(\lambda)$, where λ is the total length of the preference lists. We use this result together with Lemmas 5.4.9, 5.4.10, 5.4.11 and 5.4.14 to obtain the following theorem.

Theorem 5.4.15. *For a given instance of* SRTI-ML*, algorithm* SRTI-ML-`super` *finds the unique super stable matching, or reports that none exists, in time $O(\lambda)$, where λ is the total length of the preference lists.*

5.5 SRTI-ML under strong stability

Scott [67] showed that the problem of finding a strongly stable matching, or reporting that none exists, given an instance I of SRTI is solvable in time $O(\lambda^2)$, where λ is the total length of the preference lists. In this section we describe an $O(\sqrt{n}\lambda)$ algorithm for finding a strongly stable matching, or reporting none exists, given an instance of SRTI-ML, where n is the number of agents.

Algorithm SRTI-ML-`strong` is shown in Algorithm 23. In this section the definition of the terms *considered* and *processed* are identical to the corresponding definitions given in Section 5.4. Similarly for an iteration z during an execution of algorithm SRTI-ML-`strong`, we define U_z, S_z and T_z as in Section 5.4. Finally we also use P_i to represent the head of an agent p_i's list at the iteration during which p_i is considered. We refer to P_i as p_i's *key agents*. We again show in this section that P_i is well-defined for each such p_i. That is, we show that the definition of P_i does not depend on a particular execution of the algorithm. However in contrast to the case for super-stability, a strongly stable matching for an instance I of SRTI-ML need not be unique.

Algorithm SRTI-ML-`strong` proceeds as follows: we construct a matching M that is initially empty. Then for each tie U in the master list in turn we let S denote the set of agents in U that have a non-empty preference list and who are currently unassigned. Next we identify the set T, which is constructed from the union of the sets P_i for each agent $p_i \in S$. If the cardinality of the sets S and T are different, then (as we will show) no strongly stable matching exists. Otherwise a graph G is then built using algorithm `BuildGraph` shown in Algorithm 24. The vertex set of G consists of those agents in $S \cup T$. An edge is added between each agent $p_i \in S$ to each vertex $p_j \in P_i$. A maximum cardinality matching M^+ is then constructed in G. If there exists an agent $p_i \in S \cup T$ such that p_i is unassigned in M^+ then (as we will show) no strongly stable matching exists, otherwise we add the pairs in M^+ to M. Finally for each agent $p_i \in S$, we identify p_i's assignee p_j in M, and then for each strict successor p_k of p_j on p_i's list we delete the pair $\{p_i, p_k\}$, where 'delete the pair' means to delete p_i from p_k's list and vice-versa. Similarly, for each strict successor p_k of p_i from p_j's list we delete the pair $\{p_j, p_k\}$.

In the following lemma we show that each agent considered at an iteration z of the main loop during an execution E is assigned in M^+.

Lemma 5.5.1. *Let E be an execution of* SRTI-ML-`strong` *for an instance I of* SRTI-ML. *Then at every iteration z of the main loop during E, each agent $p_i \in S_z \cup T_z$ is assigned in M^+, or else the algorithm returns null.*

Proof. This is immediate, for if an agent in $S_z \cup T_z$ is unassigned in M^+ then the algorithm would have returned null at line 12, a contradiction. □

We now show that if p_i is considered at iteration z, and P_i is the head of p_i's list at this iteration, then each agent p_j in P_i is unassigned.

Lemma 5.5.2. *Let E be an execution of* SRTI-ML-`strong` *for an instance I of* SRTI-ML *and let $p_i \in S_z \cup T_z$ at some iteration z of E. Let P_i denote the head of p_i's list at the beginning of iteration z and let $p_j \in P_i$. Then p_j is unassigned at the beginning of iteration z.*

Proof. The proof is identical to that of Lemma 5.4.2. □

In the following proof we show that if $p_i \in T_z$, for some iteration z during an execution E of SRTI-ML-`strong`, and if an agent $p_j \in S_z$ has p_i as a key agent, then p_j is a key agent of p_i.

Algorithm 23 SRTI-ML-`strong`

```
1: M := ∅;
2: for each tie U in the master list in turn do
3:    S := set of agents in U whose lists are non-empty and are currently unassigned;
4:    T := ⋃_{p_i ∈ S} P_i; /** P_i denotes the head of p_i's list at this iteration */
5:
6:    if |S| ≠ |T| then
7:       return null;
8:
9:    G := BuildGraph(S,T); /** See Algorithm 24 */
10:   M+ := maximum cardinality matching in G;
11:   if there exists p_i ∈ S ∪ T such that M+(p_i) = ∅ then
12:      return null;
13:
14:   M := M ∪ M+;
15:
16:   for each p_i ∈ S do
17:      p_j := M(p_i);
18:      for each strict successor p_k of p_j on p_i's list do
19:         delete the pair {p_i, p_k};
20:      for each strict successor p_k of p_i on p_j's list do
21:         delete the pair {p_j, p_k};
22: return M;
```

Algorithm 24 `BuildGraph`(S,T)

```
1: V := S ∪ T;
2: E := ∅
3: for each p_i ∈ S do
4:    for each p_j ∈ P_i do
5:       E := E ∪ {{p_i, p_j}};
6:
7: G := (V, E);
8: return G;
```

Lemma 5.5.3. *Let E be an execution of* SRTI-ML-`strong` *for an instance I of* SRTI-ML. *Then at each iteration z during E, if $p_i \in T_z$ and $p_i \in P_j$, where $p_j \in S_z$, then $p_j \in P_i$.*

Proof. The proof is identical to that of Lemma 5.4.4. □

Now using the lemma above we can obtain the following result.

Lemma 5.5.4. *Let E be an execution of* SRTI-ML-`strong` *for an instance I of* SRTI-ML. *If at some iteration z during E, p_i becomes assigned to p_j, then $p_j \in P_i$.*

Proof. At iteration z either (i) $p_i \in S_z$ or (ii) $p_i \in T_z \setminus S_z$.

> *Case (i):* Since p_i and p_j are assigned to one another in M^+, it follows that $\{p_i, p_j\}$ represents an edge in G at iteration z. Hence as $p_i \in S_z$, it follows by the construction of G that $p_j \in P_i$.
>
> *Case (ii):* Again, since p_i and p_j become assigned at iteration z, it follows that $\{p_i, p_j\}$ represents an edge in G at iteration z. Furthermore as $p_i \in T_z \setminus S_z$, it follows that $p_j \in S_z$ and $p_i \in P_j$. Hence by Lemma 5.5.3, we have that $p_j \in P_i$.

□

Lemma 5.5.4 tells us that no matter which maximum cardinality matching M^+ is chosen at line 10 of SRTI-ML-`strong`, the same set of pairs will be deleted at each loop iteration. We use this observation, together with Lemma 5.5.1 and Lemma 5.5.2, to show that P_i is well-defined (in the sense described in the second paragraph of this section) for each $p_i \in \mathcal{P}$.

Lemma 5.5.5. *For each agent $p_i \in \mathcal{P}$, if p_i's set of key agents P_i is defined then P_i is well-defined.*

Proof. The result follows by an identical argument to the proof of Lemma 5.4.3, together with the observation that the same set of pairs are deleted no matter which matching M^+ is chosen at each iteration for a given execution of the algorithm. □

We now show that if $p_i \in T_z$, for some iteration z during an execution E of SRTI-ML-`strong`, then all of p_i's key agents are contained in S_z.

Lemma 5.5.6. *Let E be an execution of* SRTI-ML-`strong` *for an instance I of* SRTI-ML. *Then at each iteration z during E, if $p_i \in T_z$ then $P_i \subseteq S_z$, where P_i is the head of p_i's list at iteration z.*

Proof. The proof is identical to that of Lemma 5.4.6. □

We note here that if $p_i \in T_z$ and $p_j \in P_i$ then Lemma 5.5.6 shows that $p_j \in S_z$, however it need not be the case that $p_i \in P_j$.

In the following lemma we prove that no pair which belongs to a strongly stable matching is ever deleted by the algorithm.

Lemma 5.5.7. *Algorithm* SRTI-ML-`strong` *never deletes a pair* $\{p_i, p_j\}$ *that belongs to a strongly stable matching.*

Proof. The proof is identical to that of Lemma 5.4.7. □

In the following lemma we prove that in every strongly stable matching, each agent p_i obtains a key agent.

Lemma 5.5.8. *Let* p_i *be an agent whose set of key agents* P_i *is defined and let* M *be a strongly stable matching. Then* $\{p_i, p_j\} \in M$*, for some* $p_j \in P_i$*.*

Proof. The proof is identical to that of Lemma 5.4.8. □

In the following lemma we prove that a matching output by the algorithm is indeed strongly stable.

Lemma 5.5.9. *A matching returned by* SRTI-ML-`strong` *is strongly stable.*

Proof. Let E be an execution of SRTI-ML-`strong` for an instance I of SRTI-ML and let M be a matching returned at the end of E. Now suppose for a contradiction that M is not strongly stable. Hence there exists a pair $\{p_i, p_j\}$ that blocks M. If the pair $\{p_i, p_j\}$ has been deleted then, without loss of generality, we can assume that p_i is assigned in M to an agent whom he prefers to p_j. Hence the pair $\{p_i, p_j\}$ does not block M. Therefore, the pair $\{p_i, p_j\}$ has not been deleted. Also p_i and p_j must be assigned in M, for otherwise the algorithm would have returned null at line 12. As $\{p_i, p_j\}$ blocks M, without loss generality p_i prefers p_j to his assignee p_k in M. Let z be the iteration of E where p_i becomes assigned to p_k. By Lemma 5.5.4, $p_k \in P_i$. Hence $\{p_i, p_j\}$ has already been deleted prior to iteration z, a contradiction.

□

We now show that if, at an iteration z during an execution E of SRTI-ML-`strong`, the size of S_z and T_z are not equal, then no strongly stable matching exists.

Lemma 5.5.10. *If algorithm* SRTI-ML-`strong` *returns null at line 7 then no strongly stable matching exists.*

Proof. Let E be an execution of SRTI-ML-`strong` for an instance I of SRTI-ML. Now suppose that at some iteration z during E, $|S_z| \neq |T_z|$, and suppose for a contradiction that M is a strongly stable matching in I. Consider the following two cases.

> *Case (i):* $|S_z| > |T_z|$. Then by Lemma 5.5.8 each agent in S_z is assigned to an agent in T_z in M. However clearly in this case some agent $p_i \in T_z$ is multiply assigned, and as such M is not a matching, a contradiction.
>
> *Case (ii):* $|S_z| < |T_z|$. Again using Lemma 5.5.8, each agent in $p_i \in T_z$ is assigned, in M, to an agent in P_i. Now using Lemma 5.5.6, $P_i \subseteq S_z$, therefore each such p_i must obtain an agent in S_z in M, which is impossible as $|T_z| > |S_z|$.

□

In the following proof we show that if, at an iteration z during an execution E of SRTI-ML-`strong`, some agent $p_i \in S_z \cup T_z$ is unassigned in M^+ then no strongly stable matching exists.

Lemma 5.5.11. *If algorithm* SRTI-ML-`strong` *returns null at line 12, then no strongly stable matching exists.*

Proof. Let E be an execution of SRTI-ML-`strong` for an instance I of SRTI-ML. Now suppose for a contradiction that at iteration z of E there exists an agent $p_i \in S_z \cup T_z$ such that $M^+(p_i) = \emptyset$, and that M is a strongly stable matching in I. Then since $p_i \in S_z \cup T_z$, it follows that P_i is defined and non-empty. Hence by Lemma 5.5.8, p_i is assigned to an agent p_j in M, where $p_i \in P_j$ and $p_j \in P_i$. Now if $p_i \in S_z$ then, by definition of T_z, we have that $p_j \in T_z$. Furthermore if $p_i \in T_z$ then, since $p_j \in P_i$, it follows by Lemma 5.5.6, that $p_j \in S_z$. Therefore both p_i and p_j are represented by vertices in G, and G contains the edge $\{p_i, p_j\}$. As such if p_j is unassigned in M^+, we contradict the maximality of M^+ by adding the edge $\{p_i, p_j\}$ to M^+. Therefore p_j must be assigned in M^+ to some agent p_k. Hence by Lemma 5.5.4, it follows that $p_k \in P_j$. Then since $\{p_i, p_j\} \in M$ and p_j is indifferent between p_i and p_k, either (i) p_k obtains an agent p_l strictly better than p_j in M, or (ii) p_k obtains an agent p_l in M and p_l and p_j appear in the same tie on the master list, otherwise $\{p_j, p_k\}$ blocks M.

Case (i): Then $p_j \in P_k$ by Lemma 5.5.4. Hence the pair $\{p_k, p_l\}$ has been deleted. Thus by Lemma 5.5.7, the pair $\{p_k, p_l\}$ cannot belong to any strongly stable matching, a contradiction.

Case (ii): Since p_j becomes assigned to p_k in M^+, there exists an edge $\{p_j, p_k\}$ in G. Also as $\{p_k, p_l\} \in M$, by Lemma 5.5.8, it follows that $p_k \in P_l$ and $p_l \in P_k$. Therefore if $p_k \in S_z$, since $p_l \in P_k$, it follows that $p_l \in T_z$. Similarly if $p_k \in T_z \setminus S_z$ then by Lemma 5.5.6, it follows that $p_l \in S_z$, since $p_l \in P_k$. Hence G contains the edge $\{p_k, p_l\}$. Thus p_l cannot be unassigned in M^+, for otherwise we can augment along the path $\{p_i, p_j\}, \{p_j, p_k\}, \{p_k, p_l\}$, increasing the size of M^+ and contradicting the maximality of $|M^+|$.

Let p_r denote p_l's assignee in M^+, hence there exists an edge $\{p_r, p_l\}$ in G. Thus by Lemma 5.5.4, we have that $p_r \in P_l$ and $p_l \in P_r$. Then p_r cannot be unassigned in M, as P_r is non-empty. Additionally p_r cannot obtain an agent worse than p_l, for otherwise $\{p_l, p_r\}$ blocks M (as $\{p_l, p_k\} \in M$ and p_l is indifferent between p_k and p_r). Also by Case (i), p_r cannot obtain an agent better than p_l in M. Hence in M, p_r must be assigned to an agent p_w such that p_l and p_w appear in the same tie on the master list. Thus by Lemma 5.5.8, $p_r \in P_w$ and $p_w \in P_r$. Therefore if $p_r \in S_z$, then $p_w \in T_z$ by construction of T_z, and if $p_r \in T_z$, then by Lemma 5.5.6, $p_w \in S_z$. As a result there exists an edge $\{p_r, p_w\}$ in G. Therefore p_w cannot be unassigned in M^+ for otherwise we can increase the size of M^+ by augmenting along the path $\{p_i, p_j\}, \{p_j, p_k\}, \{p_k, p_l\}, \{p_l, p_r\}, \{p_r, p_w\}$.

Let p_x denote p_w's assignee in M^+. Clearly if we continue in a manner identical to the above, since the number of agents is finite, we must reach a point where no such p_x exists, a contradiction.

□

We now bring together Lemma 5.5.10 and Lemma 5.5.11 to obtain the following lemma.

Lemma 5.5.12. *If algorithm* SRTI-ML-`strong` *returns null, then no strongly stable matching exists.*

The complexity of the algorithm is dominated by finding a maximum matching in the graph G constructed at each loop iteration. All other operations take a total of $O(\lambda)$ time, taken over the algorithm's entire execution, where λ is the total length of the preference

lists. The fastest algorithm for finding a maximum matching in a general graph $G = (V, E)$ is due to Micali and Vazirani [57], and has time complexity $O(\sqrt{|V|}|E|)$. Hence the time complexity of SRTI-ML-`strong` is $O(\sqrt{n}(m_1 + m_2 + \cdots + m_r))$, where m_z is the number of edges in G_z, where G_z is the graph at iteration z and r is the total number of loop iterations. Therefore SRTI-ML-`strong` has time complexity $O(\sqrt{n}\lambda)$.

Finally we bring together the time complexity analysis with Lemmas 5.5.9 and 5.5.12 to obtain the following theorem.

Theorem 5.5.13. *For a given instance of* SRTI-ML*, algorithm* SRTI-ML-`strong` *finds a strongly stable matching, or reports that none exists, in time* $O(\sqrt{n}\lambda)$*, where* λ *is the total length of the preference lists and* n *is the number of agents.*

Chapter 6

Stable Matching Problems with Symmetric Preferences

6.1 Introduction

In this chapter we study variants of SM, HR and SR with *symmetric preferences*. An instance I of a stable matching problem is said to have symmetric preferences when the rank of each agent p_i on p_j's list is equal to that of p_j on p_i's list. We denote an instance of SM with symmetric preferences by SM-SYM, with SR-SYM and HR-SYM being similarly defined. Figure 6.1 shows the preference lists for an instance I of SM-SYM.

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

Figure 6.1: Instance I of SM-SYM.

To understand the motivation for symmetric preferences, we consider the more general problem of SR with globally-ranked pairs (SR-GRP) [5]. An instance of SR-GRP is a restriction of SR in which preferences are derived from a ranking function $rank : E \rightarrow \mathbb{N}$ that acts on the edges of an arbitrary graph $G = (V, E)$. An agent $p_i \in V$ prefers an agent p_j to p_k if $rank(e) < rank(e')$, where $e = \{p_i, p_j\}$ and $e' = \{p_i, p_k\}$. It is known that SR-GRP can be used to model the preferences in a kidney exchange programme [2, 65, 66]. In such

a programme, there exists a set of patients each with a willing, but incompatible, donor, who would like to exchange their donor kidney for another, compatible, donor kidney. When two (donor, patient) pairs are matched together, the transplant is only carried out after results obtained using expensive last-minute compatibility test are known. As such, it is advantageous for a doctor (and patient) to use the potential success of a transplant (which may be estimated by a scoring system taking into account factors such as blood type, tissue-type etc.) as a criterion for ranking donors and patients. It is easy to see that this can be achieved using the SR-GRP model.

It is straightforward to see that SR-SYM is a special case of SR-GRP. For, given an instance I of SR-SYM, we create the underlying graph $G = (V, E)$ of I in the usual way, and we create a rank function $r : E \rightarrow \mathbb{N}$ as follows: for any edge $\{p_i, p_j\} \in E$, $r(\{p_i, p_j\})$ is the rank of p_i on p_j's preference list.

We note that given an instance of SM-SYM, the men's preference lists form a Latin square $S = [s_{i,j}]$ (as seen from the men's preferences in Figure 6.1). A matrix $T = [t_{i,j}]$ representing the women's preference lists can be derived from S as follows: $\forall i, j$ $(1 \leq i, j, \leq n)$, if $s_{i,j} = k$ then $t_{k,j} = i$, where n is the number of men in I. In the following lemma we prove that T is also a Latin square.

Lemma 6.1.1. *Let S be the Latin square derived from the men's preference lists for a given instance I of* SM-SYM*, and let T be derived from S in the following way: for each i and j $(1 \leq i, j \leq n)$ if $s_{i,j} = k$ then $t_{k,j} = i$, where n is the number of men. Then T is also a Latin square.*

Proof. Suppose that T is not a Latin square. Then $t_{i,j} = t_{i,k} = l$, for some $j \neq k$. It follows, by definition of T, that $s_{l,j} = s_{l,k} = i$, contradicting the fact that S is a Latin square. Now suppose $t_{i,j} = t_{k,j} = l$, for some $i \neq k$. Then similarly, $s_{l,j} = i$ and $s_{l,j} = k$, which is impossible. Hence T is a Latin square. □

For a given instance of SMTI, HRT, and SRTI, we consider two models arising from the interpretation of an agent's rank. Let p_i be an agent and let A_i denote the set of agents that p_i finds acceptable. Then we denote the set of agents in the r^{th} tie on p_i's list by $T_{i,r}$. For the first model, we define the rank of p_j on p_i's list, denoted by $rank(p_i, p_j)$, to be k, where $p_j \in T_{i,k}$. In the second model, we define the rank of p_j on p_i's list, denoted by $rank(p_i, p_j)$, to be $1 + |\{p_k \in A_i : p_i \text{ strictly prefers } p_k \text{ to } p_j\}|$. For example, consider the preference list for a single man m_1 in an SMTI instance shown in Figure 6.2 below.

Then in the first model, $rank(m_1, w_1) = 1$, $rank(m_1, w_3) = 2$, and $rank(m_1, w_8) = 4$, whilst in the second model $rank(m_1, w_1) = 1$, $rank(m_1, w_3) = 3$, and $rank(m_1, w_8) = 7$. Additionally, we say that an agent p_j is in r^{th} *place* on p_i's list if $rank(p_i, p_j) = r$. We denote an instance of SMT with symmetric preferences where ranks are interpreted using the first model by SMT-SYM1 (with SMTI-SYM1, HRT-SYM1, SRT-SYM1, and SRTI-SYM1 being similarly defined). Also, we denote an instance of SMT with symmetric preferences whose ranks are interpreted using the second model by SMT-SYM2 (again SMTI-SYM2, HRT-SYM2, SRT-SYM2 and SRTI-SYM2 are similarly defined). Additionally, if a result is established for a problem that is specified using an instance without the trailing number, e.g SMTI-SYM, then the result holds regardless of the rank interpretation.

$$m_1 : \quad (w_1 \quad w_2) \quad w_3 \quad (w_4 \quad w_5 \quad w_6) \quad (w_7 \quad w_8)$$

Figure 6.2: m_1's preference list.

To understand the motivation behind our first model, we observe that if an agent is genuinely indifferent between a set of agents, the rank of an agent in a given tie should not be dependent on the number of agents in preceding tie(s). The second model is analogous to the convention used in athletics events, whereby if two agents are first to cross the finishing line, and do so at the same time, they are deemed to be joint first, and the next person to cross the line is said to be third.

In this chapter we present a range of algorithmic results for stable matching problems with ties involving symmetric preferences. Some of these result are given in terms of model 1, some in terms of model 2, whilst others hold regardless of the model under consideration.

We firstly observe that given an instance of SRI-SYM, there exists a simple algorithm to find a stable matching. We simply assign each agent p_i to the first agent p_j on his preference list. It is easy to show that this is indeed a stable matching.

The main results of this chapter are as follows. In Section 6.2.1 we describe a polynomial-time algorithm that finds a weakly stable matching, given an instance I of SRTI-SYM. In Section 6.2.2, we show that for an instance I of SMTI-SYM, weakly stable matchings may have different sizes. We then show in Section 6.2.2 that, given an instance I of SMTI-SYM1, the problem of determining if a complete weakly stable matching exists is NP-complete. We give an alternative reduction to prove that the problem of determining if a complete weakly stable matching exists, given an instance of SMTI-SYM2, is also NP-complete. Next

we show, in Section 6.2.3 and Section 6.2.4 respectively, that each of the problems of finding a minimum regret weakly stable matching and an egalitarian weakly stable matching, given an instance of SMTI-SYM1, is NP-hard. Then in Section 6.2.5 we prove that, given an instance of SMTI-SYM1, the problem of determining if a (man,woman) pair belongs to a weakly stable matching is NP-complete.

In the remaining sections we describe new algorithms for the problems of finding a super-stable or strongly stable matching, or reporting that none exists, given instances of SRTI-SYM and HRT-SYM. The algorithms are simpler than, and in certain cases improve on, the time complexity of the best known algorithms [38, 39, 45, 67]. In Section 6.3.1 and Section 6.3.2, we give $O(\lambda)$ algorithms for the problems of finding a super-stable matching, or reporting that none exists, given an instance of SRTI-SYM and HRT-SYM respectively, where λ is the total length of the preference lists. Then in Section 6.4.1, we give an $O(\sqrt{n}\lambda)$ algorithm for the problem of finding a strongly stable matching, or reporting that none exists, given an instance of SRTI-SYM, where n is the number of agents. Finally in Section 6.4.2, we give an $O(\sqrt{C}\lambda)$ algorithm for finding a strongly stable matching, or reporting that none exists, given an instance of HRT-SYM, where C is the sum of the hospital capacities.

We note that a restriction of the SR-GRP model is described by Arkin et al. [8] (developed independently from the model presented in this chapter). In their paper the authors describe a model of SRTI involving 'geometric' preferences, whereby each agent's preference list is represented as a set of points in a metric space, with the distance between each pair of points indicating the mutual preference between two agents. An algorithm is presented that finds a weakly stable matching in polynomial-time, given an instance of SRTI with geometric preferences. Furthermore, polynomial-time algorithms are also presented for each of the problems of finding a strongly stable and super-stable matching, or reporting that none exists, given an instance of SRTI with geometric preferences. The author also provides a description of an algorithm for a minimum regret weakly stable matching (the actual stability definition to which the authors refer for the minimum regret problem is not clear, however we assume the matching is a minimum regret weakly stable matching). However, the problem described does not correspond to the definition of minimum regret widely used in the literature [15, 25, 26].

It can be easily shown that an instance of SR with geometric preferences described by Arkin et al. need not have symmetric preferences. Consider a triangle in the plane with

vertices u, v, w, where $d(u,v) = 1$, $d(u,w) = 2$ and $d(v,w) = 2.5$. This gives rise to the following non-symmetric preference lists:

$$u: \quad v \quad w$$
$$v: \quad u \quad w$$
$$w: \quad u \quad v$$

Hence the model considered by Arkin et al. is a special case of SR-GRP which is distinct from the SR-SYM model considered in this chapter.

6.2 Weakly stable matchings

6.2.1 Finding a weakly stable matching in SRTI-SYM

For an instance of SRT, it is known that the problem of finding a weakly stable matching is NP-hard [38, 62] (see Section 1.4.4). In this section we show that, by contrast, the problem of finding a weakly stable matching for an instance of SRTI-SYM is polynomial-time solvable.

Consider algorithm SRTI-SYM-`weak` shown in Algorithm 25. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ be the set of agents for an instance I of SRTI-SYM, and let $P_{i,r}$ denote the set of agents in r^{th} place on p_i's list. Furthermore, in model one, r^* is defined to be the maximum number of ties on any agent's list, and in model two, r^* is defined to be the maximum length of any agent's list. Then for each r $(1 \leq r \leq r^*)$, we construct a graph G_r, whose vertex set V_r consists of the union of the set of agents in $P_{i,r}$ who are unmatched, for each agent $p_i \in \mathcal{P}$. The edge set of G_r comprises edges of the form $\{p_i, p_j\}$, where $p_i \in \mathcal{V}_r$ and $p_j \in P_{i,r}$ and p_j is unmatched in M. We then find a maximal matching M_r in G_r, and add M_r to M.

We now show that the matching returned by SRTI-SYM-`weak` is weakly stable. In the proof of the following lemma, we say that a pair $\{p_i, p_j\}$ is *considered* at iteration r when $p_j \in P_{i,r}$ (and since the preference lists are symmetric, it follows that $p_i \in P_{j,r}$).

Lemma 6.2.1. *The matching returned by* SRTI-SYM-`weak` *is weakly stable.*

Proof. Let E be an execution of SRTI-SYM-`weak` for an instance I of SRTI-SYM and let M be the matching returned at the end of E. Now suppose for a contradiction that M is not weakly stable. Hence there exists a pair $\{p_i, p_j\}$ that blocks M. Let r be the unique iteration of E where the pair $\{p_i, p_j\}$ is considered. We identify the following two cases.

Algorithm 25 SRTI-SYM-`weak`

$M := \emptyset$;
$r := 1$;
while $r \leq r^*$ **do**
  $V_r := \bigcup_{p_i \in \mathcal{P}} \{p_i \in P_{i,r} : M(p_i) = \emptyset\}$;
  $E_r := \emptyset$;
  for each agent $p_i \in V_r$ **do**
    for each agent $p_j \in P_{i,r} \cap V_r$ **do**
      $E_r := E_r \cup \{\{p_i, p_j\}\}$;
  $G_r := (V_r, E_r)$;
  $M_r :=$ maximal matching in G_r;
  $M := M \cup M_r$;
  $r = r + 1$;
return M;

Case (i): p_i obtains an agent p_k in M such that p_i strictly prefers p_j to p_k. Let $r' > r$ be the iteration of E where p_i and p_k become assigned. We observe that p_j must be unassigned at the beginning of iteration r, for otherwise p_j became assigned at an iteration prior to r, and so is assigned to an agent strictly better than p_i, as such $\{p_i, p_j\}$ does not block M. Now as p_i is unassigned at the beginning of iteration r' (this follows from the fact that p_i and p_k become assigned at iteration r', and so both must be unassigned at the beginning of iteration r' by the construction of G_r), p_i must have been unassigned at the end of iteration r. Hence if p_j is unassigned at the end of iteration r, we can add $\{p_i, p_j\}$ to M_r contradicting the maximality of M_r. Therefore p_j must have become assigned to some agent p_l at iteration r, and so $rank(p_i, p_j) = rank(p_j, p_l)$. Thus $\{p_i, p_j\}$ does not block M, a contradiction.

Case (ii): p_i is unassigned in M (and so is unassigned at the end of iteration r). Then as in Case (i), p_j must be unassigned at the beginning of iteration r. Furthermore p_j must become assigned to some agent p_k during iteration r, for otherwise we can add the pair $\{p_i, p_j\}$ to M_r, contradicting the maximality of M_r. Hence p_j is indifferent between p_i and p_k, therefore $\{p_i, p_j\}$ does not block M, a contradiction.

□

We observe that the time complexity of the algorithm is dominated by finding a maximal matching in the graph G_r; all other operations take time $O(\lambda)$, where λ is the total length of the preference lists. We can find a maximal matching in G_r in time $O(|E_r|)$ using

a straightforward greedy algorithm. Therefore any two graphs constructed have disjoint sets of vertices and edges, the overall time complexity of finding all maximal matchings is $O(\lambda)$. Hence the overall time complexity for the algorithm is $O(\lambda)$. We use this result together with Lemma 6.2.1 to obtain the following theorem.

Theorem 6.2.2. *For a given instance of* SRTI-SYM*, algorithm* SRTI-SYM-`weak` *returns a weakly stable matching in time $O(\lambda)$, where λ is the total length of the preference lists.*

6.2.2 Finding a complete weakly stable matching in SMTI-SYM

In this section we focus on SMTI-SYM. We show that weakly stable matchings may be of different sizes, and that the problem of deciding whether a complete weakly stable matching exists is NP-complete. We observe that these results hold, by restriction, for SRTI-SYM and HRT-SYM. First consider instance I_1 of SMTI-SYM1 shown in Figure 6.3. Here two weakly stable matchings are $M_1 = \{(m_1, w_1), (m_3, w_2)\}$ and $M_1' = \{(m_1, w_3), (m_2, w_1), (m_3, w_2)\}$. To show that weakly stable matchings can be of different sizes for an instance of SMTI-SYM2, consider instance I_2 shown in Figure 6.4. In I_2, two possible weakly stable matchings are $M_2 = \{(m_1, w_1), (m_3, w_2), (m_4, w_4)\}$ and $M_2' = \{(m_1, w_3), (m_2, w_1), (m_3, w_2), (m_4, w_4)\}$. Now consider the decision problem COM-SMTI-SYM1 defined as follows (COM-SMTI-SYM2 can be similarly defined):

Name :	COM-SMTI-SYM1
Instance:	An SMTI-SYM1 instance I.
Question:	Does I admit a complete weakly stable matching?

Men's preferences	Women's preferences
m_1 : $(w_1 \;\; w_2) \;\; w_3$	w_1 : $(m_1 \;\; m_2)$
m_2 : w_1	w_2 : $(m_1 \;\; m_3)$
m_3 : $(w_2 \;\; w_3)$	w_3 : $m_3 \;\; m_1$

Figure 6.3: Instance I_1 of SMTI-SYM1.

In the following sections we show that both COM-SMTI-SYM1 and COM-SMTI-SYM2 are NP-complete.

Men's preferences	Women's preferences
$m_1:\ (w_1\ \ w_2)\ \ w_3$	$w_1:\ (m_1\ \ m_2)$
$m_2:\ w_1$	$w_2:\ (m_1\ \ m_3)$
$m_3:\ (w_2\ \ w_3)$	$w_3:\ (m_3\ \ m_4)\ \ m_1$
$m_4:\ (w_3\ \ w_4)$	$w_4:\ m_4$

Figure 6.4: Instance I_2 of SMTI-SYM2.

First model

Here we show that the problem COM-SMTI-SYM1 is NP-complete using a reduction from EXACT-MM in subdivision graphs, which was shown to be NP-complete in Section 2.2.2.

Theorem 6.2.3. COM-SMTI-SYM1 *is NP-complete.*

Proof. Clearly COM-SMTI-SYM1 is in NP. To prove that COM-SMTI-SYM1 is NP-hard, we reduce from EXACT-MM in subdivision graphs. Let $G = (V, E)$ (a subdivision graph of some graph G'), and K (a positive integer), be an instance of EXACT-MM. Suppose that $V = U \cup W$ is a bipartition of G, where $U = \{m_1, m_2, \ldots, m_{n_1}\}$ and $W = \{w_1, w_2, \ldots, w_{n_2}\}$. Denote the set of vertices adjacent to a vertex $m_i \in U$ in G by U_i and similarly the set of vertices adjacent to $w_i \in W$ in G by W_i.

We construct an instance I of COM-SMTI-SYM1 as follows: let $U \cup X \cup A \cup B$ be the set of men and $W \cup Y \cup A' \cup B'$ be the set of women, where $X = \{x_1, x_2, \ldots, x_{n_2-K}\}$, $Y = \{y_1, y_2, \ldots, y_{n_1-K}\}$, $A = \{a_1, a_2, \ldots, a_K\}$, $B = \{b_1, b_2, \ldots, b_K\}$, $A' = \{a'_1, a'_2, \ldots, a'_K\}$ and $B' = \{b'_1, b'_2, \ldots, b'_K\}$. The preference lists of I are shown in Figure 6.5. It may be verified that I is an instance of SMTI-SYM1. We claim that G has an exact maximal matching of size K if and only if I admits a complete weakly stable matching.

Suppose G has a maximal matching M, where $|M| = K$. We construct a matching M' in I as follows. Initially let $M' = M$. There remain $n_1 - K$ men in U that are not assigned to women in W in M'; denote these men by m_{k_i} $(1 \leq i \leq n_1 - K)$ and add (m_{k_i}, y_i) to M'. Similarly there remain $n_2 - K$ women in W that are not assigned to men in U in M'; denote these women by w_{l_j} $(1 \leq j \leq n_2 - K)$, and add (x_j, w_{l_j}) to M'. Finally we add (a_i, a'_i) and (b_i, b'_i) $(1 \leq i \leq K)$ to M'. It is easy to verify that M' is a complete matching, and it remains to prove that M' is weakly stable.

Suppose for a contradiction that M' is not weakly stable. Hence there exists a pair that blocks M'. We note that since the matching is complete, no person in $A \cup B \cup A' \cup B'$

Men's preferences

m_i : (U_i) $(y_1 \;\; y_2 \;\; \ldots \;\; y_{n_1-K})$ $(1 \leq i \leq n_1)$

x_i : b'_i (W) $(1 \leq i \leq n_2 - K)$

a_i : $(y_i \;\; a'_i)$ $(1 \leq i \leq K)$

b_i : b'_i $(1 \leq i \leq K)$

Women's preferences

w_j : (W_j) $(x_1 \;\; x_2 \;\; \ldots \;\; x_{n_2-K})$ $(1 \leq j \leq n_2)$

y_j : a_j (U) $(1 \leq j \leq n_1 - K)$

a'_j : a_j $(1 \leq j \leq K)$

b'_j : $(b_j \;\; x_j)$ $(1 \leq j \leq K)$

Figure 6.5: Preference lists for the constructed instance of COM-SMTI-SYM1.

can be involved in a blocking pair and hence neither can each person in $X \cup Y$. Therefore any pair that blocks M' must have the form (m_i, w_j), where $m_i \in U$ and $w_j \in W$. Hence $(m_i, y_l) \in M'$ and $(x_k, w_j) \in M'$, for some $x_k \in X$ and $y_l \in Y$, so it follows that m_i and w_j are unassigned in M. However if this is the case then we can add (m_i, w_j) to M, which contradicts the maximality of M.

Conversely suppose that M' is a complete weakly stable matching in I. Let $M = M' \cap E$. We show that $|M| = K$. First suppose that $|M| < K$. Thus as M' is a complete weakly stable matching, at least $n_1 - K + 1$ men in U must be assigned in M' to women in Y, which is impossible as there are only $n_1 - K$ women in Y. Now suppose $|M| > K$. Then at most $n_1 - K - 1$ women in Y are assigned in M' to men in U. Since M' is complete, there exists at least one women in Y assigned in M' to a man in A. Thus at most $K - 1$ men in A are assigned in M' to women in A'. Hence only $K - 1$ women in A' are assigned in M', contradicting the fact that M' is a complete weakly stable matching.

Finally suppose that the matching M is not maximal in G. Hence there exists an edge (m_i, w_j) in G, such that no edge in M is incident to m_i or w_j. Therefore in M', m_i is assigned to some woman $y_l \in Y$, and w_j is assigned to some man $x_k \in X$. Hence (m_i, w_j) blocks M' in I, contradicting the weak stability of M'. Therefore M is indeed maximal in G. □

The following remark is used in Sections 6.2.3, 6.2.4 and 6.2.5.

Remark 6.2.4. The instance of SMTI-SYM1 constructed in the proof of Theorem 6.2.3 can be extended to that shown in Figure 6.6 with straightforward modifications to the proof of correctness. This allows us to assume that each man and woman has exactly two ties on their list.

Men's preferences

m_i : (U_i) $(y_1 \;\; y_2 \;\; \ldots \;\; y_{n_1-K})$ $(1 \leq i \leq n_1)$

x_i : a'_i (W) $(1 \leq i \leq n_2 - K)$

a_i : $(y_i \;\; a'_i)$ c'_i $(1 \leq i \leq K)$

b_i : b'_i c'_i $(1 \leq i \leq K)$

c_i : c'_i $(a'_i \;\; b'_i)$ $(1 \leq i \leq K)$

Women's preferences

w_j : (W_j) $(x_1 \;\; x_2 \;\; \ldots \;\; x_{n_2-K})$ $(1 \leq j \leq n_2)$

y_j : a_j (U) $(1 \leq j \leq n_1 - K)$

a'_j : $(x_j \;\; a_j)$ c_j $(1 \leq j \leq K)$

b'_j : b_j c_j $(1 \leq j \leq K)$

c'_j : c_j $(a_j \;\; b_j)$ $(1 \leq j \leq K)$

Figure 6.6: Preference lists for the extended constructed instance of COM-SMTI-SYM1.

Second model

We now show that if the second interpretation of an agent's rank is used, i.e. $rank(p_i, p_j) = 1 + |\{p_k \in A_i : p_i \text{ prefers } p_k \text{ to } p_j\}|$, the problem of deciding whether a complete weakly stable matching exists remains NP-complete.

Theorem 6.2.5. COM-SMTI-SYM2 *is NP-complete.*

Proof. Clearly COM-SMTI-SYM2 is in NP. To prove that COM-SMTI-SYM2 is NP-hard we reduce from EXACT-MM restricted to subdivision graphs of cubic graphs, which as noted in Section 2.2.2 is NP-complete. Let $G = (V, E)$ (a subdivision graph of some cubic graph G'), and K (a positive integer), be an instance of EXACT-MM. Suppose that $V = U \cup W$ is a bipartition of G, where $U = \{m_1, m_2, \ldots, m_{n_1}\}$ and $W = \{w_1, w_2, \ldots, w_{n_2}\}$. Without loss of generality suppose that each vertex in U has degree 3 and each vertex in W has

degree 2. Then we denote the set of vertices adjacent to a vertex $m_i \in U$ in G by U_i and similarly the set of vertices adjacent to $w_i \in W$ in G by W_i.

We construct an instance I of COM-SMTI-SYM2 as follows: let $U \cup X \cup A \cup B \cup C \cup D \cup E$ be the set of men and $W \cup Y \cup A' \cup B' \cup C' \cup D' \cup E'$ be the set of women, where $X = \{x_1, x_2, \ldots, x_{n_2-K}\}$, $Y = \{y_1, y_2, \ldots, y_{n_1-K}\}$, $A = \{a_1, a_2, \ldots, a_K\}$, $A' = \{a'_1, a'_2, \ldots, a'_K\}$, $B = \{b_1, b_2, \ldots, b_K\}$, $B' = \{b'_1, b'_2, \ldots, b'_K\}$, $C = \{c_1, c_2, \ldots, c_K\}$, $C' = \{c'_1, c'_2, \ldots, c'_K\}$ $D = \{d_1, d_2, \ldots, d_K\}$, $D' = \{d'_1, d'_2, \ldots, d'_K\}$, $E = \{e_1, e_2, \ldots, e_K\}$, and $E' = \{e'_1, e'_2, \ldots, e'_K\}$. The preference lists of I are shown in Figure 6.7. It may be verified that I is an instance of SMTI-SYM2. We claim that G has a maximal matching of size K if and only if I admits a complete weakly stable matching.

Men's preferences

m_i :	(U_i) $(y_1 \;\; y_2 \;\; \ldots \;\; y_{n_1-K})$	$(1 \le i \le n_1)$
x_i :	$(d'_i \;\; e'_i)$ (W)	$(1 \le i \le n_2 - K)$
a_i :	$(y_i \;\; a'_i)$	$(1 \le i \le K)$
b_i :	$(y_i \;\; b'_i)$	$(1 \le i \le K)$
c_i :	$(y_i \;\; c'_i)$	$(1 \le i \le K)$
d_i :	d'_i	$(1 \le i \le K)$
e_i :	e'_i	$(1 \le i \le K)$

Women's preferences

w_j :	(W_j) $(x_1 \;\; x_2 \;\; \ldots \;\; x_{n_2-K})$	$(1 \le j \le n_2)$
y_j :	$(a_j \;\; b_j \;\; c_j)$ (U)	$(1 \le j \le n_1 - K)$
d'_j :	$(x_j \;\; d_j)$	$(1 \le j \le K)$
e'_j :	$(x_j \;\; e_j)$	$(1 \le j \le K)$
a'_j :	a_j	$(1 \le j \le K)$
b'_j :	b_j	$(1 \le j \le K)$
c'_j :	c_j	$(1 \le j \le K)$

Figure 6.7: Preference lists for the constructed instance of COM-SMTI-SYM2.

Suppose G has a maximal matching M, where $|M| = K$. We construct a matching M' in I as follows. Initially let $M' = M$. There remain $n_1 - K$ men in U that are not assigned to women in W in M'; denote these men by m_{k_i} $(1 \le i \le n_1 - K)$ and add (m_{k_i}, y_i) to M'. Similarly there remain $n_2 - K$ women in W that are not assigned to men in U in M';

denote these women by w_{l_j} $(1 \leq j \leq n_2 - K)$, and add (x_j, w_{l_j}) to M'. Finally we add (a_i, a'_i), (b_i, b'_i), (c_i, c'_i), (d_i, d'_i), (e_i, e'_i) $(1 \leq i \leq K)$ to M'. It is easy to verify that M' is complete, and it remains to prove that M' is weakly stable.

We first observe that each person in $A \cup A' \cup B \cup B' \cup C \cup C' \cup D \cup D' \cup E \cup E'$ obtains their first-choice partner so cannot be involved in a blocking pair. Hence each person in $X \cup Y$ also cannot be involved in a blocking pair. Therefore any pair that blocks M must have the form (m_i, w_j), where $m_i \in U$ and $w_j \in W$. Suppose that (m_i, w_j) blocks M'. Then $(m_i, y_l) \in M'$ for some $y_l \in Y$ and $(x_k, w_j) \in M'$ for some $x_k \in X$. Therefore in M, each of m_i and w_j is unassigned, and so we can add (m_i, w_j) to M, contradicting the maximality of M.

Conversely suppose that M' is a complete weakly stable matching in I'. Let $M = M' \cap E$. We show that M is a maximal matching in G and that $|M| = K$. First suppose $|M| < K$. Then as M' is complete, at least $n_1 - K + 1$ men in U are assigned in M' to women in Y, which is impossible as there are only $n_1 - K$ women in Y. Now suppose $|M| > K$. Then at most $n_1 - K - 1$ men in U are assigned in M' to women in Y. Therefore at least one woman $y_j \in Y$ is assigned in M' to a man $q \in A \cup B \cup C$. Suppose that $q \in A$. Then at most $K - 1$ men in A are assigned in M' to women in A'. Hence there exists a woman in A' who is unassigned in M', contradicting the fact that M' is a complete weakly stable matching. A similar argument can be used if $q \in B \cup C$. Therefore $|M| = K$ as required.

Finally we prove that M is indeed a maximal matching. For, suppose not. Hence there exists an edge $(m_i, w_j) \in E$ such that each of m_i and w_j are unassigned in M. Therefore, in M', m_i is assigned to a woman $y_l \in Y$ and w_j is assigned to a man $x_k \in X$, as M' is complete. Hence the pair (m_i, w_j) blocks M', a contradiction. □

6.2.3 Finding a minimum regret weakly stable matching in SMT-SYM1

We recall from Section 1.1.3 that a matching M has minimum regret if

$$r(M) = \max_{p \in \mathcal{M} \cup \mathcal{W}} cost_M(p)$$

is minimised over all weakly stable matchings, given an instance I of SMT-SYM1. It was shown in [54] that the problem of finding a minimum regret weakly stable matching is NP-hard, given an instance of SMT. In this section we prove that the same is true even for SMT-SYM1.

Now consider the following decision problem:

Name :	REGRET-SMT-SYM1-D
Instance:	An SMT-SYM instance I and a positive integer K.
Question:	Does I admit a weakly stable matching M with $r(M) \leq K$?

We prove in the following theorem that REGRET-SMT-SYM1-D is NP-complete.

Theorem 6.2.6. REGRET-SMT-SYM1-D *is NP-complete.*

Proof. Clearly REGRET-SMT-SYM1-D is in NP. To show that the problem is NP-hard, we reduce from the restriction of COM-SMTI-SYM1 in which each person's list has exactly two ties, which is NP-complete by Theorem 6.2.3 and Remark 6.2.4. Let I be such an instance of SMTI-SYM1, where $U = \{m_1, m_2, \ldots, m_{n_1}\}$ is the set of men and $W = \{w_1, w_2, \ldots, w_{n_2}\}$ is the set of women in I. Furthermore we lose no generality (by Theorem 6.2.3 and Remark 6.2.4) by assuming that $n_1 = n_2 = n$. For each man $m_i \in U$ $(1 \leq i \leq n)$, we denote m_i's preference list in I by U_i. Similarly for each woman $w_j \in W$ $(1 \leq j \leq n)$, we denote w_j's preference list in I by W_j.

We construct an instance I' of SMT-SYM1 as follows: let U be the set of men and W be the set of women (as in instance I). The preference lists in I' are shown in Figure 6.8. We claim that I has a complete weakly stable matching M if and only if I' has a weakly stable matching M' where $r(M') \leq 2$.

Men's preferences

$m_i : \quad U_i \quad (W \setminus U_i) \qquad (1 \leq i \leq n)$

Women's preferences

$w_j : \quad W_j \quad (U \setminus W_j) \quad (1 \leq j \leq n)$

Figure 6.8: Preference lists for the constructed instance of REGRET-SMT-SYM1-D.

Suppose that M is a complete weakly stable matching in I. Let $M' = M$. Then clearly M' is also weakly stable in I'. Additionally each man and woman must have a partner in U_i and W_j respectively. Therefore $r(M') \leq 2$, as required.

Conversely suppose that M' is a weakly stable matching in I' such that $r(M') \leq 2$. Let $M = M'$. Then clearly each man and woman in $U \cup W$ is assigned in M, and has a

partner in U_i and W_j respectively. Therefore as M' is weakly stable in I', it follows that M is weakly stable in I, as required. □

6.2.4 Finding an egalitarian weakly stable matching in SMT-SYM

We first recall from Section 1.1.3 that an egalitarian weakly stable matching M is a weakly stable matching such that

$$c(M) = \sum_{p \in \mathcal{M} \cup \mathcal{W}} cost_M(p)$$

is minimised over all weakly stable matchings in I. It was shown in [54] that the problem of finding an egalitarian weakly stable matching is NP-hard, given an instance of SMT. In this section we show that the same is true even for SMT-SYM1.

To prove that the problem of finding an egalitarian weakly stable matching is NP-hard, we observe a result of Gergely [23], shown in Theorem 6.2.7, relating to *diagonalized* Latin squares. A *transversal* of a Latin square A is a set S of n distinct entries $a_{i,j}$ of A such that $|\{i : a_{i,j} \in S\}| = n$ and $|\{j : a_{i,j} \in S\}| = n$. A Latin square is said to be diagonalized if the main diagonal is a transversal.

Theorem 6.2.7 (Gergely [23])**.** *For any integer $n \geq 3$, there exists a diagonalized Latin square of order n having a transversal which has no common entry with the main diagonal.*

Now consider the following decision problem:

Name :	EGAL-SMT-SYM1-D
Instance:	An SMT-SYM1 instance I and a positive integer K.
Question:	Does I admit a weakly stable matching M with $c(M) \leq K$?

We prove in the following theorem that EGAL-SMT-SYM1-D is NP-complete.

Theorem 6.2.8. EGAL-SMT-SYM1-D *is NP-complete.*

Proof. Clearly EGAL-SMT-SYM1-D is in NP. To show that the problem is NP-hard, we reduce from the restriction of COM-SMTI-SYM1 in which each person's list has exactly two ties, which is NP-complete by Theorem 6.2.3 and Remark 6.2.4. Let I be such an instance of SMTI-SYM1 where $U = \{m_1, m_2, \ldots, m_{n_1}\}$ is the set of men and $W = \{w_1, w_2, \ldots, w_{n_2}\}$ is the set of women. Furthermore we lose no generality (by Theorem 6.2.3 and Remark 6.2.4) by assuming that $n_1 = n_2 = n$. For each man $m_i \in U$ $(1 \leq i \leq n)$ we denote m_i's preference list in I by U_i. Similarly for each woman $w_j \in W$ $(1 \leq j \leq n)$ we denote w_j's preference list in I by W_j.

We construct an instance I' of SMT-SYM1 as follows: let $U \cup X \cup \{p\}$ be the set of men and let $W \cup Y \cup \{q\}$ be the set of women, where $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. Then we construct the preference lists in I' by considering the diagonalized Latin square of order n, as constructed using Gergely's method [23] (we note that Gergely's construction is polynomial-time computable). Let S be the constructed Latin square of order n (an example of such a Latin square for $n = 8$ is shown in Figure 6.10). We first ensure that the main diagonal in S has entries in the order $1, 2, \ldots, n$; this can be achieved by simply swapping symbols in S, e.g. in the Latin square D in Figure 6.10, we apply the permutations $\langle 1\rangle$, $\langle 2, 4, 5, 3\rangle$, $\langle 6\rangle$, $\langle 7, 8\rangle$ to obtain the desired Latin square as shown in Figure 6.11. Next we construct a Latin square T from S using the method given in Lemma 6.1.1. It is straightforward to verify that T is diagonalized with elements in order $1, 2, \ldots, n$ in the main diagonal. We then use S and T to construct the preference lists as shown in Figure 6.9. By the construction of T from S and by inspection of the remaining preference list entries, we observe that I' is an instance of SMT-SYM1 with symmetric preferences. Let $K = 2(3n + 1)$. We claim that I has a complete weakly stable matching M if and only if I' has a weakly stable matching M' where $c(M') \leq K$.

Suppose that M is a complete weakly stable matching in I. Then we construct a matching M' in I' as follows: $M' = M \cup \{(p, q)\} \cup \{(x_i, y_i) : 1 \leq i \leq n\}$. Then clearly M' is weakly stable in I', as M is weakly stable in I and every person in $X \cup Y \cup \{p, q\}$ is assigned to their first-choice partner. Also, each person $a_i \in U \cup W$ has $cost_{M'}(a_i) \leq 2$, and each person $a_j \in X \cup Y \cup \{p, q\}$ has $cost_{M'}(a_j) = 1$. Therefore $c(M') \leq (2n+n+1)+(2n+n+1) = 2(3n + 1) = K$.

Conversely suppose that M' is a weakly stable matching in I' such that $c(M') \leq K$. We observe that p and q are assigned to one another in every weakly stable matching and also that x_i is assigned to y_i $(1 \leq i \leq n)$ in every weakly stable matching. Hence each man m_i is assigned in M' to a woman in U_i or a woman in $W \setminus U_i$. Now suppose there exists a man $m_k \in U$, such that $M'(m_k) \in W \setminus U_k$. Then $cost_{M'}(m_k) = n + 3$, and since the preference lists are symmetric, $cost_{M'}(M'(m_k)) = n + 3$. Hence $c(M') \geq ((n + 3) + (n - 1) + n + 1) + ((n+3) + (n-1) + n + 1) = 3n+3+3n+3 = 6(n+1)$, contradicting the fact that that $c(M') \leq K$. Thus each man $m_i \in U$ and woman $w_j \in W$ must be assigned to a partner in U_i and W_j respectively. Now let $M = M' \setminus (\{(p, q)\} \cup \{(x_i, y_i) : 1 \leq i \leq n\})$. Then as each person $a_i \in U \cup W$ is assigned to a partner in $U_i \cup W_i$ in M and since M' is weakly stable in I', it follows that M is a complete weakly stable matching in I. □

Men's preferences

$m_i:\ U_i\ (y_1\ q)\ y_2\ \ldots\ y_n\ (W \setminus U_i) \qquad (1 \le i \le n)$

$x_1:\ y_1\ q\ (W)\ y_{s_{1,2}}\ y_{s_{1,3}}\ y_{s_{1,4}}\ \cdots\ y_{s_{1,n}}$

$x_2:\ y_2\ q\ y_{s_{2,1}}\ (W)\ y_{s_{2,3}}\ y_{s_{2,4}}\ \cdots\ y_{s_{2,n}}$

$x_3:\ y_3\ q\ y_{s_{3,1}}\ y_{s_{3,2}}\ (W)\ y_{s_{3,4}}\ \cdots\ y_{s_{3,n}}$

$\vdots$

$x_n:\ y_n\ q\ y_{s_{n,1}}\ y_{s_{n,2}}\ y_{s_{n,3}}\ y_{s_{n,4}}\ \ldots\ (W)$

$p:\ q\ (Y)\ (W)$

Women's preferences

$w_j:\ W_j\ (x_1\ p)\ x_2\ \ldots\ x_n\ (U \setminus W_j) \qquad (1 \le j \le n)$

$y_1:\ x_1\ p\ (U)\ x_{t_{1,2}}\ x_{t_{1,3}}\ x_{t_{1,4}}\ \ldots\ x_{t_{1,n}}$

$y_2:\ x_2\ p\ x_{t_{2,1}}\ (U)\ x_{t_{2,3}}\ x_{t_{2,4}}\ \ldots\ x_{t_{2,n}}$

$y_3:\ x_3\ p\ x_{t_{3,1}}\ x_{t_{3,2}}\ (U)\ x_{t_{3,4}}\ \ldots\ x_{t_{3,n}}$

$\vdots$

$y_n:\ x_n\ p\ x_{t_{n,1}}\ x_{t_{n,2}}\ x_{t_{n,3}}\ x_{t_{n,4}}\ \ldots\ (U)$

$q:\ p\ (X)\ (U)$

Figure 6.9: Preference lists for the constructed instance of EGAL-SMT-SYM1-D.

6.2.5 Finding weakly stable pairs in SMT-SYM

In this section we consider the problem of determining if, given an instance I of SMT-SYM1 and a (man,woman) pair (m_i, w_j), there exists a weakly stable matching M in I such that $(m_i, w_j) \in M$. This problem was shown to be NP-complete by Manlove et al. [54] given an instance of SMT . We show that this NP-completeness result holds even in the presence of symmetric preference lists. Consider the following decision problem:

Name :	PAIR-SMT-SYM1-D
Instance:	An SMTI-SYM instance I and a (man,woman) pair (m_i, w_j).
Question:	Does there exists a weakly stable matching M in I such that $(m_i, w_j) \in M$?

We prove in the following theorem that PAIR-SMT-SYM1-D is NP-complete.

Theorem 6.2.9. PAIR-SMT-SYM1-D *is NP-complete.*

$$\begin{bmatrix} 1 & 6 & 7 & 8 & 5 & 2 & 3 & 4 \\ 2 & 3 & 6 & 7 & 8 & 4 & 5 & 1 \\ 8 & 4 & 5 & 6 & 7 & 1 & 2 & 3 \\ 7 & 8 & 1 & 2 & 6 & 3 & 4 & 5 \\ 6 & 7 & 8 & 3 & 4 & 5 & 1 & 2 \\ 5 & 2 & 4 & 1 & 3 & 6 & 7 & 8 \\ 4 & 1 & 3 & 5 & 2 & 7 & 8 & 6 \\ 3 & 5 & 2 & 4 & 1 & 8 & 6 & 7 \end{bmatrix}$$

Figure 6.10: Latin square D.

$$\begin{bmatrix} 1 & 6 & 8 & 7 & 3 & 4 & 2 & 5 \\ 4 & 2 & 6 & 8 & 7 & 5 & 3 & 1 \\ 7 & 5 & 3 & 6 & 8 & 1 & 4 & 2 \\ 8 & 7 & 1 & 4 & 6 & 2 & 5 & 3 \\ 6 & 8 & 7 & 2 & 5 & 3 & 1 & 4 \\ 3 & 4 & 5 & 1 & 2 & 6 & 8 & 7 \\ 5 & 1 & 2 & 3 & 4 & 8 & 7 & 6 \\ 2 & 3 & 4 & 5 & 1 & 7 & 6 & 8 \end{bmatrix}$$

Figure 6.11: D after the swap.

Proof. Clearly PAIR-SMT-SYM1-D is in NP. To show that the problem is NP-hard, we reduce from the restriction of COM-SMTI-SYM1 in which each person's list has exactly two ties, which is NP-complete by Theorem 6.2.3 and Remark 6.2.4. Let I be such an instance of SMTI-SYM1, where $U = \{m_1, m_2, \ldots, m_{n_1}\}$ is the set of men and $W = \{w_1, w_2, \ldots, w_{n_2}\}$ is the set of women. Furthermore, we lose no generality (by Theorem 6.2.3 and Remark 6.2.4) by assuming that $n_1 = n_2 = n$. For each man $m_i \in U$ $(1 \leq i \leq n)$ we denote m_i's preference list in I by U_i. Similarly for each woman $w_j \in W$ $(1 \leq j \leq n)$ we denote w_j's preference list in I by W_j.

We construct an instance I' of SMT-SYM1 as follows: let $U \cup \{x, a, b\}$ be the set of men and $W \cup \{y, a', b'\}$ be the set of women. The preference lists in I' are shown in Figure 6.12. It is straightforward to verify that I' is an instance of SMT-SYM1. We claim that I has a complete weakly stable matching M if and only if there exists a weakly stable matching M' in I' such that $(x, y) \in M'$.

Suppose that M is a complete weakly stable matching in I. Then we construct a matching M' in I' as follows: $M' = M \cup \{(x, y), (a, a'), (b, b')\}$. Now as M is weakly stable in I and each of a and b have their first-choice women in I', it follows that x cannot be involved in a blocking pair of M' in I, as a', b' and each woman in W has a partner in M' whom they prefer to x. Hence M' is a weakly stable matching in I' and $(x, y) \in M'$ as required.

Conversely suppose that M' is a weakly stable matching in I' such that $(x, y) \in M'$. It follows that $(a, a') \in M'$, for otherwise (a, a') blocks M' in I'. Also it follows that $(b, b') \in M'$. Now let $M = M' \setminus \{(x, y), (a, a'), (b, b')\}$. Then each man $m_i \in U$ cannot be

Men's preferences

m_i : U_i $(y$ a' $b')$ $(W \setminus U_i)$ $(1 \leq i \leq n)$

x : a' b' (W) y

a : $(y$ $a')$ b' (W)

b : b' $(y$ $a')$ (W)

Women's preferences

w_j : W_j $(x$ a $b)$ $(U \setminus W_j)$ $(1 \leq j \leq n)$

y : a b (U) x

a' : $(x$ $a)$ b (U)

b' : b $(x$ $a)$ (U)

Figure 6.12: Preference lists for the constructed instance of PAIR-SMT-SYM1-D.

assigned to a woman in $W \setminus U_i$, for otherwise the pair (m_i, y) blocks M'. Hence each such man m_i must be assigned to a woman in U_i. Therefore M is a complete weakly stable matching in I. □

6.3 Super-stable matchings

6.3.1 The case of SRTI-SYM

In this section we describe a polynomial-time algorithm for finding a super-stable matching, or reporting that none exists, given an instance of SRTI-SYM. The algorithm has time complexity identical to the algorithm due to Irving and Manlove [38] for the general case. However, the algorithm in this section is much simpler and involves only one phase, as opposed to two phases required in the general case.

In this section we use P_i to denote the tie in first place on p_i's list (this may be a tie of size 1, representing a single agent). The algorithm SRTI-SYM-`super` (shown in Algorithm 26) proceeds as follows: for each agent $p_i \in \mathcal{P}$, we first check that P_i contains exactly one agent; if this is not the case then (as we will show), no super-stable matching exists. We then construct the set T from the union of the sets P_i, for each $p_i \in \mathcal{P}$. If $|T|$ is not equal to n, then at least two people have the same agent at the head of their list and (as we will show) no super-stable matching exists. Otherwise we add $\{p_i, p_j\}$ to M for each $p_j \in P_i$.

Algorithm 26 SRTI-SYM-`super`

1: **for each** $p_i \in \mathcal{P}$ **do**
2: **if** $|P_i| > 1$ **then**
3: **return null**;
4: $T := \cup_{p_i \in \mathcal{P}} P_i$;
5: **if** $|T| \neq n$ **then**
6: **return null**;
7: $M := \{\{p_i, p_j\} : p_i \in \mathcal{P} \wedge p_j \in P_i\}$;
8: **return** M;

Lemma 6.3.1. *If algorithm* SRTI-SYM-`super` *returns null, then no super-stable matching exists, otherwise the matching output by the algorithm is a super-stable matching.*

Proof. Let I be an instance of SRTI-SYM. We first show that in every super-stable matching, p_i obtains an agent in P_i. Clearly p_i cannot obtain an agent better than those in P_i (as P_i is the tie in first place of p_i's list). Now suppose p_i obtains an agent p_j worse than those in P_i. Then each agent $p_k \in P_i$ must obtain an agent strictly better than p_i in M, for otherwise $\{p_i, p_k\}$ blocks M. However p_i must also be at the head of p_k's list as the preferences are symmetric. Therefore p_k cannot obtain an agent strictly better than p_i.

Let E be an execution of SRTI-SYM-`super` for I. Now suppose that during E the algorithm returns null when $|P_i| > 1$, for some $p_i \in \mathcal{P}$. Let M' be a super-stable matching in I. By the result above, p_i obtains an agent $p_j \in P_i$ in M'. Hence there exists an agent $p_k \in P_i \setminus \{p_j\}$ such that p_k must obtain an agent strictly better than p_i in M', for otherwise $\{p_i, p_k\}$ blocks M'. However since $rank(p_i, p_k) = 1$ and the preference lists are symmetric, it follows that $rank(p_k, p_i) = 1$. Therefore p_k cannot obtain an agent strictly better than p_i in M', and so there exists no super-stable matching in this case.

Now suppose that during E the algorithm returns null when $|T| \neq n$. Again let M be a super-stable matching in I. Since $|T| \neq n$, it follows that there exist two agents p_i and p_j such that $p_k \in P_i$ and $p_k \in P_j$. Then as the algorithm did not return null at line 3, we have that $|P_i| = |P_j| = 1$. Hence by the result first paragraph (i.e. an agent p_x is assigned to an agent in P_x in every super-stable matching), both p_i and p_j obtain p_k in M', which is impossible. Hence no super-stable matching exists.

Let M be the assignment output by the algorithm. Then as each agent is assigned to his first-choice agent and no two agents share the same agent at the head of their list (for otherwise the algorithm would have returned null at line 6), M is a matching.

Finally suppose that M is not super-stable and that the pair $\{p_i, p_j\}$ blocks M. We first observe that each agent must be assigned in M (as the algorithm did not return null, hence each agent is assigned to the single agent at the head of his list). Hence p_j is the single agent at the head of p_i's list and p_i is the single agent at the head of p_j's list. Therefore $\{p_i, p_j\} \in M$. Thus $\{p_i, p_j\}$ does not block M. □

It is easy to verify that the algorithm runs in time linear in the size of the problem instance. We thus obtain the following theorem.

Theorem 6.3.2. *For a given instance of* SRTI-SYM*, algorithm* SRTI-SYM-`super` *returns a super-stable matching, or reports that none exists, in time $O(\lambda)$, where λ is the total length of the preference lists.*

6.3.2 The case of HRT-SYM

In this section we present an algorithm HRT-SYM-`super` that finds a super-stable matching, or reports that none exists, given an instance of HRT-SYM. The algorithm is similar to SRTI-SYM-`super` shown in Section 6.3.1. Again the algorithm has an identical time complexity to the best known algorithm by Irving et al. [39] for the general case. However, again our algorithm is simpler and avoids the complex implementation issues associated with the algorithm for the general case.

Let $\mathcal{R} = \{r_1, r_2, \ldots, r_{n_1}\}$ be the set of residents and $\mathcal{H} = \{h_1, h_2, \ldots, h_{n_2}\}$ be the set of hospitals for an instance I of HRT-SYM. We use R_i to denote the set of hospitals in first place on a resident r_i's preference list. Algorithm HRT-SYM-`super` is shown in Algorithm 27. The algorithm proceeds as follows: for each agent $r_i \in \mathcal{R}$, if R_i contains more than one hospital, then (as we will show) no super-stable matching exists. We then construct H from the union of the sets R_i, for each $r_i \in \mathcal{R}$. Then for each hospital $h_j \in H$ we construct the set T_j, which contains the residents in $\mathcal{R}$ who have h_j in first place on their list. If there are more residents in T_j than there are posts in h_j (i.e. $|T_j| > c_j$) then (as we will show) no super-stable matching exists. Otherwise we obtain M by adding (r_i, h_j) to M for each $r_i \in \mathcal{R}$ and $h_j \in R_i$.

We prove in the following lemma that if the algorithm HRT-SYM-`super` returns a matching M, then M is indeed super-stable, and that if null is returned then no super-stable matching exists.

Algorithm 27 HRT-SYM-`super`

```
1: for each r_i ∈ R do
2:     if |R_i| > 1 then
3:         return null;
4: H := ∪_{r_i ∈ R} R_i;
5: for each h_j ∈ H do
6:     T_j := {r_i ∈ R : h_j ∈ R_i};
7:     if |T_j| > c_j then
8:         return null;
9: M := {(r_i, h_j) : r_i ∈ R ∧ h_j ∈ R_i};
10: return M;
```

Lemma 6.3.3. *If algorithm* HRT-SYM-`super` *returns null, then no super-stable matching exists, otherwise the matching output by the algorithm is a super-stable matching.*

Proof. Let I be an instance of HRT-SYM. We first show that in every super-stable matching each resident $r_i \in \mathcal{R}$ must obtain a hospital in R_i. Clearly r_i cannot obtain a hospital better than those in R_i. Now suppose r_i obtains a hospital worse than those in R_i. Then each hospital $h_j \in R_i$ must be full with residents it strictly prefers to r_i. However, as the preference lists are symmetric it follows that $rank(h_j, r_i) = 1$. Therefore h_j cannot be full with residents it prefers to r_i.

Let E be an execution of HRT-SYM-`super` for I. Now suppose that during E the algorithm returns null when $|R_i| > 1$, for some $r_i \in \mathcal{R}$. Let M' be a super-stable matching in I. By the result above, r_i obtains a hospital $h_j \in R_i$ in M'. Hence there exists a hospital $h_k \in R_i \setminus \{h_j\}$ such that h_k must be full with residents strictly better than r_i, for otherwise (r_i, h_k) blocks M'. However h_k cannot be full with residents better than h_k as $rank(h_k, r_i) = 1$. Therefore no super-stable matching exists in this case.

Now suppose that the algorithm returns null during E when $|T_j| > c_j$, for some $h_j \in H$, i.e. more than c_j residents have h_j at the head of their list. Again let M' be a super-stable matching in I. By the first paragraph each resident in T_j must be assigned to h_j in M, as each resident must have a single hospital at the head of their list, for otherwise the algorithm would have returned null at line 3. Therefore M' is not a matching, a contradiction.

We now prove that the assignment M returned by the algorithm is a super-stable matching. Clearly M is a matching as each resident in $\mathcal{R}$ is assigned exactly one hospital,

for otherwise the algorithm would have returned null at line 3, and each hospital is assigned no more than c_j residents, as if this was not the case then the algorithm would have returned null at line 8. Now suppose that the pair (r_i, h_j) blocks M. Then r_i has exactly one hospital at the head of his list and is assigned to this hospital in M, hence there exists no such pair (r_i, h_j) that blocks M. □

Again as in Section 6.3.1 it is easy to verify that the algorithm's runtime is linear in the size of that problem instance. Hence we obtain the following theorem.

Theorem 6.3.4. *For a given instance of* HRT-SYM*, algorithm* HRT-SYM-`super` *returns a super-stable matching, or reports that none exists, in time $O(\lambda)$, where λ is the total length of the preference lists.*

6.4 Strongly stable matchings

6.4.1 The case of SRTI-SYM

In this section we describe an algorithm SRTI-SYM-`strong` that finds a strongly stable matching or reports that none exists, given an instance of SRTI-SYM. Our algorithm is simpler than the algorithm due to Scott [67] for the general case, and reduces the time complexity from $O(\lambda^2)$ to $O(\sqrt{n}\lambda)$ (where λ is the total length of the preference lists and n is the number of agents).

Consider SRTI-SYM-`strong` shown in Algorithm 28. As in Section 6.3.1, P_i denotes the tie in first place on an agent p_i's list. The algorithm constructs a graph $G = (V, E)$. The vertex set V is constructed from the agents in $\mathcal{P}$. The edge set E is then constructed by adding an edge $\{p_i, p_j\}$ to E, for each $p_i \in \mathcal{P}$ and $p_j \in P_i$. Next we find a maximum cardinality matching M in $G = (V, E)$. If $|M| \neq |V|/2$ (i.e. M is not a perfect matching) then (as we will show) no strongly stable matching exists, otherwise M is a strongly stable matching.

Lemma 6.4.1. *If algorithm* SRTI-SYM-`strong` *returns null, then no strongly stable matching exists, otherwise the matching M output by the algorithm is a strongly stable matching.*

Proof. Let I be an instance of SRTI-SYM. Again we can easily verify, using a similar argument to that in Lemma 6.3.1, that in every strongly stable matching M, each agent $p_i \in \mathcal{P}$ must be assigned to an agent in P_i in M.

Algorithm 28 SRTI-SYM-`strong`

1: $V := \mathcal{P}$;
2: $E := \bigcup_{p_i \in \mathcal{P}} \{\{p_i, p_j\} : p_j \in P_i\}$;
3:
4: $M :=$ maximum cardinality matching in $G = (V, E)$;
5: **if** $|M| = |V|/2$ **then**
6: **return** M;
7: **else**
8: **return** **null**;

Let E be an execution of SRTI-SYM-`strong` for I. Now suppose that during E the algorithm returns null when $|M| \neq |V|/2$. Let M' be a strongly stable matching in I. By the result above each agent $p_i \in \mathcal{P}$ must be assigned in M' to an agent in P_i. However G is constructed from the agents in P_i for each $p_i \in \mathcal{P}$, yet $|M| < |V|/2$, hence no such M' exists.

To prove that the matching M output is strongly stable, we observe that every agent is assigned in M, for otherwise the algorithm would have returned null, and by the result above each agent is assigned to an agent in P_i (their first-choice agents). Hence there is no pair that blocks M. □

We observe that the time complexity of finding a strongly stable matching is dominated by the construction of a maximum cardinality matching in $G = (V, E)$. This can be achieved in time $O(\sqrt{|V|}|E|)$ using the Hopcroft-Karp algorithm [32]. Therefore the total time required to run SRTI-SYM-`strong` is $O(\sqrt{n}\lambda)$. We use this result to obtain the following theorem.

Theorem 6.4.2. *For a given instance of* SRTI-SYM*, algorithm* SRTI-SYM-`strong` *returns a matching that is strongly stable, or reports that none exists, in time* $O(\sqrt{n}\lambda)$*, where* n *is the number of agents and* λ *is the total length of the preference lists.*

6.4.2 The case of HRT

Here we present an algorithm HRT-SYM-`strong` for finding a strongly stable matching, or reporting that none exists, given an instance I of HRT-SYM. The algorithm improves on the best known time complexity (namely $O(C\lambda)$ [45], where λ is the total length of the preference lists and C the sum of the hospital capacities) for finding a strongly stable matching, or reporting that none exists, given a general instance of HRT.

As in Section 6.3.2, R_i denotes the set of hospitals in first place on a resident r_i's list. Similarly H_j denotes the set of residents in first place on a hospital h_j's list. Algorithm HRT-SYM-`strong` is shown in Algorithm 29. The algorithm builds a capacitated bipartite graph $G = (V, E)$ with upper degree constraining function u. First we construct the set H from the union of the sets R_i, for each $r_i \in \mathcal{R}$. The vertex set V comprises the residents in $\mathcal{R}$ and the hospitals in H. An edge (r_i, h_j) is then added to E for each $r_i \in \mathcal{R}$ and $h_j \in R_i$. The upper bound $u(r_i)$ for each resident $r_i \in R$ is set to 1, and for each hospital $h_j \in H$ the upper bound $u(h_j)$ is set to be $\min\{c_j, |H_j|\}$. A maximum degree-constrained subgraph D in G is then computed, and M is set to be the edges in D. We then check to see if every resident and hospital is assigned exactly $u(v)$ assignees in M, where $v \in V$ (i.e. whether each resident is assigned in M and each hospital is assigned exactly $\min\{c_j, |H_j|\}$ residents in M). If this is not the case, then (as we will show) no strongly stable matching exists. Otherwise we will prove that M is a strongly stable matching.

Algorithm 29 HRT-SYM-`strong`

```
1: H := ⋃_{r_i ∈ R} R_i;
2: V := R ∪ H;
3: E := ⋃_{r_i ∈ R} {(r_i, h_j) : h_j ∈ R_i};
4: for each r_i ∈ R do
5:     u(r_i) := 1;
6: for each h_j ∈ H do
7:     u(h_j) := min{c_j, |H_j|};
8:
9: D := maximum degree-constrained subgraph of G = (V, E);
10: M := edges of D;
11:
12: for each v ∈ V do
13:     if |M(v)| ≠ u(v) then
14:         return null;
15:
16: return M;
```

In the following lemma we prove that if a matching is returned by the algorithm then this matching is strongly stable, and if the algorithm returns null then no strongly stable matching exists.

Lemma 6.4.3. *If algorithm* HRT-SYM-`strong` *returns null, then no strongly stable match-*

ing exists, otherwise the matching M output by the algorithm is a strongly stable matching.

Proof. Again it is easy to show, using a similar argument to that in Lemma 6.3.3, each resident r_i is assigned in every strongly stable matching to a hospital in R_i.

Now suppose that there does not exist a maximum degree-constrained subgraph D with the property specified in line 13 and that there exists a strongly stable matching M' in I. Then either (i) some resident r_i is unmatched in M' or (ii) some hospital h_j has less than $\min\{c_j, |H_j|\}$ assignees in M'.

Case (i): By the result in the first paragraph above, each resident must be assigned in M', hence no strongly stable matching exists.

Case (ii): There exists a resident $r_i \in H_j \setminus \{M(h_j)\}$, therefore (r_i, h_j), blocks M'.

Let M be the matching output by the algorithm. Suppose for a contradiction that M is not strongly stable. Hence there exists a pair (r_i, h_j) that blocks M. Then as r_i is not assigned to h_j, and the algorithm did not return null, it follows that $u(h_j) = c_j$. Hence h_j must be full in M with assignees belonging to H_j, and r_i is assigned a hospital in R_i, therefore (r_i, h_j) does not block M. □

By inspection of the algorithm we can see that the time complexity is dominated by finding a maximum degree-constrained subgraph in $G = (V, E)$. This can be achieved in time $O(\sqrt{\min\{n, C\}}|E|)$ using Gabow's algorithm [16], where n is the total number of agents and C is the sum of the upper degree constraints. Therefore we have that HRT-SYM-`strong` runs in time $O(\sqrt{C}\lambda)$, where C is the sum of the hospital capacities and λ is the total length of the preference lists. We use this result and Lemma 6.4.3 to obtain the following theorem.

Theorem 6.4.4. *For a given instance of* HRT-SYM*, algorithm* HRT-SYM-`strong` *returns a matching that is strongly stable, or reports none exists, in time $O(\sqrt{C}\lambda)$, where λ is the total length of the preference lists, and C is the sum of the hospital capacities.*

6.5 Open problems

We conclude this chapter with a selection of open problems.

6.5.1 Minimum number of strongly blocking pairs

The problem of finding a weakly stable matching with the minimum number of strongly stable blocking pairs has been shown to be NP-hard and not approximable within $n^{1-\epsilon}$, for $\epsilon > 0$, given an instance of SR-GRP where n is the total number of agents. However it is open as to whether this problem is NP-hard or polynomial-time solvable, given an instance of SMT-SYM.

6.5.2 Optimal matching problems and the stable pair problem in the second model

We showed in Section 6.2.4, using a reduction from COM-SMTI-SYM1, that EGAL-SMT-SYM1-D is NP-complete. However, a similar style of reduction from COM-SMTI-SYM2 to EGAL-SMT-SYM2-D does not appear to be obvious. We conjecture that this problem remains NP-hard, but so far no proof is known. Additionally, a straightforward extensions of the NP-completeness reductions given for REGRET-SMT-SYM1-D and PAIR-SMT-SYM1-D to their model two counterparts do not appear to be obvious. Again we conjecture that these problems are NP-complete when applied to model two, but so far no proof is known.

Chapter 7

Constraint Programming and the Stable Marriage Problem

7.1 Introduction

In previous chapters we have explored stable matching problems from a mainly theoretical perspective. For many variants of these problems we have derived NP-hardness results. The NP-hardness of a computational problem naturally leads to the question of how to cope with this complexity in practice. To this end, preceding chapters have contained approximation algorithms. However by their very definition these algorithms cannot guarantee to solve an arbitrary instance to optimality. If optimal solutions are required then, assuming P $\neq$ NP, one is forced to settle for an exponential-time algorithm. A potential objective when designing such an algorithm is that it will perform reasonably well on problem instances that are likely to be considered. Constraint Programming (CP) offers one possible technique for obtaining exact algorithms for NP-hard optimisation problems. However, it is also applicable in situations where we are faced with a variant of a polynomial-time solvable problem that involves additional criteria, such that no polynomial-time algorithm for the variant is known.

This and the next chapter concern the application of CP techniques to the Stable Marriage (SM) and Hospitals/Residents (HR) problems with the objective of showing how NP-hard variants of these classical problems can be modelled and solved. Appendix A gives a general introduction to CP.

CP approaches to SM have been the focus of much attention in the literature in recent years [6, 20–22, 24, 50]. These previous studies have generally involved formulating SM as

a constraint satisfaction problem (CSP), examining the time complexities of establishing Arc Consistency (AC) within these CSP models and the structure of the solutions that can be derived from them. A benefit of this approach is that such CSP models can easily be extended using appropriate "side constraints" to capture variants of SM that are either NP-hard or involve additional constraints that do not appear to lend themselves easily to polynomial-time algorithms. This chapter described two such encodings of SM, and provides examples of SM variants that can be modelled using side constraints.

In order to successfully model variants of SM using side constraint, CSP encodings of SM require certain structural properties to be maintained. One of the most useful of these properties is the concept of the GS-lists. As noted in Section 1.1.2, the extended Gale-Shapley (EGS) algorithm has two possible orientations, namely the *man-oriented* EGS (MEGS) algorithm and the *woman-oriented* EGS (WEGS) algorithm. The reduced preference lists created as a result of the deletions made by both the MEGS and WEGS algorithm are known as the *MGS-lists* and *WGS-lists* respectively. We recall that the *GS-lists* are created from the intersection of the MGS-lists and WGS-lists, and allow us to take advantage of the many structural properties of SM. It is therefore natural to investigate the problem of obtaining a CSP encoding for SM.

Section 7.2 gives an overview of the previous CSP encodings of SM. Thereafter, we present two new CSP encodings for SMI. In Section 7.3 an $(n+1)$-valued encoding is given, this encoding is a elegant, easy to understand and a natural way to model an instance of SMI. We show that in this model arc consistency can be established in $O(n^3)$ time. We also present structural results and a failure-free enumeration strategy for finding all stable matchings for a given instance of SMI. The second encoding, presented in Section 7.4, is a 4-valued encoding. The encoding is more complex than the first encoding present, but as a result, AC can be established in time $O(n^2)$. Again we prove certain structural properties for the encoding exist and describe a failure-free enumeration strategy for finding all the stable matchings for a given instance of SMI. Finally, in Section 7.5, we present two NP-hard variants of SM and describe the side constraints required to obtain a solution using the $(n+1)$-valued encoding.

7.2 Overview of SM encodings

This section discusses CSP encodings for SM previously proposed in the literature. The first encoding we consider is due to Aldershof et al. [6]. This encoding uses a set of inequalities to model an instance of SM with n men and n women. For each (man,woman) pair an inequality is constructed by examination of the preference lists. An algorithm is then used to reduce the number of inequalities – it should be noted that it is possible that the number of inequalities may not be reduced in certain circumstances. A discussion indicating the relationship between variables of this encoding and the GS-lists is presented, however no proof is given. In addition to this, no explicit method of enumerating all stable matchings is shown. Overall this encoding is inefficient: there are $\Omega(n^2)$ constraints, the size of each variable's domain is 2, and the arity of the constraints is $\Omega(n)$ in the worst case. The complexity of establishing AC is $\Theta(2^{2n})$ time.

Next we consider two encodings for SMI due to Gent et al. [20]. The first encoding creates a CSP instance J_1 using a set of 'conflict matrices' to encode an SMI instance I. In J_1, AC is established in $O(n^4)$ time and after AC propagation the variables' domains correspond to the GS-lists of I in a particular way. The second CSP model is a Boolean encoding creating a CSP instance that we denote by J_2. In J_2, AC is established in $O(n^2)$ time, however the variables' domains after AC propagation only correspond to a weaker structure called the XGS-lists of I (see Section 1.1.5). In both encodings the set of all stable matchings in I can be enumerated in a failure-free manner.

An extension of SM was presented in the form of a CSP encoding by Dye [12]. Here a restricted model of SPA is encoded as a CSP. Dye presents a set of constraints where ties are allowed in the preference lists of the students (men). The report aims to find a weakly stable matching, or more specifically a weakly stable matching that satisfies some additional criteria, namely load-balancing the projects a lecturer may supervise and optimising the "student-optimality" of the matching. The analysis of the encoding is from a practical point of view, and does not focus on theoretical properties of SM. As a result, no structural properties are proved with reference to the GS-lists, nor is it considered whether all the weakly stable matchings can be found in a failure-free manner. The author also notes that, in practice, this approach is not particularly efficient for a large number of students.

An encoding presented by Lustig and Puget [50] bears some resemblance to the encoding we present in Section 7.3. The constraints for this encoding are shown in Figure 7.1.

The paper compares and contrasts linear programming and constraint programming, with SM being used as a concrete example of a combinatorial problem that may be solved using constraint programming techniques. The encoding is presented to illustrate the general techniques of constraint programming, and as such, the structural properties arising from the encoding are not considered. AC propagation with this encoding is established in $O(n^4)$ time.

```
solve {
    forall(m in Men)
        husband[wife[m]] = m;
    forall(w in Women)
        wife[husband[w]] = w;
    forall(m in Men & o in Women)
        rankMen[m,o] < rankMen[m,wife[m]] =>
            rankWomen[o,husband[o]] < rankWomen[o,m];
    forall(w in Women & o in Men)
        rankWomen[w,o] < rankWomen[w,husband[w]] =>
            rankMen[o,wife[o]] < rankMen[o,w];
}
```

Figure 7.1: Constraints for SM instance found in [50].

Lastly we consider an encoding due to Green and Cohen [24]. The paper describes a framework that is used to determine if a given instance of a CSP is tractable. Here an encoding of SM is presented, and the framework is used to explain the tractability of this encoding. The model used is complicated, and is constructed to facilitate the use of the framework. The increased complexity does not reflect an increase in performance, with AC being established in $O(n^4)$ time. Additionally, no method of enumerating all stable matchings and no structural properties of SM are given.

7.3 $(n+1)$-valued encoding

We now present an $(n+1)$-valued CSP encoding for an instance of SMI. Let I be an instance of SMI with n men and n women, each of whom ranks a subset of the members of the opposite sex in strict order of preference. In I let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ denote the set of men, and $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ denote the set of women. For each man m_i and

woman w_j $(1 \leq i, j \leq n)$ in I, the length of m_i's and w_j's preference list is denoted by l_i^m and l_j^w respectively. Also, for any person z let $PL(z)$ denote the set of people on z's original preference list, and $GS(z)$ the set of people on z's GS-list. For any man $m_i \in \mathcal{M}$ and for any woman $w_j \in PL(m_i)$ we also denote the position of w_j on m_i's original preference list by $rank(m_i, w_j)$, with $rank(w_j, m_i)$ being similarly defined. If $w_j \in \mathcal{W} \backslash PL(m_i)$, then $rank(m_i, w_j)$ and $rank(w_j, m_i)$ are undefined.

We define a CSP encoding J for an instance I of SMI by introducing $2n$ variables: for each man m_i $(1 \leq i \leq n)$ in I we introduce a variable x_i to represent m_i in J; similarly, for each woman w_j $(1 \leq j \leq n)$ in I, we introduce a variable y_j to represent w_j in J. The domain of a variable x_i is denoted by $dom(x_i)$ and can be defined as follows:

$$dom(x_i) = \{rank(m_i, w_j) : w_j \in PL(m_i)\} \cup \{n+1\} = \{1, 2, \ldots, l_i^m\} \cup \{n+1\}$$

Similarly $dom(y_j)$ $(1 \leq j \leq n)$ can be defined for each woman w_j.

An intuitive meaning of the variables is now given. Informally, if $x_i = p$ $(1 \leq p \leq l_i^m)$, then m_i marries the woman w_j such that $rank(m_i, w_j) = p$, and similarly for the case that $y_j = q$ $(1 \leq q \leq l_j^w)$. If $\min(dom(x_i)) \geq p$ (often shortened to $x_i \geq p$) then the pair (m_i, w_j) has been deleted as part of the MEGS algorithm, for all w_j such that $rank(m_i, w_j) < p$. Here "the pair (m_i, w_j) has been deleted" means that m_i has been deleted from w_j's list and w_j from m_i's list. Hence if w_j is the woman such that $rank(m_i, w_j) = p$, then either m_i proposes to w_j during the execution of the MEGS algorithm or the pair (m_i, w_j) will be deleted before the proposal occurs. Similarly if $\min(dom(y_j)) \geq q$ (often shortened to $y_j \geq q$) then the pair (m_i, w_j) has been deleted as part of the WEGS algorithm, for all m_i such that $rank(w_j, m_i) < q$. Hence if m_i is the man such that $rank(w_j, m_i) = q$, then either w_j proposes to m_i during the execution of the WEGS algorithm or the pair (m_i, w_j) will be deleted before the proposal occurs. If $dom(x_i) = \{n+1\}$ then all the women on m_i's list were deleted during an execution of either the MEGS or WEGS algorithm and consequently m_i is unmatched in every stable matching. Similarly if $dom(y_j) = \{n+1\}$ then all the men on w_j's list were deleted during an execution of either the MEGS or WEGS algorithm and consequently w_j is unmatched in every stable matching.

The constraints used for the $(n+1)$-valued encoding are shown in Figure 7.3. Each constraint is present if and only if m_i finds w_j acceptable, where p denotes the rank of w_j on m_i's list and q denotes the rank of m_i on w_j's list.

1.	$x_i \geq p \Rightarrow y_j \leq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$
2.	$y_j \geq q \Rightarrow x_i \leq p$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
3.	$y_j \neq q \Rightarrow x_i \neq p$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$
4.	$x_i \neq p \Rightarrow y_j \neq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$

Figure 7.2: The constraints for the $(n+1)$-valued encoding of an instance SMI.

Interpretations of Constraints 1 and 3 are now given (a similar interpretation can be attached to Constraints 2 and 4, with the roles of the men and women reversed). First consider Constraint 1 – a stability constraint. This constraint ensures that if a man m_i obtains a partner no better than his p^{th}-choice woman w_j, then w_j obtains a partner no worse than her q^{th}-choice man m_i. Now consider Constraint 3 – a consistency constraint. This constraint ensures that if man m_i is removed from w_j's list, then w_j is removed from m_i's list.

We now prove that, given the above CSP encoding J for an SMI instance I, the variables' domains in J after AC propagation correspond to the GS-lists of I. That is, we prove that, after AC is established, for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $p \in dom(x_i)$, and similarly $m_i \in GS(w_j)$ if and only if $q \in dom(y_j)$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$.

The proof is presented using two lemmas. The first lemma shows that the arc consistent domains correspond to a subset of the GS-lists. We show that the deletions made by the MEGS and WEGS algorithms correspond to the deletions made by AC propagation. The second lemma shows that the GS-lists correspond to a subset of the domains remaining after AC propagation. This is shown by considering the domains that correspond to the GS-lists for an instance I, and proving that these domains are arc consistent in the CSP instance J.

Lemma 7.3.1. *For a given i $(1 \leq i \leq n)$, let p be an integer such that $p \in dom(x_i)$ after AC propagation. Then the woman w_j at position p on m_i's preference list belongs to the GS-list of m_i. A similar correspondence holds for the women.*

Proof. The GS-lists are constructed as a result of the deletions made by the MEGS and WEGS algorithms. It is sufficient to prove that the deletions that occur as part of the MEGS and WEGS algorithms correspond to those deletions made to the variables' domains during AC propagation. In the following proof only deletions made by the MEGS algorithm

are considered, a similar argument can be used to prove the result for an execution of the WEGS algorithm.

Let z be the number of proposals during an execution E of the MEGS algorithm. Then we prove the following by induction on z: if proposal z consists of a man m_i proposing to woman w_j, with $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$, then $x_i \geq p$, $y_j \leq q$, and for each man m_k such that $rank(w_j, m_k) = t$ $(q < t \leq l_j^w)$, $x_k \neq s$, where $rank(m_k, w_j) = s$.

First consider the base case where $z = 1$. Then $p = 1$. Since $x_i \geq 1$, propagation of Constraint 1 yields $y_j \leq q$. Then for each t $(q < t \leq l_j^w)$ propagation of Constraint 3 yields $x_k \neq s$, where $rank(w_j, m_k) = t$ and $rank(m_k, w_j) = s$.

Now suppose that $z = c > 1$ and assume that the result holds for $z < c$. We now consider the cases where (i) $p = 1$ and (ii) $p > 1$.

Case (i) The proof is similar to that of the base case.

Case (ii) Let w_r be a woman such that $rank(m_i, w_r) = s < p$. Then w_r has been deleted from m_i's list during the MEGS algorithm. Now suppose $rank(w_r, m_i) = t_1$. Then m_i was deleted from w_r's preference list because she received a proposal from a man m_k whom she prefers to m_i, with $rank(w_r, m_k) = t_2 < t_1$. Since m_k proposed to w_r before the c^{th} proposal, we have by the induction hypothesis that $y_r \leq t_2$. In particular, $y_r \neq t_1$ and thus $x_i \neq s$. But w_r was arbitrary and hence $x_i \neq s$ for $1 \leq s \leq p-1$, therefore $x_i \geq p$. The rest of the proof is similar to that of the base case.

□

Lemma 7.3.2. *For each i $(1 \leq i \leq n)$, define a domain of values $dom(x_i)$ for the variable x_i as follows: if $GS(m_i) = \emptyset$, then $dom(x_i) = \{n+1\}$; otherwise $dom(x_i) = \{rank(m_i, w_j) : w_j \in GS(m_i)\}$. The domain of each y_i $(1 \leq j \leq n)$ is defined analogously. Then the domains so defined are arc consistent in J.*

Proof. Suppose that the variables are assigned the values defined in the statement of the lemma. We are required to show, by considering each constraint shown in Figure 7.3, that the variables' domains are arc consistent in J.

First consider Constraint 1, and suppose that $x_i \geq p$. Then during an execution of the MEGS algorithm either (i) m_i proposed to w_j, or (ii) the pair (m_i, w_j) was deleted, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$. Consider the two cases below:

Case (i) If m_i proposes to w_j during the MEGS algorithm, all men ranked below m_i on w_j's list are removed, i.e. $y_j \leq q$ as required.

Case (ii) If (m_i, w_j) was deleted as part of the MEGS algorithm, then w_j must have received a proposal from a man m_k whom she prefers to m_i, where $rank(w_j, m_k) = t$ $(t < q)$. Therefore the MEGS algorithm deletes all those men m_z from w_j's list such that $rank(w_j, m_z) > t$, i.e. $y_j \leq t < q$ as required.

Next consider Constraint 3. Suppose that $y_j \neq q$. Then as part of the MEGS/WEGS algorithm m_i is deleted from w_j's list, where $rank(w_j, m_i) = q$. To ensure the preference lists are consistent, the MEGS/WEGS algorithm deletes w_j from m_i's list, i.e. $x_i \neq p$, where $rank(m_i, w_j) = p$, as required.

Verifying Constraints 2 and 4 is similar to the above with the roles of the men and women reversed and the MEGS algorithm exchanged for the WEGS algorithm. □

In general, each constraint in this encoding can be revised in constant time. Arc consistency can therefore be establish in $O(ed)$ time [72], where e is the number of constraints, and d is the domain size. For this encoding we have $e = O(n^2)$, and $d = O(n)$. Therefore it can be seen that AC is established in $O(n^3)$ time. The time complexity of this encoding is poorer than that of the Gale-Shapley algorithm. However, the encoding is concise, easy to understand, and naturally models instances of SMI. In addition to this, the GS-lists are also returned after AC has been established.

The two lemmas above, and the fact that AC algorithms find the unique maximal set of arc consistent domains, leads to the following theorem.

Theorem 7.3.3. *Let I be an instance of* SMI*, and let J be a CSP instance obtained using the (n+1)-valued encoding. Then:*

– *AC can be established in $O(n^3)$ time;*

– *the domains remaining after AC propagation in J correspond exactly to the GS-lists.*

We now show that for an instance I of SMI, the CSP encoding J presented in this section can be used to enumerate all the solutions of I in a failure-free manner using AC propagation combined with a value-ordering heuristic.

Theorem 7.3.4. *Let I be an instance of* SMI *and let J be a CSP instance obtained from I using the $(n+1)$-valued encoding. Then the following search process enumerates all solutions in I without repetition and without ever failing due to an inconsistency:*

- *AC is established as a preprocessing step, and after each branching decision including the decision to remove a value from a domain;*
- *if all domains are arc consistent and some variable x_i has two or more values in its domain then search proceeds by setting x_i to the minimum value p in its domain. On backtracking, the value p is removed from the domain of x_i;*
- *when a solution is found, it is reported and backtracking is forced.*

Proof. Let T be the search tree as defined above. We prove by induction on T that each node in T corresponds to an arc consistent CSP instance J', which in turn corresponds to the GS-lists I' for an SMI instance derived from I such that every stable matching in I' is also stable in I. To prove this we first show that it holds for the root node of T. We then assume that the statement is true for an arbitrary branch node u of T, and show that it is true for the two children of u.

The root node of T corresponds to the CSP instance J' with arc consistent domains, where J' is obtained from J by forcing AC propagation. By Theorem 7.3.3, J' corresponds to the GS-lists I' for an SMI instance I. Using the properties of the GS-lists shown in Theorem 1.1.2, every stable matching in I' is also stable in I.

Now suppose that we have reached the branching node u of T described above. By the induction hypothesis we have, associated with u, a CSP instance J' with arc consistent domains. Furthermore, J' corresponds to the GS-lists I' for an SMI instance derived from I such that every stable matching in I' is stable in I. Then since u is a branching node, there exists a variable x_i $(1 \leq i \leq n)$ such that the domain of x_i contains at least two values. Hence in T, u has two children, namely v_1 and v_2, each having an associated CSP instance J'_1 and J'_2 derived from J' in the following way. In J'_1, x_i is assigned the smallest value p (which corresponds to the rank of m_i's most preferable partner in I') in its domain, and in J'_2, p is removed from x_i's domain.

First consider instance J'_1. During AC propagation in J'_1 we consider the revisions made by Constraint 4 when x_i is assigned the value p. Let w_j be the woman such that $rank(m_i, w_j) = p$. Then if $x_i = p$, for each woman w_r where $rank(m_i, w_r) > p$, AC propagation in J'_1 forces $y_r \neq t$, where $rank(w_r, m_i) = t$. After such revisions, J'_1 corresponds to the SMI instance I'_1 obtained from I' by deleting the pairs (m_i, w_r), where $r \neq j$. We now verify that every stable matching M in I'_1 is stable in I'. Suppose that the pair (m, w) blocks M in I'. If $w \in PL(m)$ in I'_1, then (m, w) blocks M in I'_1, therefore

(m, w) was deleted when obtaining I'_1 from I'. Hence $(m, w) = (m_i, w_r)$ for some w_r such that $rank(m_i, w_r) > p$. Let M_0 denote the man-optimal stable matching in I'. Then $(m_i, w_j) \in M_0$, and we can easily verify that M_0 is stable in I'_1. Since the same set of men and women are matched in all stable matchings, the Rural Hospitals Theorem (Theorem 1.2.1) implies that m_i is matched in M. In particular, $(m_i, w_j) \in M$ as w_j is the only woman on m_i's list in I'_1. Hence $(m, w) = (m_i, w_r)$ cannot block M after all, as m_i prefers w_j to w_r. Therefore M is stable in I' and hence by the induction hypothesis is also stable in I. At node v_1, AC is established in J'_1 giving instance J''_1 which we associate with this node. By Theorem 7.3.3, J''_1 corresponds to the GS-lists I''_1 of SMI instance I'_1. The properties of the GS-lists given in Theorem 1.1.2 imply that every stable matching in I''_1 is stable in I'_1, which in turn is stable in I by the preceding argument.

We now consider J'_2. During AC propagation in J'_2, we consider the revisions made when p is removed from the domain of x_i. Let $rank(w_j, m_i) = q$. Propagation of Constraint 4 (or Constraint 1) forces $y_j \neq q$. After this revision J'_2 corresponds to an SMI instance I'_2 obtained from I' by deleting the pair (m_i, w_j). We can now verify that every stable matching M in I'_2 is stable in I'. Suppose that (m, w) blocks M in I'. Then $(m, w) = (m_i, w_j)$, for if this is not the case (m, w) blocks M in I'_2. In I', m_i has a list of length at least 2, by the assumption at the branch node. Hence w_j must also have a list of length at least 2. Therefore w_j is matched in the man-pessimal stable matching for instance I', which is stable in I'_2. Since the same set of men and women are matched in all stable matchings (by the Rural Hospitals Theorem – Theorem 1.2.1), w_j must be matched in every stable matching in I'_2. In particular, w_j is matched in M to a man whom she prefers to m_i. Therefore (m_i, w_j) cannot block M in I'. So M is stable in I', and hence by the induction hypothesis is also stable in I. Now at node v_2, AC is established in J'_2 giving instance J''_2 which we associate with this node. The rest of the proof is similar to that used in for instance J'_1 above. Hence by induction the claim is true for all nodes in T.

We now show that the branching process never fails due to an inconsistency, since setting the variable x_i to p leaves the man-optimal stable matching, while excluding p leaves the man-pessimal stable matching. Also, since all areas of the search space are explored by the branching process, all possible stable matchings for an SMI instance I are listed. Finally we show that there are no repeated solutions. First observe that the leaf nodes of T correspond to the stable matchings in I. Suppose for a contradiction that leaf nodes l_1 and l_2 correspond to the same stable matching M in I. Let b be the lowest

common ancestor of l_1 and l_2 in T. Without loss of generality assume l_1 is reached by taking the path from the left child of b, and l_2 is reached by taking the path from the right child of b. We know that node b corresponds to the GS-lists I' for a particular SMI instance derived from I. Furthermore, there exists a variable x_i which has at least two values in its domain. Thus in I' there exists a man m_i who has a GS-list of size greater than one. Then the left child of b is obtained by forcing m_i to obtain the woman w_j at the head of his list in I', and similarly the right child of b is obtained by removing w_j from m_i's list. So l_1 corresponds to a stable matching M_1 where $(m_i, w_j) \in M_1$, and l_2 corresponds to a stable matching M_2 where $(m_i, w_j) \notin M_2$, i.e. $M_1 \neq M_2$. Therefore each leaf node corresponds to a unique stable matching. □

7.4 4-Valued Encoding

In this section we present a 4-valued CSP encoding for SMI. The encoding is more compact that the $(n+1)$-valued encoding presented in Section 7.3, and as a result it has an improved time complexity. Additionally, after AC propagation the variables' domains correspond to the GS-list of the original instance. Again we consider an SMI instance I with n men and n women, and denote the set of men and women by $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ and $\mathcal{W} = \{w_1, w_2, \ldots, w_n\}$ respectively. The length of a man m_i's preference list is denoted by l_i^m $(1 \leq i \leq n)$, with l_j^w being defined similarly.

Here a CSP encoding J of I consists of λ variables, each of which represents a preference list entry, where λ is the total length of the preference lists in I. In the encoding J we introduce for each man m_i $(1 \leq i \leq n)$ a set of l_i^m variables $x_{i,p}$ $(1 \leq p \leq l_i^m)$, and similarly for each woman w_j $(1 \leq j \leq n)$ a set of l_j^w variables $y_{j,q}$ $(1 \leq q \leq l_j^w)$. As before the domain of a variable z is denoted by $dom(z)$. In the CSP model, each variable $x_{i,p}$ and $y_{j,q}$ has the initial domain $\{0, 1, 2, 3\}$. The proposals and deletions made by the MEGS and WEGS algorithms are expressed by the removal of values from a variable's domain as shown in Figure 7.3.

An intuitive meaning of the variables' values is given in Figure 7.3. The table indicates that deletions carried out by the MEGS and WEGS algorithms applied to I are reflected by the removal of elements from the relevant variables' domains. In particular, removal of the value 2 (respectively 3) from a variable's domain corresponds to a preference list entry being deleted by the MEGS (respectively WEGS) algorithm applied to I. Note

that potentially a given preference list entry could be deleted by both algorithms. Also, if the value 0 is removed from $dom(x_{i,p})$ $(1 \leq i \leq n,\ 1 \leq p \leq l_i^m)$, then either m_i proposes to w_j during the MEGS algorithm (where $rank(m_i, w_j) = p$) or the entry is deleted prior to the proposal occurring. Similarly if the value 0 is removed from $dom(y_{j,q})$ $(1 \leq j \leq n,\ 1 \leq q \leq l_j^w)$, then either w_j proposes to m_i during the WEGS algorithm (where $rank(w_j, m_i) = q$) or the entry is deleted prior to the proposal occurring.

i.	$0 \notin x_{i,p}$	$\Leftrightarrow$	$p = 1$ or $2 \notin x_{i,s}$ $(1 \leq s < p)$;
ii.	$2 \notin x_{i,p}$	$\Leftrightarrow$	man m_i's p^{th}-choice woman has been removed from his list as part of the MEGS algorithm;
iii.	$3 \notin x_{i,p}$	$\Leftrightarrow$	man m_i's p^{th}-choice woman has been removed from his list as part of the WEGS algorithm;
iv.	$0 \notin y_{j,q}$	$\Leftrightarrow$	$q = 1$ or $3 \notin y_{i,t}$ $(1 \leq t < q)$;
v.	$2 \notin y_{j,q}$	$\Leftrightarrow$	woman w_j's q^{th}-choice man has been removed from her list as part of the MEGS algorithm;
vi.	$3 \notin y_{j,q}$	$\Leftrightarrow$	woman w_j's q^{th}-choice man has been removed from her list as part of the WEGS algorithm.

Figure 7.3: Variable definitions for the 4-valued SMI encoding.

The constraints for this encoding are listed in Figure 7.4. For each i and j $(1 \leq i, j \leq n)$, the constraints marked (†) are present if and only if m_i finds w_j acceptable. In the context of Constraints 4 and 10, j is the integer such that $rank(m_i, w_j) = p$; also $q = rank(w_j, m_i)$. In the context of Constraints 5 and 9, i is the integer such that $rank(w_j, m_i) = q$; also $p = rank(m_i, w_j)$. Further, we remark that Constraints 4 and 9 are present only if $q + 1 \leq l_j^w$ and $p + 1 \leq l_i^m$ respectively.

The interpretation of each constraint is now given. First consider Constraint 1. This constraint is used to start the proposal sequence and can be interpreted as each man initially proposing to the first woman on his list during the MEGS algorithm. Constraint 2 says if (m_i, w_r) has been deleted by the MEGS algorithm for all w_r, such that $rank(m_i, w_r) < p$, and (m_i, w_j) has also been deleted, then (m_i, w_r) has been deleted by the MEGS algorithm for all w_r such that $rank(m_i, w_r) \leq p$. Also if a woman's q^{th}-choice partner is deleted during an iteration of the MEGS algorithm she cannot obtain a partner

further down her list, hence her $(q+1)^{th}$-choice partner should also be deleted – this is modelled by Constraint 3. Constraint 4 shows a stability constraint: this is used to ensure that if a man m_i proposes to woman w_j, then w_j obtains a partner no worse than m_i. Lastly Constraint 5 is a consistency constraint: this ensures that if m_i is removed from w_j's list during the MEGS algorithm, then w_j is removed from m_i's list. Constraints 6-10 have a similar meaning with roles of the men and women reversed, and with MEGS replaced by WEGS.

We now prove that given the above CSP encoding J of an SMI instance I, the domains of the variables in J after AC propagation correspond to the GS-lists of I. That is, we show that, after AC is established, for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $\{2,3\} \subseteq dom(x_{i,p})$, and similarly $m_i \in GS(w_j)$ if and only if $\{2,3\} \subseteq dom(y_{j,q})$, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$. First some terminology is introduced.

We define the *GS-domains* for the variables in J as follows. Initially let each variable in J have domain $\{0,1,2,3\}$. Run the MEGS algorithm on instance I. Then use (i), (ii) and (v) in Figure 7.3 to remove 0s and 2s from the appropriate domains. Next run the WEGS algorithm on instance I. Now use (iii), (iv) and (vi) in Figure 7.3 to remove 0s and 3s from the appropriate domains.

As in Section 7.3, we use two lemmas to prove that after AC propagation in the CSP encoding J, obtained from an instance I of SMI, the variables' domains correspond to the

1.	$x_{i,1} > 0$	$(1 \leq i \leq n)$
2.	$(x_{i,p} \neq 2 \wedge x_{i,p} > 0) \Rightarrow x_{i,p+1} > 0$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m - 1)$
3.	$y_{j,q} \neq 2 \Rightarrow y_{j,q+1} \neq 2$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w - 1)$
4.	$x_{i,p} > 0 \Rightarrow y_{j,q+1} \neq 2$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$ (†)
5.	$y_{j,q} \neq 2 \Rightarrow x_{i,p} \neq 2$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$ (†)
6.	$y_{j,1} > 0$	$(1 \leq j \leq n)$
7.	$(y_{j,q} \neq 3 \wedge y_{j,q} > 0) \Rightarrow y_{j,q+1} > 0$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w - 1)$
8.	$x_{i,p} \neq 3 \Rightarrow x_{i,p+1} \neq 3$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m - 1)$
9.	$y_{j,q} > 0 \Rightarrow x_{i,p+1} \neq 3$	$(1 \leq j \leq n, 1 \leq q \leq l_j^w)$ (†)
10.	$x_{i,p} \neq 3 \Rightarrow y_{j,q} \neq 3$	$(1 \leq i \leq n, 1 \leq p \leq l_i^m)$ (†)

Figure 7.4: The constraints for the 4-valued encoding of an instance SMI.

GS-lists of I. The first lemma shows that the variables' domains after AC propagation correspond to a subset of the GS-lists. This is achieved by proving that if a deletion is made as part of the MEGS and WEGS algorithm, then a corresponding deletion is made by the constraints during AC propagation. In particular if a variable, say $x_{i,p}$, contains the values $\{2,3\}$ then neither the MEGS nor the WEGS algorithms have deleted the woman w_j at position p on m_i's list. Hence w_j belongs to m_i's GS-list. A similar correspondence holds for the women. The second lemma is then used to prove that the GS-lists correspond to a subset of the domains after AC propagation. We do this by proving that the GS-domains (corresponding to the GS-lists in I) are arc consistent in J.

Lemma 7.4.1. *For a given i $(1 \leq i \leq n)$, let p be an integer such that $\{2,3\} \subseteq dom(x_{i,p})$ after AC propagation. Then the woman at position p on m_i's preference list belongs to the GS-list of m_i. A similar correspondence holds for the women.*

Proof. The GS-lists are obtained for an instance of SMI through deletions made by the MEGS and WEGS algorithms. We prove that the deletions made by an execution of each algorithm correspond exactly to the deletions made in the domains of the relevant variables during AC propagation. Suppose $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$. Then we prove:

1. (m_i, w_j) is deleted during MEGS algorithm $\Leftrightarrow 2 \notin dom(x_{i,p})$ and $2 \notin dom(y_{j,q})$.
2. (m_i, w_j) is deleted during WEGS algorithm $\Leftrightarrow 3 \notin dom(x_{i,p})$ and $3 \notin dom(y_{j,q})$.

In this proof only the deletions made by the MEGS algorithm are considered, a similar argument can be used to prove the same result for the deletions made by the WEGS algorithm.

It suffices to prove the following by induction on the number of proposals z during an execution E of the MEGS algorithm: if proposal z consists of a man m_i proposing to a woman w_j, with $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$, then $x_{i,p} > 0$, $y_{j,t} \neq 2$ $(q < t \leq l_j^w)$, and for each man m_k such that $rank(w_j, m_k) = t$ $(q < t \leq l_j^w)$, $x_{k,s} \neq 2$ where $rank(m_k, w_j) = s$.

First consider the base case where $z = 1$. Then $p = 1$. By Constraint 1, $x_{i,1} > 0$, and hence using Constraint 4 we obtain $y_{j,q+1} \neq 2$. Also by Constraint 3 it follows that $y_{j,t} \neq 2$ for $(q + 2 \leq t \leq l_j^w)$. Combining these results we obtain $y_{j,t} \neq 2$ for $q < t \leq l_j^w$. Then for each man m_k at position t in w_j's list $(q < t \leq l_j^w)$, the propagation of Constraint 5 ensures $x_{k,s} \neq 2$, where $rank(w_j, m_k) = t$ and $rank(m_k, w_j) = s$.

Now assume $z = c > 1$, and that the result holds for $z < c$. We consider two cases.

Case (i) For $p = 1$ the proof follows from the base case.

Case (ii) Now assume that $p > 1$. Let w_r be a woman such that $rank(m_i, w_r) = s < p$. Then w_r has been deleted from m_i's list during the MEGS algorithm. Now suppose $rank(w_r, m_i) = t_1$. Then w_r was deleted from m_i's list during the MEGS algorithm when w_r received a proposal from a man m_k whom she prefers to m_i, where $rank(w_r, m_k) = t_2 < t_1$. Then since m_k must have proposed to w_r before the c^{th} proposal, by the induction hypothesis it follows that $x_{i,s} \neq 2$. However since w_r was arbitrary $x_{i,v} \neq 2$ for $1 \leq v \leq p-1$. From Constraint 1 we have $x_{i,1} > 0$ and hence propagation of Constraint 2 ($p-1$ times) yields $x_{i,p} > 0$. The rest of the proof is similar to that of the base case.

□

Lemma 7.4.2. *The GS-domains (corresponding to the GS-lists in I) are arc consistent in J.*

Proof. We consider each constraint in turn to show that the GS-domains are arc consistent.

Clearly Constraint 1 is satisfied as $p = 1$ in rule (i), i.e. $x_{i,1} > 0$. Now consider Constraint 4. Suppose that $x_{i,p} > 0$. Then during the execution of the MEGS algorithm either (i) m_i is proposed to his first choice partner w_j, or (ii) the pair (m_i, w_j) has been deleted, where $rank(m_i, w_j) = p$ and $rank(w_j, m_i) = q$. Assuming $q + 1 \leq l_j^w$, we consider the two cases.

Case (i) If m_i proposed to w_j during the execution of the MEGS algorithm then w_j deletes all those men ranked below m_i on her preference list, and in particular $y_{j,q+1} \neq 2$.

Case (ii) If the pair (m_i, w_j) is deleted during the MEGS algorithm then w_j must have received a proposal from a man m_k whom she prefers to m_i. As a result of this all men ranked below m_k on w_j's list, including m_i, are deleted by the MEGS algorithm, and in particular $y_{j,q+1} \neq 2$.

Now suppose that $y_{j,q} \neq 2$. By construction of the GS-domains, the MEGS algorithm deleted the pair (m_i, w_j), where $rank(w_j, m_i) = q$. Hence $x_{i,p} \neq 2$, where $rank(m_i, w_j) = p$, satisfying Constraint 5. Furthermore, a woman only deletes a man m_i from her list

when she receives a proposal from a man whom she prefers to m_i. Hence as $y_{j,q} \neq 2$, w_j received a proposal from a man m_k whom she prefers to m_i, where $rank(w_j, m_k) = t_1 < q$. As a result of this proposal all those ranked below m_k on w_j's list are deleted by the MEGS algorithm. Therefore 2 is removed from the domain of y_{j,t_2} $(t_1 < q < t_2 \leq l_j^w)$, i.e. $y_{j,q+1} \neq 2$. Thus Constraint 3 is satisfied.

Now consider Constraint 2 and suppose that $x_{i,p} \neq 2$ and $x_{i,p} > 0$. Then w_j has been removed from the list of m_i during the MEGS algorithm, where $rank(m_i, w_j) = p$. Then $x_{i,p} > 0$ implies either (i) $p = 1$, or (ii) $x_{i,s} \neq 2$ for $(1 \leq s < p)$. Consider the following two cases below.

> *Case (i)* For $p = 1$, we have $x_{i,1} \neq 2$, and therefore m_i either proposed to his second-choice woman or she was deleted from m_i's list during the MEGS algorithm. In either case $x_{i,2} > 0$ by construction of the GS-domains.
>
> *Case (ii)* From the definition of $x_{i,p} > 0$, it follows that $x_{i,s} \neq 2$ $(1 \leq s < p)$. Then since $x_{i,p} \neq 2$ we have $x_{i,s} \neq 2$ $(1 \leq s \leq p)$, and by construction of the GS-domains $x_{i,p+1} > 0$.

A similar argument can be used to verify that Constraints 6-10 are satisfied. Here the roles of the men and women are reversed and the MEGS algorithm replaced by the WEGS algorithm. □

Using the formula for calculating the complexity of AC shown in Section 7.3, we obtain the following results for the 4-valued encoding: $e = O(n^2)$ and $d = 4$. This means that AC is established in $O(n^2)$ time, and is therefore comparable to the time required by the original Gale-Shapley algorithm. This is also an improvement over the encoding presented in Section 7.3.

The two lemmas above, and the fact that AC algorithms find the unique maximal set of arc consistent domains, lead to the following theorem.

Theorem 7.4.3. *Let I be an instance of* SMI*, and let J be a CSP instance obtained by the 4-valued encoding. Then:*

- *AC can be established in $O(n^2)$ time;*
- *the GS-domains correspond in a precise way to the GS-lists in the following sense: for any i, j $(1 \leq i, j \leq n)$, $w_j \in GS(m_i)$ if and only if $\{2, 3\} \subseteq dom(x_{i,p})$, and*

similarly $m_i \in GS(w_j)$ *if and only if* $\{2,3\} \subseteq dom(y_{j,q})$*, where* $rank(m_i, w_j) = p$ *and* $rank(w_j, m_i) = q$*.*

During the search process of a constraint solver a solution is normally found by reducing the domain of each variable to a single value. However, with the 4-valued encoding each variable's domain need not be reduced to a single value to obtain a solution. Theorems 7.4.3 and 1.1.2(iii) show that we can find a solution to the CSP giving the man-optimal stable matching M_0 without search: for each man $m_i \in \mathcal{M}$, if $\{2,3\} \not\subseteq dom(x_{i,r})$ for each r $(1 \leq r \leq l_i^m)$ then m_i is unmatched in M_0, otherwise we let p be the unique integer such that $dom(x_{i,p}) = \{1,2,3\}$ and define the partner of m_i to be the woman $w_j \in \mathcal{W}$ such that $rank(m_i, w_j) = p$. Considering the y_j variables in a similar way gives the woman-optimal stable matching M_z.

We now show that for an instance I of SMI, the CSP encoding J presented in this section can be used to enumerate all the solutions of I in a failure-free manner using AC propagation combined with a value-ordering heuristic.

Theorem 7.4.4. *Let* I *be an instance of* SMI *and let* J *be a CSP instance obtained from* I *using the* 4*-valued encoding. Then the following search process enumerates all solutions in* I *without repetition and without ever failing due to an inconsistency:*

- *AC is established as a preprocessing step, and after each branching decision including the decision to remove a value from a domain;*
- *if all domains are arc consistent and some variable* $x_{i,s}$ *has* $\{0,1,2,3\}$ *in its domain, then we let* p *be the unique integer such that* $dom(x_{i,p}) = \{1,2,3\}$ *and we choose* p' *to be the minimum integer* $(p < p')$ *such that* $dom(x_{i,p'}) = \{0,1,2,3\}$*;*
- *search proceeds by removing 3 from the domain of* $x_{i,p'}$*. On backtracking, the value 2 is removed from the domain of* $y_{j,q}$*, where* $rank(m_i, w_j) = p$ *and* $rank(w_j, m_i) = q$*;*
- *when a solution is found, it is reported and backtracking is forced.*

Proof. The proof uses a similar argument to that of Theorem 7.3.4. Once again we consider instances J_1' and J_2' at nodes v_1 and v_2 respectively. In J_1', the value 3 is removed from the domain of $x_{i,p'}$, and in J_2', the value 2 is removed from the domain of $y_{j,q}$.

First consider instance J_1'. Then during AC propagation in J_1' we consider the revisions made by Constraints 8 and 10 when 3 is removed from the domain of $x_{i,p'}$. Constraint 8

forces 3 to be removed from the the domain of $x_{i,u}$ ($p' < u \leq l_i^m$) during AC propagation. Let w_r be the woman such that $rank(m_i, w_r) = u$ ($p' < u \leq l_i^m$). Then the consistency constraint, Constraint 10, ensures that if $x_{i,u} \neq 3$, then $y_{r,t} \neq 3$, where $rank(w_r, m_i) = t$. After such revisions, J_1' corresponds to the SMI instance I_1' obtained from I' by deleting the pair (m_i, w_r) where $r \neq j$. A similar argument to that used in the proof of Theorem 7.3.4 can now be used to show that any stable matching in I_1' is stable in I', which in turn is stable in I by the induction hypothesis given in the proof of Theorem 7.3.4. The rest of the proof is similar to that for instance J_1' in Theorem 7.3.4.

Now we consider instance J_2'. Then during AC propagation in J_2' we consider the revision made by Constraint 5 when $y_{j,q} \neq 2$. Here Constraint 5 forces $x_{i,p} \neq 2$ during AC propagation. The revisions in J_2' correspond to an SMI instance I_1' obtained by deleting the pair (m_i, w_j). Again a similar argument to that used in Theorem 7.3.4 can be used to prove that any stable matching in I_2' is stable in I', which is in turn stable in I by the induction hypothesis. The rest of the proof is similar to that for instance J_2' in Theorem 7.3.4.

To prove that the branching process never fails due to an inconsistency, that all solutions are enumerated, and that we obtain no repeat solutions we once again refer to Theorem 7.3.4. □

7.5 Constraint versatility

One of the key motivations for considering constraint-based models of SM is the versatility that they offer. For example, as shown in Sections 7.3.4 and 7.4.4, we can find all stable matchings using AC propagation and the standard search process. This is in contrast to obtaining all stable matchings using a combinatorial algorithmic approach, which requires the implementation of a complex algorithm given in [26]. We can also easily add side-constraints to our models to solve variants of SM that are NP-hard. Below we consider two such extensions.

7.6 Balanced SM problem

In the Balanced Stable Marriage problem we seek to find a matching M where we minimise the maximum cost (as defined in Section 1.1.3) of the men and the women with respect

to M. That is, we require to find a stable matching M that minimizes:

$$\max\{\sum_{m_i \in \mathcal{M}} cost_M(m_i), \sum_{w_j \in \mathcal{W}} cost_M(w_j)\}.$$

Feder [13] showed that Balanced SM is NP-hard. We describe a side-constraint C that acts as an objective function, which is defined in terms of the variables in our model. Most CP toolkits include `minimise` and `maximize` functions as standard. These typically use a branch and bound technique to find a solution. An example of the side-constraint that can be used with the $(n+1)$-valued encoding to obtain an balanced stable matching is shown below.

$$\texttt{minimise}\{\max\{\sum x_i, \sum y_i\}\}$$

7.6.1 Sex-equal SM problem

In the sex-equal stable marriage problem we seek to find a matching M where the total cost of M with respect to the men is as close as possible to the total cost of M with respect to the women. That is, we require to find a stable matching M that minimizes:

$$|\sum_{m_i \in \mathcal{M}} cost_M(m_i) - \sum_{w_j \in \mathcal{W}} cost_M(w_j)|.$$

It was shown by Kato [44] that the problem of finding a sex-equal stable matching is NP-hard given an instance of SM. However with the use of side-constraints we can easily model the sex-equal stable matching problem. An example constraint for the $(n+1)$-valued encoding is shown below.

$$\texttt{minimise}\{|\sum x_i - \sum y_i|\}$$

7.7 Open problem

We finish this chapter with an open problem. Can we find a CP encoding for an instance of SR that uses polynomial space where AC can be established in polynomial time? Also can we use the encoding to quickly identify the non-existence of a stable matching for such an instance or alternatively enumerate all stable matchings in a failure-free manner?

Chapter 8

Constraint Programming and the Hospitals/Residents Problem

8.1 Introduction

In Chapter 7 we presented two CP encodings of SM, each with the property that the domains remaining after AC propagation yield the GS-lists, and a failure-free enumeration strategy may be used to list all stable matchings. The motivation for these models is that, in each case, side constraints can be added easily to solve hard variants of SM. In this chapter we extend the first encoding of Chapter 7 to the HR case, and subsequently to the case of HRT under weak stability. The motivation is again provided by the facility to add side constraints to solve hard variants of HR. However, our ultimate goal is that of modelling and solving, via the second encoding, the NP-hard problem of finding a maximum weakly stable matching, given an HRT instance.

Centralised matching schemes such as the Scottish Foundation Allocation Scheme (SFAS), which allocates graduating medical students to hospital posts in Scotland, allow hospitals' preference lists to contain ties. In the case of SFAS and other similar schemes it is desirable to match as many students as possible. This motivates the problem of finding a maximum weakly stable matching, given an instance of HRT. As discussed in Section 1.1.7, this problem is NP-hard. This naturally leads us to consider if using CSP techniques can help with this problem.

As indicated in Chapter 7, CP models of SM that identify a correspondence with the GS-lists following AC propagation, yield techniques for establishing a failure-free enumeration of all stable matchings. Our constraint model of HR aims to establish a similar

structural correspondence involving the GS-lists for an HR instance. We recall from Section 1.2 that the deletions which occur as part of the resident-oriented (RGS) algorithm result in a reduced set of preference lists called the RGS-lists. Similarly, the deletions that occur as part of the hospital-oriented (HGS) algorithm result in a reduced set of preference lists called the HGS-lists. The intersection of the RGS-lists and the HGS-lists is known as the GS-lists.

In this chapter we present in Section 8.2 a CSP encoding for an instance of HR, together with a correctness proof, and a strategy for a failure-free enumeration of all stable matchings. The HR encoding is then extended to HRT under weak stability in Section 8.3. We described an enumeration strategy for finding all weakly stable matchings for an instance of HRT and hence, with the aid of branch and bound, a maximum weakly stable matching.

8.2 HR encoding

An instance I of HR involves a set $\mathcal{R} = \{r_1, \ldots, r_n\}$ of *residents* and a set $\mathcal{H} = \{h_1, \ldots, h_m\}$ of *hospitals*. Each resident $r_i \in R$ has an *acceptable* set of hospitals $A_i \subseteq H$; moreover r_i ranks A_i in strict order of preference. For each $h_j \in H$, denote by $B_j \subseteq R$ those residents who find h_j acceptable; h_j ranks B_j in strict order of preference. Finally each hospital h_j has capacity c_j $(1 \leq j \leq m)$, indicating the number of *posts* that h_j offers. For each $r_i \in R$, let l_i^r denote the length of r_i's preference list, and for each $h_j \in H$, let l_j^h denote the length of h_j's preference list; we assume that $c_j \leq l_j^h$.

We construct a CSP instance J involving variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_{j,k} : 1 \leq j \leq m \wedge 0 \leq k \leq c_j\}$. Initially the variables' domains are defined as follows:

$$
\begin{aligned}
&dom(x_i) = \{1, 2, \ldots, l_i^r\} \cup \{m+1\} && (1 \leq i \leq n)\\
&dom(y_{j,0}) = \{0\} && (1 \leq j \leq m)\\
&dom(y_{j,k}) = \{k, k+1, \ldots, l_j^h\} \cup \{n+k\} && (1 \leq j \leq m \wedge 1 \leq k \leq c_j)
\end{aligned}
$$

For the x_i variables $(1 \leq i \leq n)$, the value $m+1$ corresponds to the case that r_i's GS-list is empty, whilst the remaining values correspond to the ranks of preference list entries that belong to the GS-lists. A similar meaning applies to the $y_{j,k}$ variables $(1 \leq j \leq m$, $1 \leq k \leq c_j)$, except that the value $n+k$ corresponds to the case that h_j's GS-list contains fewer than k entries.

More specifically, if $x_i \geq p$ $(1 \leq p \leq l_i^r)$, then during the RGS algorithm, r_i applies to

his p^{th}-choice hospital or worse, so that in M_0, either r_i is assigned to such a hospital or is unassigned. Similarly if $x_i \leq p$, then during the HGS algorithm, r_i was offered a post by his p^{th}-choice hospital or better, so that r_i is assigned to such a hospital in M_z.

From the hospitals' point of view, if $y_{j,k} \geq q$ $(1 \leq q \leq l_j^h)$, then during the HGS algorithm, h_j offers its k^{th} post to its q^{th}-choice resident or worse, so that in M_z, either h_j's k^{th} post is filled by such a resident, or is unfilled. Similarly if $y_{j,k} \leq q$, then during the RGS algorithm, some resident r_i applied to h_j's k^{th} post, where $rank(h_j, r_i) \leq q$, so that h_j's k^{th} post is filled by r_i or better in M_0.

The constraints in J are given in Figure 8.1. In the context of lines 2-5, p denotes the rank of h_j in r_i's list and q denotes the rank of r_i in h_j's list. An interpretation of the constraints is now given. Constraint 1 ensures that h_j's filled posts are occupied by residents in preference order, and that if post $k-1$ is unfilled then so is post k. Constraint 2 states that if h_j's k^{th} post is filled by a resident no better than r_i or is unfilled, then r_i must be assigned to a hospital no worse than h_j. Constraints 3 and 5 reflect the consistency of deletions carried out by the HGS and RGS algorithms respectively (i.e. if h_j is deleted from r_i's list, then r_i is deleted from h_j's list, and vice versa). Finally Constraint 4 states that if r_i is assigned to a hospital no better than h_j or is unassigned, and h_j's first $k-1$ posts are filled by residents better than r_i, then h_j's k^{th} post must be filled by a resident at least as good as r_i.

1.	$y_{j,k} < y_{j,k+1}$	$(1 \leq j \leq m, 1 \leq k \leq c_j - 1)$
2.	$y_{j,k} \geq q \Rightarrow x_i \leq p$	$(1 \leq j \leq m, 1 \leq k \leq c_j, 1 \leq q \leq l_j^h)$
3.	$x_i \neq p \Rightarrow y_{j,k} \neq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^r, 1 \leq k \leq c_j)$
4.	$(x_i \geq p \wedge y_{j,k-1} < q) \Rightarrow y_{j,k} \leq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^r, 1 \leq k \leq c_j)$
5.	$y_{j,c_j} < q \Rightarrow x_i \neq p$	$(1 \leq j \leq m, c_j \leq q \leq l_j^h)$

Figure 8.1: Constraints for the CSP encoding for an HR instance.

It turns out that establishing AC in J yields a set of domains that correspond to the GS-lists in I. We prove this using three lemmas. The first two lemmas show that the arc consistent domains correspond to subsets of the HGS-lists and the RGS-lists respectively. The third lemma shows that the GS-lists correspond to arc consistent domains. In the following proofs we use the terminology "r_i applies (or is assigned) to h_j's k^{th} post" to mean that h_j is currently assigned $k-1$ residents whom it prefers to r_i, and prefers r_i

to any of its other assignees. Also given a stable matching M, and given any k such that $1 \leq k \leq |M(h_j)|$, we denote the resident who is assigned to h_j's k^{th} post in M by $M_k(h_j)$.

Lemma 8.2.1. *(i) For a given j $(1 \leq j \leq m)$, let q be an integer $(q \leq n)$ such that $q \in dom(y_{j,k})$ for some k $(1 \leq k \leq c_j)$ after AC propagation. Then the resident r_i at position q on hospital h_j's preference list belongs to the HGS-list of h_j.*
(ii) For a given i $(1 \leq i \leq n)$, let p be an integer $(p \leq m)$ such that $p \in dom(x_i)$ after AC propagation. Then hospital h_j at position p on resident r_i's preference lists belongs to the HGS-list of r_i.

Proof. The HGS-lists are constructed as a result of the deletions made by the HGS algorithm. We show that the corresponding deletions are made to the variables' domains during AC propagation.

The following proof uses induction on the number of iterations of the main loop during an execution E of the HGS algorithm to show that, if iteration z consists of some hospital h_j offering some resident r_i its k^{th} post, then $x_i \leq p$, proving (ii) above, $y_{j,k} \geq q$, and $y_{v,b} \neq t$ $(1 \leq b \leq c_v)$, proving (i) above, for each hospital h_v such that $rank(r_i, h_v) > p$, where $t = rank(h_v, r_i)$, $p = rank(r_i, h_j)$ and $q = rank(h_j, r_i)$.

Let z be the number of iterations of the main loop during an execution E of the HGS algorithm. We prove the following by induction on z: if iteration z consists of hospital h_j offering r_i its k^{th} post, then $x_i \leq p$, $y_{j,k} \geq q$, and for each h_v such that $rank(r_i, h_v) = s > p$, it follows that $y_{v,b} \neq t$ $(1 \leq b \leq c_v)$, where $t = rank(h_v, r_i)$, $p = rank(r_i, h_j)$ and $q = rank(h_j, r_i)$.

First consider the case where $z = 1$. On the first iteration of the main loop, hospital h_j offers resident r_i its k^{th} post, where $q = 1 = rank(h_j, r_i)$ and $p = rank(r_i, h_j)$. By domain initialisations, $y_{j,k} \geq 1$ $(1 \leq k \leq c_j)$, therefore propagation of Constraint 2 yields $x_i \leq p$. Finally, consider each hospital h_v where $rank(r_i, h_v) > p$. By propagation of Constraint 3 we obtain $y_{v,b} \neq t$ $(1 \leq b \leq c_v)$, where $t = rank(h_v, r_i)$, giving us the result as required.

Now suppose that $z = d > 1$, and that the result holds for $z < d$. We consider the two cases where $k = 1$ and $k > 1$.

Case (i) Suppose $k = 1$. Then consider the two subcases where $q = 1$ and $q > 1$.

Subcase (a) For $q = 1$ the proof is similar to the base case.

Subcase (b) Now suppose that $q > 1$. Then h_j is offering its 1^{st} post to the resident r_i at position q. Let r_{u_1} be a resident such that $rank(h_j, r_{u_1}) = t_1 < q$.

Then r_{u_1} has been deleted from h_j's list. Now suppose $rank(r_{u_1}, h_j) = s_1$. Then r_{u_1} must have received an offer from some hospital h_v whom he prefers to h_j, where $rank(r_{u_1}, h_v) = s_2 < s_1$ and $rank(h_v, r_{u_1}) = t_2$. Therefore h_v offered its a^{th} post, for some a $(1 \leq a \leq c_v)$, to r_{u_1} before the d^{th} iteration. So by the induction hypothesis we have $y_{v,a} \geq t_2$, $x_{u_1} \leq s_2$ and $y_{j,k} \neq t_1$ $(1 \leq k \leq c_j)$, where $t_1 = rank(h_j, r_{u_1})$. However r_{u_1} was arbitrary, and since $y_{j,k} \neq t_1$ for all t_1 $(1 \leq t_1 \leq q-1)$, $y_{j,k} \geq q$. The rest of the proof is similar to the base case.

Case (ii) Now suppose $k > 1$. Let r_{u_1} be the resident to which h_j offered its $(k-1)^{th}$ post. This occurred during the g^{th} iteration for some $g < d$. Suppose that $rank(h_j, r_{u_1}) = t_1 < q$. Then by the induction hypothesis we have $y_{j,k-1} \geq t_1$, therefore propagation of Constraint 1 yields:

$$y_{j,k} \geq t_1 + 1 \tag{8.1}$$

Now consider the two subcases where $q = t_1 + 1$ and $q > t_1 + 1$.

Subcase (a) If $q = t_1 + 1$, then the rest of the proof is similar to the base case.

Subcase (b) Now suppose $q > t_1+1$. Let r_{u_2} be a resident such that $rank(h_j, r_{u_2})$ $= t_2$ $(t_1+1 \leq t_2 \leq q-1)$. Then r_{u_2} has been deleted from h_j's list. Now suppose $rank(r_{u_2}, h_j) = s_2$. Then r_{u_2} must have received an offer from some hospital h_v whom he prefers to h_j, where $rank(r_{u_2}, h_v) = s_3 < s_2$ and $rank(h_v, r_{u_2}) = t_3$. Therefore h_v offered its a^{th} post to r_{u_2} for some a $(1 \leq a \leq c_v)$ before the d^{th} iteration. By the induction hypothesis, $y_{v,a} \geq t_3$, $x_{u_2} \leq s_3$ and $y_{j,k} \neq t_2$ $(1 \leq k \leq c_j)$. However, r_{u_2} was arbitrary, so:

$$y_{j,k} \neq t_2 \text{ for } t_1 + 1 \leq t_2 \leq q - 1 \tag{8.2}$$

Thus from Inequalities 8.1 and 8.2, we have $y_{j,k} \geq q$. The rest of the proof is similar to the base case.

□

Lemma 8.2.2. *(i) For a given i $(1 \leq i \leq n)$, let p be an integer $(p \leq m)$ such that $p \in dom(x_i)$ after AC propagation. Then hospital h_j at position p on resident r_i's preference lists belongs to the RGS-list of r_i.*

(ii) For a given j $(1 \leq j \leq m)$, let q be an integer $(q \leq m)$ such that $q \in dom(y_{j,k})$ for

some k $(1 \leq k \leq c_j)$ after AC propagation. Then the resident r_i at position q on h_j's preference list belongs to the RGS-list of h_j.

Proof. The RGS-lists are constructed as a result of the deletions made by the RGS algorithm. We show that the corresponding deletions are made to the variables' domains during AC propagation.

The following proof uses induction on the number of iterations of the main loop during an execution E of the RGS algorithm to show that if hospital h_j becomes full, and residents $r_{i_1}, \ldots, r_{i_{c_j}}$ are assigned to h_j, then $y_{j,k} \leq q_k$ $(1 \leq k \leq c_j)$, where $rank(h_j, r_{i_k}) = q_k$ and $0 < q_1 < q_2 < \cdots < q_{c_j}$ $(1 \leq k \leq c_j)$. We use this result to show that (ii) above is satisfied, and propagation of Constraint 5 shows that (i) is also satisfied.

Let z be the number of iterations of the main loop during an execution E of the RGS algorithm. We prove the following by induction on z: if iteration z involves some resident r_i applying to a hospital h_j and at the end of this iteration, residents $r_{i_1}, \ldots, r_{i_{d_j}}$ are assigned to h_j, where $d_j \leq c_j$, then $y_{j,k} \leq q_k$ $(1 \leq k \leq d_j)$, where $q_k = rank(h_j, r_i)$ and $0 < q_1 < q_2 < \cdots < q_{d_j}$, and $x_{i_k} \geq p_{i_k}$, where $p_{i_k} = rank(r_{i_k}, h_j)$.

First consider the base case where $z = 1$. Then during the first iteration of the main loop, some resident r_i applies for the first post at hospital h_j, where $p = 1 = rank(r_i, h_j)$, and $q = rank(h_j, r_i)$. Thus, $x_i \geq p$ (by construction of the x_i variables' domains), and $y_{j,k-1} < q$, since $k = 1$ and $y_{j,0} = 0$ by definition. Therefore propagation of Constraint 4 yields $y_{j,k} \leq q$ as required.

Now suppose that $z = d > 1$, and that the result holds for $z < d$. Then during the d^{th} iteration resident r_i applies to hospital h_j, and we let d_j denote the number of residents assigned to h_j just before r_i applies, where $d_j \geq 0$ and $rank(r_i, h_j) = p$. We consider the cases where $p = 1$ and $p > 1$.

> *Case (i)* Suppose $p = 1$, and therefore $x_i \geq p$ by initialisation of the variables' domains. We first note that if $d_j = 0$, the proof is similar to the base case. Now suppose that $d_j \geq 1$. Then there exists an iteration $g < d$ of the main loop where some resident applies to h_j, such that iteration g' of the main loop, for $g < g' < d$, does not involve a resident applying to h_j. Then at the end of the g^{th} iteration, residents $r_{i_1}, \ldots, r_{i_{d_j}}$ are assigned to h_j, and by the induction hypotheses $y_{j,k} \leq q_k$ $(1 \leq k \leq d_j)$, where $q_k = rank(h_j, r_{i_k})$ and $0 < q_1 < q_2 < \cdots < q_{d_j}$. Now consider the two subcases where $d_j < c_j$ and $d_j = c_j$.

Subcase (a) Suppose $d_j < c_j$. If $q > q_{d_j}$, then at the d^{th} iteration, r_i is assigned to h_j's $(d_j + 1)^{th}$ post. From above we have that $y_{j,d_j} \leq q_{d_j} < q$ and since $x_i \geq p$, propagation of Constraint 4 yields $y_{j,d_j+1} \leq q$, as required. Now suppose that $q < q_{d_j}$. Then there exists b $(1 \leq b \leq d_j)$ such that $q_{b-1} < q < q_b$ (for convenience we define $q_0 = 0$). Therefore at the d^{th} iteration, r_i becomes assigned to h_j's b^{th} post. Thus from above $y_{j,b-1} \leq q_{b-1} < q$, and since $x_i \geq p$, propagation of Constraint 4 yields $y_{j,b} \leq q$. Furthermore, $y_{j,b} \leq q < q_b$, and by the induction hypothesis $x_{i_b} \geq p_{i_b}$, where $p_{i_b} = rank(r_{i_b}, h_j)$. Again propagation of Constraint 4 yields $y_{j,b+1} \leq q_b$. Continuing in this manner we obtain $y_{j,k} \leq q_{k-1}$, for all k $(b + 1 \leq k \leq d_j + 1)$, as required.

Subcase (b) Now suppose that $d_j = c_j$. Hence when r_i applies to h_j at the d^{th} iteration, h_j becomes over-subscribed. During the g^{th} iteration of the main loop, h_j must have become full. When this happens as part of the RGS algorithm, the worst assigned resident is identified, and all its successors on h_j's list are deleted. It follows that $q < q_{c_j}$. The remainder of the proof is similar to that used in Subcase (a) when $q < q_{d_j}$.

Case (ii) Now suppose that $p > 1$. Let h_v be a hospital such that $rank(r_i, h_v) = s_1 < p$. Now suppose $rank(h_v, r_i) = t_1$. Then h_v has been deleted from r_i's list during the execution of the RGS algorithm. This can only happen if h_v became full at the g^{th} iteration $(g < d)$ of the RGS algorithm. At this point the worst resident r_u assigned to h_v is identified, where $rank(h_v, r_u) = t_2 < t_1$. Since h_v is full, it follows that r_u is assigned to h_v's c_v^{th} post at the end of the g^{th} iteration, hence by the induction hypothesis $y_{v,c_v} \leq t_2 < t_1$. Thus propagation of Constraint 5 yields $x_i \neq s_1$. But h_v was arbitrary and hence $x_i \neq s_2$ for all s_2 such that $1 \leq s_2 \leq p - 1$, so $x_i \geq p$. The rest of the proof is similar to that used in Case (i).

□

We now prove that the domains corresponding to the GS-lists of an instance I of HR are arc consistent. First we introduce some new notation.

For each j $(1 \leq j \leq m)$ we define the set S_j as follows:

$$S_j \quad = \quad \{rank(h_j, r_i) : r_i \in GS(h_j)\}.$$

Let d_j denote the number of residents assigned to hospital h_j in any stable matching in I. Then define $q_{j,k} = rank(h_j, M_{z_k}(h_j))$ and $t_{j,k} = rank(h_j, M_{0_k}(h_j))$ for each k $(1 \leq k \leq d_j)$, where M_z and M_0 are the hospital-optimal and resident-optimal stable matchings respectively. The *GS-domains* for the variables in J are defined as follows:

$$dom(x_i) = \begin{cases} \{rank(r_i, h_j) : h_j \in GS(r_i)\}, & \text{if } GS(r_i) \neq \emptyset \\ \{m+1\}, & \text{otherwise.} \end{cases}$$

$$dom(y_{j,k}) = \begin{cases} \{q \in S_j : q_{j,k} \leq q \leq t_{j,k}\}, & \text{if } 1 \leq k \leq d_j \\ \{n+k\}, & \text{if } d_j + 1 \leq k \leq c_j. \end{cases}$$

Lemma 8.2.3. *The GS-domains are arc consistent in J.*

Proof. First consider Constraint 1, and suppose that $k < d_j$. Then $\min(dom(y_{j,k+1})) = q_{j,k+1} > q_{j,k} = \min(dom(y_{j,k}))$. Now suppose that $d_j \leq k < c_j$. Then $y_{j,k+1} = n+k+1 > n+k = y_{j,k}$.

Now consider Constraint 2 and once again suppose that $y_{j,k} \geq q$. Then during the execution of the HGS algorithm either (i) hospital h_j offered the resident r_i at position q its a^{th} post for some a $(1 \leq a \leq c_j)$, or (ii) the pair (r_i, h_j) was deleted, where $p = rank(r_i, h_j)$ and $q = rank(h_j, r_i)$. Now consider the two cases below:

> *Case (i)* If h_j offered resident r_i its a^{th} post as part of the HGS algorithm, then r_i deletes all those hospitals ranked lower than h_j on his preference list, i.e. $x_i \leq p$.
>
> *Case (ii)* If the pair (r_i, h_j) is deleted, then resident r_i must have received an offer from a hospital h_v which he prefers to h_j, where $rank(r_i, h_v) = s < p$. Since r_i deletes all hospitals in his preference list ranked below h_v when he receives such an offer, $x_i \leq s$. In particular $x_i \leq p$.

Consider Constraint 3, and suppose that $x_i \neq p$. Then hospital h_j has been deleted from resident r_i's preference list, where $p = rank(r_i, h_j)$, by either the RGS or HGS algorithm. The same algorithm ensures that the preference lists are consistent and removes r_i from the list of h_j, i.e. $y_{j,k} \neq q$ $(1 \leq k \leq c_j)$, where $q = rank(h_j, r_i)$.

For Constraint 4, suppose that $x_i \geq p$ and $y_{j,k-1} < q$, where $p = rank(r_i, h_j)$ and $q = rank(h_j, r_i)$. If $t_{j,k} \leq q$, then $y_{j,k} \leq q$, since $y_{j,k} \leq t_{j,k}$ by definition, as required. Now suppose for a contradiction that $t_{j,k} > q$. Then $t_{j,a} < q$, for $1 \leq a \leq k-1$, and $t_{j,a} > q$, for $k \leq a \leq d_j$. Hence r_i is not assigned to h_j in M_0, so (r_i, h_j) was deleted as part of

the RGS algorithm, since either r_i is unmatched in M_0 or prefers h_j to $M_0(r_i)$. As (r_i, h_j) has been deleted, h_j must have become full during an execution of RGS algorithm with residents that it prefers to r_i. Thus h_j is full in M_0, so $d_j = c_j$, and moreover $t_{j,c_j} < q$. This is a contradiction to the earlier assumption that $t_{j,k} > q$, hence $t_{j,k} \leq q$.

Finally consider Constraint 5 and suppose that $y_{j,c_j} < q$. Then during an execution of the RGS or HGS algorithm, hospital h_j removed the resident r_i from its list where $p = rank(r_i, h_j)$ and $q = rank(h_j, r_i)$. To ensure consistency the same algorithm then deletes h_j from r_i's list, i.e. $x_i \neq p$. □

In general, following AC propagation in J, matchings M_0 and M_z may be obtained as follows. Let $x_i \in X$. If $x_i = m+1$, resident r_i is unassigned in both M_0 and M_z. Otherwise, in M_0 (respectively M_z), r_i is assigned to the hospital h_j such that $rank(r_i, h_j) = p$, where $p = \min(dom(x_i))$ (respectively $p = \max(dom(x_i))$).

The constraints for this encoding can be revised in $O(1)$ time. Arc consistency can therefore be established in $O(ed)$ time [72], where e is the number of constraints, and d is the domain size. For this encoding we have $e = O(nmC)$, where $C = \max\{c_j : h_j \in H\}$, $d = O(n+m)$, and therefore AC can be established in $O(nmC(n+m)) = O((n+m)^3 C)$.

The two lemmas and time complexity discussion above, in addition to the fact that AC algorithms find the unique maximal set of arc consistent domains, leads to the following theorem.

Theorem 8.2.4. *Let I be an instance of* HR*, and let J be a CSP instance obtained by the encoding in Figure 8.1. Then:*

– *AC can be established in $O((n+m)^3 C)$ time;*

– *the domains remaining after AC propagation in J correspond exactly to the GS-lists.*

We now show that, for an instance I of HR, the encoding presented above can be used to enumerate all the solutions of I in a failure-free manner using AC propagation with a value-ordering heuristic. We note that if x_i has at least two values in its domain then $m+1$ does not belong to the domain. For, suppose that j ($j \leq m$) and $m+1$ belong to $dom(x_i)$ after AC propagation. Then r_i became assigned to some hospital during the RGS algorithm, so r_i is matched in the resident-optimal stable matching. Hence by the Rural Hospitals Theorem (Theorem 1.2.1) r_i is matched in the hospital-optimal stable matching. Thus r_i received a proposal during the HGS algorithm, hence $m+1$ must have been removed from x_i's domain during AC propagation.

Theorem 8.2.5. *Let I be an instance of* HR *and let J be a CSP instance obtained from I using the encoding in Figure 8.1. Then the following search process enumerates all solutions in I without repetition and without ever failing due to an inconsistency:*

- *AC is established as a preprocessing step, and after each branching decision including the decision to remove a value from a domain;*
- *if all domains are arc consistent and some variable x_i has two or more values in its domain then search proceeds by setting x_i to the minimum value p in its domain. On backtracking, the value p is removed from the domain of x_i;*
- *when a solution is found, it is reported and backtracking is forced.*

Proof. Let T be the search tree as defined above. We prove by induction on T that each node in T corresponds to an arc consistent CSP instance J', which in turn corresponds to the GS-lists I' for an HR instance derived from I such that every stable matching in I' is also stable in I. To prove this we first show that it holds for the root node of T. Then we assume it is true at any branch node u in T and show that it is true for each child of u.

The root node of T corresponds to the CSP instance J' with arc consistent domains, where J' is obtained from J by forcing AC propagation. Therefore by Theorem 8.2.4, J' corresponds to the GS-lists I' for the HR instance I. Using the properties of the GS-lists shown in Theorem 1.1.2, every stable matching in I' is also stable in I.

Now suppose that we have reached a branching node u of T. By the induction hypothesis we have, associated with u, a CSP instance J' with arc consistent domains. Furthermore, J' corresponds to the GS-lists I' for an HR instance derived from I, such that every stable matching in I' is stable in I. Then since u is a branching node, there exists a variable x_i $(1 \leq i \leq n)$ such that the domain of x_i contains at least two values. Hence in T, u has two children, namely v_1 and v_2, each having an associated CSP instance J'_1 and J'_2 derived from J' in the following way. In J'_1, x_i is assigned the smallest value p (which corresponds to the rank of r_i's best stable partner h_j in I') in its domain, and in J'_2, p is removed from x_i's domain.

First consider instance J'_1. During AC propagation in J'_1, we consider the revisions made by Constraint 3 when x_i is assigned the value p. Suppose $x_i = p$. Let h_v be a hospital such that $rank(r_i, h_v) > p$. Then AC propagation in J'_1 forces $y_{v,k} \neq t$ $(1 \leq k \leq c_v)$, where $t = rank(h_v, r_i)$. After such revisions, J'_1 corresponds to an HR instance I'_1 obtained from I' by deleting the pairs (r_i, h_v), where $v \neq j$. We now verify that every stable matching

M in I_1' is stable in I'. Suppose that the pair (r, h) blocks M in I'. If $h \in PL(r)$ in I_1', then (r, h) must block M in I_1', hence (r, h) must have been deleted in I_1'. Hence $(r, h) = (r_i, h_v)$ for some v such that $rank(r_i, h_v) > p$. Let M_0 denote the resident-optimal stable matching in I'. In M_0 each resident obtains his best possible stable partner in I'. It can be easily verified that M_0 is also stable in I_1', hence $(r_i, h_j) \in M_0$. By the Rural Hospitals Theorem (Theorem 1.2.1), the same set of hospitals and residents are matched in every stable matching, therefore r_i is matched in M. In particular, $(r_i, h_j) \in M$, as h_j is the only hospital on r_i's list in I_1'. Thus (r, h) cannot block in M in I' after all, as r_i prefers h_j to h_v (as h_j is the hospital at the head of r_i's list in I'). Therefore M is stable in I', and hence by the induction hypothesis is also stable in I. Therefore at node v_1, AC is established in J_1' giving instance J_1'' which we associate with this node. By Theorem 8.2.4, J_1'' corresponds to the GS-lists I_1'' of HR instance I_1'. Using the properties of the GS-lists given in Theorem 1.1.2, we have that every stable matching in I_1'' is stable in I_1', which in turn is stable in I by the preceding argument.

We now consider J_2'. Let $q = rank(h_j, r_i)$. Then during AC propagation in J_2', we consider the revisions made when p is removed from the domain of x_i. Propagation of Constraint 3 forces $y_{j,k} \neq q$ $(1 \leq k \leq c_j)$. Then propagation of Constraint 4 gives $y_{j,1} \leq q$. However $y_{j,1} \neq q$, so $y_{j,1} < q$. Hence further propagation of Constraint 4 yields $y_{j,2} \leq q$, and hence $y_{j,2} < q$. Continuing in this way we obtain $y_{j,k} < q$ for $1 \leq k \leq c_j$. Hence after such revisions J_2' corresponds to an HR instance I_2' obtained from I' by deleting the pairs (r_u, h_j), where $rank(h_j, r_u) \geq q$. We now verify that every stable matching M in I_2' is stable in I'. Suppose that (r, h) blocks M in I'. Then $(r, h) = (r_u, h_j)$ for some r_u such that $rank(h_j, r_u) \geq q$, for otherwise (r_u, h_j) blocks M in I_2'. Consider M_z, the resident-pessimal stable matching in I', where each resident obtains his worst possible stable assignment in I'. We can easily verify that M_z is stable in I_2'. Thus in M_z, r_i is assigned to the hospital at the end of his list, and since r_i's list contains at least two entries (by assumption at the branch node) $M_z(r_i) \neq h_j$. Therefore h_j must be full in M_z with residents it prefers to r_i, by the stability of M_z in I_2'. Hence by the Rural Hospitals Theorem (Theorem 1.2.1) applied to I_2', h_j is full in M. However h_j must be full with assignees whom it prefers to r_u by construction of I_2'. Hence M is stable in I', and by the induction hypothesis is stable in I. Now at node v_2, AC is established in J_2' giving instance J_2'' which we associate with this node. The rest of the proof is similar to that used for instance J_1' above. Hence by induction the claim is true for all nodes in T.

We can now see that the branching process never fails due to an inconsistency, as setting the variable x_i to p leaves the resident-optimal stable matching, while excluding p always leaves the resident-pessimal stable matching. Also, since we explore all areas of the search space with the branching process, all possible stable matchings for an HR instance I are listed.

We can also prove that there are no repeated solutions. First observe that the leaf nodes of T correspond to the stable matchings in I. Suppose for a contradiction that leaf nodes l_1 and l_2 correspond to the same stable matching M in I. Then let b be the lowest common ancestor of l_1 and l_2 in T. Without loss of generality, assume l_1 is reached by taking the path from the left child of b, and l_2 is reached by taking the path from the right child of b. We know that node b corresponds to the GS-lists I' for a particular HR instance derived from I, such that variable x_i has at least two values in its domain. This means that in I' there exists some resident r_i who has a GS-list of size greater than one. Then the left child of b is obtained by forcing r_i to be assigned to hospital h_j at the tail of his list in I', and similarly the right child of b is obtained by removing h_j from r_i's list. So l_1 corresponds to a stable matching M_1 where $(r_i, h_j) \in M_1$, and l_2 corresponds to a stable matching M_2 where $(r_i, h_j) \notin M_2$, i.e. $M_1 \neq M_2$. Therefore we have that each leaf node corresponds to a unique stable matching. □

8.3 HRT encoding

In this section we present a CSP encoding for an instance I of HRT. We use the same notation for the components of I as defined in Section 8.2 for an instance of HR. However, we define *pos* used throughout this section. For all $r_i \in R$,] let $pos(r_i, \cdot) : A_i \to \{1, 2, \ldots, l_i^r\}$ be a bijective function. Similarly[for all $h_j \in H$, let $pos(h_j, \cdot) : B_j \to \{1, 2, \ldots, l_j^h\}$ be a bijective function such that:

$$\forall\, r_i \in R \cdot \forall\, h_j, h_k \in A_i \quad \cdot \quad rank(r_i, h_j) < rank(r_i, h_k) \Rightarrow pos(r_i, h_j) < pos(r_i, h_k)$$
$$\forall\, h_j \in H \cdot \forall\, r_i, r_k \in B_j \quad \cdot \quad rank(h_j, r_i) < rank(h_j, r_k) \Rightarrow pos(h_j, r_i) < pos(h_j, r_k).$$

If $pos(r_i, h_j) = p$ then we say that h_j occurs at *position* p on r_i's list, and similarly if $pos(h_j, r_i) = q$ we say that r_i occurs at position q on h_j's list. We also define a bijective function

$$pref(r_i, \cdot) : \{1, 2, \ldots, l_i^r\} \to A_i,$$

such that $pref(r_i, k) = h_j$ if and only if $pos(r_i, h_j) = k$. Similarly define a bijective function

$$pref(h_j, \cdot) : \{1, 2, \ldots, l_j^h\} \rightarrow B_j,$$

such that $pref(h_j, k) = r_i$ if and only if $pos(h_j, r_i) = k$. Then given any $p \in \{1, 2, \ldots, l_i^r\}$, for some resident r_i, we define:

$$\begin{aligned} p^+ &= \max\{k \in \{1, 2, \ldots, l_i^r\} \quad : \quad rank(r_i, pref(r_i, k)) = rank(r_i, pref(r_i, p))\} \\ p^- &= \min\{k \in \{1, 2, \ldots, l_i^r\} \quad : \quad rank(r_i, pref(r_i, k)) = rank(r_i, pref(r_i, p))\}. \end{aligned}$$

Also given any $q \in \{1, 2, \ldots, l_j^h\}$ we define:

$$\begin{aligned} q^+ &= \max\{k \in \{1, 2, \ldots, l_j^h\} \quad : \quad rank(h_j, pref(h_j, k)) = rank(h_j, pref(h_j, q))\} \\ q^- &= \min\{k \in \{1, 2, \ldots, l_j^h\} \quad : \quad rank(h_j, pref(h_j, k)) = rank(h_j, pref(h_j, q))\}. \end{aligned}$$

We now give the intuition behind p^+ and p^-, with q^+ and q^- being similarly defined. Then p^+, with respect to some resident r_i, denotes the position of the last hospital in the same tie as the hospital in position p on r_i's list. Similarly p^-, with respect to some resident r_i, denotes the position of the first hospital in the same tie as the hospital in position p. Both q^+ and q^- are analogously defined for the hospitals.

Consider the preference list of r_1 shown in Figure 8.2. We illustrate the usage of the above notation with respect to r_1's preference list. In r_1's list we have $pos(r_1, h_4) = 3$, $rank(r_1, h_4) = 2$, and $pref(r_1, 3) = h_4$. Also for $p = 5$ (i.e. $pref(r_1, p) = h_2$) we have that $p^+ = 6$ and $p^- = 4$.

$$r_1 : \quad (h_1 \quad h_5) \quad h_4 \quad (h_3 \quad h_2 \quad h_6)$$

Figure 8.2: r_1's preference list.

We construct a CSP instance J involving variables $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_{j,k} : 1 \leq j \leq m \wedge 0 \leq k \leq c_j\}$. Initially the variables' domains are defined as follows:

$$\begin{aligned} dom(x_i) &= \{1, 2, \ldots, l_i^r\} \cup \{m + 1\} && (1 \leq i \leq n) \\ dom(y_{j,0}) &= \{0\} && (1 \leq j \leq m) \\ dom(y_{j,k}) &= \{k, k + 1, \ldots, l_j^h\} \cup \{n + k\} && (1 \leq j \leq m \wedge 1 \leq k \leq c_j). \end{aligned}$$

An important difference between the constraints for the HR model, and that of the model for HRT, is that the values in the variables' domains for the HRT model correspond to positions on an agent's preference list as opposed to ranks.

1.	$y_{j,k} < y_{j,k+1}$	$(1 \leq j \leq m, 1 \leq k \leq c_j - 1)$
2.	$x_i > p^+ \Rightarrow y_{j,k} \leq q^+$	$(1 \leq i \leq n, 1 \leq p \leq l_i^r, 1 \leq k \leq c_j)$
3.	$x_i \neq p \Rightarrow y_{j,k} \neq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^r, 1 \leq k \leq c_j)$
4.	$(x_i = p \wedge y_{j,k-1} < q) \Rightarrow y_{j,k} \leq q$	$(1 \leq i \leq n, 1 \leq p \leq l_i^r, 1 \leq k \leq c_j)$
5.	$y_{j,c_j} < q \Rightarrow x_i \neq p$	$(1 \leq j \leq m, c_j \leq q \leq l_j^h)$

Figure 8.3: Constraints for the CSP model of HRT instance.

The constraints in J are given in Figure 8.3. In the context of Constraint 2, p denotes the position of some hospital h_v on r_i's preference list, where $1 \leq p \leq l_i^r$. Then p^+ denotes the position of the last hospital in the same tie as h_v (possibly $p = p^+$). Similarly p^- denotes the position of the first hospital in the same tie as h_v (possibly $p = p^-$). Both q^+ and q^- are analogously defined for the hospitals. Then h_j denotes a hospital such that $p^- \leq pos(r_i, h_j) \leq p^+$. In the context of Constraints 3–5, $p = pos(r_i, h_j)$ and $q = pos(h_j, r_i)$.

An interpretation of the constraints is now given. Constraint 1 ensures that h_j's filled posts are occupied by residents in the order in which they appear on its list, and that if post $k-1$ is unoccupied then so is post k. Constraint 2 is a stability constraint and ensures that if r_i obtains a hospital strictly worse than h_j, then h_j is filled with residents at least as good as r_i. Constraints 3 and 5 are consistency constraints, i.e. if h_j is deleted from r_i's list then r_i is deleted from h_j's list, and vice versa. Finally Constraint 4 states that if r_i is assigned hospital h_j, and h_j's first $k-1$ posts are filled by residents who appear before r_i on h_j's list, then h_j's k^{th} post must be filled by r_i or a resident who appears before r_i on h_j's list.

In contrast to the CSP encoding for HR, enforcing AC on a CSP instance of HRT does not, in general, yield a weakly stable matching. Furthermore, there is no guarantee that all weakly stable matchings can be found in a failure-free manner. To find all weakly stable matchings the following enumeration process is used.

- AC is established as a preprocessing step, and after each branching decision including the decision to remove a value from a domain;
- if all domains are arc consistent and some variable x_i has two or more values in its domain then search proceeds by setting x_i to the minimum value p in its domain. On backtracking, the value p is removed from the domain of x_i;

– when a solution is found, it is reported and backtracking is forced.

We now show that if during search a solution to the CSP encoding shown in Figure 8.3 is output by the constraint solver, then the solution corresponds to a weakly stable matching in M.

Lemma 8.3.1. *Let I be an instance of* HRT *and let M be an assignment relation output as a solution to the CSP encoding shown in Figure 8.3. Then M is a weakly stable matching.*

Proof. In order to establish the correctness of the result above, our first goal is to prove that M is a matching, and secondly that M is weakly stable.

To prove that M is a matching, the aim is to show that, in M, each resident is assigned to at most one hospital and the number of assignees of each hospital does not exceed its capacity. First we note that each resident r_i can only be assigned to at most one hospital, as a solution is output only when r_i's domain contains a single value. Now let h_j be a hospital with d_j assignees, where $d_j > c_j$. Then there exist d_j residents assigned to h_j, namely $r_{i_1}, r_{i_2}, \ldots, r_{i_{d_j}}$, where $pos(h_j, r_{i_k}) = q_k$ $(1 \leq k \leq d_j)$, and $q_1 < q_2 < \cdots < q_{d_j}$. Therefore $x_{i_k} = p_k$ $(1 \leq k \leq d_j)$, where $p_k = pos(r_{i_k}, h_j)$. By Constraint 4, since $x_{i_1} = p_1$ and $y_{j,0} < q_1$, we obtain $y_{j,1} \leq q_1$. Continuing in this manner, we obtain $y_{j,c_j} \leq q_{c_j}$. Therefore propagation of Constraint 5 yields $x_{i_b} \neq p_b$, for all b $(c_j < b \leq d_j)$, a contradiction to the fact that $x_{i_k} = p_k$ for all k $(1 \leq k \leq d_j)$. Hence $d_j \leq c_j$, as required. Since h_j was arbitrary, the result holds for all hospitals.

We now establish that M is a weakly stable matching. Suppose for a contradiction that (r_i, h_j) blocks M in I. Let $p = pos(r_i, h_j)$ and $q = pos(h_j, r_i)$. As (r_i, h_j) blocks M in I, either r_i is unmatched, or r_i strictly prefers h_j to $M(r_i)$, and either h_j is under-subscribed or h_j strictly prefers r_i to its worst assignee. First, suppose that r_i is unmatched in M, therefore $x_i = m + 1 > p^+$. Hence propagation of Constraint 2 yields $y_{j,k} \leq q^+$ $(1 \leq k \leq c_j)$. Then in M, h_j is full with c_j assignees, none of whom are worse than r_i, a contradiction to the fact that (r_i, h_j) blocks M in I. Therefore r_i must be matched in M, and r_i strictly prefers h_j to his assignee h_v in M, where $pos(r_i, h_v) = s > p^+$. Then $x_i > p^+$. Now using a similar argument to that above, we can easily verify that, in M, h_j must be full with c_j assignees, none of whom are worse than r_i. Therefore M is a weakly stable matching. □

We now establish that every weakly stable matching corresponds to a set of arc consistent domains. First we introduce some notation. Let M be a weakly stable matching

in which each $h_j \in H$ is assigned d_j residents, namely $r_{i_1}, \ldots, r_{i_{d_j}}$, where $q_k = pos(h_j, r_{i_k})$ $(1 \leq k \leq d_j)$, and $q_1 < q_2 < \cdots < q_{d_j}$. Then the *M-domains* for M are defined as follows:

$$dom(x_i) = \begin{cases} \{pos(r_i, M(r_i))\}, & \text{if } M(r_i) \neq \emptyset \\ \{m+1\}, & \text{otherwise} \end{cases}$$

$$dom(y_{j,k}) = \begin{cases} \{q_k\}, & \text{if } 1 \leq k \leq d_j \\ \{n+k\}, & \text{if } d_j + 1 \leq k \leq c_j \end{cases}$$

Lemma 8.3.2. *The M-domains are arc consistent in J.*

Proof. Suppose that the variables are assigned values as defined by the M-domains. Then we show that the variables' domains are arc consistent by considering each constraint in turn.

First consider Constraint 1. By definition, if $1 \leq k < d_j$, $y_{j,k} = q_k$ and $y_{j,k+1} = q_{k+1}$, where $q_k < q_{k+1}$. Also for any k such that $d_j \leq k \leq c_j$, it follows that $y_{j,k} = n + k < n + k + 1 = y_{j,k+1}$. Hence Constraint 1 is satisfied.

Now consider Constraint 2, and suppose that $x_i > p^+$. Then x_i is assigned to some hospital h_v, where $pos(r_i, h_v) = s_1 > p^+$. Let h_j be any hospital such that $p^- \leq pos(r_i, h_j) \leq p^+$. Then by the weak stability of M, h_j must be full with c_j residents, none of whom are worse than r_i, i.e. $y_{j,k} \leq q^+$ for all k $(1 \leq k \leq c_j)$, where $q = pos(h_j, r_i)$.

Consider Constraint 3. Now suppose that $x_i \neq p$ and let k $(1 \leq k \leq c_j)$ be given. Then r_i is not assigned to hospital h_j, where $p = pos(r_i, h_j)$. Let $q = pos(h_j, r_i)$. Then h_j is not assigned to r_i, therefore $y_{j,k} \neq q$ for $(1 \leq k \leq c_j)$.

For Constraint 4, suppose that $x_i = p$ and $y_{j,k-1} < q$. Hence r_i is assigned to h_j, where $pos(r_i, h_j) = p$ and $pos(h_j, r_i) = q$. Let $pos(h_j, r_u) = t < q$, where r_u is the resident occupying h_j's $(k-1)^{th}$ post. Hence the position of the resident holding h_j's k^{th} post must be at least as good as q, i.e. $y_{j,k} \leq q$. Therefore Constraint 4 is satisfied.

Finally consider Constraint 5, and suppose $y_{j,c_j} < q$, where $q = pos(h_j, r_i)$. Then $y_{j,k} < q$ for all k $(1 \leq k \leq c_j)$, therefore h_j is not assigned to r_i. Hence $x_i \neq p$, where $p = pos(r_i, h_j)$. □

The two lemmas above lead to the following theorem.

Theorem 8.3.3. *Let I be an instance of* HRT*. Then the set of solutions returned by the enumeration process coincides with the set of weakly stable matchings in I.*

We note that by an argument similar to that in Theorem 8.2.5, no weakly stable matching is repeated during the enumeration.

The constraints shown in Figure 8.3 can be revised in $O(1)$ time during AC propagation, assuming that upper and lower bounds for the variables' domains are maintained. Hence the time complexity of establishing AC is $O(ed)$ [72]. For this encoding we have $e = O(nmC)$ and $d = O(n + m)$, therefore AC may be established in $O((m + n)^3C)$, where $C = \max\{c_j : h_j \in H\}$.

To find a maximum weakly stable matching using the encoding in Figure 8.3 we can add a constraint C similar to those described in Section 7.6. Here the constraint C seeks to maximise the number of residents assigned over all the weakly stable matchings. An example of such a constraint is shown below:

$$\texttt{maximise}\{|\{x_i \in X : x_i \neq m + 1\}|\}$$

8.4 Open problems

Finally we present some open problems.

8.4.1 Optimal CP encoding for HR

For the HR encoding presented in Section 8.2, AC can be established in time $O((n+m)^3C)$, therefore it remains open as to whether there exists an encoding for HR for which AC can be established in time $O(\lambda)$ (yielding the GS-lists after AC propagation as before), where λ is the total length of the preference lists.

8.4.2 Value/variable ordering heuristics for HRT encodings

Section 8.3 describes a CP encoding for an instance of HRT. However a good value and variable ordering heuristic can make large difference to the time taken to find an optimal solution. Can we find such a value or variable ordering heuristic that works well with our HRT encoding?

8.4.3 Stable fixtures problem

The Stable Fixtures problem is a many-to-many generalisation of SR. The problem is known to be polynomial-time solvable in the case where no ties are allowed in an agent's list [42]. Can the encoding described in this chapter be extended to the Stable Fixtures problem?

Chapter 9

Conclusions

In this thesis we have presented a range of algorithmic results for stable matching problems. These include polynomial-time algorithms, and NP-hardness or NP-completeness results. These complexity results are summarised in Figure 9.1, whose rows correspond to the stable matching problems in Chapters 2-6. In the table the 'no ties' column indicates the complexity of finding a stable matching when agents' lists are strictly ordered. Similarly the columns labelled 'Weak', 'Strong', and 'Super' indicate the complexity of finding a weakly, strongly and super-stable matching respectively. The term '(max)' is used with respect to weak stability to denote the problem of finding a maximum weakly stable matching. Furthermore we use '(u)' to indicate that the matching output by the corresponding algorithm is in fact the unique matching of the given type. Additionally, we indicate the section number in round brackets where the results can be found.

Although this study has involved a diverse range of problems, we make here a number of remarks about common properties that can be observed from the contributions of this thesis.

1. Weak stability

   (a) A weakly stable matching (on its own) is usually easy to find in polynomial time. In the case of SMTI and HRT it is already known that simply breaking the ties arbitrarily and running the extended Gale/Shapley algorithm for SMI and HRT respectively yields a weakly stable matching. By contrast, the problem of deciding whether an instance of SRT admits a weakly stable matching is NP-complete. However, Figure 9.1 also shows that when we consider certain restrictions of an SRTI instance, a weakly stable matching can be found in

	No ties	Weak	Strong	Super
SPA-PW	NP-hard (max) (2.2.2)			
SPA-PS	P (2.3)			
$(3,4)$-MAX-SMTI $(2,\infty)$-MAX-SMTI $(2,\infty)$-COM-SMTI $(3,\infty)$-COM-SMTI		NP-hard (3.5) P (3.3) P (3.4) NP-complete (3.6)		
HR/HRT-1ML	P (u)(4.2)	P (4.2) NP-hard(max) (4.3)	P (4.5)	P (u)(4.4)
HR/HRT-2ML	P (u)(4.2)	NP-hard(max) (4.3)	P (4.5)	P (u)(4.4)
SRI/SRTI-ML	P (u)(5.2)	P (5.2) NP-hard(max) (5.3)	P (5.5)	P (u)(5.4)
SMI/SMTI-SYM HR/HRT-SYM	P (u)(5.2)	NP-hard(max) (6.2.2) NP-hard(max) (6.2.2)	P (6.4.2) P (6.4.2)	P (u)(6.3.2) P (u)(6.3.2)
SRI/SRTI-SYM	P (u)(6.1)	P (6.2.1) NP-hard(max) (6.2.2)	P (6.4.1)	P (u)(6.3.1)

Figure 9.1: Thesis contribution.

polynomial time.

(b) If we add any additional conditions (such as maximum cardinality, minimum regret, etc to the problem of finding a weakly stable matching) then NP-hardness usually prevails. In Chapter 6 we show this to be true for various problems involving finding weakly stable matchings given an SMT/SMTI instance even if the preference lists are subject to the restriction that they are symmetric. Also, from Figure 9.1, we can see that this is true for the problem of finding a maximum weakly stable matching even if preference lists on one or both sides are of bounded length, or are derived form one or two master lists.

(c) By contrast to the results mentioned in Part (b), there is a restricted version of SMTI for which the problem of finding a maximum cardinality weakly stable matching is solvable in polynomial time, namely the case where the men's lists are of length at most 2.

2. Strong/super-stability

   (a) In each of the cases of SMT, HRT and SRT, it is straightforward to formulate an example instance that admits no strongly stable or super-stable matching [35]. The same is true for SPA-PS, where the stability definition is analogous to strong stability, despite there being no ties in the preference lists of a SPA-PS instance.

   (b) Again, in each of the case of SMT, HRT and SRT, there are polynomial-time algorithms for each of the problems of finding a strongly stable or super-stable matching, or reporting that none exists [35, 38–40, 67]. The same is true for SPA-PS. Moreover for restricted cases of SMT, HRT and SRT, we are able to simplify and (in the case of strong stability) improve on the time complexity of these algorithms. We further observe that, in the case of stable matching problems with master lists and symmetric preferences, a super-stable matching returned by our algorithms is in fact the unique super-stable matching for the instance in question.

3. Constraint programming

   (a) It has been observed that the ability to add side constraints to model and solve NP-hard variants and versions of problems for which no polynomial-time algorithm is currently known is an attractive property of CP. We have used this to show that not only can instances of SM and HR be modelled simply and efficiently, but side constraints and simple extensions are presented, in Sections 7.5 and 8.3 respectively, that extend these encodings to NP-hard variants of the respective problem.

   (b) We also note that the versatility of CP allows for the development of a modular software system in relation to matching problems. Using CP eliminates the need to change the inner workings of an algorithm to deal with special cases that may arise. A constraint based approach allows for looser coupling between the different criteria that a matching must satisfy.

The study leaves open some interesting avenues for future research. Open problems are listed at the end of most chapters, each relating to the particular problem context to which they apply.

Appendix A

An Introduction to Constraint Programming

For many years in the field of AI (see [49]) *constraint satisfaction problems* (CSPs) have been an important area of research. A CSP has a set of variables, each with a domain of values, and a set of constraints involving the given variables. A solution to a CSP is an assignment to each variable a single value from its domain such that all constraints are satisfied. One method of finding a set of possible values that provide a satisfying assignment is known as *constraint programming* (CP).

CP is typically used to obtain solutions to problems that are known to be hard. There are many free, and commercial, CP toolkits that are used to solve a wide variety of problems. In particular, timetabling and scheduling problems that arise in practical applications have been formulated as CSPs and solved using CP toolkits.

We now formally define a CSP by the following components:

Variables: A set $X = \{x_1, x_2, ..., x_n\}$ of n variables that represent the values that we are attempting to find.

Variable Domains: Each variable $x_i \in X$ has a set of possible values called its *domain*, denoted by D_i. The domain may be finite or infinite. Here we only consider finite domains.

Constraints: Constraints model the restrictions between variables. A simple constraint might be $x_1 < x_2$.

Typically when dealing with constraints, we are interested in *binary constraints*. A binary constraint C_{ij} between variables x_i and x_j is a subset of the Cartesian product $D_i \times D_j$ that specifies the allowed pairs of values between x_i and x_j. We say that a binary constraint has *arity* two, meaning that each constraint contains two variables. A *binary constraint satisfaction problem* is a CSP that contains only binary and unary constraints (a unary constraint is a constraint with arity one). It should also be noted that any CSP can be transformed into a binary CSP [71, Section 1.4].

A solution to a binary CSP with variables $x_1, \ldots, x_n$ is a tuple $(a_1, \ldots, a_n)$, where $a_i \in D_i$ $(1 \leq i \leq n)$, and $(a_i, a_j) \in C_{ij}$ for each constraint C_{ij}. We say that a solution satisfies a constraint C_{ij} if $(a_i, a_j) \in C_{ij}$.

Let C_{ij} be a constraint between variables x_i and x_j. Then $b \in D_j$ is called a *support* for value $a \in D_i$ with respect to C_{ij} if (a, b) satisfies C_{ij}. A value $a \in D_i$ is said to be *viable* if for every variable x_j such that C_{ij} exists, a has a support in D_j. We say that a CSP instance J is *arc consistent* if for each domain D_i in J, all values in D_i are viable. Arc Consistency (AC) is the process of removing values from a variable's domain that are not viable.

AC algorithms have a long history. The first well-known algorithm (known as AC-2) was described by Waltz [73] in 1972. Then in 1977 Mackworth [51] presented an algorithm (AC-3) that is a simpler and more general version of Waltz's algorithm. The complexity of AC-3 was shown to be $O(ed^3)$ [52], where e is the total number of binary constraints, and d is the size of the largest variable domain. An enhanced algorithm, AC-4, was presented by Mohr and Henderson [58] in 1986. AC-4 has a theoretically optimal $O(ed^2)$ bound, however this is at the sacrifice of space-complexity over AC-3. Furthermore, although optimal, AC-4 may under certain circumstances run slower than AC-3, and in practice, AC-3 is often used instead of AC-4 when domain sizes are large. Further variations and improvements on AC algorithms have been developed; in particular van Hentenryck et al. [72] presented AC-5, which can achieve AC in $O(ed)$ time for a number of important classes of constraints, namely monotonic, functional and anti-functional constraints (for definitions of these constraint types see [72]). The latest AC algorithm is AC-7 [10].

An example graph showing the constraints between two variables is shown in Figure A.1(a). The graph represents a CSP instance with two variables x and y, each with domain $\{1 \ldots 5\}$. In (b) (x, y) has been made arc consistent, and as a result the domain of x is now $D_x = \{4, 5\}$. This is achieved by checking which values in D_x are consistent with

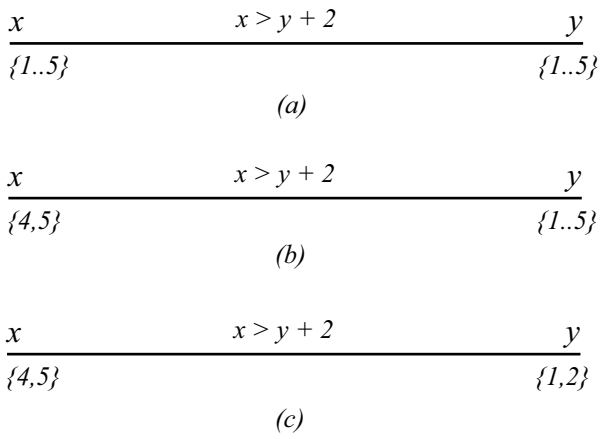

Figure A.1: Arc consistency of arc (x, y) and (y, x).

constraint C_{xy} $(x > y + 2)$. Lastly Figure A.1(c) shows the values after the arc (y, x) has been made arc consistent.

After AC propagation, if every variable's domain contains only a single value, then a solution has been found. Otherwise we have to *search* to obtain a solution. Search consists of instantiating a variable x with a value in its domain and then propagating the effects of this – if this instantiation cannot ultimately lead to a solution then the search backtracks and tries another value in x's domain. The search process typically involves an exponential number of steps depending on the problem and instance. To improve search capabilities, techniques have been developed to improve worst case performance in practice. These include symmetry reductions, backtracking, forward checking, and maintaining arc consistency [71]. In many practical CSP's performance is affected by the choice of a variable's value, or by the order in which variables are instantiated during search. Value and variable ordering heuristics can be used to dramatically improve the search time in such cases.

Bibliography

[1] D. Abraham, P. Biró, and D.F. Manlove. "Almost Stable" matching in the roommates problem. In *Proceedings of WAOA 2005: the 3rd Workshop on Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[2] D.J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings of EC '07: the 8th ACM Conference on Electronic Commerce*, pages 295–304. ACM, 2007.

[3] D.J. Abraham, R.W. Irving, and D.F. Manlove. The Student-Project Allocation Problem. In *Proceedings of ISAAC 2003: the 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 474–484. Springer, 2003.

[4] D.J. Abraham, R.W. Irving, and D.F. Manlove. Two algorithms for the Student-Project Allocation problem. *Journal of Discrete Algorithms*, 5(1):73–90, 2007.

[5] D.J. Abraham, A. Levavi, D.F. Manlove, and G. O'Malley. The stable roommates problem with globally-ranked pairs. To appear in *Proceedings of WINE 2007: the 3rd International Workshop On Internet and Network Economics*. Springer, 2007.

[6] B. Aldershof, O.M. Carducci, and D.C. Lorenc. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.

[7] A.A. Anwar and A.S. Bahaj. Student project allocation using integer programming. *IEEE Transactions on Education*, 46(3):359–367, 2003.

[8] E.M. Arkin, A. Efrat, J.S.B. Mitchell, and V. Polishchuk. Geometric Stable Roommates. Manuscript, 2007.
`http://www.ams.sunysb.edu/~kotya/pages/geomSR.pdf`.

[9] V. Bansal, A. Agrawal, and V.S. Malhotra. Stable marriages with multiple partners: efficient search for an optimal solution. In *Proceedings of ICALP '03: the 30th International Colloquium on Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 527–542. Springer, 2003.

[10] C. Bessière, E.C. Freuder, and J-C Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[11] L.E. Dubins and D. Freedman. Machiavelli and the Gale-Shapley algorithm. *American Mathematical Monthly*, 88:485–494, 1981.

[12] 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.

[13] T. Feder. *Stable Networks and Product Graphs*. PhD thesis, Stanford University, 1990.

[14] T. Feder. Network flow and 2-satisfiability. *Algorithmica*, 11:291–319, 1994.

[15] T. Fleiner, R.W. Irving, and D.F. Manlove. Efficient algorithms for generalised stable marriage and roommates problems.

[16] H.N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of STOC '83: the 15th Annual ACM Symposium on Theory of Computing*, pages 448–456. ACM, 1983.

[17] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[18] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[19] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[20] I.P. Gent, R.W. Irving, D.F. Manlove, P. Prosser, and B.M. Smith. A constraint programming approach to the stable marriage problem. In *Proceedings of CP '01: the 7th International Conference on Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2001.

[21] I.P. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *Proceedings of ECAI '02: the 15th European Conference on Artificial Intelligence*, pages 141–145. IOS Press, 2002.

[22] I.P. Gent and P. Prosser. SAT encodings of the stable marriage problem with ties and incomplete lists. In *Proceedings of SAT '02: The Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, 2002. `http://gauss.ececs.uc.edu/Conferences/SAT2002/Abstracts/gent.ps`.

[23] E. Gergely. A simple method for constructing doubly diagonalized latin squares. *Journal of Combinatorial Theory, Series A*, 16(2):266–272, 1974.

[24] M.J. Green and D.A. Cohen. Tractability by approximating constraint languages. In *Proceedings of CP '03: the 9th International Conference on Principles and Practice of Constraint Programming*, volume 2833 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2003.

[25] D. Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16(1):111–128, 1987.

[26] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[27] 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.

[28] M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation of the stable marriage problem. In *Proceedings of ESA '03: the 11th European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2003.

[29] M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Randomised approximation of the stable marriage problem. In *Proceedings of COCOON '03: the 9th Computing and Combinatorics Conference*, volume 2697 of *Lecture Notes in Computer Science*, pages 339–350. Springer, 2003.

[30] F. Harary. Maximum versus minimum invariants for graphs. *Journal of Graph Theory*, 7:275–284, 1983.

[31] 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.

[32] 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.

[33] J.D. Horton and K. Kilakos. Minimum edge dominating sets. *SIAM Journal on Discrete Mathematics*, 6:375–387, 1993.

[34] R.W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

[35] R.W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.

[36] R.W. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98: the Sixth European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 1998.

[37] 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.

[38] R.W. Irving and D.F. Manlove. The Stable Roommates Problem with Ties. *Journal of Algorithms*, 43:85–105, 2002.

[39] R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT 2000: the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2000.

[40] R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS 2003: the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 2003.

[41] R.W. Irving, D.F. Manlove, and S. Scott. The stable marriage problem with master preference lists. 2007. Submitted for publication.

[42] R.W. Irving and S. Scott. The stable fixtures problem - a many-to-many extension of stable roommates. Discrete Applied Mathematics, 155(16):2118–2129, 2007.

[43] K. Iwama, S. Miyazaki, and K. Okamoto. A $(2 - c\frac{\log n}{n})$-approximation algorithm for the stable marriage problem. In *Proceedings of SWAT '04: the 9th Scandinavian Workshop on Algorithm Theory*, volume 3111 of *Lecture Notes in Computer Science*, pages 349–361. Springer, 2004.

[44] A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics*, 10:1–19, 1993.

[45] 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 2004: the 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2004.

[46] D. Kazakov. Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science, 2002.

[47] D.E. Knuth. *Mariages Stables*. 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.

[48] D.E. Knuth. *Stable Marriage and its Relation to Other Combinatorial Problems*, volume 10 of *CRM Proceedings and Lecture Notes*. American Mathematical Society, 1997. English translation of *Mariages Stables*, Les Presses de L'Université de Montréal, 1976.

[49] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13:32–44, 1992.

[50] I.J. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.

[51] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[52] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[53] D.F. Manlove. Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, January 1999.

[54] 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.

[55] D.F. Manlove and G. O'Malley. Modelling and solving the stable marriage problem using constraint programming. Technical Report TR-2005-192, University of Glasgow, Department of Computing Science, 2005.

[56] D.F. Manlove and G. O'Malley. Stable marriage with ties and bounded length preference lists. 2007. Submitted to *Journal of Discrete Algorithms*.

[57] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of FOCS '80: the 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 17–27. IEEE Computer Society, 1980.

[58] R. Mohr and T.C. Henderson. Arc and path consistency revised. *Artificial Intelligence*, 25:65–74, 1986.

[59] C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.

[60] C. Ng and D.S. Hirschberg. Three-dimensional stable matching problems. *SIAM Journal on Discrete Mathematics*, 4:245–252, 1991.

[61] National Resident Matching Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.

[62] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[63] 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.

[64] A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.

[65] A.E. Roth, T. Sonmez, and M. U. Unver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005.

[66] A.E. Roth, T. Sonmez, and M.U. Unver. Kidney exchange. *Quarterly Journal of Economics*, 119(2):457–488, 2004.

[67] S. Scott. *A study of stable marriage problems with ties*. PhD thesis, University of Glasgow, Department of Computing Science, 2005.

[68] `http://www.nes.scot.nhs.uk/sfas/` (Scottish Foundation Allocation Scheme website).

[69] 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.

[70] M. Thorn. A constraint programming approach to the student-project allocation problem. BSc Honours project report, University of York, Department of Computer Science, 2003.

[71] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[72] P. van Hentenryck, Y. Deville, and C-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[73] D.L. Waltz. Understanding line drawings of scenes with shadows. In E. Winston, editor, *The Psychology of Computer Vision*, pages 19–91, New York, 1975. McGraw-Hill.

[74] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, 18(1):364–372, 1980.

Index to first usage of major terminology