Algorithmics of Two-Sided Matching Problems

David J. Abraham

Submitted for the degree of

Master of Science,

Department of Computing Science,

University of Glasgow,

October, 2003

©2003 David J. Abraham
Abstract

In this thesis, we study several types of two-sided matching problems. Such a problem involves two disjoint sets of participants, say $U$ and $W$, each of whom ranks a subset of the other set of participants in order of preference. A matching in this context is a pairing of members of $U$ with members of $W$ that satisfies certain problem-specific cardinality and ranking constraints. Chapter 1 contains a brief introduction to two-sided matching problems, and provides the necessary background for the remaining thesis.

In Chapter 2, we introduce the student-project allocation problem (SPA), which generalizes the classical hospitals/residents problem (HR). An instance of SPA consists of two sets of participants, namely students and projects, where each project is offered by a unique lecturer. Each student ranks a subset of the projects in order of preference, and similarly, each lecturer ranks a subset of the students in order of preference. We present two optimal linear-time algorithms for finding a stable matching of students to projects, where the stability property generalizes the corresponding concept in HR. The first algorithm finds a student-optimal stable matching, which is simultaneously the best possible stable matching for all students. The second algorithm finds a lecturer-optimal stable matching, which is simultaneously the best possible stable matching for all lecturers.

In Chapter 3, we study the exchange-stable matching problem ESM. An instance of ESM consists of a set of applicants and a set of posts, where each applicant ranks a subset of the posts in order of preference. A matching of applicants to posts is exchange-stable if no applicant can obtain a better allocation without requiring some other applicant to obtain a worse allocation. We give several properties of the set of all exchange-stable matchings for an arbitrary instance of ESM. For example, we present three different algorithms to prove that the problem of finding a maximum cardinality exchange-stable matching is polynomial-time solvable. We also give a polynomial-time checkable characterization of the set of ESM instances that admit a unique exchange-stable matching. Finally, we introduce the concept of an exchange-stable matching signature to show a relationship between ESM and the classical stable marriage problem with incomplete lists.

In Chapter 4, we introduce the tutorial allocation problem (TA). An instance of TA consists of a set of students, and a set of tutorials, where each tutorial has a specified capacity, and each student may only be available for a subset of the tutorials. The TA problem is to allocate each student to exactly one tutorial without exceeding the capacity of any tutorial. We consider the minimum tutorial cover (MTC) variant of TA, in which we seek a maximum cardinality allocation with the minimum number of non-
empty tutorials. We present a polynomial-time solvable restriction of MTC, but prove that, in general, MTC is NP-hard. Finally, we give a new algorithm for finding a balanced allocation, which distributes students amongst tutorials as evenly as possible.

In Chapter 5, we introduce half-strong stability, which is a new type of stability for the stable marriage problem with ties and incomplete lists (SMTI). We place half-strong stability in context with the three classical types of stability for SMTI, namely weak, strong and super-stability. We then consider the problem of (i) determining if an instance of SMTI admits a half-strongly stable matching, and (ii) finding such a matching, if one exists. We give two polynomial-time solvable special cases of this problem, but prove that, in general, it is NP-hard.

In Chapter 6, we consider Gusfield and Irving’s ninth open problem [28], which is to determine if there is a reduction from the stable roommates problem (SR) to the stable marriage problem (SM), such that there is a correspondence between the stable matchings in SR and the stable matchings in SM. We give a reduction from SR to a variant of SMTI, which, although not directly answering the open problem, should provide some intuition and machinery to find the required reduction, or prove that no such reduction exists.

In Chapter 7, we present results on two miscellaneous problems. Firstly, we introduce the partner swapping problem (PSP), giving a characterization of the set of stable matchings admitted by an instance I of PSP. Secondly, we consider the minimum maximal matching problem (MMM) from graph theory, giving three new approximation algorithms. The last two algorithms use a restricted brute force approach to improve on existing approximation algorithms. These algorithms may be viewed weaker forms of polynomial-time approximation schemes, where the approximation guarantee converges to some constant greater than 1. We extend this idea to give improved approximation algorithms for minimum vertex cover and maximum satisfiability.
# Contents

## 1 Introduction

1.1 Complexity Theory ................................................. 2
1.1.1 Decision Problems ............................................. 2
1.1.2 Optimization Problems ........................................ 3

1.2 Graph Matching ......................................................
  1.2.1 Unweighted Graphs ........................................... 4
  1.2.2 Weighted Graphs ............................................. 6
  1.2.3 b-matching ................................................... 7
  1.2.4 Flow Networks ............................................... 7

1.3 Stable Matching .....................................................
  1.3.1 Practical Applications ....................................... 9
  1.3.2 Stable Marriage Problem ..................................... 9
  1.3.3 Preference List Generalizations of SM ....................... 10
  1.3.4 Hospitals/Residents Problem ................................. 13
  1.3.5 Stable Roommates Problem .................................. 14

## 2 Student-Project Allocation

2.1 Introduction ....................................................... 15
2.2 Simplified Model .................................................. 15
2.3 One-sided preferences ............................................ 18
2.4 Two-sided Preferences ...........................................
  2.4.1 Overview of Algorithm SPA-student .......................... 22
  2.4.2 Correctness of Algorithm SPA-student ....................... 23
  2.4.3 Analysis of Algorithm SPA-student .......................... 26
  2.4.4 Properties of the Student-Project Allocation Problem .... 28
  2.4.5 Overview of Algorithm SPA-lecturer ........................ 29
  2.4.6 Correctness of algorithm SPA-lecturer ...................... 30
  2.4.7 Analysis of Algorithm SPA-lecturer ....................... 35

2.5 Conclusions and Open Problems ................................. 39

## 3 Exchange-Stability

3.1 Problem Definition ............................................... 41
3.2 Background ....................................................... 42
3.3 Preliminary Results and Observations ........................... 42
  3.3.1 Checking Exchange-Stability ................................. 42
  3.3.2 Existence of Exchange-Stable Matchings ..................... 43
  3.3.3 Sizes of Exchange-Stable Matchings ....................... 44

3.4 Maximum Cardinality Exchange-Stable Matchings ............... 45
3.5 Uniqueness and Applicant-Optimality ........................... 51
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Approximation Classes.</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Augmenting path algorithm for finding a maximum matching.</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Gale/Shapley Algorithm.</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>An instance of the Simplified Student-Project Allocation problem.</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Network Flow Models for the simplified student-project allocation problem.</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>An instance of the Student-Project Allocation problem [1].</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>An instance of the Student-Project Allocation problem.</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>Algorithm for finding a student-optimal stable matching.</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>Instance $I_1$ of the Student-Project Allocation problem.</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Instance $I_2$ of the Student-Project Allocation problem.</td>
<td>29</td>
</tr>
<tr>
<td>11</td>
<td>Algorithm for finding a lecturer-optimal stable matching.</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>An instance of the Student-Project Allocation problem.</td>
<td>35</td>
</tr>
<tr>
<td>13</td>
<td>Implementation of algorithm SPA-lecturer.</td>
<td>38</td>
</tr>
<tr>
<td>14</td>
<td>Algorithm Greedy-ESM.</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>An instance of ESM.</td>
<td>45</td>
</tr>
<tr>
<td>16</td>
<td>Algorithm Stabilize-ESM.</td>
<td>46</td>
</tr>
<tr>
<td>17</td>
<td>Algorithm for Lex-ESM.</td>
<td>49</td>
</tr>
<tr>
<td>18</td>
<td>An instance of SMI which admits no stable matching which is also man-exchange-stable.</td>
<td>55</td>
</tr>
<tr>
<td>19</td>
<td>An instance of TA.</td>
<td>61</td>
</tr>
<tr>
<td>20</td>
<td>$G[I]$ for TA instance in Figure 19.</td>
<td>62</td>
</tr>
<tr>
<td>21</td>
<td>Constructing $M'$ from $M$.</td>
<td>63</td>
</tr>
<tr>
<td>22</td>
<td>MinWMCM on $G[I]$.</td>
<td>63</td>
</tr>
<tr>
<td>23</td>
<td>An algorithm for finding a balanced matching.</td>
<td>67</td>
</tr>
<tr>
<td>24</td>
<td>Relationship between stability definitions.</td>
<td>73</td>
</tr>
<tr>
<td>25</td>
<td>An instance of SMT/SMTI with no strongly stable matching.</td>
<td>73</td>
</tr>
<tr>
<td>26</td>
<td>An instance of SMTI with no half-strongly stable matching</td>
<td>74</td>
</tr>
<tr>
<td>27</td>
<td>Relationship between stability definitions, where no ties occur on the women’s side.</td>
<td>75</td>
</tr>
<tr>
<td>28</td>
<td>Instance of SMTI admitting half-strongly stable matchings with different cardinalities</td>
<td>75</td>
</tr>
<tr>
<td>29</td>
<td>Relationship between stability definitions, where no ties occur on the men’s side.</td>
<td>76</td>
</tr>
<tr>
<td>30</td>
<td>Reduction preference lists.</td>
<td>77</td>
</tr>
<tr>
<td>31</td>
<td>Instance of SR that admits no stable matchings [45].</td>
<td>80</td>
</tr>
<tr>
<td>32</td>
<td>Instance of SMTI/SMRI.</td>
<td>81</td>
</tr>
</tbody>
</table>
Instance of SR and two corresponding preference structures in MAX-SMRI. 83
Greedy algorithm for finding a maximal matching. 86
Reducing path approximation algorithm for MMM. 90
ApproxMMM1 on a tree. 91
A \((2 - \frac{4}{n+2})\)-approximation algorithm for MMM. 92
\( \left( r - \frac{2cr}{n+2c(r-1)} \right) \)-approximation algorithm for MMM. 94
Approximation Performance of ApproxMMM3 to 5 decimal places. 95
\(G[2, 1] \) - a worst case graph for ApproxMMM3, with \( r = 2 \). 96
Improved approximation algorithm for MINIMUM VERTEX COVER. 96
Declarations

No part of this thesis has been previously submitted by the author for a degree at any other university, and all results contained within, unless otherwise stated, are claimed as original. The proof of Theorem 5.10 and the reduction from SR to MAX-SMRI in Section 6.2 are due to David Manlove. Algorithm SPA-student is due to Rob Irving and David Manlove, however the correctness proof and run-time analysis are original.

Publications

D. Abraham, R. Irving and D. Manlove. The Student-Project Allocation Problem. In the 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-Verlag, 2003. (This paper is based on Sections 2.4 - 2.4.4.)
Acknowledgements

I would like to thank David Manlove for giving me the opportunity to study two-sided matching. This was my first choice for graduate study, but without David’s support it would not have been possible. David has been outstanding supervisor. I have learnt much from his approach to research, his insightful comments on my work, and his encyclopaedic knowledge of algorithmics.

I would also like to thank my second supervisor, Rob Irving. Rob’s past work forms much of the basis of two-sided matching theory, and hence strongly influences this thesis. My meetings with Rob were always interesting, and our discussions inspired several ideas addressed in this work.

I am thankful to the following organizations, each of which provided financial support for my research.

- Engineering and Physical Sciences Research Council (studentship associated with grant GR/R/84597/01)
- Universities UK (Overseas Research Student Award)
- University of Sydney (Barker Travelling Scholarship)

Finally, thanks go to my family and Liz for their love and support.

Dedicated to memory of Oscar and Phyllie.
1 Introduction

A two-sided matching problem $\Pi = (U, W)$ consists of two disjoint sets of participants, $U$ and $W$, each of whom submits a list of acceptable participants from the other set, which may be ranked in order of preference. We say that two participants $m \in U$ and $w \in W$ find each other acceptable if both $m$ and $w$ rank each other on their respective preference lists. A matching $M$ of $\Pi$ is subset of $U \times W$, where (i) for each $(m, w) \in M$, $m$ and $w$ find each other acceptable, and (ii) $M$ satisfies certain problem-specific capacity constraints.

For example, consider the following real-world two-sided matching problem. Let $U$ be a set of high school graduates, and let $W$ be a set of university courses, where each course $c \in C$ has a capacity $\text{cap}(c)$, indicating the maximum number of graduates it may admit. Each graduate submits a preference list ranking the subset of courses that he/she finds acceptable. Depending on the problem, each course may or may not supply a preference list ranking those students that find it acceptable. In this context, a matching must satisfy the following two capacity constraints:

(i) Each graduate may be allocated to at most one course.

(ii) No course $c$ may be allocated more than $\text{cap}(c)$ students.

There are several other real-world examples of two-sided matching problems (see [64] and Section 1.3.1), each of which typically involves thousands of participants. In many cases, organizations have established centralized processes to solve these problems: preference lists are collected from participants, and an algorithm is run to find an optimal matching, where the definition of optimality is problem specific.

Because these problems can involve so many participants, we are concerned with finding efficient matching algorithms. Section 1.1 contains a review of complexity theory, which deals with the efficiency of algorithms.

There is a strong connection between two-sided matching problems and graph matching (vertices correspond to participants, and edges correspond to two participants finding each other acceptable). A review of graph matching can be found in Section 1.2.

Finally, we remark that various two-sided matching problems have been extensively studied. When preference lists are on both sides, the notion of an optimal matching usually involves stability. A matching $M$ of $\Pi$ is stable unless there is some pair of participants $(m, w) \in U \times W \setminus M$ such that $m$ and $w$ prefer each other to their assignment in $M$. A review of previous work on finding stable matchings is contained in Section 1.3.
1.1 Complexity Theory

In this section, we give a brief review of the complexity class hierarchy for decision problems and optimization problems. This theory will subsequently allow us to analyze the complexity of two-sided matching problems.

1.1.1 Decision Problems

A decision problem $\Pi = (\mathcal{I}, \pi)$ consists of a Turing-recognizable set $\mathcal{I}$ of instances, and an (implicit) function $\pi: \mathcal{I} \to \{0, 1\}$. For any instance $I \in \mathcal{I}$, we say that $I$ is a yes-instance of $\Pi$ if $\pi(I) = 1$, and a no-instance of $\Pi$ otherwise. The decision problem $\Pi$ for $I$ is to determine if $I$ is a yes-instance of $\Pi$. An algorithm $A$ solves $\Pi$ if $A$ maps $I$ to $\{0, 1\}$, and $A(I) = \pi(I)$ for all $I \in \mathcal{I}$.

Denote by $|I|$ the length of a reasonable string encoding $^1$ of $I$. We say that $A$ runs in polynomial time if there is some non-negative integer $k$, such that a deterministic Turing machine can compute $A(I)$ in $O(|I|^k)$ time, for all $I \in \mathcal{I}$. Denote by $P$ the class of all decision problems that are solvable by some polynomial-time algorithm. Any problem (decision problem or otherwise) that does not admit a polynomial-time algorithm is said to be intractable. Garey and Johnson [26, pages 7-8] highlight the difference in actual running times between algorithms with polynomial time and super-polynomial time complexity functions.

A non-deterministic algorithm $A'$ solves $\Pi$ if $A'(I) = 1$ if and only if $I$ is a yes-instance of $\Pi$. We say that $A'$ runs in non-deterministic polynomial time if there is some non-negative integer $k$ such that a non-deterministic Turing machine can compute $A'(I) = 1$ in $O(|I|^k)$ time, for all yes-instances $I$ of $\Pi$. Denote by $NP$ the class of all decision problems that are solvable by some non-deterministic polynomial-time algorithm. We remark that $P \subseteq NP$.

Let $\Pi_1 = (\mathcal{I}_1, \pi_1)$ and $\Pi_2 = (\mathcal{I}_2, \pi_2)$ be any two decision problems. A polynomial-time reduction from $\Pi_1$ to $\Pi_2$ is a polynomial-time computable function $f$ from $\mathcal{I}_1$ to $\mathcal{I}_2$ such that, for all $I \in \mathcal{I}_1$, $I$ is a yes-instance of $\Pi_1$ if and only if $f(I)$ is a yes-instance of $\Pi_2$.

Suppose $\Pi_2 \in P$. Then there is some polynomial-time algorithm $A_2$ that solves $\Pi_2$. It is easy to see that $A_2(f(I)) = \pi_1(I)$ for all $I \in \mathcal{I}_1$. So, $A_1 = A_2 \circ f$ solves $\Pi_1$, and since polynomials are closed under composition, $A_1$ runs in polynomial time. Therefore, if $\Pi_2 \in P$, we have that $\Pi_1 \in P$, and contrapositively, if $\Pi_1 \notin P$, we have that $\Pi_2 \notin P$. For this reason, we say that $\Pi_2$ is at least as hard as $\Pi_1$.

---

$^1$For a discussion of the term reasonable, see [26, pages 17 - 22].
A Turing reduction generalizes the concept of a polynomial-time reduction: given any two problems \( \Pi_1 \) and \( \Pi_2 \), we say that \( \Pi_1 \) Turing-reduces to \( \Pi_2 \) if, given a hypothetical polynomial-time algorithm for solving \( \Pi_1 \), there is a polynomial-time algorithm for solving \( \Pi_2 \). A problem \( \Pi \) is \textit{NP-hard} if, for all \( \Pi' \in \text{NP} \), there is a Turing reduction from \( \Pi' \) to \( \Pi \). Furthermore, if \( \Pi \in \text{NP} \), we say that \( \Pi \) is \textit{NP-complete}.

Cook [14] showed that the class of NP-complete problems is non-empty by proving the membership of the satisfiability problem. Since then, hundreds of problems have been shown to be NP-complete (see [26, 59] for example). These problems, the hardest in NP, share the property that if any one of them is solvable in polynomial time, then every problem in NP is solvable in polynomial time (i.e. \( \text{P} = \text{NP} \)). However, no polynomial-time algorithm has been published for an NP-complete problem, so it may be the case that \( \text{P} \neq \text{NP} \), meaning that every NP-complete problem is intractable.

1.1.2 Optimization Problems

Many NP-complete problems are decision versions of \textit{optimization} problems [10, Lemma 6.1]. In this section, we review complexity classes over optimization problems.

An optimization problem \( \Pi = (\mathcal{I}, \mathcal{F}, m, g \in \{\text{minimize, maximize}\}) \) consists of a set \( \mathcal{I} \) of instances, each \( I \) of which has an associated set \( \mathcal{F}(I) \) of feasible solutions. Additionally, associated with each solution \( s \in \mathcal{F}(I) \) is a measure \( m(I, s) \) of the quality of \( s \). The objective of \( \Pi \), given by \( g \), is to find a \( s \in \mathcal{F}(I) \) that either minimizes or maximizes \( m(I, s) \). Denote by \( \text{NPO} \) the class of all optimization problems \( \Pi \), where both \( \mathcal{I} \) and the range of \( \mathcal{F} \) are recognizable in polynomial time, \( m \) is computable in polynomial time, and there is some non-negative integer \( k \) such that for all \( I \in \mathcal{I} \) and \( s \in \mathcal{F}(I) \), \( |s| \leq |I|^k \).

An algorithm \( A \) solves \( \Pi \) if for all \( I \in \mathcal{I} \), the solution \( A(I) \) satisfies the objective of \( \Pi \). Denote by \( \text{PO} \) the class of all \( \text{NPO} \) problems that are solvable in polynomial time.

For many \( \text{NPO} \) problems, no polynomial-time algorithm has been found. In practice, we deal with such problems using \textit{approximation algorithms}, which are polynomial-time algorithms that simply return some feasible solution.

Let \( A \) be an approximation algorithm for \( \Pi \) and denote by \( \text{OPT}(I) \) the measure of a minimum (respectively maximum) solution to some \( I \in \mathcal{I} \). The \textit{performance ratio} of \( A \) with respect to \( I \) for the minimization (respectively
maximization) problem $\Pi$ is:

$$R_A(I) = \frac{m(I, A(I))}{OPT(I)} \quad \left( R_A(I) = \frac{OPT(I)}{m(I, A(I))} \right)$$

Denote by $R_A$ the smallest constant $c \geq 1$ such that $R_A(I) \leq c$ for all $I \in \mathcal{I}$. If $c < \infty$, then we say that $A$ is a $c$-approximation algorithm for $\Pi$, and that $\Pi$ is approximable within $c$. Denote by APX the class of all NPO problems that are approximable within some finite constant.

An approximation scheme $A$ for some problem $\Pi \in \text{APX}$ is an algorithm that accepts both (i) an instance $I$ from $\Pi$, and (ii) an upper bound $\epsilon > 1$, and then outputs a feasible solution, where $R_A(I, \epsilon) \leq \epsilon$. We say that $A$ is a polynomial-time approximation scheme if for all $I$ and $\epsilon$, $A(I, \epsilon)$ is computable in time polynomial in $|I|$. Denote by PTAS the class of NPO problems that admit a polynomial-time approximation scheme. Additionally, if $A$ runs in time polynomial in $1/(\epsilon-1)$, then $A$ is a fully polynomial-time approximation scheme. Denote by FPTAS the class of NPO problems that admit a fully polynomial-time approximation scheme.

We summarize the relationships between these complexity classes in the following figure.

PO $\subseteq$ FPTAS $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO

Figure 1: Approximation Classes.

These approximation classes are non-empty (see Ausiello et al. [7]), and the inclusions are strict if and only if $P \neq \text{NP}$ (see Bovet and Crescenzi [10]). Finally, we remark that a problem $\Pi$ is APX-hard if for all $\Pi' \in \text{APX}$, there is an $L$-reduction \(^2\) from $\Pi'$ to $\Pi$. Furthermore, if $\Pi \in \text{APX}$, then $\Pi$ is APX-complete. No APX-complete problem admits a polynomial-time approximation scheme unless $P = \text{NP}$ [15].

### 1.2 Graph Matching

In this section, we review important results from graph matching, which we subsequently use to solve several two-sided matching problems.

#### 1.2.1 Unweighted Graphs

Let $G = (V, E)$ be any graph with $n$ vertices and $m$ edges. A matching $M$ of $G$ is a subset of $E$ such that no two edges in $M$ are adjacent. We say

\(^2\)See [7] for more details.
that a vertex \( v \) is matched in \( M \) if there is some vertex \( M(v) \in V \) such that \( \{v, M(v)\} \in M \). Otherwise, \( v \) is unmatched in \( M \).

The size or cardinality of a matching \( M \), denoted by \( |M| \), is just the number of edges in \( M \). Every graph admits the same minimum cardinality matching, \( M = \emptyset \), which has size 0. However, the size of a maximum cardinality matching, denoted by \( \beta_1(G) \), depends on the structure of \( G \). For any graph \( G \), \( 0 \leq \beta_1(G) \leq n/2 \), since no matching of \( G \) can match more than \( n \) vertices. Any matching achieving this upper bound is called a perfect or complete matching.

The following theorem, due to Hall [29], characterizes the set of all bipartite graphs that admit a perfect matching.

**Theorem 1.1 (Hall Marriage Theorem)** Let \( G = (L, R, E) \) be any bipartite graph. \( G \) admits a perfect matching if and only if \( |L| = |R| \) and for all \( L' \subseteq L \), \( |L'| \leq |N(L')| \), where \( N(L') \) is the set of all vertices in \( R \) adjacent to some vertex in \( L' \).

Tutte’s Theorem [67] generalizes Hall’s Marriage Theorem by characterizing the set of all arbitrary graphs that admit a perfect matching. We remark that neither characterization is stated in terms of a polynomial-time checkable criterion, and neither characterization helps us find maximum cardinality matchings. The following work, due to Berge [9], solves these two problems.

Let \( M \) be a matching of an arbitrary graph \( G = (V, E) \). A path \( P \) in \( G \) is a sequence of vertices \( \langle v_1, v_2, \ldots, v_k \rangle \) such that (i) \( v_i \neq v_j \) for all \( i \neq j \), and (ii) \( \{v_i, v_{i+1}\} \in E \) for \( 1 \leq i \leq k - 1 \). For exposition purposes, we sometimes regard \( P \) as the edge set \( \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\} \). An \( M \)-alternating path in \( G \) is a path in which the edges alternately belong to \( M \) and \( E \setminus M \). An \( M \)-augmenting path is an \( M \)-alternating path beginning and ending with two vertices unmatched in \( M \).

**Theorem 1.2 (Berge)** Let \( M \) be a matching of an arbitrary graph \( G = (V, E) \). \( M \) is a maximum cardinality matching of \( G \) if and only if \( G \) admits no \( M \)-augmenting path.

Suppose that \( G \) admits an \( M \)-augmenting path \( A \). It is not too hard to see that the symmetric difference \( M' = M \oplus A \) is a matching of \( G \) with size \( |M'| = |M| + 1 \). This suggests the fundamental algorithm in Figure 2 for finding a maximum matching of \( G \).

It remains to show how to find an \( M \)-augmenting path, or prove that no such path exists. Suppose \( G = (L, R, E) \) is a bipartite graph. Let \( \overline{G} \) be
MaximumMatching($G = (V, E)$)
\[ M := \emptyset; \]
\[ \textbf{while} \ (G \text{ admits an } M\text{-augmenting path } A) \]
\[ M = M \oplus A; \]
\[ \textbf{return} \ M; \]

Figure 2: Augmenting path algorithm for finding a maximum matching.

the orientation of $G$ in which all edges in $E \setminus M$ are directed from $L$ to $R$, and all edges in $M$ are directed from $R$ to $L$. It is easy to see that, starting from the unmatched vertices in $L$, a depth-first search of $\overrightarrow{G}$ finds an $M$-augmenting path if and only if one exists. This search takes $O(n + m)$ time, and since there are at most $n/2$ such searches, the overall time complexity of the MaximumMatching algorithm for bipartite graphs is $O(n(n + m))$. The problem of efficiently finding an $M$-augmenting path in an arbitrary graph was first solved by Edmonds [16]. Gabow [20] then provided an $O(n + m)$ implementation of Edmonds’ algorithm, giving an overall time complexity of $O(n(n + m))$.

Hopcroft and Karp [33] improved the MaximumMatching algorithm for bipartite graphs by requiring that a maximal set of disjoint $M$-augmenting paths be found during each loop iteration. This improvement leads to the best known algorithm for maximum matching in bipartite graphs, with a time complexity of $O(\sqrt{nm})$. Micali and Vazirani [54] generalized this improvement, leading to a $O(\sqrt{nm})$ algorithm for maximum matching in arbitrary graphs. We summarize these results in the following theorem.

\begin{theorem}
Let $G$ be an arbitrary graph, with $n$ vertices and $m$ edges. A maximum cardinality matching of $G$ can be found in $O(\sqrt{nm})$ time.
\end{theorem}

\subsection{Weighted Graphs}

Let $G = (V, E)$ be an arbitrary weighted graph (so that each edge $e \in E$ has an associated weight $w(e) \in \mathbb{N}$). We define the weight of a matching $M$ of $G$ as $w(M) = \sum_{e \in M} w(e)$. Consider the problems of finding a (i) maximum weight matching of $G$, and a (ii) maximum weight maximum cardinality matching of $G$.

Both of these problems can be solved by the same variation of Maximum-Matching. The basic idea is to repeatedly select an $M$-augmenting path $A$ that maximizes $w(M \oplus A)$. It is not too hard to show that after $i$ iterations of the loop, $M$ has maximum weight among all matchings of size $i$. We can
solve these weighted matching problems in $O(n(m + n \log n))$ time for both bipartite graphs [19] and arbitrary graphs [22].

Let $G = (V, E)$ be a graph with weight function $w$, and consider the problem of finding a minimum weight maximum cardinality matching. Let $G' = (V, E)$ be a copy of $G$ but with weight function $w'$, where, for all $e \in E$, $w'(e) = \max_{f \in E} w(f) - w(e)$. It is easy to see that any maximum weight maximum cardinality matching of $G'$ is a minimum weight maximum cardinality of $G$. So, we can solve the minimum weight maximum cardinality matching problem using the algorithm described above. In subsequent chapters, we refer to the algorithm for solving the minimum weight maximum matching problem as MinWMCM.

### 1.2.3 $b$-matching

Let $G = (V, E)$ be a graph in which each vertex $v \in V$ has an associated capacity $b(v) \geq 1$. A $b$-matching $M$ of $G$ is a subset of $E$ such that for all $v \in V$, $|e \in M : v \in e| \leq b(v)$.

In this thesis, we are concerned with (weighted) bipartite graphs $G = (L, R, E)$, in which only vertices in $R$ may have a capacity greater than 1. The problem of finding a maximum cardinality $b$-matching is solvable in $O(\sqrt{B}m)$ time, where $m$ is the number of edges in $G$, and $B$ is the total sum of vertex capacities [21]. The problem of finding a minimum weight maximum cardinality $b$-matching is solvable in $O(\sqrt{\alpha(m, m) \log mm \log (mN)})$ time, where $\alpha$ is the inverse Ackerman function and $N$ is the maximum edge weight [23].

### 1.2.4 Flow Networks

A directed graph $N = (V, E)$ is a flow network if (i) every edge $e = (u, v) \in E$ has non-negative capacity $c(u, v) \geq 0$, (ii) $V$ contains a source vertex $s$ and sink vertex $t \neq s$, where indegree$(s) = \text{outdegree}(t) = 0$, and (iii) every vertex lies on some path from $s$ to $t$.

For notational convenience, we assume that if $e = (u, v) \notin E$, then $c(u, v) = 0$. A flow in $N$ is a function $f : V \times V \rightarrow \mathbb{R}_0^+$, where

**Capacity Constraint** For all $u, v \in V$, $f(u, v) \leq c(u, v)$.

**Flow Conservation** For all $v \in V \setminus \{s, t\}$, $\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$.

We are concerned with integral flows, in which the range of $f$ is $\mathbb{N}$. The size of a flow $f$, denoted by $|f|$, is $\sum_{v \in V} f(s, v)$ (i.e the total flow out of the

---

3 In subsequent chapters, we use the term matching to refer to a $b$-matching.
source vertex). Given a flow network \( N \), the **maximum flow** problem is to find a flow of maximum size in \( N \). Ahuja et al. [2] describe several approaches to solving this problem. The best known algorithm, due to Goldberg and Rao [27], has worst-case time complexity \( O(\min(n^{2/3}, m^{1/2})m \log (n^2/m) \log U) \), where \( n \) and \( m \) are the numbers of vertices and edges in \( N \) respectively, and \( U \) is the largest edge capacity.

We now show a relationship between network flows and \( b \)-matchings of bipartite graphs.

Let \( G = (L, R, E) \) be a bipartite graph, where each vertex \( r \in R \) has an associated capacity \( b(r) \geq 1 \). Construct a flow network \( N \) with one vertex for each \( l \in L \) and one vertex for each \( r \in R \). Add an edge \((l, r)\) to \( N \) with unit capacity, whenever \( \{l, r\} \in G \). Add an edge \((s, l)\) with unit capacity from the source vertex \( s \) of \( N \) to each vertex \( l \). Finally, add an edge \((r, t)\) with capacity \( b(r) \) from each vertex \( r \) to the sink vertex \( t \).

Let \( f \) be a flow in \( N \). Consider the set \( M = \{ \{l, r\} : f(l, r) = 1 \} \). It is not too hard to see that the capacity constraints in \( N \) imply that \( M \) must be a \( b \)-matching of \( G \).

Now, let \( f \) be a maximum flow of \( N \). We claim that \( f \) describes a maximum cardinality \( b \)-matching \( M \) of \( I \). Suppose for a contradiction that there is a \( b \)-matching \( M' \) of \( I \) such that \( |M'| > |M| \). Construct the following flow \( f' \) of \( N \). For each \( \{l, r\} \in M' \), push a unit of flow from \( s \) through \( l \) and \( r \) to \( t \). It follows that since \( M' \) is a \( b \)-matching, \( f' \) is a valid flow in \( N \). But then \( |f'| > |f| \), contradicting the assumption that \( f \) is a maximum flow of \( N \). Therefore, by finding a maximum flow of the associated flow network \( N \), we can find a maximum \( b \)-matching of \( G \) in \( O(nm + n^2 \log n) \) time, where \( n \) and \( m \) are the numbers of vertices and edges in \( G \) respectively.

Now, let \( N = (V, E) \) be a flow network in which each edge \( e = (u, v) \) has an associated cost \( c(u, v) \geq 0 \), and let \( f \) be a flow of \( N \). The **cost** of a flow is defined as \( \sum_{(u, v) \in V \times V} c(u, v)f(u, v) \), where if \( (u, v) \notin E \), then \( c(u, v) = 0 \). Ahuja et al. [2] describe several approaches to finding a minimum cost maximum flow of \( N \). The best known algorithm, due to Orlin et al. [65], has worst-case time complexity \( O(m \log U(m + n \log n)) \), where \( n \) and \( m \) are the number of vertices and edges in \( N \), and \( U \) is the largest edge capacity or cost.

### 1.3 Stable Matching

Stable matching problems consist of a set of agents, each of whom submits a **preference list** ranking a subset of the other agents in order of preference. The problem is to form a matching \( M \) of the agents such that no two agents would prefer each other to their assignment in \( M \).
1.3.1 Practical Applications

Many countries have centralized matching schemes that construct stable matchings of graduating medical students to their first hospital post (based on the preferences of students over hospitals, and hospitals over students) [28]. America’s National Residents Matching Program (NRMP) is the largest such scheme, involving over 20,000 medical students each year. The NRMP was founded in 1952 in response to widespread unhappiness with the existing scheme (which did not produce stable matchings). Roth [62] gives an exposition of the situation leading up to the founding of the NRMP, convincingly arguing that any successful two-sided matching scheme must be centralized and produce stable matchings. Since 1952, many other professions have adopted similar schemes to match graduating students to their first post. 4 In countries such as Spain [60] and Australia [68], stable matching schemes are also used to assign high school students to universities. Finally, we remark that stable matching schemes may operate on a much smaller scale, such as the assignment of chess tournament pairings [47].

1.3.2 Stable Marriage Problem

An instance $I$ of the stable marriage problem (SM) consists of a set $U$ of men and a set $W$ of women, where $|U| = |W| = n > 0$. Each person $p \in U \cup W$ supplies a preference list, ranking all the members of the opposite sex in strict order of preference.

A matching $M$ of $I$ is a subset of $U \times W$ such that $M$ is bijection. If $(m, w) \in M$, we say that $m$ is matched to $w$ and that $w$ is matched to $m$. Furthermore, we denote $w$ by $M(m)$ and $m$ by $M(w)$.

A matching $M$ is stable unless it admits a blocking pair, that is, a (man, woman) pair $(m, w)$ such that

(i) $(m, w) \notin M$.

(ii) $m$ prefers $w$ to $M(m)$.

(iii) $w$ prefers $m$ to $M(w)$.

Every instance of SM admits a stable matching [24], which may be found in linear time [12] using the Gale/Shapley algorithm [24] given in Figure 3.

Let $M$ be the stable matching returned by an execution of the Gale/Shapley algorithm on some instance $I$ of SM. Gale and Shapley [24] proved that every man is matched in $M$ with the best partner he could obtain in any

---

4See [57] for several examples.
Gale/Shapley($I = (U,W)$)

$M := \emptyset$;

assign each person to be free;

while some man $m$ is free

$w :=$ first woman on $m$’s list to whom $m$ has not yet proposed;
/* $m$ proposes to $w */
if $w$ is free

$M := M \cup \{(m, w)\};$ /* $m$ and $w$ become engaged */
else if $w$ prefers $m$ to $m'$, where $m' = M(w)$

$M := M \setminus \{(m', w)\};$ /* break the engagement between $m'$ and $w */
assign $m'$ to be free;
$M := M \cup \{(m, w)\};$ /* $m$ and $w$ become engaged */
else
/* $w$ rejects $m$’s proposal */

return $M$;

Figure 3: Gale/Shapley Algorithm

stable matching of $I$. McVitie and Wilson [53] subsequently proved that every woman is matched in $M$ with the worst partner she could obtain in any stable matching of $I$. $M$ is therefore called the man-optimal/woman-pessimal stable matching of $I$, and is denoted by $M_O$. It is easy to see that if we reverse the roles of men and women in the Gale/Shapley algorithm, then the returned matching, denoted by $M_Z$, is both woman-optimal and man-pessimal.

In general, $I$ may admit exponentially many stable matchings [45]. The set of all stable matchings of $I$ form a distributive lattice\textsuperscript{5}, in which $M_O$ and $M_Z$ are the minimal and maximal elements respectively.

Irving et al. [37] use this lattice structure to find fair stable matchings that optimize the overall happiness of both men and women. In particular, they give an algorithm for finding an egalitarian stable matching, which is a stable matching that minimizes the sum of the ranks people have for their partners.

1.3.3 Preference List Generalizations of SM

In this section, we consider three generalizations of SM, each of which relaxes the definition of a preference list.

\textsuperscript{5}[45] attributes this result to Conway.
Incomplete Lists

We say that a person $p_i$ finds a member of the opposite sex $p_j$ unacceptable if $p_i$ would prefer to be unmatched than to be partnered with $p_j$. An instance $I$ of SMI is an instance of SM in which preference lists may be incomplete (i.e. preference lists only rank acceptable members of the opposite sex). A matching $M$ of $I$ is defined as a subset of $U \times W$, where

(i) for all $(m, w) \in M$, $m$ and $w$ find each other acceptable.

(ii) no person is matched in $M$ to more than one member of the opposite sex.

$M$ is stable unless it admits a blocking pair $(m, w) \notin M$, where $m$ and $w$ find each other acceptable, $m$ is either unmatched in $M$ or prefers $w$ to $M(m)$, and $w$ is either unmatched in $M$ or prefers $m$ to $M(w)$. Note that people may be unmatched in such stable matchings; Theorem 2.1 shows that such people are unmatched in all stable matchings.

Many SM results can be generalized for SMI. In particular, every instance of SMI admits a stable matching which can be found in linear time using a version of the Gale/Shapley algorithm. Also, for every instance of SMI, the set of all stable matchings forms a distributive lattice (see [28, Section 1.4.2] for more details).

Ties

An instance $I$ of SMT is an instance of SM in which preference lists may contain ties (i.e. two or more people may be ranked equally on some preference list). A matching $M$ of $I$ is defined in the same way as a matching in SM. However, Irving [36] gives three separate definitions of stability in this context, namely weak, strong and super-stability.

$M$ is weakly stable unless it admits a blocking pair $(m, w) \notin M$ such that $m$ and $w$ prefer each other to their partners in $M$. $I$ always admits at least one weakly stable matching. Such a matching may be found by breaking all ties arbitrarily, and then using the Gale/Shapley algorithm to return a stable matching of the resulting SM instance. There is no known efficient representation for the set of all weakly stable matchings of $I$. Indeed, $I$ may not even admit man-optimal and woman-optimal stable matchings [62]. Furthermore, the problem of finding an egalitarian weakly stable matching of $I$ is not approximable within $O(n)$ [31].

$M$ is strongly stable unless it admits a blocking pair $(m, w) \notin M$ such that either

(i) $m$ prefers $w$ to $M(m)$, and $w$ either prefers $m$ to $M(w)$, or is indifferent between them, or
(ii) $w$ prefers $m$ to $M(w)$, and $m$ either prefers $w$ to $M(m)$, or is indifferent between them.

Some instances of SMT admit no strongly stable matching (see Figure 25 for example). Irving [36] gives a quadratic-time algorithm to determine if $I$ admits a strongly stable matching, and to find one, if one exists. The set of all strongly stable matchings of $I$ forms a distributive lattice under a suitable equivalence relation [49].

$M$ is super-stable unless it admits a blocking pair $(m, w) \notin M$, where $m$ either prefers $w$ to $M(m)$, or is indifferent between them, and $w$ either prefers $m$ to $M(w)$, or is indifferent between them. Irving [36] gives a linear-time algorithm to determine if $I$ admits a super-stable matching, and to find one, if one exists. The set of all super-stable matchings of $I$ forms a distributive lattice [66].

Ties and Incomplete Lists

An instance $I$ of SMTI is an instance of SM in which preference lists may be incomplete and contain ties. A matching $M$ of $I$ is defined in the same way as a matching in SMI. We can extend the definitions of weak, strong and super-stability to apply in this context by replacing every occurrence of the clause, $p$ prefers $q$ to $M(q)$ (for arbitrary $p$ and $q$), with, $p$ is unmatched in $M$, or prefers $q$ to $M(q)$.

Manlove [48] extended Irving’s algorithms to determine if $I$ admits a strongly stable (respectively super-stable) matching, and to find such a matching, if one exists. Kavitha et al. [44] have recently improved on the strong stability algorithm, claiming a $O(nL)$ time complexity, where $n$ is the number of participants and $L$ is the total length of the preference lists. A weakly stable matching of $I$ may be found using the same algorithm described for the corresponding SMT problem (although, in this case, breaking the ties leads to an instance of SMI).

Weakly stable matchings may have different sizes. The problem of finding a maximum (respectively minimum) cardinality weakly stable matching is APX-complete, even in various restricted instances [31, 30], such as when all ties are on one side only, with at most one tie per list, and lists are of constant length [30]. However, every weakly stable matching of $I$ is at least half the size of a maximum cardinality weakly stable matching [51], and there exist weakly stable matchings of all sizes between the size of a minimum cardinality and maximum cardinality weakly stable matching of $I$ [50]. This last result means that weak stability is an interpolating invariant.
1.3.4 Hospitals/Residents Problem

An instance $I$ of the hospitals/residents problem (HR) consists of a set $R$ of residents, and a set $H$ of hospitals. Each resident $r_i$ supplies a preference list ranking a subset of $H$ in strict order of preference (note that preference lists may be incomplete). Each hospital $h_j$ supplies a preference list ranking in strict order all those residents that ranked $h_j$ on their own preference list. If $r_i$ and $h_j$ rank each other in their preference lists, then we say they find each other acceptable. Associated with each hospital $h_j$ is an integer capacity $c_j$ indicating the maximum number of residents that may be assigned to $h_j$.

A matching $M$ of $I$ is a subset of $R \times H$, such that

(i) $(r_i, h_j) \in M$ implies that $r_i$ and $h_j$ find each other acceptable.

(ii) For each resident $r_i \in R$, $|(r_i, h_j) \in M : h_j \in H| \leq 1$.

(iii) For each hospital $h_j \in H$, $|(r_i, h_j) \in M : r_i \in R| \leq c_j$.

If $(r_i, h_j) \in M$, then we say that $r_i$ is matched with $h_j$, and $h_j$ is matched with $r_i$. A resident $r_i$ is either unmatched in $M$, or matched to some hospital, denoted by $M(r_i)$. Denote by $M(h_j)$ the set of residents matched with hospital $h_j$. We say $h_j$ is under-subscribed, full or over-subscribed according as $|M(h_j)|$ is less than, equal to, or greater than $c_j$, respectively.

$M$ is stable unless it admits a blocking pair $(r_i, h_j) \notin M$ such that $r_i$ and $h_j$ find each other acceptable, $r_i$ is unmatched in $M$ or prefers $h_j$ to $M(r_i)$, and $h_j$ is under-subscribed or prefers $r_i$ to the worst resident in $M(h_j)$.

HR is a many-one generalization of SMI. We can extend the definition of a man-optimal/woman-optimal stable matching in SMI to resident-optimal/hospital-optimal stable matching in HR. For a given instance $I$ of HR, there exist linear-time algorithms to find such stable matchings of $I$ (see [28, Section 1.6] for example). The set of stable matchings $\mathcal{M}$ of $I$ form a distributive lattice, which is the basis of several algorithms (see [28, Section 1.6] for further details). $\mathcal{M}$ has several properties, which we outline in Theorem 2.1.

An instance $I$ of HRT is an instance of HR in which ties are permitted in the preference lists. The definitions of stability in this context are analogous to the definitions for stability in SMTI. There exist linear-time algorithms to determine if an instance of HRT admits a strong (respectively super) stable matching, and to find such a matching, if one exists [40, 39]. Finally, we remark that the problem of finding a maximum cardinality weakly stable matching is NP-hard by restriction to SMTI.
1.3.5 Stable Roommates Problem

An instance $I$ of the stable roommates problem (SR) consists of a set of people $P$, where $|P| = 2n$, for some $n > 0$. Each person $p \in P$ supplies a preference list, ranking all the members of $P \setminus \{p\}$ in strict order of preference.

A matching $M$ of $I$ is a partition of $P$ into disjoint unordered pairs. If $\{p_i, p_j\} \in M$, we say that $p_i$ is matched to $p_j$ and that $p_j$ is matched to $p_i$. Furthermore, we denote $p_j$ by $M(p_i)$ and $p_i$ by $M(p_j)$.

A matching $M$ is stable unless it admits a blocking pair $\{p_i, p_j\}$ such that

(i) $\{p_i, p_j\} \notin M$.

(ii) $p_i$ prefers $p_j$ to $M(p_i)$.

(iii) $p_j$ prefers $p_i$ to $M(p_j)$.

SR generalizes SM [28] (see Theorem 6.1), however, unlike SM, some instances of SR admit no stable matching [24] (see Figure 31 for an example). Irving [35] gives a polynomial-time algorithm that decides if an instance of SR admits a stable matching, and finds one, if one exists. The three preference list generalizations of SM (incomplete lists, ties, and both ties and incomplete lists) have been considered for for SR [28, 61, 38].
2 Student-Project Allocation

2.1 Introduction

Many university departments run project courses that require students to independently undertake one of a number of projects, each of which is supervised by a (possibly different) lecturer. In this chapter, we investigate the problem of matching students to projects. There are several ways of modelling this problem. For each model, we define an optimal matching and present an algorithm that finds such a matching.

2.2 Simplified Model

An instance of the simplified student-project allocation (SSPA) problem involves a set $S = \{s_1, s_2, \ldots, s_n\}$ of $n$ students, a set $P = \{p_1, p_2, \ldots, p_m\}$ of $m$ projects, and a set $L = \{l_1, l_2, \ldots, l_q\}$ of $q$ lecturers. If student $s_i$ is willing to undertake project $p_j$, then we say $s_i$ finds $p_j$ acceptable. Denote by $A_i$ the set of all projects that $s_i$ finds acceptable.

Each lecturer $l_k$ offers a non-empty set of projects $P_k$, where $P_1, P_2, \ldots, P_k$ partitions $P$. We denote by $B_k$ the set of all students that find some project in $P_k$ acceptable. Associated with each lecturer $l_k$ is a capacity constraint $d_k$, indicating the maximum number of students $l_k$ is willing to supervise. Similarly, each project $p_j$ has a capacity constraint $c_j$, indicating the maximum number of students that may be assigned to $p_j$. We assume that $\max\{c_j : p_j \in P_k\} \leq d_k$.

An assignment $M$ is a subset of $S \times P$ such that:

1. $(s_i, p_j) \in M$ implies that $p_j \in A_i$.
2. For each $s_i \in S$, $|(s_i, p_j) \in M : p_j \in P| \leq 1$.

If $(s_i, p_j) \in M$, we say that $s_i$ is assigned to $p_j$, and $p_j$ is assigned to $s_i$. Hence, $M \subseteq S \times P$ is an assignment if and only if each student $s_i$ is assigned to at most one project $p_j$, where $s_i$ finds $p_j$ acceptable. For notational convenience, if $s_i$ is assigned to $p_j$, we may also say that $s_i$ is assigned to $l_k$, or $l_k$ is assigned to $s_i$, where $l_k$ is the lecturer offering $p_j$.

For each student $s_i \in S$, if $s_i$ is assigned to some project $p_j$ in $M$, then we let $M(s_i)$ denote $p_j$; otherwise, we say that $s_i$ is unmatched in $M$. For each project $p_j \in P$, $M(p_j)$ denotes the set of students assigned to $p_j$ in $M$. We say that $p_j$ is under-subscribed, full, or over-subscribed according as $|M(p_j)|$ is less than, equal to, or greater than $c_j$, respectively. Similarly, for each lecturer $l_k \in L$, $M(l_k)$ denotes the set of students assigned to $l_k$ in $M$, and $l_k$
is under-subscribed, full, or over-subscribed according as $|M(l_k)|$ is less than, equal to, or greater than $d_k$, respectively.

A matching $M$ is an assignment satisfying the following two conditions:

1. For each $p_j \in P$, $|M(p_j)| \leq c_j$.
2. For each $l_k \in L$, $|M(l_k)| \leq d_k$.

In this context, we say that a matching $M$ is optimal if for all matchings $M'$, $|M| \geq |M'|$. The SSPA problem then is to find an optimal matching. An example instance of SSPA, with student set $S = \{s_1, s_2, s_3, s_4\}$, project set $P = \{p_1, p_2, p_3\}$ and lecturer set $L = \{l_1, l_2\}$, is given in Figure 4.

<table>
<thead>
<tr>
<th>Acceptable Projects</th>
<th>Lecturer Offerings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1: {p_1, p_3}$</td>
<td>$P_1: {p_1, p_2}$</td>
</tr>
<tr>
<td>$A_2: {p_1, p_2}$</td>
<td>$P_2: {p_3}$</td>
</tr>
<tr>
<td>$A_3: {p_1}$</td>
<td></td>
</tr>
<tr>
<td>$A_4: {p_2, p_3}$</td>
<td></td>
</tr>
</tbody>
</table>

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

Figure 4: An instance of the Simplified Student-Project Allocation problem.

We can solve the SSPA problem using the following network flow model. Construct one vertex for each student, project and lecturer, in addition to a special source vertex and sink vertex. Add a directed edge with unit capacity from the source vertex to every student vertex. For each student $s_i$, add a directed edge with unit capacity from $s_i$ to every project in $A_i$. For each project $p_j$, add a directed edge with capacity $c_j$ from $p_j$ to $l_k$, where $l_k$ is the lecturer offering $p_j$. Finally, for each lecturer $l_k$, add a directed edge with capacity $d_k$ from $l_k$ to the sink vertex.

Let $f$ be an integral flow in such a network. We can construct an assignment $M$ for the SSPA by assigning $s_i$ to $p_j$ if and only if $f(s_i, p_j) = 1$. It is not too hard to see that the additional capacity constraints force $M$ to also be a matching.

Now, let $f$ be a maximum flow of $N$. We claim that $f$ describes a maximum matching $M$ of $I$. Suppose for a contradiction that there is a matching $M'$ of $I$ such that $|M'| > |M|$. Construct the following flow $f'$ of $N$. For each student $s_i$ assigned to some project $p_j$ in $M'$, push a unit of flow from the source through $s_i, p_j$, the lecturer offering $p_j$, and finally on to the sink.
It follows that since $M'$ is a matching, $f'$ is a valid flow in $N$. But then $|f'| > |f|$, contradicting the assumption that $f$ is a maximum flow of $N$.

Figure 5 gives two different flows for the instance described in Figure 4. The first network flow corresponds to the matching $\{(s_1, p_1), (s_4, p_2)\}$. Lecturer $l_1$ is full in this matching and there are no unmatched students that find a project offered by $l_2$ acceptable. However, there is a larger matching, $\{(s_1, p_3), (s_2, p_1), (s_3, p_1), (s_4, p_3)\}$, which is described by the flow in the second model. This is clearly a maximum matching, since every edge from the source is saturated.

![Network Flow Models for the Simplified Student-Project Allocation Problem](image)

Figure 5: Network Flow Models for the SIMPLIFIED STUDENT-PROJECT ALLOCATION PROBLEM.

For a given instance $I$ of SSPA, the associated network $N$ of $I$ has $O(n + m + q)$ vertices, $O(\lambda)$ edges, where $\lambda = \sum_{i=1}^{n} |A_i|$, and largest edge capacity $O(n + m)$. Using the best known algorithm for finding a maximum flow (see Section 1.2.4), the running time of this approach is therefore $o(\lambda^2)$. 
2.3 One-sided preferences

We can generalize the SSPA problem model by allowing each student \(s_i\) to rank the projects of \(A_i\) in order of preference (possibly involving ties). In this context, we still seek a maximum matching, but subject to this cardinality constraint, we want to satisfy an additional criterion, which involves optimizing some function of the student preferences. There are several acceptable criteria, for example, we could

- Maximize the number of students matched with their first-choice project, and subject to this, maximize the number of students matched to their second-choice project, and so on.
- Minimize the number of students matched with their \(m\)th choice project, and subject to this, minimize the number of students matched to their \((m-1)\)th choice project, and so on.

However, here we choose the following criterion. Let \(M\) be any assignment of students to projects. For each (student, project) pair \((s_i, p_j)\) in \(M\), we associate with \(M\) a penalty of \(r_{s_i}(p_j)\), where \(r_{s_i}(p_j)\) is the rank of \(p_j\) in \(s_i\)’s preference list. An optimal matching is then defined as a maximum matching that minimizes the sum of these penalties.

We can find such a matching by using another network flow algorithm. Given an instance \(I\) of the SSPA problem, augmented with the list of student preferences, construct the flow network \(N\) for \(I\). Now, using the student preference information, associate a cost of \(r_{s_i}(p_j)\) with each directed edge from a student \(s_i\) to a project \(p_j\) in \(N\), where \(p_j \in A_i\). All other edges in \(N\) have a zero cost. It is not too hard to see that a minimum cost maximum flow of \(N\) describes an optimum matching, since the maximum flow requirement guarantees the associated matching has maximum cardinality, whilst the minimum cost requirement exactly corresponds to the secondary aim of minimizing the penalty sum. Using the minimum cost maximum flow algorithm described in Section 1.2.4, the worst-case running time of this approach is \(O(\lambda^2 \log n)\), where \(\lambda\) is the total length of the applicant preference lists and \(n\) is the number of students.

2.4 Two-sided Preferences

In this section, we consider the TWO-SIDED STUDENT-PROJECT ALLOCATION (SPA) problem, in which students express preferences over projects, and lecturers express preferences over students. As in the one-sided model, each student \(s_i\) supplies a preference list, ranking a subset of \(P\) in strict order
of preference. Additionally, each lecturer \( l_k \) supplies a preference list \( \mathcal{L}_k \), ranking all members of \( B_k \) in strict order of preference, where \( B_k \) is the set of all students that find some project in \( P_k \) acceptable. For each project \( p_j \in P_k \), we define the projected preference list of \( l_k \) for \( p_j \), denoted by \( \mathcal{L}_j^k \), which is the preference list obtained from \( \mathcal{L}_k \) by removing students in \( B_k \) that do not find \( p_j \) acceptable.

An instance of SPA with student set \( S = \{s_1, s_2, \ldots, s_7\} \), project set \( P = \{p_1, p_2, \ldots, p_8\} \), and lecturer set \( L = \{l_1, l_2, l_3\} \) is given in Figure 6. As an example, the projected preference list of \( l_1 \) for \( p_1 \) consists of \( s_1, s_3, s_2, s_5 \), ranked in the order given.

<table>
<thead>
<tr>
<th>Student preferences</th>
<th>Lecturer preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1: p_1 p_7 )</td>
<td>( l_1: s_7 s_4 s_1 s_3 s_2 s_5 s_6 )</td>
</tr>
<tr>
<td>( s_2: p_1 p_2 p_3 p_4 p_5 p_6 )</td>
<td>( l_2: s_3 s_2 s_6 s_7 s_5 )</td>
</tr>
<tr>
<td>( s_3: p_2 p_1 p_4 )</td>
<td>( l_3: s_1 s_7 )</td>
</tr>
<tr>
<td>( s_4: p_2 )</td>
<td></td>
</tr>
<tr>
<td>( s_5: p_1 p_2 p_3 p_4 )</td>
<td></td>
</tr>
<tr>
<td>( s_6: p_2 p_3 p_4 p_5 p_6 )</td>
<td>Project capacities: ( c_1 = 2, c_i = 1 \ (2 \leq i \leq 8) )</td>
</tr>
<tr>
<td>( s_7: p_5 p_3 p_8 )</td>
<td>Lecturer capacities: ( d_1 = 3, d_2 = 2, d_3 = 2 )</td>
</tr>
</tbody>
</table>

Figure 6: An instance of the Student-Project Allocation problem [1].

Let \( I \) be any instance of SPA. Given a matching \( M \) of \( I \), we say that a (student, project) pair \((s_i, p_j)\) \( \in (S \times P) \setminus M \) blocks \( M \) if:

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. 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 \( l_k \) prefers \( s_i \) to the worst student \( s' \) in \( M(l_k) \) or \( s_i = s' \), 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 \( p_j \).

We call \((s_i, p_j)\) a blocking pair of \( M \). A matching is stable if it admits no blocking pair. For a given instance \( I \) of SPA, the SPA problem is to find a stable matching \( M \) of \( I \). We also consider two variants of this basic problem,
in which we require $M$ to have the additional property that, among all stable matchings of $I$, $M$ is simultaneously best for (i) every student, and (ii) every lecturer. Any such stable matching is referred to as a (i) student-optimal, and (ii) lecturer-optimal.

Our definition of a blocking pair attempts to encapsulate all the scenarios in which $s_i$ and $l_k$ could both simultaneously improve, relative to $M$, by permitting an assignment between $s_i$ and $p_j$. For this to occur, $s_i$ must find $p_j$ acceptable (Condition 1), and either be unmatched in $M$ or prefer $p_j$ to $M(s_i)$ (Condition 2). From $l_k$’s perspective, there must be a free place for $s_i$ (Condition 3(a)), or alternatively, $l_k$ must be able to make a free place in $p_j$ by rejecting an existing student $s’$ already assigned to $l_k$ (Condition 3(b) and (c)). Of course, $l_k$ would reject such a student $s’$ only if $l_k$ prefers $s_i$ to $s’$. There are two small subtleties.

Firstly, if $s_i$ is already assigned to $l_k$ (so $M(s_i) \in P_k$) and $p_j$ is under-subscribed, then we assume that since $l_k$ is indifferent about switching $s_i$ from $M(s_i)$ to $p_j$, he/she would not prevent such a switch from happening (Hence, in Condition 3(b), $s_i$ may equal $s’$). However, and secondly, if $p_j$ is full in $M$, then the only way such a switch could occur is if $l_k$ rejects a student $s’$ from $p_j$. But, since $s_i$ was already assigned to $l_k$, and now $l_k$ has rejected $s’$, the number of students assigned to $l_k$ has decreased by 1.

This situation is demonstrated by the instance in Figure 7. Consider the matching $M_1 = \{(s_1, p_2), (s_2, p_1)\}$. According to the definition of a blocking pair given above, $(s_1, p_1)$ blocks $M_1$. Hence, both $s_1$ and $l_1$ permit the assignment between $s_1$ and $p_1$, resulting in the matching $M_2 = \{(s_1, p_1)\}$, which is the only stable matching. However, it is clear that in going from $M_1$ to $M_2$, $l_1$ has lost a student, and hence $l_1$ may not agree to such a switch.

$$\begin{align*}
\text{Student preferences} & \quad \text{Lecturer preferences} \\
 s_1 : & \quad p_1 \quad p_2 & l_1 : & \quad s_1 \quad s_2 & l_1 \text{ offers } p_1 \text{ and } p_2 \\
 s_2 : & \quad p_1 \\
\end{align*}$$

Project capacities: $c_1 = c_2 = 1$

Lecturer capacities: $l_1 = 2$

Figure 7: An instance of the Student-Project Allocation problem.

Given that $l_1$ loses a student under this definition, one could alter Condition 3(c) to prevent such a switch occurring. However, we make two counter-arguments to such an alteration.

Firstly, by allowing $M_1$ to be stable, we introduce an element of strategy into the problem; rather than submit his/her true preference list, a student
could submit a shorter preference list in order to obtain a more preferred assignment. In the instance above, for example, if $s_1$’s preference list only consisted of $p_1$, then $s_1$ would be matched to $p_1$ under either definition of Condition 3(c). On the other hand, by not listing every project he/she finds acceptable, a student assumes an increased risk of being unmatched in the final matching.

Secondly, from a practical perspective, allowing both $M_1$ and $M_2$ to be stable implies that stable matchings may have different sizes. Under such a definition, we would want to find a maximum cardinality stable matching of $I$, for otherwise we would not be matching as many of the participants as possible. However, several other stable matching problems admit solutions of different sizes, such as SMTI under weak stability, and in each case, the problem of finding a maximum cardinality stable matching is NP-hard. We conjecture that by altering Condition 3(c), SPA would also be NP-hard. Furthermore, using the current definition of Condition 3(c), we have been able to prove several desirable properties of SPA in Theorems 2.6, 2.13 and 2.8. These properties, including the existence of a stable matching that is simultaneously optimal for every student, do not hold under a revised Condition 3(c). For example, in Figure 7, student $s_1$ prefers matching $M_2$ to $M_1$, while student $s_2$ prefers matching $M_1$ to $M_2$.

We remark that HR (Section 1.3.4) is a special case of SPA in which projects and lecturers are indistinguishable. More formally, each lecturer $l_k$ offers exactly one project $p_j$, where $d_k = c_j$. In the HR restriction, projects/lecturers are referred to as hospitals, while students are referred to as residents. At least two linear-time algorithms are known for finding a stable matching in an instance $I$ of HR. The resident-oriented algorithm [28, Section 1.6.3] finds the resident optimal stable matching of $I$, in which each student is assigned the best hospital that he/she could have in any stable matching. On the other hand, the hospital-oriented algorithm [28, Section 1.6.2] finds the hospital optimal stable matching $M$ of $I$. Each hospital is assigned the same number of students in all stable matchings, but $M$ has the additional property that there is no stable matching $M'$ of $I$ in which $M'(h) \setminus M(h)$ contains a student preferable to the worst student in $M(h)$.

HR also has several interesting properties, that together form the Rural Hospitals Theorem. For a full exposition of this theorem, see [28, Section 1.6.4].

Theorem 2.1 (Rural Hospitals) For a given instance of HR,

(i) each hospital is assigned the same number of residents in all stable matchings [25].
(ii) exactly the same set of residents are unassigned in all stable matchings [25].

(iii) any hospital that is under-subscribed in one stable matching is matched with precisely the same set of residents in all stable matchings [63].

In this chapter, we extend some of these results from HR to SPA. Firstly, we generalize the resident-oriented algorithm for HR to form algorithm SPA-student. For any instance $I$ of SPA, this algorithm finds the student-optimal stable matching of $I$, in which each student is assigned to the best project he/she can have in any stable matching. Secondly, we generalize the hospital-oriented algorithm for HR to form algorithm SPA-lecturer. This algorithm finds the lecturer-optimal stable matching $M$ of $I$. In this matching, each lecturer $l_k$ is assigned the maximum number of students he/she obtains in any stable matching. Also, there is no stable matching $M'$ of $I$ in which $M'(l_k) \setminus M(l_k)$ contains a student preferable to the worst student in $M(l_k)$. Both algorithms have linear time complexity and are therefore asymptotically optimal, since SM, a special case of HR, has a linear-time lower bound [56]. Finally, we generalize the Rural Hospitals Theorem, although some of the properties we discussed above do not hold for SPA.

2.4.1 Overview of Algorithm SPA-student

Algorithm SPA-student begins with the empty assignment, in which all students are free, and every project and lecturer is totally under-subscribed. As long as there is a free student $s_i$ with a non-empty preference list, $s_i$ applies to the first project $p_j$ on his/her preference list. The result of this application is that $s_i$ is provisionally assigned to $p_j$ and $l_k$, where $l_k$ is the lecturer offering $p_j$.

Now, if $p_j$ is over-subscribed, then $l_k$ breaks the provisional assignment between $p_j$ and the worst student $s_r$ assigned to $p_j$. Similarly, if $l_k$ is over-subscribed, then $l_k$ rejects the worst student $s_r$ assigned to $l_k$ under any project $p_t$.

Each iteration of the algorithm finishes with a number of delete operations. We use the phrase *delete the pair* $(s, p)$ to refer to the operation of deleting $p$ from the preference list of $s$, and deleting $s$ from the projected preference list of $p$. These deletions occur in two (possibly non-disjoint) cases. Firstly, if $p_j$ is full, we let $s_r$ be the worst student assigned to $p_j$, and delete any pair $(s_t, p_j)$, where $l_k$ prefers $s_r$ to $s_t$. Secondly, if $l_k$ is full, we let $s_r$ be the worst student assigned to $l_k$, and delete any pair $(s_t, p_u)$, where $l_k$ prefers $s_r$ to $s_t$, and $p_u \in P_k$. 

22
Figure 8 gives a more precise description of algorithm SPA-student. We will now prove that, once the main loop terminates, the assigned pairs constitute a matching, which is both stable and student-optimal.

**SPA-student**

assign each student to be free;
assign each project and lecturer to be totally unsubscribed;

while (some student $s_i$ is free) and ($s_i$ has a non-empty list)

$p_j :=$ first project on $s_i$’s list;
$l_k :=$ lecturer who offers $p_j$;
/* $s_i$ applies to $p_j$ */
provisionally assign $s_i$ to $p_j$; /* and to $l_k$ */
if ($p_j$ is over-subscribed)

$s_r :=$ worst student assigned to $p_j$; /* according to $L^j_k$ */
break provisional assignment between $s_r$ and $p_j$;
else if ($l_k$ is over-subscribed)

$s_r :=$ 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)

$s_r :=$ worst student assigned to $p_j$; /* according to $L^j_k$ */
for (each successor $s_t$ of $s_r$ on $L^j_k$)
delete the pair ($s_t, p_j$);
if ($l_k$ is full)

$s_r :=$ worst student assigned to $l_k$;
for (each successor $s_t$ of $s_r$ on $L_k$)

for (each project $p_u \in P_k \cap A_t$)
delete the pair ($s_t, p_u$);
return $\{(s_i, p_j) \in S \times P : s_i$ is provisionally assigned to $p_j\}$;

Figure 8: Algorithm for finding a student-optimal stable matching.

### 2.4.2 Correctness of Algorithm SPA-student

The correctness of the algorithm, together with the optimality property of the constructed matching, may be established by the following sequence of lemmas.

**Lemma 2.2** Algorithm SPA-student terminates with a matching.

*Proof:* Each iteration involves a free student $s_i$ applying to the first project $p_j$ on his/her preference list. No student can apply to the same project twice,
since, for example, once $s_i$ is freed from $p_j$, the pair $(s_i, p_j)$ is deleted. The total number of iterations is therefore bounded by the overall length of student preference lists. Finally, it is clear that, once the main loop terminates, the assigned pairs constitute a matching.

Lemma 2.3 No pair deleted during an execution of algorithm SPA-student can block the constructed matching.

Proof: Let $E$ be an arbitrary execution of the algorithm in which some pair $(s_i, p_j)$ is deleted. Suppose for a contradiction that $(s_i, p_j)$ blocks $M$, the matching generated by $E$. Now, $(s_i, p_j)$ is deleted in $E$ because either (i) $p_j$ becomes full, or (ii) $l_k$ becomes full, where $l_k$ is the lecturer offering $p_j$. We will show that in Case (i), $(s_i, p_j)$ fails (a), (b) and (c) of Condition 3 of a blocking pair. Case (ii) is easier: $(s_i, p_j)$ cannot block $M$, since once full, a lecturer never becomes under-subscribed, and is only ever assigned more preferable students. We now deal with Case (i), and further consider the three sub-cases of Condition 3 of a blocking pair.

(a) $p_j$ is under-subscribed and $l_k$ is under-subscribed.
Condition (a) requires that $p_j$ subsequently becomes under-subscribed — something that can only happen if $l_k$ becomes over-subscribed and one of his/her assignments involving $p_j$ is broken. However, it is not possible for $l_k$ to subsequently become under-subscribed, contradicting the first clause of Condition (a).

(b) $p_j$ is under-subscribed, $l_k$ is full, and either $l_k$ prefers $s_i$ to the worst student $s'$ in $M(l_k)$, or $s_i = s'$.
Condition (b) requires that $p_j$ becomes under-subscribed at some point after the deletion of $(s_i, p_j)$. Let $(s, p_j)$ be the pair, whose deletion by the over-subscribed $l_k$ results in $p_j$ becoming under-subscribed. Now, $l_k$ prefers $s$ to $s_i$, and by Condition (b), $l_k$ either prefers $s_i$ to $s'$, or $s_i = s'$. It follows then that $l_k$ prefers $s$ to $s'$, and so, immediately after $(s, p_j)$ is deleted, the algorithm will ensure that $(s', M(s'))$ is also deleted. This is a contradiction, since $M$ is a matching of undeleted pairs.

(c) $p_j$ is full and $l_k$ prefers $s_i$ to the worst student $s'$ in $M(p_j)$.
Condition (c) gives us that $l_k$ prefers $s_i$ to $s'$, and since $(s_i, p_j)$ is deleted, $(s', p_j)$ must also be deleted. This is a contradiction, since $M$ is a matching of undeleted pairs.

■
Lemma 2.4 A matching generated by algorithm SPA-student is stable.

Proof: Let $M$ be the matching generated by an arbitrary execution $E$ of the algorithm, and let $(s_i, p_j)$ be any pair blocking $M$. We will show that $(s_i, p_j)$ must be deleted in $E$, thereby contradicting Lemma 2.3. Suppose not. Then $s_i$ must be matched to some project $M(s_i) \neq p_j$, for otherwise $s_i$ is free with a non-empty preference list (containing $p_j$), thereby contradicting the termination property established in Lemma 2.2. Now, when $s_i$ applies to $M(s_i)$, $M(s_i)$ is the first undeleted project on his/her list. Hence, $(s_i, p_j)$ must be deleted, since for $(s_i, p_j)$ to block $M$, $s_i$ must prefer $p_j$ to $M(s_i)$.

For a given instance $I$ of SPA, we say that a (student, project) pair $(s_i, p_j)$ is stable, if $s_i$ is matched with $p_j$ in some stable matching of $I$. The next lemma concerns the deletion of stable pairs in algorithm SPA-student.

Lemma 2.5 No stable pair is deleted during an execution of algorithm SPA-student.

Proof: Suppose for a contradiction that $(s_i, p_j)$ is the first stable pair deleted during an arbitrary execution $E$ of the algorithm. Let $M$ be the matching immediately after the deletion in $E$, and let $M'$ be any stable matching containing $(s_i, p_j)$. Now, $(s_i, p_j)$ is deleted in $E$ because either (i) $p_j$ becomes full, or (ii) $l_k$ becomes full, where $l_k$ is the lecturer offering $p_j$. We consider each case in turn.

(i) Suppose $(s_i, p_j)$ is deleted because $p_j$ becomes full during $E$. Immediately after the deletion, $p_j$ is full, and $l_k$ prefers all students in $M(p_j)$ to $s_i$. Now, $s_i \in M'(p_j) \setminus M(p_j)$, and since $p_j$ is full in $M$, there must be some $s \in M(p_j) \setminus M'(p_j)$. We will show that $(s, p_j)$ forms a blocking pair, contradicting the stability of $M'$.

Firstly, since $(s_i, p_j)$ is the first stable pair deleted in $E$, $s$ prefers $p_j$ to any of his/her stable partners (except possibly for $p_j$ itself). Additionally, since $(s_i, p_j) \in M'$ and $l_k$ prefers $s$ to $s_i$, it follows that $l_k$ prefers $s$ to both the worst student in $M'(p_j)$ and $M'(l_k)$. Clearly then, for any combination of $l_k$ and $p_j$ being full or under-subscribed, $(s, p_j)$ satisfies all the conditions to block $M'$.

(ii) Suppose that $(s_i, p_j)$ is deleted because $l_k$ becomes full during $E$. Immediately after the deletion, $l_k$ is full, and $l_k$ prefers all students in $M(l_k)$ to $s_i$. We consider two cases: $|M'(p_j)| > |M(p_j)|$ and $|M'(p_j)| \leq |M(p_j)|$. Suppose firstly that $|M'(p_j)| > |M(p_j)|$. Since $l_k$ is full in $M$, and $(s_i, p_j) \not\in M$, there must be some project $p \in P_k \setminus \{p_j\}$ such that
\[ |M'(p)| < |M(p)|. \] We remark that \( p \) is therefore under-subscribed in \( M' \). Now, let \( s \) be any student in \( M(p) \setminus M'(p) \). Since \((s, p)\) is the first stable pair deleted, \( s \) prefers \( p \) to any of his/her stable partners (except possibly for \( p \) itself). Also, \( l_k \) prefers \( s \) to \( s_i \), and hence to the worst student in \( M'(l_k) \). So, in either case that \( l_k \) is full or under-subscribed, \((s, p)\) blocks \( M' \).

Now suppose that \( |M'(p)| \leq |M(p)| \). Then there is some \( s \neq s_i \in M(p) \setminus M'(p) \). Now, \( p \) is under-subscribed in \( M \), for otherwise \((s_i, p)\) is deleted because \( p \) becomes full, contradicting the assumption that deletion occurs because \( l_k \) becomes full. Therefore, \( p \) is under-subscribed in \( M' \). As above, \( s \) prefers \( p \) to any of his/her stable partners (except possibly for \( p \) itself), since \((s_i, p)\) is the first stable pair deleted. Also, \( l_k \) prefers \( s \) to \( s_i \), and hence to the worst pair in \( M'(l_k) \). So, in either case that \( l_k \) is full or under-subscribed, \((s, p)\) blocks \( M' \).

The following theorem collects together Lemmas 2.2-2.5.

**Theorem 2.6** For a given instance of SPA, any execution of algorithm SPA-student constructs the student-optimal stable matching.

**Proof:** By Lemma 2.2, let \( M \) be a matching generated by an arbitrary execution \( E \) of the algorithm. In \( M \), each student is assigned to the first project on his/her reduced preference list, if any. By Lemma 2.4, \( M \) is stable, and so each of these (student, project) pairs is stable. Also, by Lemma 2.5, no stable pair is deleted during \( E \). It follows then that in \( M \), each student is assigned to the best project that he/she can obtain in any stable matching.

For example, in the SPA instance given by Figure 6, the student-optimal stable matching is \{((s_1, p_1), (s_2, p_5), (s_3, p_4), (s_4, p_2), (s_7, p_3))\}.

We now show how to implement algorithm-SPA efficiently.

### 2.4.3 Analysis of Algorithm SPA-student

The algorithm’s time complexity depends on how efficiently we can execute ‘apply’ operations and deletions, each of which occur at most once for any (student, project) pair. It turns out that both operations can be implemented to run in constant time, giving an overall time complexity of \( O(\lambda) \), where \( \lambda \) is the total length of all the preference lists. We briefly outline the non-trivial aspects of such an implementation.

For each student \( s_i \), build an array, \( \text{rank}_{s_i} \), where \( \text{rank}_{s_i}(p_j) \) is the index of project \( p_j \) in \( s_i \)’s preference list. Represent \( s_i \)’s preference list by embedding doubly linked lists in an array, \( \text{preference}_{s_i} \). For each project \( p_j \in A_i \),
preference\textsubscript{s\i}(rank\textsubscript{s\i}(p\textsubscript{j})) stores the list node containing \(p\textsubscript{j}\). This node contains two next pointers (and two previous pointers) – one to the next project in \(s\textsubscript{i}'\)s list (after deletions, this project may not be located at the next array position), and another pointer to the next project \(p'\) in \(s\textsubscript{i}'\)s list, where \(p'\) and \(p\textsubscript{j}\) are both offered by the same lecturer. Construct this list by traversing through \(s\textsubscript{i}'\)s preference list, using a temporary array to record the last project in the list offered by each lecturer. Use virtual initialization (described in [11, p.149]) for these arrays, since the overall \(\Theta(nq)\) initialization cost may be super-linear in \(\lambda\). Clearly, using these data structures, we can find and delete a project from a given student in constant time, as well as efficiently delete all projects offered by a given lecturer.

Represent each lecturer \(l\textsubscript{k}'\)s preference list \(L\textsubscript{k}\) by an array \(preference\textsubscript{l\textsubscript{k}}\), with an additional pointer, \(last\textsubscript{l\textsubscript{k}}\). Initially, \(last\textsubscript{l\textsubscript{k}}\) stores the index of the last position in \(preference\textsubscript{l\textsubscript{k}}\). However, once \(l\textsubscript{k}\) is full, make \(last\textsubscript{l\textsubscript{k}}\) equivalent to \(l\textsubscript{k}'\)s worst assigned student through the following method. Perform a backwards linear traversal through \(preference\textsubscript{l\textsubscript{k}}\), starting at \(last\textsubscript{l\textsubscript{k}}\), and continuing until \(l\textsubscript{k}'\)s worst assigned student is encountered (each student stores a pointer to their assigned project, or a special null value if unassigned). All but the last student on this traversal must be deleted, and so the cost of the traversal may be attributed to the cost of the deletions in the student preference lists.

For each project \(p\textsubscript{j}\) offered by \(l\textsubscript{k}\), construct a preference array corresponding to \(L'\textsubscript{k}\). These project preference arrays are used in much the same way as the lecturer preference array, with one exception. When a lecturer \(l\textsubscript{k}\) becomes over-subscribed, the algorithm frees \(l\textsubscript{k}'\)s worst assigned student \(s\textsubscript{i}\) and breaks the assignment of \(s\textsubscript{i}\) to some project \(p\textsubscript{j}\). If \(p\textsubscript{j}\) was full, then it is now under-subscribed, and \(last\textsubscript{p\textsubscript{j}}\) is no longer equivalent to \(p\textsubscript{j}'\)s worst assigned student. Rather than update \(last\textsubscript{p\textsubscript{j}}\) immediately, which could be expensive, wait until \(p\textsubscript{j}\) is full again. The update then involves the same backwards linear traversal described above for \(l\textsubscript{k}\), although we must be careful not to delete pairs already deleted in one of \(l\textsubscript{k}'\)s traversals. Since we only visit a student at most twice during these backwards traversals, once for the lecturer and once for the project, the asymptotic running time remains linear.

The implementation issues discussed above lead to the following conclusion.

**Theorem 2.7** Algorithm SPA-student may be implemented to run in \(\Theta(\lambda)\) time and \(\Theta(mn)\) space, where \(\lambda\) is the total length of the preference lists, \(m\) is the number of projects, and \(n\) is the number of students.
2.4.4 Properties of the Student-Project Allocation Problem

We now prove a result similar to Theorem 2.1, the Rural Hospitals result for HR.

**Theorem 2.8** For a given SPA instance:

(i) each lecturer has the same number of students in all stable matchings;

(ii) exactly the same students are unmatched in all stable matchings;

(iii) a project offered by an under-subscribed lecturer has the same number of students in all stable matchings.

**Proof:** Let $M$ be the student-optimal stable matching, and let $M'$ be any other stable matching.

(i) Suppose $|M'(l_k)| < |M(l_k)|$ for some lecturer $l_k$. There must be some project $p_j \in P_k$ such that $|M'(p_j)| < |M(p_j)|$. So, $l_k$ and $p_j$ are both under-subscribed in $M'$. Also, there exists $s_i \in M(p_j) \setminus M'(p_j)$ who is unmatched in $M'$ or prefers $p_j$ to $M'(s_i)$, since $M$ is student-optimal. Hence, $(s_i, p_j)$ blocks $M'$, and, therefore, $|M'(l_k)| \geq |M(l_k)|$ for all $l_k$. It follows that $|M'| \geq |M|$. However, $|M'| \leq |M|$, since $M$ is student-optimal and therefore matches the maximum number of students of any stable matching. Therefore, $|M'| = |M|$, and for all $l_k$, $|M'(l_k)| = |M(l_k)|$.

(ii) Let $U$ and $U'$ be the sets of students unmatched in $M$ and $M'$ respectively. By Theorem 2.6, $U \subseteq U'$, since no student unmatched in $M$ can be matched in $M'$. But $|U| = |U'|$, by (i), and so it follows that $U = U'$.

(iii) Let $l_k$ be any lecturer under-subscribed in $M'$. Suppose there is some project $p_j \in P_k$ such that $|M'(p_j)| < |M(p_j)|$. So $p_j$ is under-subscribed in $M'$, and there exists $s_i \in M(p_j) \setminus M'(p_j)$ who is unmatched in $M'$ or prefers $p_j$ to $M'(s_i)$. Hence, $(s_i, p_j)$ blocks $M'$, and, therefore, $|M'(p_j)| \geq |M(p_j)|$. Now, by (i) above, $|M'(l_k)| = |M(l_k)|$, and so $|M'(p_j)| = |M(p_j)|$ for all $p_j \in P_k$.

However, it turns out that two key properties of the Rural Hospitals Theorem have no analogue for SPA.

Figure 9 gives a SPA instance, $I_1$, in which a lecturer who is undersubscribed in one stable matching need not obtain the same set of students in
all stable matchings. This contrasts with HR, in which an undersubscribed hospital obtains the same set of residents in all stable matchings.

Instance $I_1$ admits the stable matchings $M = \{(s_1, p_3), (s_2, p_1)\}$ and $M' = \{(s_1, p_1), (s_2, p_3)\}$. Lecturer $l_1$ is under-subscribed in $M$ (and hence in $M'$ by Part (i) of Theorem 2.8). However $M(l_1) = \{s_2\}$ whilst $M'(l_1) = \{s_1\}$.

Figure 10 gives a SPA instance, $I_2$, in which a project offered by a lecturer who is full in one stable matching need not obtain the same number of students in all stable matchings. This contrasts with HR, in which each hospital obtains the same number of residents in all stable matchings.

<table>
<thead>
<tr>
<th>Student preferences</th>
<th>Lecturer preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1: p_1, p_3, p_2, p_4$</td>
<td>$l_1: s_3, s_4, s_1, s_2$</td>
</tr>
<tr>
<td>$s_2: p_1, p_4, p_3, p_2$</td>
<td>$l_2: s_1, s_2, s_3, s_4$</td>
</tr>
<tr>
<td>$s_3: p_3, p_1, p_2, p_4$</td>
<td></td>
</tr>
<tr>
<td>$s_4: p_3, p_2, p_1, p_4$</td>
<td></td>
</tr>
</tbody>
</table>

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

Figure 10: Instance $I_2$ of the Student-Project Allocation problem.

Instance $I_2$ admits the stable matchings $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_3), (s_4, p_3)\}$ and $M' = \{(s_1, p_3), (s_2, p_4), (s_3, p_1), (s_4, p_2)\}$. Lecturer $l_1$ is full in $M$ (and hence in $M'$ by Part (i) of Theorem 2.8). However $M(p_1) = \{s_1, s_2\}$ whilst $M'(p_1) = \{s_3\}$.

2.4.5 Overview of Algorithm SPA-lecturer

Algorithm SPA-lecturer begins with the empty assignment, in which all students are free, and every project and lecturer is totally under-subscribed.
The algorithm then enters a loop, each iteration of which involves an under-subscribed lecturer $l_k$ offering a project $p_j \in P_k$ to a student $s_i$. This student must be the first student on $l_k$’s list that prefers an under-subscribed project in $P_k$ to his/her current provisional assignment. Additionally, $p_j$ must be the first such under-subscribed project from $P_k$ on $s_i$’s preference list. This offer is always accepted, and after breaking any existing assignment involving $s_i$, $s_i$ is provisionally assigned to $p_j$ and $l_k$. Following this assignment, any pair $(s_i, p)$, where $s_i$ prefers $p_j$ to $p$ is deleted, which means that $p$ is removed from $s_i$’s preference list, and $s_i$ is removed from the projected preference list of $l_k$ for $p$. The algorithm continues in this loop until no such $l_k$, $p_j$ and $s_i$ can be found.

Figure 11 gives a more precise description of algorithm SPA-lecturer. We will then prove that, once the main loop terminates, the assigned pairs constitute a matching, which is both stable and lecturer-optimal.

### SPA-lecturer($I$)

assign each student, project and lecturer to be free;

while (some lecturer $l_k$ is under-subscribed) and (there is some (student, project) pair $(s_i, p_j)$ where $s_i$ is not provisionally assigned to $p_j$ and $p_j \in P_k$ is under-subscribed and $s_i \in L^i_k$)

\[
  s_i := \text{first such student on } l_k\text{’s list;}
  p_j := \text{first such project on } s_i\text{’s list;}
  \text{if } (s_i \text{ is provisionally assigned to some project } p) \\
  \quad \text{break the provisional assignment between } s_i \text{ and } p;
  \quad \text{// } l_k \text{ offers } p_j \text{ to } s_i
  \quad \text{provisionally assign } s_i \text{ to } p_j; /* \text{ and to } l_k */
  \quad \text{for each successor } p \text{ of } p_j \text{ on } s_i\text{’s list}
  \quad \text{delete } (s_i, p);
\]

return \{(s_i, p_j) \in S \times P : s_i \text{ is provisionally assigned to } p_j\};

Figure 11: Algorithm for finding a lecturer-optimal stable matching.

### 2.4.6 Correctness of algorithm SPA-lecturer

**Lemma 2.9** Algorithm SPA-lecturer terminates with a matching.

**Proof:** Each iteration involves a provisional assignment: either the first assignment for a student, or an assignment the student prefers to his/her previous assignment. Therefore, the maximum number of iterations is bounded
by the total length of the student preference lists, which is linear in the size of the input. Finally, it is clear that, once the main loop terminates, the assigned pairs constitute a matching.

Lemma 2.10 No pair deleted during an execution of algorithm SPA-lecturer can block the constructed matching.

Proof: Let $E$ be an arbitrary execution of the algorithm in which some pair $(s_i, p_j)$ is deleted. Suppose for a contradiction that $(s_i, p_j)$ blocks $M$, the matching generated by $E$. Now, $(s_i, p_j)$ is deleted because $s_i$ is provisionally assigned to some project $p$, where $s_i$ prefers $p$ to $p_j$. On subsequent iterations, $s_i$ can only improve his/her assignment, and so, by transitivity, $s_i$ prefers his/her final assignment to $p_j$. Therefore, $(s_i, p_j)$ cannot form a blocking pair.

Lemma 2.11 A matching generated by algorithm SPA-lecturer is stable.

Proof: Let $M$ be the matching generated by an arbitrary execution $E$ of the algorithm. Suppose for a contradiction that $M$ is blocked by the pair $(s_i, p_j)$, where $l_k$ is the lecturer offering $p_j$. Now, by Lemma 2.10, we have that $(s_i, p_j)$ is not deleted, and so, after termination, $s_i \in \mathcal{L}_k$. Also, we have that $(s_i, p_j)$ must satisfy (a), (b) or (c) of Condition 3 for a blocking pair. We show a contradiction in each case.

(a) $p_j$ is under-subscribed and $l_k$ is under-subscribed.

Student $s_i$, project $p_j$ and lecturer $l_k$ satisfy the loop condition, contradicting the termination property established in Lemma 2.9.

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

Let $T_1$ be the point in the execution immediately after $s'$ obtains his/her final assignment $p' \in P_k$, and all subsequent deletions involving $s'$ have occurred. Let $M'$ be the matching at $T_1$, and let $B = \{s'\} \cup \{s : l_k$ prefers $s$ to $s'\}$. Define also the following set.

$$F = \left\{ p \in P_k : \begin{array}{l} \text{there exists a student } s_t \in B \text{ such that } p \in A_t, \\ (s_t, p) \notin M' \text{ and } (s_t, p) \text{ is not deleted before } T_1 \end{array} \right\}$$

The following properties of $F$ must hold.
1. Any assignment to $l_k$ after $T_1$ must involve a project from $F$, since $s'$ is the worst student in $M(l_k)$.

2. Every $p \in F$ is full at $T_1$, otherwise $l_k$ would not have offered $p'$ to $s'$.

3. $p_j \in F$, since $l_k$ either prefers $s_i$ to $s'$, or $s_i = s'$ by Condition (b), and $(s_i, p_j)$ is not deleted by Lemma 2.10.

Now since $p_j \in F$, the number of students assigned to $l_k$ in $M'$ is given by

$$|M'(l_k)| = \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M'(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)| \leq d_k \quad (1)$$

The number of students assigned to $l_k$ in $M$ is given by

$$|M(l_k)| = \sum_{p_f \in F \setminus \{p_j\}} |M(p_f)| + |M(p_j)| + \sum_{p_g \in P_k \setminus F} |M(p_g)|$$

Now, since all assignments to $l_k$ after $T_1$ only involve projects from $F$ (Property 1) and all projects in $F$ are full in $M'$ (Property 2), we have that

$$|M(l_k)| \leq \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)|$$

Finally, we are given that $p_j$ is under-subscribed at the termination of $E$ (Condition (b)). Therefore

$$|M(l_k)| < \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M'(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)|$$

$$= |M'(l_k)| \leq d_k$$

So, $l_k$ is under-subscribed at the termination of $E$, contradicting Condition (b).

(c) $p_j$ is full and $l_k$ prefers $s_i$ to the worst student $s'$ assigned to $p_j$.

We have that $l_k$ prefers $s_i$ to $s'$, and so at the time $l_k$ offered $p_j$ to $s'$, $(s_i, p_j)$ must have been deleted (otherwise $l_k$ would have offered $p_j$ to $s_i$). This is a contradiction, since by Lemma 2.10, $(s_i, p_j)$ blocks $M$ only if it is not deleted.
Lemma 2.12  No stable pair is deleted during an execution of algorithm SPA-lecturer.

Proof: Suppose, for a contradiction, that \((s_i, p_j)\) is the first stable pair deleted during an arbitrary execution \(E\) of the algorithm. This deletion occurs because \(s_i\) is provisionally assigned to a project \(p'\), where \(s_i\) prefers \(p'\) to \(p_j\). Let \(l'\) be the lecturer offering \(p'\), and let \(c'\) and \(d'\) be the capacities of \(p'\) and \(l'\) respectively.

Now, the number of stable pairs \((s', p')\) in which \(l'\) prefers \(s'\) to \(s_i\) must be less than \(c'\), for otherwise, one of these pairs must be deleted before \(s_i\) is assigned to \(p'\), contradicting the assumption that \((s_i, p_j)\) is the first stable pair deleted in \(E\). Therefore, in any stable matching without \((s_i, p')\), either (i) \(p'\) is under-subscribed, or (ii) \(p'\) is full and assigned a student inferior to \(s_i\).

Let \(M\) be any stable matching containing \((s_i, p_j)\). We will prove that \((s_i, p')\) blocks any such matching \(M\), contradicting the stability of \((s_i, p_j)\).

Firstly, we have that \(s_i\) prefers \(p'\) to \(p_j\), and so \((s_i, p')\) satisfies Condition 1 and 2 of a blocking pair. It remains to show that \((s_i, p')\) satisfies Condition 3(a), (b) or (c) of a blocking pair.

Now, since \((s_i, p_j) \in M\), it must be the case that \((s_i, p') \notin M\), and so, by the argument above, either (i) or (ii) holds for \(M\). If (ii) holds, then \(p'\) is full and assigned a student inferior to \(s_i\) in \(M\). Therefore, \((s_i, p')\) satisfies Condition 3(c). Otherwise, (i) holds, and \(p'\) is under-subscribed in \(M\).

If \(l'\) is under-subscribed in \(M\), then \((s_i, p')\) satisfies Condition 3(a). Otherwise \(l'\) is full in \(M\), and the only way \((s_i, p')\) cannot satisfy Condition 3(b) is if \(l'\) is assigned \(d'\) students in \(M\), each of whom he/she prefers to \(s_i\). We will show a contradiction for this case.

Since \(M\) is a stable matching, each of these \(d'\) assignments form stable pairs. Now, for \(l'\) to offer \(p'\) to \(s_i\) in \(E\), only \(0 \leq z < d'\) of these stable pairs are assigned (since \(l'\) must be under-subscribed to make an offer). However, none of the \(d'\) stable pairs is deleted before the offer to \(s_i\) in \(E\), for otherwise \((s_i, p_j)\) is not the first stable pair deleted. So, it must be the case that for the \(d' - z\) unassigned stable pairs in \(E\), each of the projects in these pairs is full (otherwise, the next offer from \(l'\) in \(E\) would involve one of the unassigned stable pairs, not \(s_i\) and \(p')\). But then \(l'\) is full when the offer of \(p'\) is made to \(s_i\) in \(E\), giving the required contradiction.

The following theorem collects together Lemmas 2.9-2.12.
Theorem 2.13 For a given instance of SPA, any execution of algorithm SPA-lecturer constructs the stable matching in which (i) every lecturer is assigned the best set of students he/she has in any stable matching, (ii) each project $p_j$, for some integer $h$, is assigned the first $h$ students not deleted from the projected preference list for $p_j$, and (iii) each student is unmatched or assigned the worst project he/she has in any stable matching.

Proof: By Lemma 2.11, let $M$ be the stable matching constructed by an arbitrary execution $E$ of the algorithm. We will prove each statement in turn.

(i) Firstly, we remark that, by Theorem 2.8(i), every lecturer $l_k$ is assigned in $M$ the maximum number of students he/she has in any stable matching.

Now, let $s'$ be the worst student in $M(l_k)$, and let $s_1$ be any student not in $M(l_k)$, where $l_k$ prefers $s_1$ to $s'$. We will show that $s_1$ cannot be assigned to any project offered by $l_k$ in any stable matching.

Suppose for a contradiction that $(s_1, p_1)$ belongs to some stable matching $M'$, where $p_1 \in P_k$. Then, by Lemma 2.12, $p_1$ is in $s_1$'s preference list at the termination of $E$, and $s_1$ is either unmatched in $M$, or prefers $p_1$ to $M(s_1)$.

Now, $p_1$ is full in $M$, for otherwise $(s_1, p_1)$ forms a blocking pair of $M$. Therefore, since $s_1 \not\in M(p_1)$ and $s_1 \in M'(p_1)$, there must be some student $s_2 \in M(p_1) \setminus M'(p_1)$, where, by the stability of $M$, $l_k$ prefers $s_2$ to $s_1$. Now, since $M'$ is stable, $s_2$ must be assigned to a project $p_2$ in $M'$, where $s_2$ prefers $p_2$ to $p_1$.

Project $p_2$ must be full in $M$, for otherwise $(s_2, p_2)$ forms a blocking pair of $M$. Therefore, since $s_2 \not\in M(p_2)$ and $s_2 \in M'(p_2)$, there must be some student $s_3 \in M(p_2) \setminus M'(p_2)$, where, by the stability of $M$, the lecturer offering $p_2$ prefers $s_3$ to $s_2$. Now, since $M'$ is stable, $s_3$ must be assigned to a project $p_3$ in $M'$, where $s_3$ prefers $p_3$ to $p_2$.

This process forms an infinite chain, where in each step, we prove that some student $s_i$ prefers $M'(s_i)$ to $M(s_i)$. We remark that it is possible to select a different student for each step. For example, if $p_3 = p_1$ above, then $|M'(p_1)| \geq 2$, and since $p_1$ is full in $M$, $|M(p_1)| \geq 2$. Therefore, there must be some student $s_4 \neq s_2$ in $M(p_1)$, where $l_k$ prefers $s_4$ to $s_3$. Otherwise, if $p_3 \neq p_1$, we can select $s_4$ to be any student in $M(p_3) \setminus M'(p_3)$.

This gives the required contradiction, since the number of students is finite.
(ii) Suppose there is some (student, project) pair \((s_i, p_j) \notin M\), where \(s_i\) is not deleted from the projected preference list of \(p_j\), and \(l_k\), the lecturer offering \(p_j\), prefers \(s_i\) to the worst student \(s'\) in \(M(p_j)\). Since \((s_i, p_j)\) is not deleted in \(E\), \(s_i\) is either unmatched in \(M\), or prefers \(p_j\) to \(M(s_i)\). So, \((s_i, p_j)\) is a blocking pair, contradicting the stability of \(M\).

(iii) Let \(s_i\) be any student matched in \(M\). Algorithm SPA-lecturer deletes all successors of \(M(s_i)\) from \(s_i\)'s preference list. Now, by Lemma 2.12, no stable pair is deleted, and so \(s_i\) can have no worse partner than \(M(s_i)\) in any stable matching. Hence, each student is either unmatched in \(M\), and therefore in any stable matching (Theorem 2.8), or assigned to the worst project he/she has in any stable matching.

For example, in the SPA instance given by Figure 6, the lecturer-optimal stable matching is \(\{(s_1, p_1), (s_2, p_5), (s_3, p_4), (s_4, p_2), (s_7, p_3)\}\), which, in this case, is the same as the student-optimal stable matching. We now show how to implement algorithm SPA-lecturer efficiently.

### 2.4.7 Analysis of Algorithm SPA-lecturer

Even with the specialized data structures discussed for algorithm SPA-student (Section 2.4.3), it is not immediately obvious that algorithm SPA-lecturer can be implemented in linear time. For example, consider the instance in Figure 12, and the execution trace below.

<table>
<thead>
<tr>
<th>Student preferences</th>
<th>Lecturer preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_1): (p_1) (p_2)</td>
<td>(l_1): (s_2) (s_1) (s_3) (s_4) (s_5)</td>
</tr>
<tr>
<td>(s_2): (p_4) (p_1)</td>
<td>(l_2): (s_2)</td>
</tr>
<tr>
<td>(s_3): (p_2)</td>
<td></td>
</tr>
<tr>
<td>(s_4): (p_3)</td>
<td></td>
</tr>
<tr>
<td>(s_5): (p_1) (p_2) (p_3)</td>
<td></td>
</tr>
</tbody>
</table>

Project capacities: \(c_1 = c_2 = c_3 = c_4 = 1\)

Lecturer capacities: \(d_1 = 3\), \(d_2 = 1\)

Figure 12: An instance of the Student-Project Allocation problem.

(i) \(l_1\) offers \(p_1\) to \(s_2\); \(p_1\) becomes full;
(ii) $l_1$ offers $p_2$ to $s_1$; $p_2$ becomes full;

(iii) $l_1$ offers $p_3$ to $s_4$; $l_1$ and $p_3$ become full;

(iv) $l_2$ offers $p_4$ to $s_2$; $s_2$ is freed from $p_1$; $p_1$ becomes under-subscribed; $l_2$ and $p_4$ become full; $(s_2, p_1)$ is deleted;

(v) $l_1$ offers $p_1$ to $s_1$; $s_1$ is freed from $p_2$; $p_2$ becomes under-subscribed; $p_1$ becomes full; $(s_1, p_2)$ is deleted;

(vi) $l_1$ offers $p_2$ to $s_3$; $p_2$ becomes full;

The sequence of offers made by $l_1$, $\{(s_2, p_1), (s_1, p_2), (s_4, p_3), (s_1, p_1), (s_3, p_2)\}$, reveals two important behaviours of algorithm SPA-lecturer not seen in the hospital-oriented algorithm for HR. Firstly, a lecturer can make more than one offer to the same student ($l_1$ offers both $p_2$ and $p_1$ to $s_1$). Secondly, a lecturer’s sequence of offers may not agree with his/her order of preference ($l_1$ offers $p_3$ to $s_4$ before $s_3$ is made an offer).

Of course, the main reason for both these behaviours is that a given project may be full at one step in the execution, but subsequently may become under-subscribed. For example, at step (ii) in the execution above, $p_1$ is full, and so $s_1$ is assigned to his/her second preference, $p_2$. This means that $p_2$ is now full, and so $s_3$ misses out on any assignment at all. However, $s_2$ subsequently accepts a more preferable project in step (iv), freeing $p_1$ for $s_1$ in step (v), which then frees $p_2$ for $s_3$ in step (vi).

In general, after a partial execution of algorithm SPA-lecturer, $P_k$ may contain several under-subscribed projects that were previously full, where $l_k$ is an under-subscribed lecturer. Consider the set of students $O_k$ that have an under-subscribed project from $P_k$ in their preference list at this point of the execution, and let $s$ be the student to whom $l_k$ last made an offer. Now, $O_k$ may contain several students, some of whom $l_k$ may rank at least as high as $s$, and some of whom $l_k$ may rank lower than $s$. For example, immediately after step (iv) of the execution above, $p_1$ has just become under-subscribed, and so $O_1$ consists of both $s_1$ and $s_5$. The main implementation problem is that, subject to our overall linear time goal, we need to be able to efficiently determine which student $l_k$ ranks highest from $O_k$.

It turns out that we can overcome this problem by restricting the nondeterministic choice of $l_k$ in the main loop. Before outlining this restriction, we define two variables, which will help in the discussion. Also, throughout this discussion, the projected preference list variables are assumed to reflect any deletions made in the execution.
For a given project $p_j$, $next_{pj}$ is the first student in $L_j^i$ not assigned to $p_j$. Note that $next_{pj}$ must be the first student in $L_j^i$ after the worst student assigned to $p_j$. For a given lecturer $l_k$, $next_{lk}$ is the first student $s_i \in O_k$ after the poorest student to whom $l_k$ has offered a project so far.

Initially, $next_{pj}$ and $next_{lk}$ refer to the first student in $L_j^i$ and $L_k^i$ respectively. During an execution of algorithm SPA-lecturer, both variables take on a sequence of values, or students. Importantly, these sequences are ordered according to the original ordering in $L_j^i$ and $L_k^i$ respectively. And, if either of these variables becomes undefined, the variable remains undefined until the end of the execution. It is not too hard to see that, since these variables only traverse their respective preference lists once, we can maintain them within the linear time bound.

Initially, all lecturers $l_k$ make offers to $next_{lk}$. However, whenever a project $p_j \in P_k$ goes from being full to under-subscribed, $l_k$ may prefer $next_{pj}$ to $next_{lk}$, and hence $l_k$’s next offer must be made to $next_{pj}$. Such an offer results in $p_j$ becoming full, and so, at this point, $l_k$ reverts to making offers to $next_{lk}$. Alternatively, if $l_k$ prefers $next_{lk}$ to $next_{pj}$, then student $next_{pj}$ is in the scope of variable $next_{lk}$, and so $l_k$ can revert to simply making offers to $next_{lk}$.

Our implementation of SPA-lecturer allows a non-deterministic choice of $l_k$ in the main loop, with one exception. Suppose a project $p_j \in P_k$ goes from being full to under-subscribed. At this point of the execution, $l_k$’s next offer can only involve one of two students, $next_{pj}$ or $next_{lk}$, a decision that can be made in constant time. If $next_{pj}$ is defined and either $next_{lk}$ is undefined or $l_k$ prefers $next_{pj}$ to $next_{lk}$, then we require that $l_k$ makes an offer to $next_{pj}$ in the next loop iteration. This requirement avoids the problem of deciding between several students for $l_k$’s next offer, which might involve a priority queue, or additional linear search. Figure 13 gives the pseudocode for this implementation of SPA-lecturer.

We briefly outline the data structures used in the linear time implementation. For each student $s_i$, construct a linked list, preference$_{s_i}$, where the $i$th node in preference$_{s_i}$ stores the $i$th ranked project in $s_i$’s preference list. As for algorithm SPA-student, each node has two next pointers (and two previous pointers) - one to the next project in $s_i$’s preference list, and another pointer to the next project on $s_i$’s list offered by the same lecturer.

Using preference$_{s_i}$, we can efficiently find the first under-subscribed project $p_j$ offered by a given lecturer $l_k$, and then delete all successors of $p_j$ on $s_i$’s preference list.

For each lecturer $l_k$, build an array, rank$_{lk}$, where rank$_{lk}(s_i)$ is the index of student $s_i$ in $l_k$’s preference list. We represent $l_k$’s preference list by an array,
SPA-lecturer($l$)
assign each student, project and lecturer to be free;
assign $p$ to be undefined;
while (some lecturer $l_k$ is under-subscribed) and
(two some (student, project) pair ($s_i, p_j$) where
$s_i$ is not provisionally assigned to $p_j$ and
$p_j \in P_k$ is under-subscribed and $s_i \in \mathcal{L}_k^j$) {
if ($p$ is defined)
$\begin{align*}
    p_j &:= p; \\
    l_k &:= \text{lecturer who offers } p_j; \\
    s_i &:= \text{next}_{p_j}; \\
    \text{assign } p \text{ to be undefined;}
\end{align*}$
else
/* next$_{l_k}$ is defined since while loop has not terminated */
$\begin{align*}
    s_i &:= \text{next}_{l_k}; \\
    p_j &:= \text{first under-subscribed project from } P_k \text{ in } s_i\text{'s list;}
\end{align*}$
if ($s_i$ is provisionally assigned to some project $p'$ and lecturer $l'$)
$\begin{align*}
    \text{if (} p' \text{ is full) and (} \text{next}_{p'} \text{ is defined) and} \\
    \text{(} \text{next}_{p'} \text{ is undefined or } l' \text{ prefers } \text{next}_{p'} \text{ to } \text{next}_{l'}) \\
    p &:= p'; \\
    \text{break the provisional assignment between } s_i \text{ and } p'; \\
    \text{provisionally assign } s_i \text{ to } p_j; /* \text{and to } l_k */ \\
    \text{update } \text{next}_{p_j} \text{ and } \text{next}_{l_k}; /* \text{see commentary for details} */
\end{align*}$
for each successor $p'$ of $p_j$ on $s_i$'s list
$\begin{align*}
    \text{delete } (s_i, p'); \\
\end{align*}$
$\text{return } \{(s_i, p_j) \in S \times P : s_i \text{ is provisionally assigned to } p_j\};$

Figure 13: Implementation of algorithm SPA-lecturer.

.preference$_{l_k}$, where preference$_{l_k}(\text{rank}_{l_k}(s_i))$ stores student $s_i$. Each lecturer $l_k$
also stores a count of the number of students to which they are provisionally
assigned, and a pointer next$_{l_k}$ into preference$_{l_k}$, which we described earlier.

For each project $p_j$ offered by $l_k$, build an array rank$_{p_j}$, where rank$_{p_j}(s_i)$
is the index of student $s_i$ in $\mathcal{L}_k^j$. We represent $\mathcal{L}_k^j$ by embedding a
doubly linked list in an array, preference$_{p_j}$. For each student $s_i \in \mathcal{L}_k^j$, preference$_{p_j}(\text{rank}_{p_j}(s_i))$ stores the list node containing $s_i$. This node has a pointer
to the next student in preference$_{p_j}$ and one to the previous student in preference$_{p_j}$. Each project also stores a count of the number of students to
which it is provisionally assigned, and a pointer, next$_{p_j}$, to the first student

38
in preference\(_{p_j}\) not assigned to \(p_j\).

Using these data structures, we can find and delete a student from a project’s preference list in constant time. For each preference list, we can also compare the ranks of any two students, and efficiently traverse through the sequence of students, missing out any students that have been deleted.

The implementation issues discussed above lead to the following conclusion.

**Theorem 2.14** Algorithm \(\text{SPA-lecturer}\) may be implemented to run in \(\Theta(\lambda)\) time and \(\Theta(mn)\) space, where \(\lambda\) is the total length of the preference lists, \(m\) is the number of projects, and \(n\) is the number of students.

### 2.5 Conclusions and Open Problems

In this chapter, we introduced SPA, which is a generalization of HR. We then presented student-oriented and lecturer-oriented algorithms for solving SPA. For any instance \(I\) of SPA, these algorithms return the student optimal and lecturer optimal stable matchings of \(I\) respectively. We also proved an analogue of the Rural Hospitals Theorem for SPA.

A number of open problems remain. For example,

- We can extend the SPA model so that the preference lists of students and lecturers may contain ties. In this context, as with SMT and HRT, there are several possible definitions of stability. It remains open to determine if we can efficiently find stable matchings under these definitions.

- We can extend the SPA model so that lecturers have preferences over (student, project) pairs. In this context, it is an open problem to formulate a definition of stability that avoids the strategic issues described in Section 2.4.

- We can transform an instance \(I\) of HR to an instance \(J\) of SMI such that there is bijection between stable matchings of \(J\) and stable matchings of \(I\). This transformation essentially involves constructing \(c_k\) clones of each hospital \(h_k\), where each clone has capacity 1, and \(c_k\) is the capacity of \(h_k\) (see [1] for more details). So, we can find a stable matching of \(I\) by first transforming \(I\) into \(J\), and then using the Gale/Shapley algorithm to find a stable matching of \(J\). It turns out, however, that it is more efficient to simply use the direct stable matching algorithm for HR, since the cloning transformation may result in a much larger instance of SMI. Here, we ask if there is a similar transformation from SPA to
HR. If there is such a transformation, then we conjecture that using the direct algorithms, SPA-student and SPA-lecturer, will be asymptotically faster than first transforming SPA into HR and then using a direct algorithm for HR. Certainly, this transformation method could never be asymptotically faster, since SPA-student and SPA-lecturer are both asymptotically optimal.
3 Exchange-Stability

3.1 Problem Definition

An instance $I$ of the EXCHANGE-STABLE MATCHING (ESM) problem involves a set $A = \{a_1, a_2, \ldots, a_m\}$ of applicants, and a set $P = \{p_1, p_2, \ldots, p_n\}$ of posts. If an applicant $a_i$ is willing to take a post $p_j$, then we say that $a_i$ finds $p_j$ acceptable, and we denote by $A_i$ the set of all posts that $a_i$ finds acceptable. Each applicant $a_i$ supplies a preference list for $I$ ranking $A_i$ in strict order of preference. Denote by $L$ the total length of all applicant preference lists.

A matching $M$ of $I$ is a subset of $A \times P$ such that

1. $(a_i, p_j) \in M$ implies that $p_j \in A_i$.
2. For each $a_i \in A$, $|(a_i, p_j) \in M : p_j \in P| \leq 1$.
3. For each $p_j \in P$, $|(a_i, p_j) \in M : a_i \in A| \leq 1$.

If $(a_i, p_j) \in M$, then we say that $a_i$ is matched to $p_j$, and $p_j$ is matched to $a_i$. So, an applicant $a_i$ is either unmatched in $M$, or matched to some post, which we denote by $M(a_i)$. Similarly, a post $p_j$ is either unmatched in $M$, or matched to some applicant $M(p_j)$. For exposition purposes, whenever we write $M(a_i)$ (respectively $M(p_j)$), we assume that $a_i$ (respectively $p_j$) is matched in $M$.

Informally, a matching $M$ of $I$ is exchange-stable unless some applicant can be matched to a more preferable post, without requiring some other applicant to be matched to a less preferable post. Formally, a matching $M$ is exchange-stable unless $M$ satisfies at least one of the following blocking conditions.

1. There is some (applicant, post) pair $(a_i, p_j)$ such that $a_i$ and $p_j$ are both unmatched in $M$, and $p_j \in A_i$.
2. There is some (applicant, post) pair $(a_i, p_j)$ such that $a_i$ is matched in $M$, $p_j$ is unmatched in $M$, and $a_i$ prefers $p_j$ to $M(a_i)$.
3. For $q > 1$, there is some applicant sequence $\langle a_1, a_2, \ldots, a_q \rangle \in A^q$ such that $a_i$ prefers $M(a_{i+1})$ to $M(a_i)$, where $1 \leq i < q$, and $a_q$ prefers $M(a_1)$ to $M(a_q)$. We say that such a sequence $\langle a_1, a_2, \ldots, a_q \rangle$ forms a coalition.

A matching said to be (i) maximal if blocking condition 1 does not hold, (ii) trade-in-free if blocking condition 2 does not hold, and (iii) coalition-free if blocking condition 3 does not hold. So, a matching is exchange-stable if and only if it is maximal, trade-in-free and coalition-free.
3.2 Background

Alcalde [3] introduced exchange-stability to deal with situations in which participants have property rights. For example, consider the problem of assigning $2n$ students to $n$ two-bed rooms. Let $M$ be a matching that pairs the students, and assigns the pairs to rooms and beds. Now, suppose that two students, $s_i$ and $s_j$, prefer each other to their partners in $M$. Although $M$ is not stable in the classical sense (see Section 1.3.5), there is no separate room for $s_i$ and $s_j$ to occupy, and, if both $M(s_i)$ and $M(s_j)$ exercise their property rights by refusing to swap rooms, then $s_i$ and $s_j$ cannot change their allocation. However, we can certainly say that $M$ is not stable against changes if $s_i$ prefers $M(s_j)$ and $s_j$ prefers $M(s_i)$, since in this case, $s_i$ and $s_j$ may swap beds. Clearly, $s_i$ and $s_j$ form a coalition for $M$.

More recently, Cechlárová and Manlove [13] have studied exchange-stability in the context of SM. In their work, (i) matchings are complete (so every matching is necessarily maximal and trade-in-free), (ii) coalitions have size 2, and (iii) both the set of men and the set of women must be exchange-stable. Under this definition, they prove that determining if an instance of SM admits an exchange-stable matching is NP-complete. Furthermore, they restrict their definition of stability to apply only to men, and prove that every instance of SM admits a man-exchange-stable matching. Additionally, they give an algorithm for finding a maximum cardinality man-exchange-stable matching when the instance has incomplete preference lists.

3.3 Preliminary Results and Observations

3.3.1 Checking Exchange-Stability

Let $I$ be an instance of ESM. We can determine if a matching $M$ of $I$ is exchange-stable by determining in turn if $M$ is maximal, trade-in-free and coalition-free. It is trivial to test the first two blocking conditions; we only remark that both tests can be performed in $O(L)$ time, where $L$ is the length of the applicant preference lists. Determining if $M$ is coalition-free is less trivial.

The preference graph $G$ of $M$ in $I$ consists of one vertex for each applicant, with a directed edge $(a_i, a_j)$ between any two applicants $a_i$ and $a_j$, where either $a_i$ is unmatched in $M$ and $M(a_j) \in A_i$, or $a_i$ prefers $M(a_j)$ to $M(a_i)$. It is not too hard to see that there is a bijective correspondence between coalitions in $M$ and cycles in $G$. Therefore, $M$ is coalition-free if and only if $G$ is acyclic. We can test if $G$ is acyclic by using any cycle detection algorithm, such as depth-first search. This test takes $O(L)$ time.

We summarize the preceding discussion in the following theorem.
Theorem 3.1 Let $M$ be a matching of some instance $I$ of ESM. We can determine if $M$ is exchange-stable in $O(L)$ time, where $L$ is the length of the applicant preference lists.

3.3.2 Existence of Exchange-Stable Matchings

In this section, we show that it possible to find an exchange-stable matching for any instance of ESM in linear time.

Theorem 3.2 Every instance of ESM admits an exchange-stable matching, which can be found in $O(L)$ time, where $L$ is the length of the applicant preference lists.

Proof: Let $I$ be any instance of ESM, and let $M$ be the set returned by an arbitrary execution $E$ of algorithm Greedy-ESM (see Figure 14) on $I$. We will show that $M$ is an exchange-stable matching.

\begin{verbatim}
Greedy-ESM(I = (A, P))
    assign each applicant and post to be unmatched;
    $M := \emptyset$;
    for each applicant $a_i \in A$
        if $A_i$ contains an unmatched post
            $p_j :=$ first unmatched post in $A_i$;
            /* match $a_i$ with $p_j$ */
            $M := M \cup (a_i, p_j)$;
    return $M$;
\end{verbatim}

Figure 14: Algorithm Greedy-ESM.

It is clear from the algorithm description that the set $M$ is a matching of $I$. $M$ must be maximal, since if an applicant $a_i$ is unmatched in $M$, then $A_i$ contains no unmatched posts. Furthermore, $M$ must be trade-in-free, since whenever an applicant $a_i$ is matched with $M(a_i)$ in $E$, every post $p_j$ that $a_i$ prefers to $M(a_i)$ has already been matched. We will now show that $M$ is coalition-free.

Suppose for a contradiction that $M$ admits the coalition $C = \langle a_1, a_2, ..., a_q \rangle$. Without loss of generality, assume that $C$ has been cyclically rotated so that $a_1$ is the first applicant in $C$ to be matched during $E$. Now, since $C$ is a coalition, $a_1$ prefers $M(a_2)$ to $M(a_1)$. But, at the time $a_1$ is matched to $M(a_1)$ in $E$, $M(a_2)$ is unmatched. This contradicts the fact that Greedy-ESM matches each applicant with the first unmatched post on his/her preference list, if
any. Hence, $M$ is coalition-free, and therefore, $M$ is also an exchange-stable matching of $I$.

Finally, it is not too hard to see that Greedy-ESM runs in $O(L)$ time, since the algorithm makes at most one complete traversal of the applicant preference lists.

Roth and Sotomayor [64, Example 4.3] prove that if Greedy-ESM is used as a centralized matching mechanism, then no applicant can improve his/her final allocation by strategically misrepresenting his/her preferences. Cechlárová and Manlove [13] were the first to prove that Greedy-ESM returns an exchange-stable matching, although in their work, coalitions must contain exactly two applicants, and all preference lists are complete (see below).

Roth and Sotomayor [64, Example 4.3] also remark that a variant of Greedy-ESM is used by the United States Naval Academy to assign graduating students to their first post as a Naval officer. This variant differs from Greedy-ESM in that the main loop is deterministic - students are given the opportunity to select a post in non-decreasing order of graduation results. We briefly revisit a generalization of this variant in Theorem 3.22, where we prove that, given any exchange-stable matching $M$, there exists an execution of Greedy-ESM that will return $M$.

### 3.3.3 Sizes of Exchange-Stable Matchings

We say that an applicant $a_i$ has a complete preference list if $A_i = P$. If every applicant in some instance $I$ of ESM has a complete preference list, then we say that $I$ is an instance of ESM with complete lists. The next proposition proves that, for such an instance $I$, all exchange-stable matchings of $I$ have the same size.

**Proposition 3.3** Let $I$ be an instance of ESM with complete lists, where $A$ and $P$ are the sets of applicants and posts of $I$ respectively. Then the cardinality of every exchange-stable matching of $I$ is $\min(|A|, |P|)$.

**Proof:** Let $M$ be any exchange-stable matching of $I$, and suppose for a contradiction that $|M| < \min(|A|, |P|)$. Since fewer than $|A|$ applicants are matched in $M$, there must be some applicant $a_i$ who is unmatched in $M$. Similarly, there must be some post $p_j$ that is unmatched in $M$. Now, $A_i = P$, and so $p_j$ is a member of $A_i$. Therefore $M$ is not maximal, contradicting the exchange-stability of $M$. ■
Let $I$ be an instance of ESM in which some applicant $a_i$ has an incomplete preference list (i.e. $A_i \subset P$). It turns out that $I$ may admit exchange-stable matchings of different cardinalities, and that Algorithm Greedy-ESM may not find the largest such matching. Consider, for example, the instance in Figure 15, with applicant set $A = \{a_1, a_2, a_3\}$, and post set $P = \{p_1, p_2, p_3\}$. It is not too hard to see that the only exchange-stable matchings of this instance are $M_1 = \{(a_1, p_1)\}$, $M_2 = \{(a_1, p_2), (a_2, p_1)\}$ and $M_3 = \{(a_1, p_2), (a_3, p_1)\}$. We remark that $M_1$ has smaller cardinality than $M_2$ and $M_3$, and that $|M_2| = |M_3| < \min(|A|, |P|)$. Also, note that if Greedy-ESM selects $a_1$ as the first applicant in the main loop, then the returned matching will be $M_1$, which is not the largest exchange-stable matching.

\[
\begin{align*}
  a_1 & : p_1 \ p_2 \\
  a_2 & : p_1 \\
  a_3 & : p_1
\end{align*}
\]

Figure 15: An instance of ESM.

We summarize the preceding discussion in the following proposition.

**Proposition 3.4** An instance $I$ of ESM may admit exchange-stable matchings of different cardinalities.

**Corollary 3.5** The set of applicants unmatched in one exchange-stable matching may not be the same as the set of applicants unmatched in another exchange-stable matching.

### 3.4 Maximum Cardinality Exchange-Stable Matchings

In this section, we present three different algorithms for finding a maximum cardinality exchange-stable matching. We then prove that such a matching has maximum cardinality among all matchings, even those which are not exchange-stable. Finally, we prove an interpolation result on the cardinalities of exchange-stable matchings.

Firstly, we introduce some new terminology. Let $I$ be an instance of ESM with applicant set $A$ and post set $P$. The underlying graph $G = (A \cup P, E)$ of $I$ is the bipartite graph with left vertex set $A$, right vertex set $P$, and edge set $E = \{(a_i, p_j) \subseteq A \times P : p_j \in A_i\}$. Additionally, associated with each edge $(a_i, p_j) \in E$ is a weight, which is the rank of $p_j$ in $a_i$'s preference list.
The first algorithm we present, Stabilize-ESM, is based on the exchange-stability checking algorithm informally described in Section 3.3.1. This algorithm begins by finding a maximum cardinality matching \( M \) of \( I \), which may not be exchange-stable. The algorithm then proceeds through two additional phases.

In the second phase, the algorithm repeatedly finds (applicant, post) pairs \((a_i, p_j)\) that cause \( M \) to satisfy blocking condition 2. Whenever such a pair is found, the algorithm breaks the existing assignment involving \( a_i \), and proceeds to match \( a_i \) to \( p_j \). In the final phase, the algorithm repeatedly constructs the preference graph \( G \) of \( M \), rotating the partners of any coalition found in these graphs. More formally, in a rotation of coalition \( C = \langle a_1, a_2, \ldots, a_q \rangle \), we partner \( a_i \) with \( M(a_{i+1}) \) for all \( 1 \leq i \leq q \) (where \( a_{q+1} = a_1 \)), after first breaking all original assignments involving applicants from \( C \).

Stabilize-ESM\((I = (A, P))\)

assign each applicant and post to be unmatched;
\( G := \) underlying graph of \( I \);
\( M := \) any maximum matching of \( G \);
/* \( M \) is maximal */

while there exists an applicant \( a_i \) matched in \( M \) and
a post \( p_j \) unmatched in \( M \), where \( a_i \) prefers \( p_j \) to \( M(a_i) \)
\( M := M \setminus \{(a_i, M(a_i))\} \);
\( M := M \cup \{(a_i, p_j)\} \);
/* \( M \) is trade-in-free */
\( H := \) preference graph of \( M \);

while there exists a cycle \( C \) in \( H \)
/* \( C \) represents coalition for \( M \)*/
Cyclically rotate the posts assigned to applicants in \( C \);
\( H := \) preference graph of \( M \);
/* \( M \) is coalition-free */
return \( M \);

Figure 16: Algorithm Stabilize-ESM.

**Theorem 3.6** Let \( I \) be an instance of ESM with \( m \) applicants and \( n \) posts. Then Stabilize-ESM returns a maximum cardinality exchange-stable matching of \( I \) in \( O(L^2) \) time, where \( L \) is the total length of the applicant preference lists.
Proof: In the first phase of Stabilize-ESM, we find a maximum matching \( M \) of \( G \). Every applicant that is matched in \( M \) remains matched throughout the algorithm. So, the returned matching has maximum cardinality, and must therefore be maximal. This phase takes \( O(\sqrt{\min(m,n)L}) \) time.

In the second phase of Stabilize-ESM, we check to see if any applicant \( a_i \) can trade-in his/her existing post for a unmatched post \( p_j \), where \( a_i \) prefers \( p_j \) to \( M(a_i) \). It is clear that finding such a pair \((a_i, p_j)\), or proving that no such pair exists, takes \( O(L) \) time. We need to repeat this at most \( L \) times, since each trade-in results in some applicant improving his/her assignment, and the total length of the preference list is \( L \). Therefore, it takes \( O(L^2) \) time to ensure that \( M \) is trade-in-free. We remark that every post that is (un)matched in \( M \) at this point remains (un)matched after the third phase of the algorithm (since the final phase only involves partner swapping). Hence, the returned matching is trade-in-free.

In the final phase of the algorithm, we repeatedly check for the existence of a coalition in \( M \). This check takes \( O(L) \) time, and since every member of the coalition improves his/her allocation, the maximum number of iterations is \( O(L) \). Hence, it takes \( O(L^2) \) time to ensure that \( M \) is coalition-free.

Therefore, Algorithm Stabilize-ESM returns a maximum cardinality exchange-stable matching of \( I \) in \( O(L^2) \) time. Finally, we remark that since the returned matching has maximum cardinality of all matchings of \( I \), it must also have maximum cardinality of all exchange-stable matchings of \( I \).

Corollary 3.7 Let \( I \) be an instance of ESM with applicant set \( A \) and post set \( P \). The size of a maximum cardinality exchange-stable matching of \( I \) is equal to the size of a maximum cardinality matching of \( I \).

The second algorithm we present is based on the observation in Theorem 3.8 that any minimum weight maximum cardinality matching of an underlying graph \( G \) is also a maximum cardinality exchange-stable matching in the corresponding instance \( I \) of ESM. Let \( p = n + m \) and \( L \) be the number of vertices and edges in \( G \) respectively. We can find a minimum weight maximum cardinality matching of \( G \) in \( O(p(L + p \log p)) \) time, using the algorithm due to Fredman and Tarjan [19].

Theorem 3.8 Let \( G \) be the underlying graph of some instance \( I \) of ESM. Any minimum weight maximum cardinality matching of \( G \) is a maximum cardinality exchange-stable matching of \( I \).

\[ \text{Corollary 3.7 Let } I \text{ be an instance of ESM with applicant set } A \text{ and post set } P. \text{ The size of a maximum cardinality exchange-stable matching of } I \text{ is equal to the size of a maximum cardinality matching of } I. \]

The second algorithm we present is based on the observation in Theorem 3.8 that any minimum weight maximum cardinality matching of an underlying graph \( G \) is also a maximum cardinality exchange-stable matching in the corresponding instance \( I \) of ESM. Let \( p = n + m \) and \( L \) be the number of vertices and edges in \( G \) respectively. We can find a minimum weight maximum cardinality matching of \( G \) in \( O(p(L + p \log p)) \) time, using the algorithm due to Fredman and Tarjan [19].

\[ \text{Theorem 3.8 Let } G \text{ be the underlying graph of some instance } I \text{ of ESM. Any minimum weight maximum cardinality matching of } G \text{ is a maximum cardinality exchange-stable matching of } I. \]

\[ \text{Corollary 3.7 Let } I \text{ be an instance of ESM with applicant set } A \text{ and post set } P. \text{ The size of a maximum cardinality exchange-stable matching of } I \text{ is equal to the size of a maximum cardinality matching of } I. \]
Proof: Let $M$ be a minimum weight maximum cardinality matching of $G$. Since $M$ is a maximum matching, we immediately have that $M$ is maximal.

Suppose for a contradiction that $M$ is not trade-in-free due to some (applicant, post) pair $(a_i, p_j)$. Consider the matching $M' = (M \setminus (a_i, M(a_i))) \cup (a_i, p_j)$. We have that $|M'| = |M|$, and so $M'$ is a maximum cardinality matching of $G$. Also, since $a_i$ prefers $p_j$ to $M(a_i)$, the weight of $M'$ is smaller than the weight of $M$, contradicting the assumption that $M$ has minimum weight among all maximum cardinality matchings of $G$.

Now, suppose for a contradiction that $M$ admits a coalition $C = \{a_1, a_2, \ldots, a_q\}$. Consider the matching $M' = (M \setminus \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_q, M(a_q))\}) \cup \{(a_1, M(a_2)), (a_2, M(a_3)), \ldots, (a_q, M(a_1))\}$. The same contradiction follows, since $M'$ is a maximum matching of $G$ with smaller weight than $M$ (each applicant in $C$ is matched to a more preferable post in $M'$ than in $M$).

The third algorithm we present, Lex-ESM, begins with an arbitrary exchange-stable matching $M$, and repeatedly augments $M$ by matching an additional applicant and post (though not in general to each other). Lex-ESM is therefore a classical augmenting path algorithm, although here, we require that the augmenting paths preserve the exchange-stability from one matching to the next.

In the following discussion, a string $S = \langle s_1, s_2, \ldots, s_q \rangle$ is a fixed permutation of some underlying set $\{s_1, s_2, \ldots, s_q\}$. We denote by $S[i]$ the $i$th ranked element in $S$, where $S[1]$ is the first element and $|S|$ is the length of $S$. A string $S'$ is a substring of $S$ if there exists some integer $b$ such that $S'[i] = S[b + i]$ for all $1 \leq i \leq |S'|$. We define the term Prefix($S$, $s$) to be the substring $\langle S[1], S[2], \ldots, S[k] \rangle$ of $S$, where $S[k] = s$. Similarly, we define Suffix($S$, $s$) to be the substring $\langle S[k], S[k + 1], \ldots, S[|S|] \rangle$ of $S$, where $S[k] = s$. Note that these two terms are well-defined since all elements of $S$ are distinct. Given two strings $S = \langle s_1, s_2, \ldots, s_q \rangle$ and $T = \langle t_1, t_2, \ldots, t_l \rangle$, where the intersection of the underlying sets is empty, the concatenation of $S$ and $T$ is the string $S \cdot T = \langle s_1, s_2, \ldots, s_q, t_1, t_2, \ldots, t_l \rangle$.

Recall that an alternating string $A = \langle a_1, p_1, \ldots, a_q, p_q \rangle$ of applicants and posts is an augmenting path of some matching $M$ if

(i) $a_1$ is an unmatched applicant in $M$,

(ii) $p_q$ is an unmatched post in $M$

(iii) $p_i = M(a_{i+1})$ for all $1 \leq i \leq q - 1$,

(iv) $a_i$ finds $p_i$ acceptable for all $1 \leq i \leq q$. 

48
We denote by $\text{Aug}(M)$ the set of all augmenting paths with respect to $M$. For each applicant $a_i$, denote by $\text{Aug}(M, a_i)$ the set \{\(\Lambda \in \text{Aug}(M) : \Lambda[1] = a_i\)\}. Let $\Lambda = \langle a_1, p_1, \ldots, a_q, p_q \rangle$ be any augmenting path in $\text{Aug}(M)$. We associate with $\Lambda$ a vector $\text{rank}(\Lambda)$, which consists of the ranks of $a_i$ for $1 \leq i \leq q$. Also, we define a strict partial ordering on $\text{Aug}(M)$: for all $\Lambda, \Lambda' \in \text{Aug}(M)$, $\Lambda < \Lambda'$ if and only if $\Lambda, \Lambda' \in \text{Aug}(M, a_i)$, for some $i$, and $\text{rank}(\Lambda)$ is lexicographically smaller than $\text{rank}(\Lambda')$. Finally, for an augmenting path $\Lambda \in \text{Aug}(M)$, we denote by $M \oplus \Lambda$ the matching $M$ augmented by $\Lambda$.

**Lex-ESM(I)**

\[
M := \text{any exchange-stable matching of } I; \\
\textbf{while } \text{Aug}(M) \neq \emptyset \\
\quad \Lambda := \text{minimal element of } \text{Aug}(M); \\
\quad M := M \oplus \Lambda;
\]

Figure 17: Algorithm for Lex-ESM.

Let $I$ be an instance of ESM, and let $M$ be any exchange-stable matching of $I$. If $M$ is not a maximum cardinality matching, then we know from basic augmenting path theory that $\text{Aug}(M) \neq \emptyset$. Let $\Lambda$ be any minimal element of $\text{Aug}(M)$, and consider the matching $M' = M \oplus \Lambda$.

**Lemma 3.9** $M' = M \oplus \Lambda$ is a maximal matching of $I$.

**Proof:** Suppose for a contradiction that $M'$ is not maximal due to some (applicant, post) pair $(a_i, p_j)$. Now, since $a_i$ and $p_j$ are unmatched in $M'$, they must both be unmatched in $M$. This means $M$ is not maximal, giving the required contradiction.

**Lemma 3.10** $M' = M \oplus \Lambda$ is a trade-in-free matching of $I$.

**Proof:** Suppose for a contradiction that $M'$ is not trade-in-free due to some (applicant, post) pair $(a_i, p_j)$. Let $\Lambda'$ be the string $\langle a_i, p_j \rangle$ if $a_i \notin \Lambda$, or $\text{Prefix}(\Lambda, a_i), \langle p_j \rangle$ otherwise. Now, since $p_j$ is unmatched in $M'$, $p_j$ is also unmatched in $M$, and so $p_j \notin \Lambda$. Therefore, $\Lambda' \in \text{Aug}(M)$, and since $a_i$ prefers $p_j$ to $M'(a_i)$, $\Lambda' < \Lambda$. This contradicts the minimality of $\Lambda$.

**Lemma 3.11** $M' = M \oplus \Lambda$ is a coalition-free matching of $I$.
Proof: Suppose for a contradiction that \( M' \) admits a coalition \( C = \langle a_1, a_2, \ldots, a_q \rangle \).

At least one applicant from \( C \) must also be in \( \Lambda \), for otherwise \( M \) also admits \( C \). Let \( a_i \) be the first applicant in \( \Lambda \) who is also in \( C \), and for the following argument, any applicant \( a_{kq+t} \) refers to \( a_t \), where \( k \) is some integer, and \( 1 \leq t \leq q \).

Since \( a_i \in C \), we have that \( a_i \) prefers \( M'(a_{i+1}) \) to \( M'(a_i) \). Now, \( M'(a_{i+1}) \) cannot be unmatched in \( M \), for otherwise \( M \) admits the augmenting path \( \Lambda' = \text{Prefix}(\Lambda, a_i) \cdot \langle M'(a_{i+1}) \rangle \), which is less than \( \Lambda \). Also, \( M'(a_{i+1}) \) cannot appear in \( \Lambda \) before \( a_i \), for otherwise \( a_{i+1} \) precedes \( M'(a_{i+1}) \) in \( \Lambda \), and \( a_i \) is not the first applicant in \( \Lambda \) who is also in \( C \). Furthermore, \( M'(a_{i+1}) \) cannot appear in \( \Lambda \) after \( a_i \), for otherwise \( M \) admits the augmenting path \( \Lambda' = \text{Prefix}(\Lambda, a_i) \cdot \text{Suffix}(\Lambda, M'(a_{i+1})) \), which is less than \( \Lambda \).

So, it must be the case that \( M'(a_{i+1}) \) is matched in \( M \) and does not appear in \( \Lambda \). Now, consider the string \( S = \langle M'(a_{i+1}), a_{i+1}, \ldots, M'(a_{i+j-1}), a_{i+j-1} \rangle \), where \( a_{i+j} \) is the first applicant after \( a_i \) in \( C \) that is also in \( \Lambda \). Note, there must exist such an applicant \( a_{i+j} \), since \( a_i \in \Lambda \) and \( a_{i+q} = a_i \). The string \( S \) has the following properties, the last two of which mirror the final two properties of an augmenting path.

(i) The intersection of the underlying sets of \( S \) and \( \Lambda \) is empty, since an applicant \( a \) is not a member of \( \Lambda \) if and only if \( M'(a) \) is not a member of \( \Lambda \).

(ii) For all \( 1 \leq k < j \), \( M'(a_{i+k}) \) is matched with \( a_{i+k} \) in \( M \).

(iii) For all \( 1 \leq k < j \), \( a_{i+k} \) prefers \( M'(a_{i+k+1}) \) to \( M'(a_{i+k}) \), since \( C \) is a coalition for \( M' \).

Now, if \( a_i = a_{i+j} \), then \( M'(a_{i+j}) \) appears after \( a_i \) in \( \Lambda \). This is true also if \( a_i \neq a_{i+j} \), for otherwise, since \( a_{i+j} \) precedes \( M'(a_{i+j}) \) in \( \Lambda \), \( a_i \) would not be the first applicant in \( \Lambda \) to appear in \( C \). So, the string \( \Lambda' = \text{Prefix}(\Lambda, a_i) \cdot S \cdot \text{Suffix}(\Lambda, M'(a_{i+j})) \) forms a valid augmenting path. Finally, since \( a_i \) prefers \( M'(a_{i+1}) \) to \( M'(a_i) \), \( \Lambda' \) is less than \( \Lambda \), contradicting the minimality of \( \Lambda \).

We now show how to efficiently find a minimal element of \( \text{Aug}(M) \). Construct the underlying graph \( G \) of \( I \). For each edge \( (a_i, p_j) \), if \( M(a_i) = p_j \), then replace this edge with \( (p_j, a_i) \). Recall that, in general, an augmenting path from \( \text{Aug}(M) \) can be found by performing a depth-first search (DFS) of \( G \), where each tree in the resulting forest is rooted by an applicant unmatched in \( M \). It is not too hard to see that we can find a minimal element of \( \text{Aug}(M) \) by performing a well-known variant of depth-first search, which we call ordered depth-first search (ODFS). During this search, whenever we visit
a_i, the next post visited must currently be unvisited, as in DFS, and, subject to this constraint, the post must be the endpoint of the smallest weight edge out of a_i.

In general graphs, ODFS is asymptotically slower than DFS, since ODFS has the added overhead of finding a smallest weight edge (as opposed to any edge). However, if the graph is already represented as an adjacency list, where for each vertex v, v’s adjacency list stores the edges out of v in non-decreasing order of weight, then the runtime of ODFS is the same as DFS. We can build G in this way in only linear time, since the preference list of each applicant is already given to us in non-decreasing order of rank. Hence, given a matching M, we can find a minimal element of Aug(M) or show that Aug(M) = ∅ in O(L) time, where L is the total length of the applicant preference lists.

The following theorem collects together Lemmas 3.9 to 3.11, as well as the preceding discussion on the time complexity of finding a minimal element of Aug(M).

**Theorem 3.12** For an arbitrary instance I of ESM, Lex-ESM returns a maximum cardinality exchange-stable matching of I in O(min(m, n)L) time, where m and n are the numbers of applicants and posts in I, and L is the total length of the applicant preference lists.

**Theorem 3.13** Let M^- and M^+ be minimum and maximum cardinality exchange-stable matchings of some instance I of ESM. There exist exchange-stable matchings of I of all cardinalities between |M^-| and |M^+|.

**Proof:** Let E be an execution of Lex-ESM on I, beginning from the matching M = M^- . On each iteration of E, Lex-ESM generates a new matching M ⊕ Δ of I, where |M ⊕ Δ| = |M| + 1, and by Lemmas 3.9-3.11, M ⊕ Δ is exchange-stable. This process continues until Lex-ESM generates a maximum cardinality exchange-stable matching of I (Theorem 3.12). Hence, there exist exchange-stable matchings of I of all cardinalities between |M^-| and |M^+|.

### 3.5 Uniqueness and Applicant-Optimality

Let M be an exchange-stable matching of some instance I of ESM. We say that M is unique if I admits no exchange-stable matching other than M. In this section, we give a polynomial-time characterization of the set of ESM instances that admit a unique exchange-stable matching.
Let $M_1$ and $M_2$ be any two matchings of some instance $I$ of ESM. We say that an applicant $a_i$ prefers $M_1$ to $M_2$ if

(i) $a_i$ is matched in $M_1$, and

(ii) $a_i$ is unmatched in $M_2$, or $a_i$ prefers $M_1(a_i)$ to $M_2(a_i)$.

We say that an exchange-stable matching $M$ of $I$ is applicant-optimal if every applicant either prefers $M$ to any other exchange-stable matching of $I$, or is indifferent between them. The following lemma will help us show that there is a close connection between uniqueness and applicant-optimality.

**Lemma 3.14** Let $M_1 \neq M_2$ be any two exchange-stable matchings of some instance $I$ of ESM with applicant set $A$ and post set $P$. Then, at least one applicant in $A$ must prefer $M_1$ to $M_2$, and at least one other applicant in $A$ must prefer $M_2$ to $M_1$.

**Proof:** Let $A_1 = \{a_i \in A : a_i$ prefers $M_1$ to $M_2\}$, and let $A_2 = \{a_i \in A : a_i$ prefers $M_2$ to $M_1\}$. For any $A' \subseteq A$, and for any matching $M$ of $I$, denote by $P(A', M)$ the set $P(A', M) = \{p_j \in P : M(p_j) \in A'\}$.

Now, since $M_1 \neq M_2$, it must be the case that, without loss of generality, $A_1 \neq \emptyset$. Suppose for a contradiction that $A_2 = \emptyset$. We will prove that a subset of $A_1$ forms a coalition for $M_2$.

Firstly, we show that $P(A_1, M_1) \subseteq P(A_1, M_2)$. Suppose for a contradiction that there exists a post $p_j \in P(A_1, M_1) \setminus P(A_1, M_2)$. We remark that $p_j \notin P(A_2, M_2)$, since $A_2 = \emptyset$. Also, $p_j \notin P(A \setminus \{A_1 \cup A_2\}, M_2)$, since $P(A \setminus \{A_1 \cup A_2\}, M_2) = P(A \setminus \{A_1 \cup A_2\}, M_1)$, and $p_j \in P(A_1, M_1)$. Therefore, $p_j$ is unmatched in $M_2$.

Now, since $p_j \in P(A_1, M_1)$, $p_j$ is matched in $M_1$ with some applicant $a_i \in A_1$. It follows that $a_i$ prefers $M_1$ to $M_2$, and so either (i) $a_i$ is unmatched in $M_2$, in which case $M_2$ is not maximal due to the pair $(a_i, p_j)$, or (ii) $a_i$ is matched to some post $M_2(a_i)$, where $a_i$ prefers $p_j$ to $M_2(a_i)$. Thus, $M_2$ is not trade-in-free due to $(a_i, p_j)$, giving the required contradiction. Hence, $P(A_1, M_1) \subseteq P(A_1, M_2)$.

Now, $A_1 \neq \emptyset$, so there must be some $a_i \in A_1$, who prefers $M_1$ to $M_2$ and is therefore matched to some $p_1$ in $M_1$. So, $p_1 \in P(A_1, M_1)$, and, furthermore, $p_1 \in P(A_1, M_2)$, since $P(A_1, M_1) \subseteq P(A_1, M_2)$. Hence, there exists an $a_1 \in A_1$ such that $(a_1, p_1) \in M_2$. Now, since $a_1 \in A_1$, $a_1$ prefers $M_1$ to $M_2$ and must therefore be matched to some $p_2$ in $M_1$. So, $p_2 \in P(A_1, M_1)$, and, furthermore, $p_2 \in P(A_1, M_2)$, since $P(A_1, M_1) \subseteq P(A_1, M_2)$. Hence, there exists an $a_2 \in A_1$ such that $(a_2, p_2) \in M_2$.

Eventually, this process must cycle. It is easy to see that the applicants in this cycle form a coalition for $M_2$, giving the required contradiction.
Corollary 3.15  An exchange-stable matching $M$ is unique if and only if $M$ is applicant-optimal.

The next lemma leads to a different characterization of applicant-optimality.

Lemma 3.16 Let $I$ be an instance of ESM with applicant set $A$ and post set $P$, and let $a_i \in A$ be any applicant with $A_i \neq \emptyset$. Then there is some exchange-stable matching $M$ of $I$ such that $a_i$ is matched in $M$ with his/her first-choice post.

Proof: We can find such a matching $M$ for $a_i$ by running Greedy-ESM on $I$ and forcing $a_i$ to be the first applicant to be assigned a post.

Corollary 3.17  An exchange-stable matching $M$ is applicant-optimal if and only if every applicant $a_i$ with $A_i \neq \emptyset$ is matched in $M$ with his/her first-choice post.

We summarize the preceding results in the following theorem.

Theorem 3.18 Let $M$ be an exchange-stable matching of some instance $I$ of ESM. The following statements are equivalent: (i) $M$ is unique, (ii) $M$ is applicant-optimal, and (iii) every applicant $a_i$ with $A_i \neq \emptyset$ is matched in $M$ with his/her first-choice post.

So we can test if an instance $I$ of ESM admits a unique exchange-stable matching by checking that no two applicants with a non-empty preference list have the same first-choice post.

3.6 Generating all Exchange-Stable Matchings

In this section, we consider the problem of generating the set of all exchange-stable matchings for some instance $I$ of ESM. We require that this generation be efficient, meaning that the generation can only take polynomial time for each exchange-stable matching.

Let $M$ be any exchange-stable matching of $I$. If $M$ is unique, we are done. Otherwise, by Theorem 3.18, some applicant must not be matched in $M$ to his/her first choice post. As described in the proof of Lemma 3.16, we can generate a limited number of additional exchange-stable matchings by successively identifying an applicant $a_i$ who is not assigned his/her first
ranked post \(p_j\) in \(M\), and then constructing a new exchange-stable matching that includes \((a_i, p_j)\).

Here, we present a more general approach, which we describe for the following restricted version of the generation problem: Given an exchange-stable matching \(M\) of \(I\), generate a second exchange-stable matching of \(I\) that matches exactly the same set of applicants as \(M\), or determine that no such matching exists.

Construct the graph \(G' = (V, E')\) by augmenting the preference graph \(G = (V, E)\) of \(M\) with the following edges and weights. For each applicant pair \((a_i, a_j)\), where \(a_i\) is matched in \(M\) and prefers \(M(a_i)\) to \(M(a_j)\), add the edge \((a_i, a_j)\) to \(E'\) with weight 0. For any edge \((a_i, a_j)\) in \(E\), we know that \(a_i\) is either unmatched in \(M\) or prefers \(M(a_j)\) to \(M(a_i)\). Assign a weight of \(-1\) to all such edges.

Suppose that \(G'\) admits a negative weight cycle \(C = \langle a_1, a_2, \ldots, a_q \rangle\). We remark that since vertices corresponding to unmatched applicants have no incoming edges, every member of \(C\) must be matched in \(M\). Construct the matching \(M' = (M \cup \{(a_1, M(a_1)), (a_2, M(a_2)), \ldots, (a_q, M(a_q))\}) \cup \{(a_1, M(a_2)), (a_2, M(a_3)), \ldots, (a_q, M(a_1))\}\). Now, at least one applicant \(a_i\) in \(C\) prefers \(M'\) to \(M\), since \(C\) is a negative weight cycle. Therefore, \(M' \neq M\), and the set of applicants (respectively posts) matched in \(M'\) is exactly the same set of applicants (respectively posts) matched in \(M\). It follows immediately that, by the exchange-stability of \(M\), \(M'\) must be maximal.

However, in general, \(M'\) may not be trade-in-free or coalition-free. Our aim now is to stabilize \(M'\), ensuring that we do not transform \(M'\) back into \(M\).

Let \(M''\) be the result of running the second and third phase of Stabilize-ESM on \(M'\). It is easy to see that every applicant either prefers \(M''\) to \(M'\), or is indifferent between them. In particular, \(a_i\) prefers \(M''(a_i)\) to \(M(a_i)\), and so \(M'' \neq M\). Hence, \(M''\) is a second exchange-stable matching of \(I\).

Finally, we remark that if \(I\) admits a second exchange-stable matching that matches the same set of applicants as \(M\), then by Lemma 3.14, \(G'\) must admit a negative weight cycle. Hence, this approach solves the restricted generation problem. We leave open the problem of extending this approach to efficiently generate the set of all exchange-stable matchings.

### 3.7 Relationship with Stable Marriage

For each instance \(J\) of SMI, we can construct an instance \(I\) of ESM by ignoring the preference lists of women in \(J\). In this section, we examine the relationship between exchange-stability in \(I\), and classical stability in \(J\). For exposition purposes, we now regard \(I\) as consisting of a set \(U\) of men and a
set $W$ of women, where each man supplies a preference list ranking a subset of $W$ in strict order of preference. Women express no preference in $I$. We also rename exchange-stability as man-exchange-stability.

The following theorem, due to Knuth [45], is used in subsequent proofs.

**Theorem 3.19 (Knuth [45])** Let $M$ and $M'$ be stable matchings of some instance $J$ of SMI, and suppose that there is some (man, woman) pair $(m, w)$ in $M$ but not in $M'$. Then, one of $m$ and $w$ prefers $M$ to $M'$, and the other prefers $M'$ to $M$.

**Theorem 3.20** Let $J$ be an instance of SMI, and let $I$ be the corresponding instance of ESM. A stable matching $M$ of $J$ is a man-exchange-stable matching of $I$ only if $M = M_O$.

**Proof:** Suppose for a contradiction that $M$ is a man-exchange-stable matching of $I$, where $M \neq M_O$. We remark that, by Theorem 2.1, $M$ and $M_O$ match the same set of people.

Let $C$ be the set of men that prefer $M_O$ to $M$. Now, $C \neq \emptyset$, since $M \neq M_O$, and so there is some man $m_1 \in C$.

Let $w_1 = M_O(m_1)$. It follows that since $m_1$ prefers $M_O$ to $M$, $(m_1, w_1) \in M_O \setminus M$. Therefore, by Theorem 3.19, $w_1$ prefers $M$ to $M_O$.

Let $m_2 = M(w_1)$. It follows that since $w_1$ prefers $M$ to $M_O$, $(m_2, w_1) \in M \setminus M_O$. Therefore, by Theorem 3.19, $m_2$ prefers $M_O$ to $M$.

Eventually this process must cycle. It is easy to see that the men in this cycle form a coalition for $M$, since each such man $m_i$ prefers $M_O(m_i)$ to $M(m_i)$, where $M_O(m_i) = M(m_{i+1})$. Therefore, $M$ is not man-exchange-stable, giving the required contradiction.

We now give a small example to demonstrate that some man-optimal stable matchings are not man-exchange-stable. Consider the instance $I$ of SMI in Figure 18. The man-optimal stable matching of $I$ is $M_O = \{(m_1, w_1), (m_2, w_2), (m_3, w_3)\}$. This matching is blocked by the coalition $\langle m_1, m_2 \rangle$, since $m_1$ prefers $w_2$ to $w_1$, and $m_2$ prefers $w_1$ to $w_2$.

![Figure 18: An instance of SMI which admits no stable matching which is also man-exchange-stable.](image-url)
Let \( M \) be a man-exchange-stable matching of some instance \( I \) of ESM, and let \( G \) be the preference graph for \( M \). Since \( G \) is acyclic, it must admit a topological ordering of the men in \( I \). Let \( \sigma \) be any reversed topological ordering of \( G \). We call \( \sigma \) a signature of \( M \), and remark that every man-exchange-stable matching has at least one signature. We will use this fact in the following theorem.

**Theorem 3.21** Let \( I \) be an instance of ESM. Every man-exchange-stable matching of \( I \) is a man-optimal stable matching for some instance of SMI.

*Proof:* Let \( M \) be a man-exchange-stable matching of \( I \), and let \( \sigma \) be a signature of \( M \). Construct the instance \( J \) of SMI, where \( I \) is the restriction of \( J \), and every woman in \( J \) inherits her preference list from \( \sigma \) (i.e. each woman ranks the men of \( J \) in order of \( \sigma \), omitting any man that does not find her acceptable). We claim that \( M \) is the man-optimal stable matching of \( J \).

Suppose for a contradiction that \( M \) is blocked by some (man, woman) pair \((m, w)\). Now, \( w \) must be matched in \( M \), for otherwise \( M \) is either not maximal (if \( m \) is unmatched in \( M \)) or not trade-in-free (if \( m \) is matched in \( M \)), contradicting the man-exchange-stability of \( M \). So, \( m \) must appear before \( M(w) \) in \( \sigma \), since \( w \) prefers \( M(m) \) to \( M(w) \).

We also have that \( m \) is unmatched in \( M \) and finds \( w \) acceptable, or \( m \) prefers \( w \) to \( M(m) \). In either case, there must be a directed edge from \( m \) to \( M(w) \) in \( G \). Therefore, \( m \) must appear after \( M(w) \) in \( \sigma \), giving the required contradiction.

Finally, since \( M \) is both a stable matching of \( J \) and a man-exchange-stable matching of \( I \), \( M \) must be man-optimal by Theorem 3.20.

### 3.8 Signature Results

In this section, we use the concept of a signature to prove several miscellaneous results.

**Theorem 3.22** Every exchange-stable matching can be generated by an execution of Greedy-ESM.

*Proof:* Let \( M \) be any exchange-stable matching of some instance \( I \) of ESM. Let \( G \) be the preference graph of \( M \), and let \( \sigma \) be a signature of \( M \). We claim that by processing the applicants of \( I \) in order of \( \sigma \), Greedy-ESM returns the matching \( M' = M \).

Suppose for a contradiction that \( M' \neq M \). By Lemma 3.14, there must be some applicant that prefers \( M' \) to \( M \). Let \( a_i \) be the first such applicant in
\( \sigma \), and let \( p_j = M'(a_i) \). Now, \( p_j \) must be matched in \( M \) to some \( a'_i = M(p_j) \), for otherwise \( M \) is not maximal (if \( a_i \) unmatched in \( M \)), or \( M \) is not trade-in-free (if \( a_i \) is matched in \( M \)). It follows that \( G \) contains an edge from \( a_i \) to \( a'_i \), and therefore, that \( a'_i \) appears before \( a_i \) in \( \sigma \).

Now, since Greedy-ESM processes the applicants in order of \( \sigma \), \( p_j \) is free at the time \( a'_i \) is selected in Greedy-ESM. Furthermore, since \( M'(p_j) = a_i \), \( a'_i \) must be assigned a partner in \( M' \) that he/she prefers to \( p_j \). Therefore, \( a'_i \) prefers \( M' \) to \( M \), contradicting the assumption that \( a_i \) is the first applicant in \( \sigma \) to prefer \( M' \) to \( M \).

Hence \( M' = M \), and the result follows.

We remark that every signature corresponds to a unique exchange-stable matching, though such a matching may have several signatures.

**Lemma 3.23** Let \( M \) be an exchange-stable matching of some instance \( I \) of ESM. Suppose an applicant \( a_i \) is matched in \( M \) with his/her \((k + 1)\)th-choice post. Then there is some exchange-stable matching of \( I \) in which \( a_i \) is matched with his/her \( k \)th-choice post.

**Proof:** Denote by \( p_j \) the \( k \)th-choice post of \( a_i \). Now, \( p_j \) must be matched in \( M \), say to \( a = M(p_j) \), for otherwise, \( M \) is not trade-in-free.

Let \( \sigma \) be a signature of \( M \). We have that \( a_i \) prefers \( p_j = M(a) \) to \( M(a_i) \), and so \( a \) must precede \( a_i \) in \( \sigma \). Let \( \sigma' \) be a reordering of \( \sigma \), in which \( a_i \) and \( a \) switch positions, and all other entries are unchanged. It is easy to see that in the unique exchange-stable matching \( M' \) corresponding to \( \sigma' \), \( a_i \) is matched to \( p_j \).

We generalize the preceding lemma in the following theorem.

**Theorem 3.24** Let \( M \) be an exchange-stable matching of some instance \( I \) of ESM. Suppose an applicant \( a_i \) is matched in \( M \) with his/her \( k \)th-choice post. Then there is some exchange-stable matching of \( I \) in which \( a_i \) is matched with his/her \( j \)th-choice post, where \( 1 \leq j \leq k \).

**Theorem 3.25** In any non-empty exchange-stable matching \( M \), at least \( k \) applicants are matched with their \( k \)th-ranked post or better, where \( 1 \leq k \leq |M| \).

**Proof:** Let \( \sigma \) be a signature of \( M \) after removing any applicants unmatched in \( M \). So, \( \sigma \) contains \(|M|\) applicants, all of whom are matched in \( M \). Consider the execution \( E \) of Greedy-ESM with ordering \( \sigma \).
Let $a_i$ be the $k$th applicant in $\sigma$, for any $1 \leq k \leq |M|$. Now, if $|A_i| < k$, then since every applicant in $\sigma$ is matched in $M$, $a_i$ must be matched in $M$ to some post that he/she ranks better than $k$.

Otherwise, let $p_j$ be the $k$th-choice post of $a_i$. Consider the point in $E$ immediately before $a_i$ is matched. Since $a_i$ is the $k$th applicant in $\sigma$, exactly $(k - 1)$ posts have been matched by this point in $E$. It is easy to see then that $a_i$ must be matched in $M$ with $p_j$ or better.

The result follows by a simple inductive argument.

\section{3.9 Conclusion and Open Problems}

In this chapter, we gave three algorithms for finding a maximum cardinality exchange-stable matching. We then gave an efficient characterization of the set of ESM instances that admit a unique exchange-stable matching. Finally, we introduced the concept of a signature to show the connection between exchange-stable matchings in ESM and classical stable matchings in SMI.

A number of open problems remain. For example,

- We can find a maximum cardinality exchange-stable matching in time $O(\min (m, n)L)$, where $m$ and $n$ are the numbers of applicants and posts, and $L$ is the total length of the applicant preference lists. This matching is a maximum matching of applicants to posts. In general, we can find a maximum matching (which is not necessarily exchange-stable) in only $O(\sqrt{\min (m, n)L})$ time [33]. It is an open problem to determine if we can find a maximum matching that is also exchange-stable within this time bound.

- Let $I$ be an instance of ESM, and consider the problem of finding a minimum cardinality exchange-stable matching $M$ of $I$ (MIN-ESM). Although it is not known if MIN-ESM is polynomial-time solvable, we conjecture that the problem is NP-hard. We base this conjecture on the obvious similarity between MIN-ESM (noting that $M$ must be maximal) and the problem of finding a minimum cardinality maximal matching of a bipartite graph (MMM), which is well-known to be NP-hard [69].

The only positive result we have for MIN-ESM is a 2-approximation: any exchange-stable matching of $I$ is no larger than twice the size of a minimum cardinality exchange-stable matching. This follows from the maximality property of exchange-stability. For a full proof, see Lemma 7.1.
• We can extend the ESM model so that the applicant preference lists may contain ties. In this context, as with SMT and HRT, there are several possible definitions of stability. It remains open to determine if we can efficiently find stable matchings under these definitions.

• Given an exchange-stable matching, we solved the problem of determining if there is another exchange-stable matching (which matches the same set of applicants), and finding such a matching, if one exists. However, the problem of efficiently generating all exchange-stable matchings remains open. One approach to this problem is to give an algorithm that finds an exchange-stable matching $M$, where $M$ is required to (not) contain a given (applicant, post) pair. Again, solving this problem is open.
4 Tutorial Allocation

4.1 The Model

An instance $I$ of the TUTORIAL ALLOCATION problem (TA) consists of a set $S = \{s_1, s_2, \ldots, s_m\}$ of students, and a set $T = \{t_1, t_2, \ldots, t_n\}$ of tutorials. Each tutorial $t_j \in T$ has a positive integer capacity $c_j$, indicating the maximum number of students that can be allocated to $t_j$. Each student $s_i \in S$ supplies a subset $A_i$ of tutorials, each of which he/she is free to attend. If $t_j \in A_i$, we say that $s_i$ finds $t_j$ acceptable.

A matching $M$ of $I$ is a subset of $S \times T$ such that,

(i) $(s_i, t_j) \in M$ only if $t_j \in A_i$.

(ii) For each student $s_i \in S$, $|(s_i, t_j) \in M : t_j \in T| \leq 1$.

(iii) For each tutorial $t_j \in T$, $|(s_i, t_j) \in M : s_i \in S| \leq c_j$.

If $(s_i, t_j) \in M$, we say that $s_i$ is matched to $t_j$, and $t_j$ is matched to $s_i$. Denote by $M(t_j)$ the set of students matched to $t_j$ in $M$. Similarly, a student $s_i$ is either unmatched in $M$, or matched to some tutorial, which we denote by $M(s_i)$.

The TUTORIAL ALLOCATION problem is to find a maximum cardinality matching of students to tutorials. Figure 19 gives an example instance of TA with student set $S = \{s_1, s_2, s_3\}$, and tutorial set $T = \{t_1, t_2, t_3, t_4\}$. This instance admits several maximum cardinality matchings, such as $M_1 = \{(s_1, t_1), (s_2, t_2), (s_3, t_4)\}$ and $M_2 = \{(s_1, t_2), (s_2, t_2), (s_3, t_4)\}$.

Let $I$ be an instance of TA. The underlying graph $G$ of $I$ consists of one vertex for each student $s_i$, one vertex for each tutorial $t_j$, and an edge between $s_i$ and $t_j$ whenever $t_j \in A_i$. Each student vertex has capacity 1, while each tutorial vertex $t_j$ has capacity $c_j$.

The problem of finding a maximum cardinality matching of $I$ is equivalent to the problem of finding a maximum cardinality $b$-matching of $G$ (see Section 1.2.3 for more details). In the rest of this chapter, we look at several variants of TA in which we require a maximum cardinality matching with some additional property.

4.2 Minimum Tutorial Cover

Let $I$ be an instance of TA, and let $M$ be any matching of $I$. We say that a tutorial $t_j$ is empty in $M$ if $|M(t_j)| = 0$. Suppose there is a fixed non-zero cost associated with running each non-empty tutorial. Our aim is to find an allocation of students to tutorials that minimizes the overall financial cost,
Acceptable Tutorials

\[ s_1 : \{ t_1, t_2 \} \]
\[ s_2 : \{ t_2, t_3 \} \]
\[ s_3 : \{ t_4 \} \]

Tutorial capacities: \( c_1 = 1, c_2 = c_3 = c_4 = 2 \)

Figure 19: An instance of TA.

without sacrificing the number of matched students, or violating any tutorial capacity constraints. More formally, the minimum tutorial cover problem (MTC) is to find a maximum matching \( M \) of \( I \) with the minimum number of non-empty tutorials.

It turns out that MTC is NP-hard. We prove this with a reduction from the minimum set cover problem (MSC) ([26, problem SP5]). An instance \( J \) of MSC consists of a base set \( \beta \) and a family \( \mathcal{F} \) of subsets of \( \beta \), where \( \bigcup_{F \in \mathcal{F}} F = \beta \). A set cover of \( J \) is a subset \( C \) of \( \mathcal{F} \), such that every member of \( \beta \) is in some member of \( C \). The problem of finding a minimum cardinality set cover is NP-hard [43], and NP-hard to approximate within \( o(lg(|\beta|)) \) [17]. We remark that Johnson [42] gives a \((1+ln|\beta|)\)-approximation algorithm for MSC.

Given an instance \( J \) of MSC, construct the following instance \( I \) of MTC. For each element \( F \in \mathcal{F} \), construct a tutorial, denoted by \( t(F) \), with capacity \(|F|\). For each element \( b \in \beta \), construct a student, denoted by \( s(b) \), who finds acceptable any tutorial \( t(F) \), where \( b \in F \). This construction is clearly polynomial-time computable.

Now, let \( M \) be any maximum matching of \( I \) with \( k \) non-empty tutorials. Since \( \bigcup_{F \in \mathcal{F}} F = \beta \) and every tutorial \( t(F) \) has capacity \(|F|\), it must be the case that every student is matched in \( M \). Hence, the non-empty tutorials in \( M \) describe a set cover \( C \) of \( J \), where \(|C| = k \).

Conversely, let \( C \) be any set cover of \( J \), where \(|C| = k \). We can construct a matching \( M \) of \( I \) by arbitrarily matching each student \( s(b) \) to exactly one tutorial \( t(F) \), where \( b \in F \) and \( F \in C \). It is not too hard to see that \( M \) is a maximum matching with at most \( k \) non-empty tutorials.

We summarize the preceding discussion in the following theorem.

**Theorem 4.1** MTC is NP-hard and NP-hard to approximate within \( o(lg(|S|)) \).

Although MTC is NP-hard, the problem is polynomial-time solvable under certain restrictions. For example, suppose that \( c_j = 1 \) for every tutorial.
t_j in a given instance I of MTC. Then, every maximum matching of I has the same number of non-empty tutorials, and so MTC is equivalent to TA.

More generally, let I be an instance of MTC in which every tutorial has capacity at most 2. We remark that if m = 1, then MTC is trivial. Otherwise, we can solve MTC for I in polynomial time by finding a certain type of matching in the following weighted graph.

Let G[I] be the graph consisting of four disjoint vertex sets: S, T, T' and T''. Each student is represented by a vertex in set S, while each tutorial \( t_j \) is represented by three vertices, namely \( t_j \in T, t'_j \in T' \) and \( t''_j \in T'' \). These last three vertices are connected by the edges \( \{t_j, t'_j\} \) and \( \{t'_j, t''_j\} \), with weight 0 and \( m \) respectively. For each student \( s_i \), there is an edge between \( s_i \) and \( t_j \in T \) if \( s_i \) finds \( t_j \) acceptable. Furthermore, if \( c_j = 2 \), there is also an edge between \( s_i \) and \( t'_j \in T' \). Both these edges, if defined, have weight 1. Figure 20 gives an example of this graph for the instance in Figure 19.

We associate with every matching \( M' \) of \( G[I] \) a corresponding matching \( M \) of I, where \( M = \{(s_i, t_j) : \{s_i, t_j\} \in M' \} \) or \( \{s_i, t'_j\} \in M' \}. Similarly, we associate with every matching \( M \) of I a corresponding matching \( M' \) of \( G[I] \), where \( M' \) is defined according to the rules in Figure 21.

Notice that whenever \(|M(t_j)| = 1\), \( M' \) includes an edge with maximum weight among all edges in \( G[I] \). These maximum weight edges act as penalties for matching tutorials to exactly one student.
| $|M(t_j)|$ | $M(t_j)$ | Edges in $M'$ | Weight of edges in $M'$ |
|--------|---------|-------------|-------------------|
| 0      | $\emptyset$ | $\{t_j, t_j'\}$ | 0                |
| 1      | $s_a$   | $\{s_a, t_j'\}, \{t_j', t_j''\}$ | $1 + m$          |
| 2      | $s_a, s_b$ | $\{s_a, t_j\}, \{s_b, t_j'\}$ | 2                |

Figure 21: Constructing $M'$ from $M$.

Now, recall from Section 1.2.2 that algorithm MinWMCM builds a minimum weight maximum cardinality matching $M'$ by repeatedly finding and applying minimum weight $M'$-augmenting paths. Consider an execution $E$ of MinWMCM on $G[I]$. It is not too hard to see that, due to the structure of $G[I]$, we can partition $E$ into the following four distinct phases.

In the first phase, every $M'$-augmenting path has weight 0, matching each tutorial vertex $t_j \in T$, with the corresponding vertex $t_j' \in T'$. In the second phase, every $M'$-augmenting path has weight 2, beginning and ending with distinct unmatched students. At the termination of this phase, every tutorial $t_j$ has either $|M'(t_j)| = 2$ or $|M'(t_j)| = 0$. In the third phase, every $M'$-augmenting path has weight $1 + m$, beginning and ending with an unmatched student and a tutorial vertex $t_j'' \in T''$. In the final phase, every $M'$-augmenting path has weight $2m$, beginning and ending with unmatched tutorial vertices from $T''$. Figure 22 shows the result of phase 2 and phase 3 of MinWMCM on the graph $G[I]$ from Figure 20.

![Figure 22: MinWMCM on $G[I]$.](image-url)
Let $M'$ be the matching of $G[I]$ immediately after phase 3 has terminated. We claim that the corresponding matching $M$ of $I$ is a maximum cardinality matching with the fewest number of non-empty tutorials.

**Lemma 4.2** Let $I$ be an instance of MTC in which there are $m > 1$ students, and each tutorial has capacity at most 2. Let $M'$ be the matching of $G[I]$ immediately after phase 3 of MinWMCM has terminated. Then the matching $M$ of $I$ corresponding to $M'$ is a maximum cardinality matching.

**Proof:** Suppose for a contradiction that $M$ is not a maximum cardinality matching of $I$. It follows that $M$ admits some shortest length augmenting path $A = \langle s_0, t_1, s_1, \ldots, t_{k-1}, s_{k-1}, t_k \rangle$ in the underlying graph of $I$. Since $A$ has shortest length, it must be the case that $t_k$ is the first tutorial $t_i$ in $A$ with $|M(t_i)| < c_i$. Also, since $(s_i, t_i) \in M$ for $1 \leq i \leq k - 1$, we have that $\{s_i, t_i\} \in M'$, where $t_i$ is the vertex representing $t_i$ in either $T$ or $T'$.

Now, if $|M(t_k)| = 0$, then $\{t_k, t'_k\} \in M'$, and $G[I]$ admits the phase 3 $M'$-augmenting path $\langle s_0, t_1, s_1, \ldots, t_{k-1}, s_{k-1}, t_k, t'_k, s_k, t'_k, \rangle$. This contradicts the assumption that phase 3 has terminated.

So, it must be the case that $|M(t_k)| = 1$ and $\{t'_k, t''_k\} \in M'$. Consider the matching $M''$ of $G$, where $M'' = (M' \setminus \{t'_k, t''_k\}) \oplus (s_0, t_1, s_1, \ldots, t_{k-1}, s_{k-1}, t_k)$. Now, it is easy to see that $|M''| = |M'|$ and $w(M'') = w(M') - m + 1$. Since $m > 1$, we have that $w(M'') < w(M')$, which contradicts the property that the matching $M'$ maintained by MinWMCM has minimum weight among all matchings of size $|M'|$. Therefore, $M$ is a maximum cardinality matching of $I$.

\[\Box\]

**Theorem 4.3** Let $I$ be an instance of MTC in which there are $m > 1$ students, and each tutorial has capacity at most 2. Let $M'$ be the matching of $G[I]$ immediately after phase 3 of MinWMCM has terminated. Then the matching $M$ of $I$ corresponding to $M'$ is a maximum cardinality matching with the fewest number of non-empty tutorials.

**Proof:** Suppose for a contradiction that there is some maximum matching $M^*$ of $I$ that has fewer non-empty tutorials than $M$. Let $E^*$, $U^*$ and $D^*$ be the number of tutorials in $M^*$ with 0, 1, and 2 students respectively. Similarly, Let $E$, $U$ and $D$ be the number of tutorials in $M$ with 0, 1, and 2 students respectively.

By Lemma 4.2, we have that $|M^*| = 2D^* + U^* = 2D + U = |M|$. Also, by assumption, we have that $D^* + U^* < D + U$, which together with the last equality, implies that $U^* < U$.
Let $M''$ be the matching of $G[I]$ associated with $M^*$. Now, the weight of $M''$ is $2D^* + U^*(1 + m) = 2D^* + U^* + U^*m = 2D + U + U^*m < 2D + U + Um$, which is the weight of $M'$ in $G[I]$. This contradicts the property that the matching $M'$ maintained by MinWMCM has minimum weight among all matchings of size $|M'|$. Therefore, $M$ is a maximum cardinality matching of $I$ with the fewest number of non-empty tutorials.

We remark that MSC is polynomial-time solvable when every subset of the base set has cardinality at most 2. However, there doesn’t appear to be a simple reduction from MTC to MSC under these restrictions, since several students may find any one tutorial acceptable. Therefore, we cannot see any way of using the polynomial-time algorithm for MSC to solve MTC.

In contrast to Theorem 4.3, if we restrict $|A_i| = 2$ for each student $s_i$, then MTC is NP-hard. We prove this with a reduction from the MINIMUM VERTEX COVER problem (MVC) ([26, problem GT1]). An instance $J$ of MVC consists of an undirected graph $G = (V, E)$. A vertex cover of $G$ is a subset $C$ of $V$ such that every edge in $E$ has at least one endpoint in $C$. The problem of finding a minimum vertex cover is well-known to be NP-hard [43].

Given an instance $J$ of MVC consisting of a graph $G = (V, E)$, construct the following instance $I$ of MTC. For each vertex $v \in V$, construct a tutorial, denoted by $t(v)$, with capacity $\deg(v)$. For each edge $\{u, v\} \in E$, construct a student, denoted by $s(\{u, v\})$, who finds acceptable $t(u)$ and $t(v)$ (so each student finds exactly two tutorials acceptable). This construction is equivalent to the subdivision graph of $G$, and is clearly polynomial-time computable.

Now, let $M$ be any maximum matching of $I$ with $k$ non-empty tutorials. Since each tutorial $t(v)$ has capacity $\deg(v)$, it must be the case that every student is matched in $M$. Hence, the non-empty tutorials in $M$ describe a vertex cover $C$ of $J$, where $|C| = k$.

Conversely, let $C$ be any vertex cover of $J$, where $|C| = k$. So, for each edge $\{u, v\}$, at least one of $u$ and $v$ must be in $C$. Therefore, we can construct a maximum matching $M$ of $I$, in which each student $s(\{u, v\})$ is matched with either $t(u)$, if $u \in C$, or $t(v)$, if $u \notin C$. It is not too hard to see that $M$ has at most $k$ non-empty tutorials.

We summarize the preceding discussion in the following theorem.

**Theorem 4.4** MTC is NP-hard, even when each student finds at most two tutorials acceptable.

**Theorem 4.5** MTC is NP-hard, even when each student finds at most two tutorials acceptable, and each tutorial has capacity 3.
Proof: MVC is APX-complete even for cubic graphs [4], that is, graphs in which every vertex has degree 3. Using the reduction function above, we can transform any cubic graph into an instance of MTC, where each tutorial has capacity 3. The result follows immediately.

4.3 Balanced Matchings

Let $M$ be a matching of some instance $I$ of TA. Define the load vector of $M$ as $l(M) = (|M(t_1)|, |M(t_2)|, \ldots, |M(t_n)|)$. We measure the imbalance of $M$, denoted by $||l(M)||_p$, by the $L_p$-norm of $l(M)$, where for $p \geq 1$,

$$||l(M)||_p = \left(\sum_{t_j \in T} |M(t_j)|^p\right)^{1/p}$$

A balanced matching of $I$ is a maximum cardinality matching of $I$ with minimum imbalance for every $p \geq 1$.

Remark 4.6 Let $I$ be an instance of TA. Any maximum matching of $I$ has minimum imbalance using the $L_1$-norm.

Proof: Let $M_1$ and $M_2$ be any two maximum matchings of $I$. Then $||l(M_1)||_1 = |M_1| = |M_2| = ||l(M_2)||_1$

Alon et al. [5] have studied a similar problem in the context of scheduling. In their work, $S$ is a set of jobs and $T$ is a set of machines. Each job $s_i$ can be processed in unit time by any machine in a specified subset $A_i$ of $T$. Unlike TA, machines have infinite capacity and every problem instance must admit a complete matching of jobs to machines. Alon et al prove the existence of a strongly-optimal assignment, which is a complete matching with minimum $L_p$-norm for any $p \geq 1$. Their algorithm runs in $O(v^3e)$, where $v$ and $e$ are the numbers of vertices and edges in the underlying graph.

In this section, we present RRBalance (Figure 23), which is a new algorithm for finding a balanced matching. We show below that RRBalance runs in $O(ve)$ time, giving a factor of $v^2$ improvement on the algorithm due to Alon et al. The correctness proof for RRBalance also independently proves the existence of a strongly-optimal assignment.

An execution of RRBalance consists of a sequence of rounds, where each round corresponds to an iteration of the outer loop. During each round, RRBalance performs a sequence of steps, where, initially, there is one step for each tutorial $t_j$. During each step, RRBalance attempts to find and apply
RRBalance($I$)
\[
M := \emptyset;
\]
\[
C := T;
\]
while ($C \neq \emptyset$)
\[
\text{for each } (t_j \in C)
\]
if (\[|M(t_j)| = c_j \text{ or there is no } t_j\text{-augmenting path } P\])
\[
C := C \setminus \{t_j\};
\]
else
\[
M := M \oplus P;
\]
return $M$;

Figure 23: An algorithm for finding a balanced matching.

a $t_j$-augmenting path, that is, an augmenting path with endpoint $t_j$ (similarly,
a $t_j$-alternating path is an alternating path with endpoint $t_j$). If $t_j$ is already
matched with $c_j$ students, or no $t_j$-augmenting path can be found, then $t_j$
is removed from the set $C$ of candidate tutorials, and no subsequent step
of RRBalance involves the search for a $t_j$-augmenting path. The algorithm
continues until there are no candidate tutorials.

So, for each step in RRbalance, we either remove a candidate tutorial or
apply an augmenting path. The maximum number of steps in RRbalance is
therefore $n + \min (m, N) = O(v)$, where $N$ is the total capacity sum of all
the tutorials. We can store candidate tutorials in a linked list, which allows
efficient traversal and deletion. We can perform each step in $O(e)$ time using
a depth first search. Therefore, the overall runtime of RRbalance is $O(ve)$.

Let $I$ be a instance of TA with tutorial set $T$ and underlying graph $G$. Let
$E$ be an arbitrary execution of RRBalance on $I$, and let $M_i$ be the matching
of $I$ immediately before step $i$ of $E$. The following definitions and results are
used in Theorem 4.13, which proves the correctness of RRBalance.

Definition 4.7

(i) $\text{Aug}(i) = \{t_k \in T : G \text{ admits a } t_k\text{-augmenting path at step } i \text{ of } E\}$.

(ii) $\text{Alt}(i) = \{t_k \in T : G \text{ admits a } t_k\text{-alternating path ending with a student}
unmatched in } M_i \text{ at step } i \text{ of } E\}$.

(iii) $\text{Alt}_{t_j}(i) = \{t_j\} \cup \{t_k \in T : t_k \text{ is reachable from } t_j \text{ by an } M_i\text{-alternating}
path beginning with an unmatched edge incident to } t_j\}$.

Based on these definitions, we make the following remarks.

Remark 4.8
(i) $\text{Aug}(i) \subseteq \text{Alt}(i)$.

(ii) $t_j \in \text{Alt}(i) \setminus \text{Aug}(i)$ implies $|M_i(t_j)| = c_j$.

(iii) $t_k \in \text{Alt}_{t_j}(i)$ implies $\text{Alt}_{t_k}(i) \subseteq \text{Alt}_{t_j}(i)$.

(iv) $t_j \in \text{Alt}(i)$ if and only if there is a $t_k \in \text{Alt}_{t_j}(i)$ such that $t_k$ is adjacent in $G$ to some student unmatched in $M_i$.

**Lemma 4.9** If $t_j \notin \text{Alt}(i)$, then $t_j \notin \text{Alt}(i + 1)$.

**Proof:** Suppose $t_j \notin \text{Alt}(i)$. Then, by Remark 4.8(iv), no tutorial in $\text{Alt}_{t_j}(i)$ is adjacent to a student unmatched in $M_i$. Therefore, $\text{Alt}_{t_j}(i) \cap \text{Alt}(i) = \emptyset$. Let $S'$ be the set of all students matched to some tutorial in $\text{Alt}_{t_j}(i)$, and let $A$ be any $M_i$-augmenting path.

Now, since $\text{Alt}_{t_j}(i) \cap \text{Alt}(i) = \emptyset$, $A$ cannot include any member of $\text{Alt}_{t_j}(i) \cup S'$. So, if $M_{i+1} = M_i \oplus A$, we have that $\text{Alt}_{t_j}(i + 1) = \text{Alt}_{t_j}(i)$. It follows then that every tutorial in $\text{Alt}_{t_j}(i + 1)$ can only be adjacent to members of $S'$, all of whom are matched in $M_{i+1}$. Hence, by Remark 4.8(iv), $t_j \notin \text{Alt}(i + 1)$.

**Corollary 4.10** If $t_j \notin \text{Alt}(i)$, then for all $k \geq 0$, $t_j \notin \text{Alt}(i + k)$.

**Corollary 4.11** If $t_j \notin \text{Alt}(i)$, then $M(t_j) = M_i(t_j)$.

**Lemma 4.12** If $t_j \notin \text{Aug}(i)$, then for all $k \geq 0$, $t_j \notin \text{Aug}(i + k)$.

**Proof:** Suppose $t_j \notin \text{Aug}(i)$. Now, if $|M_i(t_j)| < c_j$, Remark 4.8(ii) gives us that $t_j \notin \text{Alt}(i)$. Therefore, by Corollary 4.10, $t_j \notin \text{Alt}(i + k)$ for any $k \geq 0$. Now, since $\text{Aug}(i + k) \subseteq \text{Alt}(i + k)$ (Remark 4.8(i)), $t_j \notin \text{Aug}(i + k)$ for any $k \geq 0$. Otherwise, $|M_i(t_j)| = c_j$, and $t_j$ trivially cannot be the endpoint of any subsequent augmenting path.

**Theorem 4.13** Let $I$ be an instance of TA, and let $M$ be the matching of $I$ returned by an arbitrary execution $E$ of RRBalance. Then $M$ is a balanced matching of $I$.

**Proof:** We remark that $M$ is a maximum matching of $I$, since once a tutorial $t_j$ is removed from $C$, $G$ never subsequently admits a $t_j$-augmenting path (Lemma 4.12). Therefore, by Remark 4.6, $M$ has minimum imbalance for $p = 1$. 68
We now show that $M$ has minimum imbalance whenever $p > 1$. Let $M'$ be any matching of $I$, and suppose for a contradiction that $||l(M^*)||_p < ||l(M)||_p$. Intuitively, $M^*$ has less imbalance (in the $L_p$-norm) than $M$ because it more evenly distributes the students among tutorials. More formally, consider the symmetric difference $M \oplus M^*$ between $M$ and $M^*$, which consists of a set of alternating cycles and paths. Now, since $||l(M^*)||_p < ||l(M)||_p$ and $|M| = |M^*|$, $G$ must admit an $M$-alternating path $A = \langle t_1, s_1, t_2, \ldots, s_{r-1}, t_r \rangle$ and matching $M' = M \oplus A$, where

(i) $(s_i, t_i) \in M$, for all $1 \leq i \leq r - 1$.

(ii) $(s_i, t_{i+1}) \in M^*$ and $(s_i, t_{i+1}) \in M'$, for all $1 \leq r - 1$.

(iii) $|M(t_1)|^p + |M(t_r)|^p > (|M(t_1)| - 1)^p + (|M(t_r)| + 1)^p = |M'(t_1)|^p + |M'(t_r)|^p$.

It follows from (iii) that $|M(t_1)|^p - (|M(t_1)| - 1)^p > (|M(t_r)| + 1)^p - |M(t_r)|^p$. We will show that $|M(t_1)| > |M(t_r)| + 1$, which is central to the remaining argument.

Denote by $f$ the function $f(x) = x^p$, where $x \geq 0$ and $p > 1$. Now, suppose that $f(a) - f(a-1) > f(b+1) - f(b)$, where $a > 0$ and $b \geq 0$. Since $a - (a-1) = 1 = (b+1) - b$, it follows from Lagrange's Mean Value Theorem that there is a $c \in (a-1, a)$ and $c' \in (b, b+1)$ such that,

$$f'(c) = \frac{f(a) - f(a-1)}{a - (a-1)} > \frac{f(b+1) - f(b)}{(b+1) - b} = f'(c')$$

Now, since $f'$ is an increasing function, we have that $a > c > c' > b$. Hence, $a > b + 1$, since $a$ and $b$ are integers, and therefore, $|M(t_1)| > |M(t_r)| + 1$.

Let $i$ be the step of $E$ in which RRBalance removes $t_r$ from $C$ (so, by Lemma 4.12, $t_r \notin \text{Aug}(i+k)$ for any $k \geq 0$). Now, since $|M'(t_r)| = |M(t_r)| + 1$, $M_i(t_r) < c_r$, and therefore, by Remark 4.8(ii), $t_r \notin \text{Alt}(i)$. So, by Corollary 4.11, $M(t_r) = M_i(t_r)$, and hence, $s_{r-1}$ is matched to some tutorial $t \neq t_r$ in $M_i$. It is easy to see that $t \in \text{Alt}(t_r)$, and since $t_r \notin \text{Alt}(i)$, we have that $t \notin \text{Alt}(i)$. Therefore, by Corollary 4.11, $M(t) = M_i(t)$, and so $t = t_{r-1}$.

We can use a similar argument to prove that no tutorial $t \in A$ is a member of $\text{Alt}(i)$. This is a contradiction, since $|M(t_1)| > |M(t_r)| + 1$ implies that at step $i$ of $E$, $t_1$ admits at least one more augmenting path. Hence, $t_1 \in \text{Aug}(i)$, which by Remark 4.8(i), is a subset of $\text{Alt}(i)$.

\[ \blacksquare \]
4.4 Repairing Broken Matchings

Let $I$ be an instance of TA with student set $S$ and tutorial set $T$, and let $M$ and $M'$ be any two subsets of $S \times T$. We define the similarity between $M$ and $M'$ as $s(M, M') = |M \cap M'|$.

In this section, we consider the problem of finding a maximum cardinality matching $M$ of $I$ that has maximum similarity with some subset $M'$ of $S \times T$. This problem has several practical applications, say in the following situation. Let $M'$ be a matching of some instance $J$ of TA. Suppose that a subset $\Delta$ of students from $J$ now change the tutorials they find acceptable. The effect of this change is create a new instance $I$ of TA. Our aim is to find a maximum cardinality matching $M$ of $I$, such that the fewest number students are forced to change their allocation from $M'$ to $M$.

We can solve this in polynomial time with the following transformation to the maximum weight maximum cardinality $b$-matching problem.

Construct the underlying graph $G$ of $I$. For each pair $(s_i, t_j) \in M'$, if $s_i$ finds $t_j$ acceptable in $I$, assign the edge $\{s_i, t_j\}$ in $G$ a weight of 1. All other edges in $G$ have weight 0. It is easy to see that a maximum weight maximum cardinality $b$-matching (see Section 1.2.3) of this graph gives a maximum cardinality matching of $I$ with maximum similarity to $M'$.

4.5 Conclusions and Open Problems

In this chapter, we introduced a bipartite $b$-matching problem called TA. We proved that a variant MTC of TA is NP-hard in general, but polynomial-time solvable in the special case that each tutorial has capacity at most 2. We then gave a new algorithm for finding a balanced matching. This algorithm has better worst-case performance than the previous best algorithm. Finally, we solved the maximum similarity problem with a reduction to the weighted $b$-matching problem.

The following problems remain open.

- Let $I$ be an instance of TA with student set $S$. Consider the bipartition of $S$ into the set of males and the set of females. The minimum independent female problem (MIF) is to find a maximum matching of $I$, which minimizes the number of tutorials with only one female. We conjecture that both this problem and the more general minimum independent student problem (MIS), in which we do not distinguish between males and females, are NP-hard.

- Consider the generalization of TA in which students rank the tutorials they find acceptable. In this context, we still seek a matching with
the maximum cardinality property, but, now, we might additionally require that the matching be one-sided exchange stable (see Chapter 3). Besides the basic TA problem, all the problems we discussed in this chapter are open in this more general context.
5 Half-Strong Stability

5.1 Introduction

Let $I$ be an instance of SMTI with a set $U$ of $n$ men, and a set $W$ of $n$ women. A matching $M$ of $I$ is half-strongly stable unless $M$ admits a blocking pair $(m, w) \in U \times W$, such that

(i) $(m, w) \notin M$.

(ii) $m$ is unmatched in $M$ or prefers $w$ to $M(m)$.

(iii) $w$ is either unmatched in $M$, or prefers $m$ to $M(w)$ or is indifferent between them.

Note that we may switch the roles of $m$ and $w$ in (ii) and (iii) to obtain a different definition of half-strong stability. These definitions are symmetrical, and so, for exposition purposes, we only consider the definition given above.

In the following discussion, we place half-strong stability in context with the existing types of stability for SMTI, namely weak, strong and super-stability (see Section 1.3.3 for more detail). Let $M$ be a matching of some instance $I$ of SMTI.

Remark 5.1 $M$ is half-strongly stable if and only if (i) $M$ is weakly stable, and (ii) for all matched women $w$ in $M$, if $w$ is indifferent between $M(w)$ and $m \neq M(w)$, then $m$ is matched in $M$ and either prefers $M(m)$ to $w$ or is indifferent between them.

If $m$ is either unmatched in $M$ or prefers $w$ to $M(m)$, then $m$ may attempt to bribe $w$ to switch partners from $M(w)$ to $m$. If $w$ is indifferent between $M(w)$ and $m$, this bribe may be all the incentive that $w$ requires to ignore her allocation in $M$ and partner with $m$. A weakly stable matching in which this situation cannot arise is half-strongly stable. We say that half-strongly stable matchings are more robust than weakly stable matchings, since there are fewer opportunities for bribery.

Remark 5.2 $M$ is strongly stable if and only if (i) $M$ is half-strongly stable, and (ii) for all matched men $m$ in $M$, if $m$ is indifferent between $M(m)$ and $w \neq M(m)$, then $w$ is matched in $M$ and either prefers $M(w)$ to $m$ or is indifferent between them.

If $w$ is either unmatched in $M$ or prefers $m$ to $M(w)$, then $w$ may attempt to bribe $m$ to switch partners from $M(m)$ to $w$. If $m$ is indifferent between
M(m) and w, this bribe may be all the incentive that m requires to ignore his allocation in M and partner with w. A half-strongly stable matching in which this situation cannot arise is strongly stable. Strongly stable matchings are more robust than half-strongly stable matchings.

**Remark 5.3** M is super-stable if and only if (i) M is strongly stable, and (ii) for all matched people p in M, if p is indifferent between M(p) and q ≠ M(p), then q is matched in M and prefers M(q) to p.

So, every super-stable matching is strongly stable, every strongly stable matching is half-strongly stable and every half-strongly stable matching is weakly stable. Figure 24 summarizes the relationship between the different definitions of stability for an arbitrary instance I of SMTI. The terms super, strong, half-strong and weak refer to the set of super-stable, strongly stable, half-strongly stable and weakly stable matchings of I respectively.

**super ⊆ strong ⊆ half-strong ⊆ weak**

Figure 24: Relationship between stability definitions

In practice, we may want to find a matching that is at least strongly stable (and possibly super-stable as well), since strong stability is more robust than half-strong stability or weak stability. However, it turns out some instances of SMT/SMTI admit no strongly stable matching [36]. For example, consider the instance in Figure 25. Any strongly stable matching of this instance must have cardinality 2, but neither M₁ = \{(m₁, w₁), (m₂, w₂)\} nor M₂ = \{(m₁, w₂), (m₂, w₁)\} is strongly stable due to the pairs (m₂, w₁) and (m₂, w₂) respectively. It is easily verified, however, that both M₁ and M₂ are half-strongly stable.

\[
\begin{align*}
m₁ & : \ w₁ \ w₂ \\
m₂ & : \ (w₁ \ w₂)
\end{align*}
\]

\[
\begin{align*}
w₁ & : \ m₁ \ m₂ \\
w₂ & : \ m₂ \ m₁
\end{align*}
\]

Figure 25: An instance of SMT/SMTI with no strongly stable matching

Manlove [48] gives a polynomial-time algorithm to determine if an instance of SMTI admits a strongly stable matching, and to find such a matching, if one exists. If no such matching exists, then before settling for a weakly stable matching, one could search for a half-strongly stable matching. Although some participants may have an incentive to bribe in a half-strongly stable matching, we can at least guarantee that such participants only come
from one side of the matching. This is more than a weakly stable matching can guarantee, and it might be all that is required, say if one set of the participants (here, the men) have more to offer by way of a bribe. One example of this is the allocation of medical residents to hospitals: a hospital may certainly have the financial means to offer a resident a bribe, whereas it is less likely that a resident $r$ could bribe a hospital $h$, when $h$ has no preference for $r$ over its existing allocation.

5.2 Preliminary Observations

Although an instance of SMTI is more likely to admit a matching that is half-strongly stable than strongly stable, it turns out that some instances admit no half-strongly stable matching. An example of this is given in Figure 26. Matching $M_1 = \{(m_1, w_1), (m_2, w_2)\}$ is blocked by $(m_2, w_1)$, while matching $M_2 = \{(m_1, w_2), (m_2, w_1)\}$ is blocked by $(m_1, w_1)$. No other matching besides $M_1$ and $M_2$ is even weakly stable.

$$
m_1 : w_1 \ w_2 \quad w_1 : (m_1 \ m_2)
m_2 : w_1 \ w_2 \quad w_2 : m_1 \ m_2$$

Figure 26: An instance of SMTI with no half-strongly stable matching

The main problem we are concerned with here then is how to efficiently determine if a given instance of SMTI admits a half-strongly stable matching, and how to find such a matching, if one exists. Before dealing with this problem in general, we present two special cases.

In the first special case, we consider the set of all SMTI instances in which no ties occur in the women’s preference lists.

Proposition 5.4 In any instance of I SMTI with no ties on the women’s side, (i) every weakly stable matching is half-strongly stable, and (ii) every strongly stable matching is super-stable.

Proof:

(i) Let $M$ be any weakly stable matching of $I$. Since $I$ has no ties on the women’s side, no woman is indifferent between any two men, and so by Remark 5.1, $M$ must also be half-strongly stable.

(ii) Let $M$ be any strongly stable matching of $I$, and suppose for a contradiction that $M$ is not super-stable. It follows that $M$ admits a blocking
pair \((m, w)\), where, by Remark 5.3, \(m\) and \(w\) are indifferent between each other and their partners in \(M\). This is a contradiction, since there are no ties on the women’s side. Therefore, \(M\) is super-stable.

Figure 27 summarizes the relationship between the different definitions of stability for this first special case of SMTI.

\[
\text{super} = \text{strong} \subseteq \text{half-strong} = \text{weak}
\]

Figure 27: Relationship between stability definitions, where no ties occur on the women’s side

**Corollary 5.5** Any instance \(I\) of SMTI with no ties on the women’s side admits a half-strongly stable matching, which we can find in linear-time.

*Proof:* By Proposition 5.4, the set of half-strongly stable matchings of \(I\) is equal to the set of weakly stable matchings of \(I\). Therefore, we can use Irving’s linear-time algorithm [36] to return any weakly stable matching of \(I\).

**Corollary 5.6** An instance of SMTI may admit half-strongly stable matchings of different cardinalities.

*Proof:* Consider the instance given in Figure 28. It is easily verified that this instance admits exactly two half-strongly stable matchings, \(M_1 = \{(m_1, w_1)\}\) and \(M_2 = \{(m_1, w_2), (m_2, w_1)\}\), where \(|M_1| < |M_2|\).

\[
\begin{align*}
m_1 : & (w_1 \ w_2) & \quad w_1 : & m_1 \ m_2 \\
m_2 : & w_1 & \quad w_2 : & m_1
\end{align*}
\]

Figure 28: Instance of SMTI admitting half-strongly stable matchings with different cardinalities

This last result is not too surprising since, by Proposition 5.4, every weakly stable matching is half-strongly stable, and instances of SMTI can admit weakly stable matchings of different cardinalities [51].

The next special case considers the set of all SMTI instances in which no ties occur in the men’s preference lists.
Proposition 5.7 In any instance $I$ of SMTI with no ties on the men’s side, (i) every half-strongly stable matching is strongly stable, and (ii) every strongly stable matching is super-stable.

Proof:

(i) Let $M$ be any half-strongly stable matching of $I$. Since $I$ has no ties on the men’s side, no man is indifferent between any two women, and so by Remark 5.2, $M$ must also be strongly stable.

(ii) This proof is analogous to part (ii) of Proposition 5.4.

Figure 29 summarizes the relationship between the different definitions of stability for this second special case of SMTI.

$$\text{super} = \text{strong} = \text{half-strong} \subseteq \text{weak}$$

Figure 29: Relationship between stability definitions, where no ties occur on the men’s side

Corollary 5.8 In any instance $I$ of SMTI with no ties on the men’s side, we can determine if $I$ admits a half-strongly stable matching, and find such a matching if one exists, in linear time.

Proof: By Proposition 5.7, the set of half-strongly stable matchings of $I$ is equal to the set of super-stable matchings of $I$. Therefore, we can use Manlove’s linear-time algorithm [48] to determine if $I$ admits a super-stable matching, and to find such a matching if one exists.

5.3 Complexity of Half-Strong Stability

In this section, we use the following theorem to prove that the problem of determining if an instance of SMT/SMTI admits a half-strongly stable matching is NP-complete.

Let $I$ be an instance of SMTI in which ties occur only on the men’s side.

Theorem 5.9 ([51]) The problem of determining if $I$ admits a complete weakly stable matching is NP-complete.
Denote by $U$ and $W$ the set of men $\{m_1, m_2, \ldots, m_n\}$ and set of women $\{w_1, w_2, \ldots, w_n\}$ in $I$ respectively. For each person $p \in U \cup W$, denote by $P(p)$ the preference list of $p$ in $I$.

Construct the instance $J$ of SMT with men $U' = U \cup \{m_{n+1}, m_{n+2}, \ldots, m_{2n}\}$ and women $W' = W \cup \{w_{n+1}, w_{n+2}, \ldots, w_{2n}\}$. For each person $p \in U' \cup W'$, denote by $P'(p)$ the preference list of $p$ in $J$. We describe $P'$ in Figure 30: parentheses denote ties, square brackets denote arbitrary order, and $U'$ (respectively $W'$) at the end of a preference list $P'(p)$ denotes all members of $U'$ (respectively $W'$) that have not already appeared in $P'(p)$.

\[
\begin{align*}
P'(m_i) & : P(m_i) \ w_{n+i} \ [W'] \\
P'(m_{n+i}) & : w_{n+i} \ [W'] \\
P'(w_i) & : P(w_i) \ m_{n+i} \ [U'] \\
P'(w_{n+i}) & : (m_{n+i} \ m_i) \ [U']
\end{align*}
\]

Figure 30: Reduction preference lists

We remark that $J$ is an instance of SMT, since each person $p \in U' \cup W'$ ranks every member of the opposite sex exactly once in $P'(p)$. Furthermore, it is clear that $J$ can be constructed from $I$ in polynomial time.

Suppose that $I$ admits a complete weakly stable matching $M$. Let $M' = M \cup \{(m_{n+i}, w_{n+i}) : 1 \leq i \leq n\}$, and suppose for a contradiction that $M'$ is not a half-strongly stable matching of $J$ due to some blocking pair $(m, w)$. Now, since each man $m_{n+i}$ is matched in $M'$ with his first-choice partner, it must be the case that $m \in U$. Therefore, $M'(m) = M(m) \in W$, and so by construction of $P'(m)$, $w \in W$. Now, since $(m, w)$ blocks $M'$, $m$ prefers $w$ to $M'(m) = M(m)$, and $w$ prefers $m$ to $M'(w) = M(w)$ (note that $w$ cannot be indifferent between $m$ and $M(w)$, since $P'(w)$ contains no ties). So, $M$ is not a weakly stable matching of $I$ due to the pair $(m, w)$, giving the required contradiction. Hence, $M'$ is a half-strongly stable matching of $J$.

Conversely, suppose that $J$ admits a half-strongly stable matching $M'$. We will show that $M'$ describes a complete weakly stable matching in $I$. Firstly, $(m_{n+i}, w_{n+i}) \in M'$, for $1 \leq i \leq n$, since otherwise $(m_{n+i}, w_{n+i})$ blocks $M'$. Also, each $m_i \in U$ is matched to some $w_j \in W$, for otherwise, $(m_i, w_{n+i})$ blocks $M'$. Therefore, $M = M' \cap (U \times W)$ is a complete matching of $I$. Furthermore, $M$ is weakly stable, for otherwise, any pair that blocks the weak stability of $M$ also blocks half-strong stability of $M'$. Hence, $M$ is a complete weakly stable matching of $I$.

Finally, it is clear that the problem of determining if an instance of SMT/SMTI admits a half-strongly stable matching is in NP. The next theorem summarizes the preceding result.
Theorem 5.10 The problem of determining if an instance of SMT/SMTI admits a half-strongly stable matching is NP-complete.

5.4 Conclusion and Open Problems

In this chapter, we introduced a new definition of stability for SMTI, called half-strong stability. We placed this definition in context with the existing types of stability, showing that it is more robust than weak stability, and less robust than strong stability. We also showed that, when no ties occur on the women’s side, half-strong stability is equivalent to weak stability, and, when no ties occur on the men’s side, half-strong stability is equivalent to super-stability. Finally, despite the fact that we can solve the existence problems for weak stability and strong stability in polynomial time, we proved that the corresponding problem for half-strong stability is NP-hard.

Given this hardness result, a next step is to examine the more general problem of finding a weakly stable matching with the minimum number of half-strongly stable blocking pairs. It is not known if this problem is in APX.
6 Reducing Roommates to Stable Marriage

6.1 Background

Recall that an instance $I$ of the stable roommates problem (SR) consists of an even cardinality set of people $P$, each of whom ranks every other person in strict order of preference. A matching of $I$ is a partition of $P$ into disjoint pairs. A matching $M$ is unstable if there are two distinct people, each of whom prefers the other to their partner in $M$. These two people are said to form a blocking pair for $M$. If $M$ admits no blocking pair, then $M$ is stable.

Given an instance $I$ of SR, the stable roommates problem is to find a stable matching of $I$, or determine that no such matching exists.

SR generalizes the stable marriage problem (SM) by the following result.

**Theorem 6.1 (Gusfield and Irving [28])** For every instance $J$ of SM, there is an instance $I$ of SR such that there is a bijection between the stable matchings of $I$ and the stable matchings of $J$.

**Proof:** Given an instance $J$ of SM with $n$ men and $n$ women, construct the following instance $I$ of SR. For each man and each woman $p$ in $J$, construct a person $p$ in $I$, whose preference list consists of $p$’s preference list in $J$ followed by all other members of the same sex as $p$ in arbitrary order. Since there are $n$ men and $n$ women in $J$, $I$ has $2n$ people, each of whom ranks every other constructed person. Hence, $I$ is an instance of SR.

Let $M$ be any stable matching of $J$, and let $M' = \{(m_i, w_j) | (m_i, w_j) \in M\}$. We will show that $M'$ is a stable matching of $I$. Suppose for a contradiction that $\{p_i, p_j\}$ blocks $M'$. Now, since $M$ consists of (man, woman) pairs, and all people in $I$ prefer members of the opposite sex to members of the same sex, it must be the case that, without loss of generality, $p_i$ is a man and $p_j$ is a woman in $J$. But then $(p_i, p_j)$ forms a blocking pair for $M$, giving the required contradiction.

Conversely, let $M'$ be any stable matching of $I$. Suppose that $\{p_i, p_j\} \in M'$, where $p_i$ and $p_j$ are both men in $J$. Since $M'$ is a complete matching, $M'$ must also contain a pair $\{p'_i, p'_j\}$, where $p'_i$ and $p'_j$ are both women in $J$. But then $\{p_i, p'_j\}$ blocks $M'$, since all people in $I$ prefer members of the opposite sex to members of the same sex. Hence, $M'$ only contains \{man, woman\} pairs. We can trivially construct a matching $M$ of $J$ from these pairs of $M'$, and it is clear that $M$ is stable, for otherwise, any blocking pair for $M$ gives a blocking pair for $M'$.

\[ \blacksquare \]
An immediate corollary of this theorem is that several properties of SM also apply to SR. For example, the maximum number of stable matchings in an instance I of SR grows exponentially with the size of I (see Knuth's analogous result for SM [45]). Also, since the reduction in Theorem 6.1 is computable in linear time, the linear time lower bound on SM [56] also holds for SR.

A natural conjecture arising from Theorem 6.1 is that the reverse holds. That is, for every instance I of SR, there is an instance J of SM such that there is a bijection between the stable matchings of J and the stable matchings of I. One small problem with this conjecture is that, unlike SM, an instance of SR may not admit any stable matchings. Figure 31 gives an example of such an instance: It is easy to see that a matching containing \{p_1, p_4\}, \{p_2, p_4\} or \{p_3, p_4\} is blocked by \{p_1, p_3\}, \{p_2, p_1\} or \{p_3, p_3\} respectively [45]. We therefore restrict our attention to solvable instances of SR. Determining the truth of this restricted conjecture is the ninth open problem of Gusfield and Irving [28].

\[
\begin{align*}
{p_1}: & \quad p_2 \quad p_3 \quad p_4 \\
{p_2}: & \quad p_3 \quad p_1 \quad p_4 \\
{p_3}: & \quad p_1 \quad p_2 \quad p_4 \\
{p_4}: & \quad \text{arbitrary}
\end{align*}
\]

Figure 31: Instance of SR that admits no stable matchings [45].

In this chapter, we give a polynomial-time reduction from SR to a generalization of SM, which we call MAX-SMRI. Under this reduction, there is a bijection between stable matchings in an instance I of SR and complete weakly stable matchings in the constructed instance J of MAX-SMRI. We regard this correspondence as a first step towards solving Gusfield and Irving’s ninth open problem.

### 6.2 Reduction from SR to MAX-SMRI

Recall that SMTI is a generalization of SM in which preference lists may contain ties and be incomplete. Here, we define a new problem, SMRI, which generalizes SMTI by permitting non-transitive acyclic preference lists. So, for example, if \(m\) prefers \(w\) to \(w'\), and \(w'\) to \(w''\), then unless explicitly specified, it is not the case that \(m\) prefers \(w\) to \(w''\). Preference lists in SMRI must still be acyclic, so, in the previous example, \(m\) cannot prefer \(w''\) to \(w\).
Although the definition of SMRI may seem unnatural, non-transitive preference lists do have some justification in the literature [18]. In fact, May [52] has experimental evidence that people sometimes even have cyclic preferences. In this experiment, May gave 62 students a sequence of binary comparisons between three potential marriage partners, denoted by $x$, $y$ and $z$. The marriage partners were described in terms of three categories: intelligence, appearance and financial situation.

<table>
<thead>
<tr>
<th>Partner</th>
<th>Intelligence</th>
<th>Appearance</th>
<th>Financial Situation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>Very Intelligent</td>
<td>Plain</td>
<td>Well Off</td>
</tr>
<tr>
<td>$y$</td>
<td>Intelligent</td>
<td>Very Good Looking</td>
<td>Poor</td>
</tr>
<tr>
<td>$z$</td>
<td>Fairly Intelligent</td>
<td>Good Looking</td>
<td>Rich</td>
</tr>
</tbody>
</table>

Of the 62 students, 17 preferred $x$ to $y$, $y$ to $z$ and $z$ to $x$. These cyclic preferences arise because $x$ has 2 of 3 attributes superior to $y$ (intelligence and financial situation), $y$ has 2 of 3 attributes superior to $z$ (intelligence and looks), and $z$ has 2 of 3 attributes superior to $x$ (looks and financial situation). Although this example demonstrates that non-transitive (even cyclic) preferences can occur in practice, we are considering SMRI since it is the most restrictive version of SM to which we are currently able to reduce instances of SR.

Let $J$ be an instance of SMTI (and therefore of SMRI). A matching $M$ of $J$ is weakly stable if there is no (man, woman) pair not in $M$, each of whom is either unmatched in $M$ or prefers the other to their partner in $M$. Figure 32 gives an instance of SMTI/SMRI that admits two distinct weakly stable matchings, $M_1 = \{(m_1, w_1)\}$ and $M_2 = \{(m_1, w_2), (m_2, w_1)\}$.

We say that $M_2$ is a complete weakly stable matching, since it matches every participant. MAX-SMTI (respectively MAX-SMRI) is the problem of deciding if an instance of SMTI (respectively SMRI) admits a complete weakly stable matching, and finding such a matching if one exists.

$$
m_1 : w_1 w_2 \quad w_1 : (m_1 m_2)\\
m_2 : w_1 \quad w_2 : m_1$$

Figure 32: Instance of SMTI/SMRI.

Iwama et al [41] proved that MAX-SMTI is NP-hard. This result holds for MAX-SMRI as well, since MAX-SMRI generalizes MAX-SMTI. Hence, the reduction we present below is from a polynomial-time solvable problem, namely SR [35], to an NP-hard problem, namely MAX-SMRI. Although this
makes the reduction of little use from a computational perspective, we are primarily interested in exploring the structural relationship between the two problems, which, ultimately, we hope will help solve Gusfield and Irving’s ninth open problem. We now present the reduction.

Given an instance \( I \) of SR, the reduction function constructs an instance \( J \) of MAX-SMRI according to Gusfield and Irving’s broad suggestion that each person \( p_i \in I \) be mapped to a male copy, \( m_i \), and female copy, \( w_i \). Our plan is, given any complete weakly stable matching \( M_0 \) of \( J \), to construct a stable matching \( M \) of \( I \) such that if \( (m_i, w_j) \in M_0 \), then \( (p_i, p_j) \in M \).

This plan imposes two immediate constraints on \( M_0 \), and therefore on the constructed instance \( J \):

**Irreflexivity:** \( (m_i, w_i) \notin M' \).

**Symmetry:** \( (m_i, w_j) \in M' \) only if, for \( k \neq i \), \( (m_j, w_k) \notin M' \), and for \( k \neq j \), \( (m_k, w_i) \notin M' \).

The irreflexivity constraint ensures that \( p_i \) is never matched with him/herself in \( M \). This constraint is easily satisfied - the reduction function need only declare \( m_i \) and \( w_i \) mutually unacceptable in \( J \).

The symmetry constraint ensures that if \( p_i \) and \( p_j \) are matched in \( M \), then \( p_j \) cannot also be matched with \( p_k \neq p_i \). We haven’t found any way to enforce this constraint in a reduction to SM, and hence the reduction presented here uses the more complex preference list structures allowed by MAX-SMRI. These structures are detailed below.

The preference list of each man \( m_i \) consists of a copy of \( p_i \)’s preference list, in which each \( p_j \) is converted to \( w_j \). Transitivity of preference holds between all triples of these women. Additionally, for each woman \( w_j \) in \( m_i \)’s preference list, we add a new woman \( y_{i,j} \) such that,

1. \( m_i \) prefers \( y_{i,j} \) to \( w_j \).
2. if \( m_i \) prefers \( w_k \) to \( w_j \), then \( m_i \) prefers \( w_k \) to \( y_{i,j} \).

There are no other transitive preferences in \( m_i \)'s list. So, for example, \( m_i \) is indifferent between \( y_{i,j} \) and any other \( y_{i,k} \) on his list, and also between \( y_{i,j} \) and any woman \( w_k \) such that \( m_i \) prefers \( w_j \) to \( w_k \). Each woman \( w_i \) has a symmetrical construction, with the new type of men labelled \( x_{i,j} \).

The reduction’s proof of correctness will show that whenever \( m_i \) is matched to \( w_j \) in a complete weakly stable matching of \( J \), it is necessarily the case that \( m_j \) is matched to \( y_{i,j} \), and \( w_i \) is matched to \( x_{i,j} \). So, when we construct the roommates matching, the symmetry property is guaranteed to hold.
To enforce this situation in $J$, the additional men, $X$, and women, $Y$, must have special preference lists. Before giving these lists, we firstly remark that if $I$ has $2n$ people, then $|X| = 2n C_2$, since there are $2n$ women copied from $I$, and the indices of $x_{i,j}$ must be distinct by the irreflexivity property. Of these $2n C_2$ men, only $n$ will be used in the matching we are aiming for. This leaves $2n C_2 - n$ men from $X$ remaining, and since the matching must be complete, we construct a further set of $2n C_2 - n$ women $V$, which we call garbage collectors. The garbage collectors are indifferent between all men in $X$. Finally, we can specify the preference lists of each $x_{i,j} \in X$: $x_{i,j}$ is indifferent between $w_i$ and $w_j$, and prefers either to all women in $V$, who are ranked equally. This same construction applies to the women in $Y$, and so we have a corresponding set $U$ of male garbage collectors, where $Y$ and $U$ have analogous preference structures to $X$ and $V$ respectively.

Figure 33 gives a concrete instance of SR with 4 people, $p_1, p_2, p_3$ and $p_4$. The figure also gives the corresponding preference structure in MAX-SMRI of $m_1$, the male copy of $p_1$, as well as $x_{1,2}$. In these structures, a vector from participant $p_i$ to $p_j$ indicates that $p_i$ is preferred to $p_j$.

$$p_1 : p_4 \ p_2 \ p_3$$
$$p_2 : p_1 \ p_3 \ p_4$$
$$p_3 : p_2 \ p_4 \ p_1$$
$$p_4 : p_2 \ p_1 \ p_3$$

Figure 33: Instance of SR and two corresponding preference structures in MAX-SMRI.

This reduction function is clearly polynomial-time computable. We now show that every stable matching in the roommates instance has a corresponding complete weakly stable matching in the marriage instance.

Suppose $M$ is a stable matching for the roommates instance $I$. Construct a matching $M'$ of $J$ in the following way:

1. For all $\{p_i, p_j\} \in M$ where $i < j$, add $(m_i, w_j)$, $(m_j, y_{i,j})$, and $(x_{i,j}, w_i)$
to \( M' \).

2. Add an arbitrary complete matching between unmatched members of \( X \) and members of \( V \).

3. Add an arbitrary complete matching between unmatched members of \( Y \) and members of \( U \).

It follows from the construction that \( M' \) is a complete matching in \( J \). Now, suppose for a contradiction that \((b_m, b_w)\) is a blocking pair for \( M' \). We remark that \( b_m \notin U \) and \( b_w \notin V \) since no members of \( U \) or \( V \) have any strict preferences and \( M' \) is a complete matching. Also, members of \( X \) and \( Y \) are mutually unacceptable, therefore \( b_m \) and \( b_w \) cannot simultaneously belong to \( X \) and \( Y \) respectively. It must be the case then that at least one of \( b_m \) and \( b_w \) corresponds to a person in \( I \). We deal with each case in turn.

Suppose \( b_m = x_{\{r,s\}} \in X \). Given \( b_m \)'s preference list, \( b_w \) can only be \( w_r \) or \( w_s \). Without loss of generality then, we will assume that \((x_{\{r,s\}}, w_r)\) forms a blocking pair for \( M' \). Now, \( w_r \) prefers \( x_{\{r,s\}} \) to her partner in the matching, who, by construction, must therefore be \( m_s \). So, \((m_s, w_r) \in M' \), which implies that \((m_r, y_{\{r,s\}}) \) and \((x_{\{r,s\}}, w_s) \) are in \( M' \). But this is a contradiction, since \( x_{\{r,s\}} \) is indifferent between \( w_r \) and \( w_s \), and therefore does not prefer \( w_r \) to his partner in the matching. A symmetrical argument proves that no \( y_{\{r,s\}} \) can be a member of a blocking pair either.

The only remaining case is that some pair \((m_i, w_j)\) blocks \( M' \). Firstly, we remark that \( \{p_i, p_j\} \notin M \) for otherwise (i) \((m_i, w_j) \in M' \), a contradiction, or (ii) \((m_i, y_{\{i,j\}}) \in M' \), which again leads to a contradiction since \( m_i \) prefers \( y_{\{i,j\}} \) to \( w_j \). Let \( \{p_i, p_s\} \neq \{p_r, p_j\} \) be matched roommates in \( M \), so that, by construction, \( m_i \) is matched with \( w_s \) or \( y_{\{i,s\}} \), and \( w_j \) is matched with \( m_r \) or \( x_{\{r,j\}} \). Now, if \( m_i \) is matched to \( w_s \), then since \((m_i, w_j) \) blocks \( M' \), \( m_i \) prefers \( w_j \) to \( w_s \). Otherwise, \( m_i \) is matched to \( y_{\{i,s\}} \), and so \( m_i \) prefers \( w_j \) to \( y_{\{i,s\}} \) and therefore to \( w_s \). In either case then, \( m_i \) prefers \( w_j \) to \( w_s \), and similarly, \( w_j \) prefers \( m_i \) to \( m_r \). Now, if we project these preferences back into the roommates instance, it is clear that \( p_i \) prefers \( p_j \) to their partner \( p_s \) in \( M \), and \( p_j \) prefers \( p_i \) to their partner \( p_r \) in \( M \). So \( \{p_i, p_j\} \) forms a blocking pair, contradicting the stability of \( M \).

Hence, \( M' \) has no blocking pairs and is therefore a complete weakly stable matching in the transformed instance \( J \).

Conversely, we show that every complete weakly stable matching in the marriage instance has a corresponding stable matching in the roommates instance.

Suppose \( M' \) is a complete weakly stable matching in the marriage instance \( J \). Since \( M' \) is complete, every garbage collector in \( U \) must be matched to
some woman in Y, and every garbage collector in V must be matched to some man in X. This leaves n people in both X and Y, and, by construction, these people must be matched with male and female copies of people in I. Since there are 2n men and 2n women corresponding to people in I, there are exactly n pairs of the form \((m_i, w_j) \in M'\).

Construct M so that for every pair \((m_i, w_j) \in M', \{p_i, p_j\} \in M\). We have already shown that M' is irreflexive (since \(m_i\) and \(w_i\) are mutually unacceptable), and therefore M must also be irreflexive. Suppose for a contradiction that M' is not symmetric because, for example, \((m_i, w_j) \in M'\) and \((m_j, w_k) \in M'\), where \(w_i \neq w_k\). Now, \(w_k\) prefers \(x_{\{j,k\}}\) to \(m_j\), and since \(x_{\{j,k\}}\) is not matched to \(w_k\) (\(m_j\)’s partner) or \(w_j\) (\(m_i\)’s partner), \((x_{\{j,k\}}, w_k)\) forms a blocking pair for M'. This is a contradiction, since M' is weakly stable, and therefore M' must satisfy the necessary symmetry constraint, which together with the cardinality and irreflexivity arguments, means that M is a complete matching in the stable roommates instance.

It remains to show that M is stable in the roommates instance. Suppose for a contradiction that M is not stable due to some blocking pair \(\{p_j, p_k\}\), where \(\{p_i, p_j\}\) and \(\{p_k, p_l\}\) \(\in M\). Then, \(p_j\) prefers \(p_k\) to \(p_i\), and \(p_k\) prefers \(p_j\) to \(p_i\). It follows then that \(m_j\) prefers \(w_k\) to \(w_i\), and \(w_k\) prefers \(m_j\) to \(m_l\). Now, \(m_j\) is matched in M' to either \(w_i\) or \(y_{\{i,j\}}\), and \(w_k\) is matched to either \(m_l\) or \(x_{\{k,l\}}\). Clearly then, \((m_j, w_k)\) blocks M', giving the required contradiction.

### 6.3 Conclusion and Open Problems

In this chapter, we have presented a polynomial-time reduction from SR to MAX-SMRI with the property that there is a bijection between stable matchings in an instance I of SR and complete weakly stable matchings in a corresponding instance J of MAX-SMRI. Although this reduction is from a polynomial-time solvable problem to an NP-hard problem, we believe that the correspondence is still useful from a structural perspective. In particular, a next step might be to use similar preference list structures in a correspondence between SR and MAX-SMTI (adding a transitivity of preference requirement to MAX-SMRI). A further goal might be to strengthen the stability criterion from complete weak stability to strong or super stability, both of which are polynomial time solvable. This work, although still not directly answering Gusfield and Irving’s ninth open problem, should then provide some intuition and machinery to help find the required reduction or prove that no such reduction exists.
7 Minimum Maximal Matching in Graphs

7.1 Introduction

Recall from Section 1.3.3 that in an instance of SMTI, every weakly stable matching is at least half the size of a maximum cardinality weakly stable matching [51]. The proof of this result is based on property that every stable matching $M$ is maximal, meaning that there can be no two agents who are both unmatched in $M$ and find each other acceptable. In this chapter, we examine the concept of a maximal matching in more detail, specifically focusing on the problem of finding a minimum cardinality maximal matching.

7.2 Background

Let $G = (V, E)$ be an arbitrary undirected graph with $n$ vertices and $m$ edges. A matching $M$ of $G$ is maximal if no proper superset of $M$ is a matching. Maximal matchings can also be defined in terms of edge domination. We say that an edge $e = \{u, v\}$ dominates any edge in $\text{dom}(e) = \{e' \in E : e' \text{ is incident to } u \text{ or } v\}$. An edge dominating set of $G$ is a subset $D$ of $E$ such that every edge in $E$ is dominated by at least one edge in $D$. An independent edge dominating set is an edge dominating set in which no two edges are adjacent. A matching $M$ is maximal then if and only if $M$ is an independent edge dominating set.

The problem of finding a maximal matching is solvable in $O(n+m)$ time using the classical greedy algorithm in Figure 34. The algorithm builds an edge set $M$ by repeatedly adding elements from a candidate pool $E'$, where $E'$ is the set of all edges in $E$ that are not dominated by an edge in $M$. Once an edge $e$ is added to $M$, all edges in $\text{dom}(e)$ are removed from $E'$. This guarantees $M$ is a matching, and since the algorithm continues until $E'$ is empty, $M$ is also maximal.

GreedyMaximalMatching($G = (V, E)$)

\[
\begin{align*}
M &:= \emptyset; \\
E' &:= E; \\
\text{while } E' \neq \emptyset \\
\quad e &:= \text{any edge in } E' \\
\quad M &:= M \cup \{e\}; \\
\quad E' &:= E' \setminus \text{dom}(e); \\
\text{return } M
\end{align*}
\]

Figure 34: Greedy algorithm for finding a maximal matching.
A maximum cardinality maximal matching is just a maximum matching. The problem of finding a maximum matching is solvable in \(O(m\sqrt{n})\) time [54].

A minimum cardinality maximal matching is called a minimum maximal matching. In contrast to finding a maximum matching, Yannakakis and Gavril [69] proved that the problem of finding a minimum maximal matching (MMM) is NP-Hard, even for planar or bipartite graphs with maximum degree 3. Horton and Kilakos [34] subsequently proved hardness results for a number of other graph classes, including planar bipartite graphs and perfect claw-free graphs.

MMM has several important practical applications, such as analysing the worst case performance of certain telephone networks [69]. The problem is also important in approximating minimum vertex cover (MVC), a fundamental problem well-known to be NP-hard [43]. We say that a vertex \(v\) covers any edge incident to \(v\). A vertex cover of a graph \(G = (V, E)\) is a subset \(C\) of \(V\) such that every edge in \(E\) is covered by some vertex in \(C\). The MVC problem is to find a minimum cardinality vertex cover. Yannakakis and Gavril [69] use maximal matchings in the following 2-approximation for MVC.

Let \(M\) be any maximal matching of \(G\), and let \(C\) be the set of \(2|M|\) vertices matched by \(M\). Since \(M\) is maximal, every edge in \(E\) is adjacent to some vertex in \(C\), and so \(C\) is a vertex cover. Now, the size of a minimum vertex cover, denoted by \(\alpha_0(G)\), must be at least \(|M|\), since every edge in \(M\) must be covered, and no two edges of \(M\) are adjacent. Therefore, \(|C| = 2|M| \leq 2\alpha_0(G)\). In practice then, we prefer smaller cardinality maximal matchings, since these describe smaller vertex covers.

Along with theoretical interest, these applications have motivated the search for positive results. MMM is known to be polynomial-time solvable for various classes of graphs, including trees [55, 69], claw-free chordal graphs and the line graphs of total graphs and chordal graphs [34]. Korte and Haussmann [46] proved that the size of any maximal matching is no larger than \(2\beta_1(G)\), where \(\beta_1(G)\) denotes the size of a minimum maximal matching of \(G\). Zito [70] improved this bound to \((2 - \frac{1}{d})\beta_1(G)\), where \(G\) is a regular graph of degree \(d\). On the basis of these two results, GreedyMaximalMatching is the best known approximation algorithm for arbitrary and regular graphs. Although the reductions given in [69] indirectly prove that MMM is APX-complete, Baker [8] found a PTAS for the restricted case that \(G\) is planar. More recently, Zito [70, 71] has found upper and lower bounds on the expected value of \(\beta_1(G)\) in random graphs. This work shows that on average, we can expect the size of a maximal matching to be much less \(2\beta_1(G)\).

In this section, we present a small observation (Corollary 7.5) that allows
us to give the first improvement on Korte and Hausmann’s 2-approximation for arbitrary graphs. We use this observation as the basis of three different approximation algorithms. The first algorithm, ApproxMMM1, repeatedly finds and applies reducing paths, an approach based on Berge’s augmenting path theory [9]. The second algorithm, ApproxMMM2, uses GreedyMaximalMatching to construct several maximal matchings, returning the smallest that it finds. The third algorithm, ApproxMMM3, generalizes ApproxMMM2 by using an arbitrary r-approximation algorithm and guaranteeing to return a maximal matching with size less than \( r\beta^{-}(G) \). We briefly explore applying this idea to other NP-hard problems, improving on the best known approximation algorithms for MVC and maximum satisfiability.

7.2.1 Preliminary Results

**Lemma 7.1 (Korte and Hausmann [46])** Let \( M_1 \) and \( M_2 \) be maximal matchings of some graph \( G = (V, E) \). Then \( |M_1| \leq 2|M_2| \).

**Proof:** Suppose for a contradiction that \( |M_1| > 2|M_2| \). We remark that since \( M_1 \) is a matching, \( |M_1| \leq \alpha_0(G) \). Now, let \( C \) be the vertex cover of \( G \) consisting of the set of vertices matched in \( M_2 \). We have that \( |C| = 2|M_2| < |M_1| \leq \alpha_0(G) \). This is a contradiction, since \( \alpha_0(G) \) is the minimum size of any vertex cover of \( G \).

**Corollary 7.2** Let \( M \) be a maximal matching of some graph \( G = (V, E) \). If \( M \) is not a maximum matching, then \( |M| < 2\beta^{-}(G) \).

**Lemma 7.3** Let \( M \) be a non-empty maximal matching of some graph \( G = (V, E) \), and let \( e \) be any edge in \( M \). Then, \( M \setminus \{e\} \) is a maximal matching of \( G' = (V, E \setminus \text{dom}(e)) \).

**Proof:** Suppose for a contradiction that \( M \setminus \{e\} \) is not maximal in \( G' \). Then there must be some edge \( e' \in E \setminus \text{dom}(e) \) such that \( e' \) is not dominated by any edge in \( M \setminus \{e\} \). We also have that \( e' \notin \text{dom}(e) \) in \( G \), for otherwise \( e' \notin E \setminus \text{dom}(e) \). Hence, \( e' \) is not dominated by any edge in \( M \), contradicting the maximality of \( M \).

**Lemma 7.4** Let \( M_1 \) and \( M_2 \) be two maximal matchings of some graph \( G = (V, E) \). If \( |M_1| = 2|M_2| \), then \( M_1 \cap M_2 = \emptyset \).
Proof: Suppose for a contradiction that $|M_1| = 2|M_2|$ and $M_1 \cap M_2 \neq \emptyset$. Let $e$ be some edge in $M_1 \cap M_2$, and consider the graph $G' = (V, E \setminus \text{dom}(e))$. By Lemma 7.3, $M'_1 = M_1 \setminus \{e\}$ and $M'_2 = M_2 \setminus \{e\}$ are both maximal matchings of $G'$. Now, $|M'_1| = |M_1| - 1$ and $|M'_2| = |M_2| - 1$. So, $|M'_1| = 2|M_2| - 1 = 2(|M'_2| + 1) - 1 = 2|M'_2| + 1$, contradicting Lemma 7.1.

Corollary 7.5 Let $M$ be any maximal matching of some graph $G$. If $M$ has an edge in common with any minimum maximal matching of $G$, then $|M| < 2\beta^*_1(G)$.  

7.2.2 Reducing Paths

In this section, we define an $M$-reducing path and show how, given a maximal matching, such paths may be used to construct smaller maximal matchings.

Let $M$ be some matching of an arbitrary undirected graph $G = (V, E)$. An $M$-reducing path $P = \langle v_1, v_2, \ldots, v_{k-1}, v_k \rangle$ is an $M$-alternating path, where

1. $\{v_1, v_2\}, \{v_{k-1}, v_k\} \in M$.
2. $\{v_1, v_k\} \notin E$.
3. Any vertex adjacent to $v_1$ or $v_k$ is matched in $M$.

Norman and Rabin [58] give a more restrictive definition of an $M$-reducing path, in which $v_1$ must not be adjacent to $v_{k-1}$, and $v_k$ must not be adjacent to $v_2$. They use this definition as the basis of a polynomial-time algorithm for the minimum edge cover problem. An edge cover of a connected graph $G = (V, E)$ is a subset $C$ of $E$ such that every vertex in $V$ is incident to at least one edge in $C$. There are significant differences between maximal matchings and edge covers, making Norman and Rabin’s original definition unsuitable for our purposes.

Lemma 7.6 Let $G = (V, E)$ be an arbitrary graph that admits an $M$-reducing path $P = \langle v_1, v_2, \ldots, v_k \rangle$, where $M$ is a maximal matching of $G$. Then $M' = M \bigoplus P$ is a maximal matching of $G$ with size $|M'| = |M| - 1$.

Proof: By basic augmenting path theory, $M'$ is clearly a matching with size $|M'| = |M| - 1$. It remains to show that $M'$ is maximal. With the exception of $v_1$ and $v_k$, every vertex matched in $M$ is also matched in $M'$. So, any

---

7In fact, this statement holds for any other maximal matching of $G$, not just a minimum maximal matching. However, we will only use the corollary as given.
edge not incident to $v_1$ or $v_k$ is dominated by some edge in $M'$. Now, since there is no edge between $v_1$ and $v_k$, and every vertex adjacent to $v_1$ or $v_k$ is matched in $M$, any edge incident to $v_1$ or $v_k$ must also be dominated in $M'$. Hence, $M'$ is a maximal matching. 

\[
\text{ApproxMMM1}(G = (V, E))
\]
\[
M := \text{GreedyMaximalMatching}(G);
\]
\[
\text{while } G \text{ admits an } M\text{-reducing path } P
\]
\[
M := M \oplus P;
\]
\[
\text{return } M;
\]

Figure 35: Reducing path approximation algorithm for MMM.

The correctness of the first new approximation algorithm, ApproxMMM1, is based on Lemma 7.6. The algorithm begins with an arbitrary maximal matching $M$, and repeatedly finds and applies $M$-reducing paths. This approach follows naturally from the classical augmenting path algorithm, and so it may not be immediately clear that ApproxMMM1 can fail to find a minimum maximal matching. In fact, as we demonstrate in the next example, ApproxMMM1 can even fail on trees.

Construct the graph $G[k] = kP_4$, consisting of $k$ disjoint $P_4$ subgraphs. Now, add a connecting edge between a new vertex $v$ and exactly one degree 1 vertex in each $P_4$ component. Let $M$ be the maximal matching of $G[k]$ consisting of two edges from each of the $P_4$ subgraphs ($|M| = 2k$). Clearly, $G[k]$ admits no $M$-reducing path, since, in each $P_4$ subgraph, one of the matched edges is adjacent to a connecting edge that has the unmatched vertex $v$ as an endpoint. However, $G[k]$ admits a maximal matching of size $k+1$, consisting of one (middle) edge in each $P_4$ subgraph, along with any one of the connecting edges. Figure 36 gives two maximal matchings of $G[3]$, one a minimum maximal matching with size 4, while the other is the matching described above with size 6. In both cases, the matching edges are specified in bold.

**Theorem 7.7** For an arbitrary graph $G = (V, E)$ with $n$ vertices and $m \geq 1$ edges, ApproxMMM1 returns a maximal matching of $G$ with size at most $(2 - \frac{2}{n}) \beta_1^*(G)$.

**Proof:** Let $M$ be the matching returned by ApproxMMM1 on graph $G$. Since the algorithm begins with a maximal matching, and applying a reducing path
preserves maximality (Lemma 7.6), it must be the case that $M$ is maximal. Now, if $M$ is not a maximum matching of $G$, then by Corollary 7.2, $|M| < 2\beta_1(G)$. Otherwise, $M$ is a maximum matching. We will show that $M$ has the same size as a matching $M'$, where $M'$ is a maximal matching that has an edge in common with a minimum maximal matching $M^*$ of $G$. It will follow then from Corollary 7.5 that $|M| < 2\beta_1(G)$.

Let $e = \{u, v\}$ be any edge in $M^*$. If $e$ is in $M$, then $M' = M$ and we are done. Otherwise, since $M$ is maximal, (i) exactly one of $u$ and $v$ is matched in $M$, or (ii) both $u$ and $v$ are matched in $M$. We deal with each case in turn.

(i) exactly one of $u$ and $v$ is matched in $M$.

Suppose without loss of generality that $u$ is unmatched in $M$ and $v$ is matched in $M$ to some vertex $w$. Now, since $M$ is maximal and $u$ is unmatched, $u$ can only be adjacent to vertices that are matched in $M$. Similarly, $w$ cannot be adjacent to an unmatched vertex $x \neq u$, for otherwise $\langle x, w, v, u \rangle$ is an $M$-augmenting path, contradicting the assumption that $M$ is a maximum matching. Clearly, $M' = (M \setminus \{\{v, w\}\}) \cup \{e\}$ is a maximal matching containing $e$ with $|M'| = |M|$.

(ii) both $u$ and $v$ are matched in $M$.

Let $\{u, u'\}$ and $\{v, v'\}$ be edges in $M$. Now, if $\{u', v'\} \in E$, then $\langle u, v, v', u' \rangle$ forms an $M$-alternating cycle and we can construct $M' = (M \setminus \{\{u, u'\}, \{v, v'\}\}) \cup \{e, \{u', v'\}\}$, which has the desired properties. Otherwise, $\{u', v'\} \not\in E$. It follows that at least one of $u'$ and $v'$ must be adjacent to an unmatched vertex, for otherwise $G$ admits the $M$-reducing path $\langle u', u, v, v' \rangle$. However, $u'$ and $v'$ cannot both be adjacent to distinct unmatched vertices, say $u''$ and $v''$, for otherwise $\langle u'', u', u, v, v', v'' \rangle$ forms an $M$-augmenting path (contradicting the assumption that $M$ is a maximum matching). Suppose then that (i)
both \( u' \) and \( v' \) are adjacent to the same unmatched vertex \( u'' \), or (ii) without loss of generality, only \( u' \) is adjacent to \( u'' \). In either case, 
\[ M' = (M \setminus \{\{u, u'\}, \{v, v'\}\}) \cup \{e, \{u', u''\}\} \]
contains \( e \) and \( |M'| = |M| \).

So \( |M| \leq 2\beta_1^{-1}(G) - 1 \), which means \( |M| \) is at most \( \left(2 - \frac{1}{\beta_1^{-1}(G)}\right) \beta_1^{-1}(G) \). Now, \( \beta_1^{-1}(G) \leq \frac{n}{2} \), since no matching can match more than \( n \) vertices. Substituting this inequality for \( \beta_1^{-1}(G) \), we get that \( |M| \leq \left(2 - \frac{2}{n}\right) \beta_1^{-1}(G) \).

The time-complexity of ApproxMMM1 is not known, although we conjecture that the problem is polynomial-time solvable using techniques similar to those employed in finding a maximum matching of an arbitrary graph [54].

7.2.3 Restricted Brute Force

In this section, we give another approximation algorithm for MMM, which is again based on Corollary 7.5. The main idea of this algorithm is to find a maximal matching that has at least one edge in common with a minimum maximal matching. We subsequently extend this idea to give a third approximation algorithm, which guarantees a better worst case approximation ratio than any known algorithm for MMM.

Let \( e \) be any edge in some graph \( G = (V, E) \), and let \( M^* \) be a minimum maximal matching of \( G \). Now, since \( M^* \) is maximal, there must be some edge in \( M^* \) that dominates \( e \). Equivalently, \( M^* \cap dom(e) \neq \emptyset \). Consider the algorithm given in Figure 37.

\[
\text{ApproxMMM2}(G = (V, E))
\]
\[
\begin{align*}
&\text{if } E = \emptyset \text{ return } \emptyset; \\
&e := \text{any edge in } E; \\
&M := \emptyset; \\
&\text{for each } e' \in \text{dom}(e) \\
&M_{e'} := \text{GreedyMaximalMatching}(G' = (V, E \setminus \text{dom}(e'))); \\
&M_{e'} := M_{e'} \cup \{e'\}; \\
&M := M \cup \{M_{e'}\}; \\
&\text{return } M \in M \text{ minimizing } |M|.
\end{align*}
\]

Figure 37: A \( (2 - \frac{4}{n+2}) \)-approximation algorithm for MMM.

**Theorem 7.8** For an arbitrary graph \( G = (V, E) \) with \( n \) vertices and \( m \geq 1 \) edges, ApproxMMM2 (i) runs in \( O(n^2 + nm) \) time, and (ii) returns a maximal matching with size at most \( \left(2 - \frac{2}{n}\right) \beta_1^{-1}(G) \) \(^8\).

\(^8\)We prove a stronger bound of \( \left(2 - \frac{4}{n+2}\right) \beta_1^{-1}(G) \) in Theorem 7.9.
Proof:

(i) The maximum size of $\text{dom}(e)$ is $2n - 3$, and for each edge in $\text{dom}(e)$, ApproxMMM2 finds a maximal matching using the GreedyMaximal-Matching algorithm which runs in $O(n + m)$ time. Hence, the overall running time of ApproxMMM2 is $O(n^2 + nm)$.

(ii) Let $M$ be the matching returned by ApproxMMM2. If, for some $M' \in \mathcal{M}$, $|M|$ is strictly less than $|M'|$, then, by Corollary 7.2, $|M| < 2\beta_1^-(G)$. Otherwise, all matchings in $\mathcal{M}$ have the same cardinality, and at least one of these matchings must have an edge in common with a minimum maximal matching of $G$. Hence, by Corollary 7.5, $|M| < 2\beta_1^-(G)$, and so following the same argument given in Theorem 7.7, $|M| \leq (2 - \frac{2}{n}) \beta_1^-(G)$.

ApproxMMM2 uses a restricted brute force approach to find a maximal matching $M$ that has an edge $e$ in common with a minimum maximal matching of $G = (V, E)$. This matching, which may not ultimately be returned by the algorithm, consists of the edge $e$ and a 2-approximation of a minimum maximal matching in the subgraph $G' = (V, E \setminus \text{dom}(e))$. Of course, we are not restricted to using GreedyMaximalMatching for this 2-approximation. In practice, any $r$-approximation algorithm suffices, and indeed, for smaller $r$, ApproxMMM2 is able to guarantee a better approximation ratio. We can further improve ApproxMMM2 by insisting that the returned matching is no larger than a maximal matching that has $c > 1$ edges in common with some minimum maximal matching of $G$. These generalizations are encapsulated in the next approximation algorithm, ApproxMMM3, which accepts an integer $c \geq 0$ and an arbitrary $r$-approximation algorithm, rApproxMMM, where $r > 1$. For non-trivial graphs and $c \geq 1$, ApproxMMM3 is guaranteed to return a maximal matching with size less than $r\beta_1^-(G)$, thereby providing a stronger approximation guarantee than rApproxMMM.

**Theorem 7.9** Given an arbitrary graph $G = (V, E)$ with $n$ vertices and $m \geq 1$ edges, an integer constant $c \in [0, \beta_1^-(G)]$, and an $r$-approximation algorithm rApproxMMM with time complexity $O(T)$, ApproxMMM3 (i) runs in $O(n^c T)$ time, and (ii) returns a maximal matching no larger than $\left( r - \frac{2cr}{n + 2c(r-1)} \right) \beta_1^-(G)$.

Proof:

(i) Since $|\text{dom}(e)| = O(n)$, in the worst case $O(n^c)$ recursive calls are made to rApproxMMM, each of which costs $O(T)$. 93
ApproxMMM3(G = (V, E), c, rApproxMMM)
if E = ∅ return ∅;
if c = 0 return rApproxMMM(G);
c := any edge in E;
M := ∅;
for each e' ∈ dom(e)
  M_e' := ApproxMMM3(G' = (V, E \ dom(e')), c − 1, rApproxMMM);
  M_e' := M_e' ∪ {e'};
  M := M ∪ {M_e'};
return M ∈ M minimizing |M|.

Figure 38: \( r - \frac{2cr}{n+2c(r-1)} \)-approximation algorithm for MMM.

(ii) Let M' be a maximal matching with c edges in common with some minimum maximal matching M* of G = (V, E). Also, let G' be the graph obtained from G after removing the 2c vertices matched in both M' and M*. Now, for some real value \( r' \in [1, r] \), M' induces a maximal matching of size \( r'\beta_1^{-}(G') \) in G', and since G' has n − 2c vertices, \( \beta_1^{-}(G') \leq \frac{n−2c}{2c} \). The approximation ratio of M' in G is therefore given by:

\[
\frac{|M'|}{|M^*|} = \frac{c + r'\beta_1^{-}(G')}{c + \beta_1^{-}(G')} = r' - \frac{c(r' - 1)}{c + \beta_1^{-}(G')} \leq r' - \frac{2cr'(r' - 1)}{n + 2c(r' - 1)} = f(r')
\]

Although tedious, it is not too difficult to show that \( f(r) > f(r') \), for any \( r > r' \), and so M' is also an \( f(r) \)-approximation of M*. Using a simple inductive argument based on the correctness of ApproxMMM2, it is easy to show that ApproxMMM3 finds such a matching M', and ultimately returns a maximal matching M with size \( |M| \leq |M'| \leq (r - \frac{2cr}{n+2c(r-1)})\beta_1^{-}(G) \).

Although ApproxMMM3 is based on brute force, the running time remains polynomial in n as long as c is a constant. In general, larger values of c improve the approximation guarantee, but do so at a significant time complexity cost. This behaviour is similar to a polynomial-time approximation scheme, except that the approximation ratio does not converge to 1 as c increases. Figure 39 gives the worst case approximation ratio of ApproxMMM3 for different values of c. This example uses GreedyMaximalMatching as the basis algorithm, and the ratios given for \( c = \lg(n) \) indicate a practical upper bound.
on the approximation performance, since ApproxMMM3 is super-polynomial when \( c \) is not a constant function of \( n \).

<table>
<thead>
<tr>
<th>( c )</th>
<th>25</th>
<th>50</th>
<th>100</th>
<th>1000</th>
<th>10000</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1.85185</td>
<td>1.92308</td>
<td>1.96078</td>
<td>1.99601</td>
<td>1.9996</td>
<td>1.99996</td>
</tr>
<tr>
<td>2</td>
<td>1.72414</td>
<td>1.85185</td>
<td>1.92308</td>
<td>1.99203</td>
<td>1.9992</td>
<td>1.99992</td>
</tr>
<tr>
<td>( \lg(n) )</td>
<td>1.45825</td>
<td>1.63165</td>
<td>1.76542</td>
<td>1.96092</td>
<td>1.99470</td>
<td>1.99934</td>
</tr>
</tbody>
</table>

Figure 39: Approximation Performance of ApproxMMM3 to 5 decimal places.

We now give a class of graphs for which ApproxMMM3 may exhibit the worst case approximation ratio when the GreedyMaximalMatching algorithm is used. Construct the graph \( G[c, k] = cC_6 + kP_4 \), consisting of \( c \) disjoint 6-vertex cycles and \( k \) disjoint 4-vertex paths, where \( k \) is an arbitrary non-negative integer. Select an edge \( e_i = \{u_i, v_i\} \) from each disjoint component, where \( \deg(u_i) = \deg(v_i) = 2 \). Now, add a connecting edge between \( v_i \) and \( u_{i+1} \) for all \( 1 \leq i < k + c \), thereby connecting the graph. It is easily verified that \( \beta_1(G[c, k]) = k + 2c \), with one edge for each \( P_4 \) component and two edges for each \( C_6 \) component. However, ApproxMMM3 may return a matching as large as \( 2k + 3c \) by including 2 edges from each \( P_4 \) component, and 3 edges from each \( C_6 \) component. This worst case happens if, during the brute force phase of the algorithm, ApproxMMM3 selects one edge from each \( C_6 \) component, where the edge is not dominated by a connecting edge. Since the number of vertices in \( G[c, k] \) is \( n = 6c + 4k \), the returned matching has approximation ratio,

\[
\frac{2k + 3c}{k + 2c} = \frac{2(k + 2c) - c}{k + 2c} = 2 - c \frac{c}{k + 2c} = 2 - 4c \frac{c}{n + 2c}
\]

An example of \( G[c, k] \) is given in Figure 40 with \( c = 2 \) and \( k = 1 \). This example describes two maximal matchings of \( G[2, 1] \), one, a minimum maximal matching with size 5, while the other is the result of a worst case execution of ApproxMMM3 with size 8. In both cases, the matching edges are specified in bold.

Our approach of using brute force to improve on an approximation algorithm is applicable beyond MMM. To illustrate this, we briefly sketch new approximation algorithms for MVC and MAXIMUM SATISFIABILITY (MS).

Recall that the MVC problem for a graph \( G = (V, E) \) is to find a minimum cardinality subset \( C \) of \( V \) such that every edge in \( E \) is covered by some vertex in \( C \). It follows that for each edge \( e = \{u, v\} \), \( u \) or \( v \) must be in some minimum vertex cover. The new approximation algorithm, ApproxVC,
exploits this property by recursively constructing two vertex covers - one for \( u \) and one for \( v \). The correctness and analysis of \( \text{ApproxVC} \) is essentially the same as for \( \text{ApproxMMM3} \), so we only remark that by using Halperin’s \( (2 - 2\ln \ln(n)) (1 - o(1)) \)-approximation algorithm [32] as the basis \( r \)-approximation, \( \text{ApproxVC} \) is the best known vertex cover approximation. The algorithm is given in Figure 41, where we use the term \( \text{cov}(v) \) to denote the set of all edges incident to \( v \in V \).

\[
\text{ApproxVC}(G = (V, E), c, r\text{ApproxVC})
\]

\[
\begin{cases}
\text{if } E = \emptyset & \text{return } \emptyset; \\
\text{if } c = 0 & \text{return } r\text{ApproxVC}(G); \\
\text{e} = \{u, v\} := \text{any edge in } E; \\
C_u := \{u\} \cup \text{ApproxVC}(G' = (V, E\setminus \text{cov}(u)), c - 1, r\text{ApproxVC}); \\
C_v := \{v\} \cup \text{ApproxVC}(G' = (V, E\setminus \text{cov}(v)), c - 1, r\text{ApproxVC}); \\
\text{return } \text{smaller of } C_u \text{ and } C_v;
\end{cases}
\]

Figure 41: Improved approximation algorithm for minimum vertex cover.

\text{MAXIMUM SATISFIABILITY (MAX-SAT)} is the optimization version of the first known NP-complete problem, \text{SATISFIABILITY} [14]. In these problems, we are given a set \( V \) of \textit{variables}, and a collection \( C \) of disjunctive \textit{clauses}, each of which consists of a set of variables or their negations. The aim of MAX-SAT is to find a truth assignment for \( V \) that maximizes the number of clauses that subsequently evaluate to true. For any variable \( v \) and some truth assignment \( A \), \( v \) is either assigned \textit{true} or \textit{false} in \( A \). As in the previous two examples, we can recursively try both options, removing any true clauses, and removing from all remaining clauses any instances of \( v \) or the negation of \( v \). Again, the correctness and analysis of this approach is essentially the same as for \( \text{ApproxMMM3} \), and so we only remark that by using as the basis algorithm the 1.2746-approximation given in [6], we achieve the best known approximation for MAX-SAT.
7.2.4 Conclusions and Open Problems

In this section, we have presented three new approximation algorithms for MMM. The first algorithm uses reducing paths, which are analogues of classical augmenting paths. The last two algorithms use a restricted brute force approach to improve on existing approximation algorithms. These algorithms may be viewed as weaker forms of polynomial-time approximation schemes, where the approximation guarantee converges to some constant greater than 1.

There are a number of avenues for future work. For example, it may be possible to place the last two algorithms into a more general theory of approximation, in which we extend the definition of a polynomial-time approximation scheme. Also, the time complexity of finding a reducing path, or determining that no such path exists, is open. However, we conjecture that the problem is polynomial-time solvable.
References


