12-2010

Batch Testing, Adaptive Algorithms, and Heuristic Applications for Stable Marriage Problems

John Dabney
Clemson University, jdabney@g.clemson.edu

BATCH TESTING, ADAPTIVE ALGORITHMS, AND HEURISTIC APPLICATIONS FOR STABLE MARRIAGE PROBLEMS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
John Randolph Dabney, Jr.
December 2010

Accepted by:
Brian C. Dean, Ph.D., Committee Chair
David Jacobs, Ph.D.
Wayne Madison, Ph.D.
Brian Malloy Ph.D.

Abstract

In this dissertation we focus on different variations of the stable matching (marriage) problem, initially posed by Gale and Shapley in 1962 [16]. In this problem, preference lists are used to match n men with n women in such a way that no (man, woman) pair exists that would both prefer each other over their current partners. These two would be considered a blocking pair, preventing a matching from being considered stable. In our research, we study three different versions of this problem.

First, we consider batch testing of stable marriage solutions. Gusfield and Irving presented an open problem in their 1989 book *The Stable Marriage Problem: Structure and Algorithms* [18] on whether, given a reasonable amount of preprocessing time, stable matching solutions could be verified in less than $O(n^2)$ time. We answer this question affirmatively, showing an algorithm that will verify k different matchings in $O((m+kn)\log^2 n)$ time.

Second, we show how the concept of an adaptive algorithm can be used to speed up running time in certain cases of the stable marriage problem where the disorder present in preference lists is limited. While a problem with identical lists can be solved in a trivial $O(n)$ running time, we present an $O(n+k)$ time algorithm where the women have identical preference lists, and the men have preference lists that differ in k positions from a set of identical lists. We also show a visualization program for better understanding the effects of changes in preference lists.

Finally, we look at preference list based matching as a heuristic for cost based matching problems. In theory, this method can lead to arbitrarily bad solutions, but through empirical testing on different types of random sources of data, we show how to obtain reasonable results in practice using methods for generating preference lists "asymmetrically" that account for long-term ramifications of short-term decisions. We also discuss several ways to measure the stability of a solution and how this might be used for bicriteria optimization approaches based on both cost and stability.

Acknowledgements

I would like to thank my advisor, Dr. Brian Dean, for his help and support through all that went into both this dissertation and the master's thesis that came before it and for the experience of being able to work with him these last few years. I could not ask for a better advisor, and I say this not just because he is a brilliant professor and amazing at everything I have seen him try but because of how he goes out of his way to interact with his students and to be more than just an advisor. I have been very lucky to work with him for this short time. I also appreciate all the support and friendship that came from the Applied Algorithms Group.

I would also like to acknowledge Dr. Stephen Hedetniemi for inspiring me to go in this direction for my graduate work and for pointing me toward a new professor in the department when I was looking for an advisor. My committee over the years has included him, Dr. Dean, Dr. Robert Geist, Dr. Harold Grossman, Dr. Sandra Hedetniemi, Dr. David Jacobs, Dr. Wayne Madison, and Dr. Brian Malloy, and I would like to thank them all for everything they have done.

Finally, I wouldn't have been able to do any of this without my family and friends. Thank you all for being there for me. I needed it.

Table of Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

1 Introduction **1**
1.1 Stable Matching Background and Notation 2
1.2 The Gale-Shapley Algorithm 3
1.3 Overview of Results 4

2 Batch Testing of Stable Marriage Solutions **6**
2.1 Preliminaries 7
2.2 A Connectivity-Based Characterization 12
2.3 An Efficient Batch Algorithm for Verifying Stability 13
2.4 Concluding Remarks 14

3 Adaptive Stable Marriage Algorithms **15**
3.1 Preliminaries 16
3.2 The Algorithm 18
3.3 Future directions 20

4 Cost-Based Versus Ordinal Matching **25**
4.1 Greedy Approaches 28
4.2 "Symmetric" Preference List Construction and Greedy Solutions 30
4.3 Asymmetric Preference List Construction 33
4.4 Bicriteria Approaches 42

5 Conclusions **45**
5.1 Future Work 46
Bibliography 47

List of Tables

4.1 A comparison of greedy and optimal solutions. Percentage values are (greedy-optimal)/optimal * 100. 30
4.2 Percent improvements over greedy while using pagerank to construct preference lists. 35
4.3 Percent improvements over greedy while using additive regret to construct preference lists. 37
4.4 Percent improvement over greedy while using multiplicative regret to construct preference lists. . . . 38
4.5 Percent improvement over greedy while using a more long term look at regret to construct preference lists. 40
4.6 Percent improvement over greedy while using a combination of all methods to construct preference lists. 41
4.7 Statistics measuring instability based on number of unstable pairs and how far the most extreme blocking pairs are in each instance. 43

List of Figures

1.1 An example instance of the stable marriage problem, with men and their preference lists on the left, and women and their preference lists on the right. A stable matching is shown. 3

2.1 An example of the stable marriage problem with solutions and rotations marked. Men and their preference lists appear on the left-hand side, and women and their preference lists appear on the right-hand side. Note that the man-optimal solution (shown with boxes highlighted in bold, as well as with lines matching each man with his corresponding partner) is the leftmost marked elements on the men's side (man 1 with woman 6, man 2 with woman 4, man 3 with woman 2, and so on) and the woman-optimal solution is the rightmost marked elements on the men's side. The arrows mark rotations moving from one stable solution to the next. 9

2.2 An example instance of the stable matching problem. In (a), the men and their preference lists are shown on the left of the bipartite matching graph G, and the women on the right. The eight rotations (labeled $A \ldots H$) are shown overlaid on the preference lists, and the reduced rotation DAG is shown in (b) to the right of the matching instance. A stable matching $\mathscr{M}$ is indicated with shaded entries in the men's preference lists, and its corresponding closure is outlined in the rotation DAG. The expanded rotation graph H' is shown in (c). Note that the removal of the edges $P(i, j)$ corresponding to $(i, j) \in \mathscr{M}$ leaves H' with two connected components, one containing π^- and the other containing π^+. 11

3.1 Pseudocode for our algorithm. 19

3.2 An instance of a W-consistent stable marriage problem is shown in (a). Rectangular boxes show what the solution would have been if all men had the identical preference list $1, 2, \ldots, n$. The circles show which entries in the men's preference lists have changed from these identical lists, and the triangles show the new stable solution for individuals for whom it is different from the original. In (b), we see two snapshots showing information at different times in the execution of the algorithm on the provided instance. They show the list R of women who remain single but would not have, if preference lists had not been changed. Also shown are the list of changes C_m for the current man m and the boolean array S indicating which women are still single. 21

3.3 An example of a simple change in preference lists causing an $\Theta(n)$ change in solution. 23

4.1 Trivial example that demonstrates how bad the greedy approach can perform in the worst case. Illustration (a) shows a bipartite assignment problem with optimal solution in bold, and (b) shows the greedy solution to this problem, whose cost is arbitrarily bad compared to the cost of the optimal solution. In (c), we see that the solution of a stable marriage instance obtained by "symmetric" construction of preference lists is the same as the greedy solution. 27

4.2 Trade-off curve between cost and stability of solutions. 43

Chapter 1

Introduction

Stable matching is a computational problem involving pairing elements in such a way that no pair (or group) would all prefer to break off and change their local assignments instead of staying with their current configuration. David Gale and Lloyd Shapley proposed and solved the stable marriage problem in 1962 [18], and since that time its variants have grown into a large area of research and theory in the field of computing. They have numerous practical applications, such as roommate matching, hospital residency matching, job scheduling, and product delivery [16, 30, 9, 4].

In this dissertation, we present work that has been completed in the last three years on a collection of problems in the domain of stable matching. We explore batch testing to answer an open question proposed by Gusfield and Irving, we apply the concepts of "adaptive" algorithms to stable matching, and we experiment with the practical use of preference list versus cost based matching.

The format of the document is as follows. This chapter includes general notation, motivation, background material, and a review of relevant literature. The next three chapters each focus on the three principal areas of research undertaken in this work. Each will use terminology and notation that has been described in earlier chapters, but otherwise they will be somewhat self-contained. Notation, terminology, and details from the literature pertaining specifically to one chapter will be found in that chapter. Chapter 2 is mostly drawn from a previously published paper of the author and his advisor in Algorithmica [8], and Chapter 3 is adapted from a paper published in the proceedings of the ACM Southeast Conference [7].

1.1 Stable Matching Background and Notation

The stable marriage problem was first introduced in 1962 by Gale and Shapley [16], and forms the basis for our area of research. The first problem they considered was to match applicants and colleges based on factors such as preference and a quota q that colleges would seek to fill but not exceed. Since the colleges could not merely accept their top q applicants because some might choose to go elsewhere, and because they did not have certain details such as which other colleges were preferred by the applicants or would accept the applicants, they could only come "reasonably close" to the desired number of students. Gale and Shapley sought to achieve a solution satisfactory to both groups while exactly filling quotas if possible. They introduced the notion of an unstable assignment in which an applicant and a college are not paired together but both prefer each other to their current assignment. In this situation, both would benefit by breaking their current arrangements and accepting that applicant to that college. The college and applicant in this case are called a *blocking pair* [16].

As a special instance of stable matching, Gale and Shapley presented the *stable marriage problem*. The classical stable marriage problem has input consisting of a set M of men and a set W of women, where $|M| = |W|$. Each man $i \in M$ submits a ranked preference list over all the women, and each woman $j \in W$ submits a ranked preference list over all the men (we use variables i and j instead of m and w to avoid any potential confusion with the m edges in an instance). The goal is to find a matching $\mathscr{M}$ between M and W that is stable, meaning that it contains no *blocking pair* – a pair $(i, j) \notin \mathscr{M}$ for which i and j would both prefer to leave their current assignment in $\mathscr{M}$ and to match with each other. An example of a stable marriage problem is shown in Figure 1.1. Note that an individual always prefers being matched over being single, and therefore that every stable matching must be perfect (pairing up every man and woman) since otherwise we would have an unmatched man i and woman j who together would form a blocking pair.

Gale and Shapley proved constructively that a stable matching always exists by giving a simple $O(n^2)$ time algorithm for finding such a matching, where $n = |M| = |W|$. This running time is linear in the size of the input (so in the worst case, the process of solving a stable matching problem takes no longer than that of simply reading its input), and moreover there is a matching $\Omega(n^2)$ worst-case lower bound on the running time of any algorithm for this problem, as shown in [18]. However, McVitie and Wilson [25] show that there are possible speedups in practice, and for many common instances, the Gale-Shapley algorithm runs much faster than in $O(n^2)$ time, often computing a stable matching in closer to $O(n)$ time if most of the men end up with highly-preferred partners.

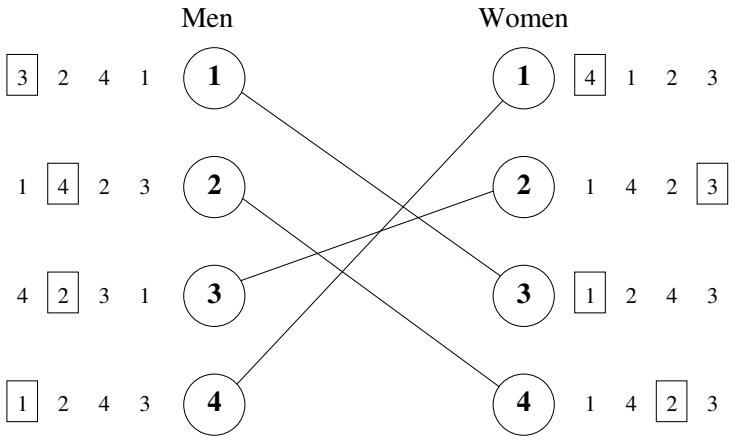

Figure 1.1: An example instance of the stable marriage problem, with men and their preference lists on the left, and women and their preference lists on the right. A stable matching is shown.

One of the most prominent examples of stable matching used in practice today is the National Resident Matching Program (NRMP), which is a centralized service to coordinate the assignment of over 30,000 medical school graduates to hospitals in the USA [30, 32, 31]. The hospitals and the applicants both submit preference lists that are used to find a stable solution.

1.2 The Gale-Shapley Algorithm

The Gale-Shapley algorithm for computing a stable matching involves a series of proposals from one sex to the other and ends with everybody married in such a way as to avoid blocking pairs. As the algorithm runs, the men propose to the women (although it works the same way if women do the proposing). Each man will ask for, in order from most to least desired, the hands of the women on his preference list in marriage. If a woman is currently single, she temporarily accepts his offer and "keeps him on a string" instead of permanently accepting him [16]. Unfortunately, this union can be broken if at some point another man proposes to that woman who is higher on her preference list than her current partner, in which case she immediately leaves for the better offer. The newly single man will then pick himself up and continue down his list, looking for another woman who will accept his proposal. When everyone is married, the algorithm terminates.

The algorithm above runs in $\Theta(n^2)$ time in the worst case time due to the possibility of n proposals to $n-1$ women and 1 proposal to the remaining woman [18]. In Chapter 3, we will see how to streamline

these proposals in order to speed up the Gale-Shapley algorithm for instances in which there is a high degree of "consistency" in the men's preference lists.

Several remarkable properties of stable matching solutions follow easily from the Gale-Shapley algorithm. First, all men and women always end up matched at termination, since every woman, once matched, stays matched forever, and a woman j cannot end up unmatched, since this would imply that there is also an unmatched man i, who would have at some point in time proposed to j. Secondly, the matching obtained by the Gale-Shapley algorithm is always stable (thereby giving a constructive proof that every instance of the stable marriage problem admits a stable solution). If the resulting matching were to contain a blocking pair (i, j), then this means i is matched with someone he prefers less than j. However, this would mean that j rejected i at some point, so j is matched with someone she prefers *more* than i, thereby contradicting the assumption that (i, j) is a blocking pair. Finally, let us say that j is a *stable partner* of i if i and j can be matched in some stable matching. It can be shown (see [18] for a simple proof) that the Gale-Shapley algorithm always outputs a stable matching $\mathscr{M}^m$ that is *man-optimal*, in which every man is matched with his most-preferred stable partner. Interestingly, $\mathscr{M}^m$ can also be shown to be *woman-pessimal*, since each woman ends up with her least-preferred stable partner (and this situation would be reversed if the women did the proposing instead of the men). It is possible that only one stable solution exists, in which case both men and women have their best and worst stable partners. Because the proposers will always end up with their best possible stable matching, an interesting corollary is that it does not matter in which order the proposals happen; the Gale-Shapley algorithm always outputs $\mathscr{M}^m$ irrespective of the men's proposal order. In Chapter 2, we will discuss in further detail the structure of the set of all possible stable solutions to an instance, along with an intriguing augmenting structure called a *rotation* that allows us to move from one stable solution to another.

1.3 Overview of Results

We narrow our focus in this work to a few specific problems dealing with stable matching. First, we have developed a batch testing algorithm for stable marriage solutions. Gusfield and Irving presented an open question of whether with "reasonable" preprocessing (they suggest $O(n^4)$) the stability of many matchings can be checked faster than checking each separately [18]. We answer this question affirmatively by describing an algorithm that will accomplish what they are asking which runs in $O((m+kn)\log^2 n)$ time, verifying each matching in $O(n\log^2 n)$ amortized time after $O(m\log^2 n)$ preprocessing. Our algorithm uses dynamic graph

connectivity data structures to test whether a certain matching cleanly splits a graph of the "rotations" that exist for given preference lists into different connected components.

Second, we introduce an "adaptive" approach to the stable marriage problem. Some problems, for example sorting, admit the possibility of algorithms that have running times not merely based on the size of the input but also some other measure of its inherent complexity. This approach cannot improve the running time in the worst case, but it can run much faster in situations where the input has some special structure, such as a mostly-sorted array in the case of sorting algorithms. For example, the worst-case running time of insertion sort on an array $A[1...n]$ in terms of problem size is $O(n^2)$, but a more precise analysis shows that the running time is actually $\Theta(n+I)$, where I denotes the number of inversions in the input array — "out of order" pairs of elements (i, j) with $i < j$ but $A[i] > A[j]$. In other words, an inversion exists when one element appears before another element in the sequence, yet the first would be considered greater than the second. Therefore, the running time of insertion sort scales gracefully between $O(n)$ and $O(n^2)$, depending on the inherent "unsortedness" of A measured in terms of inversions. These algorithms are called "adaptive" and are useful because in practice these kinds of inputs are common [12]. Given an instance of this problem with n men and n women with initially identical lists and a set of changes to k elements on lists on one side, we provide an algorithm to obtain a stable solution in $O(n+k)$ time. We explore expanding this approach to deal with changes to both sides of the preference lists and to correct a solution after adding more cycles of change. This chapter contributes a faster way to compute stable matchings for a fairly reasonable model in practice.

Finally, motivated by the fact that "ordinal" matching problems (matching problems with preference lists) can be typically solved much faster than matching problems with explicit edge costs, we investigate the use of ordinal matching as a heuristic for quickly obtaining near-optimal solutions to cost-based matching problems. In theory, this can result in much worse than optimal results in the worst case. However, through extensive computational experiments on random instances of several types, we show that in practice, one may indeed obtain reasonable solutions to cost-based problems using a variety of heuristics based on ordinal matching. In particular, we investigate several interesting ways to transform cost-based problems into problems with preference lists, where the preference lists capture not only "immediate" costs but also longer-term effects of possible choices. As opposed to the first two chapters, this chapter focuses on empirical research where we examine practical benefits and drawbacks, instead of theoretical bounds that may be rarely encountered in practice. We also look at relaxing stability requirements, showing how we can potentially consider "bicriteria" methods that seek to obtain good solutions both in terms of stability and cost.

Chapter 2

Batch Testing of Stable Marriage Solutions

For many computational problems, it seems easier from an algorithmic standpoint to verify the correctness or optimality of a candidate solution than to solve the problem outright. Even if this is not possible, it is sometimes possible to verify a *batch* of candidate solutions more efficiently than by checking each one individually, often by preprocessing an instance of the problem to build a data structure that supports fast queries to verify candidate solutions.

We show in this chapter how to perform efficient batch testing for stability of a set of matchings in the well-studied stable marriage problem. Given a stable marriage problem instance represented by a bipartite graph having $2n$ vertices and m (generally n^2, but we reduce this by methods described later) edges, we describe an algorithm that can verify the stability of k different matchings in a batch fashion in $O((m+kn)\log^2 n)$ time, effectively verifying each single matching in $O(n\log^2 n)$ amortized time after $O(m\log^2 n)$ preprocessing time. This affirmatively answers a longstanding open question from the book of Gusfield and Irving on stable marriage problems [18] as to whether stability can be verified in a batch setting (after sufficient preprocessing) in time sub-quadratic in n; this question is motivated by existence of an $\Omega(n^2)$ lower bound on the worst-case running time for verifying stability of a single matching [27]. In their 1989 book on the stable marriage problem, Gusfield and Irving presented a number of open questions "in the hope of encouraging others to enjoy the challenge and fascination of the subject" [18]. The specific problem we answer asks whether it is possible to test for stability of matchings in a batch fashion significantly faster than

testing each matching individually. It is important to note that we cannot simply list the solutions and check against them because the maximum number of solutions grows exponentially with n; in fact, enumeration of stable marriage solutions has been found to be #P-complete [22]. McVitie and Wilson do present an algorithm that will find all stable solutions [25].

Our results are built upon a new characterization of stable matchings in terms of the connectivity a certain graph related to the underlying set of rotations induced by the stable matching instance. Using this characterization, we then apply efficient data structures for fully dynamic connectivity in graphs [19, 33] to obtain an efficient test for stability.

In the next section, we describe the model we are focusing on in detail and summarize its relevant mathematical properties. Following this, we introduce our new connectivity-based characterization of stability, upon which we build our data structure for verifying the stability of a series of candidate matchings.

2.1 Preliminaries

In this chapter, we focus on a well-studied generalization of the stable marriage problem allowing for incomplete preference lists. Here, our input is described by a bipartite graph $G = (M \cup W, E)$, with $2n$ vertices and $m = |E|$ edges. Each man $i \in M$ submits a ranked preference list over the set of women $\{j \mid (i, j) \in E\}$ to which he is adjacent, and likewise for the women. The definition of stability remains the same, except we only consider couples $(i, j) \in E$ as potential blocking pairs. In this setting, we may not be able to find a stable matching $\mathscr{M} \subseteq E$ for which all men and women are assigned. However, since our results already include an allowance for preprocessing, we assume that we have "cleaned up" our instance by eliminating any men and women who are not assigned in any stable matching. Furthermore, we assume that we have removed any edges from E that are not included in any stable matching. Both steps are easily accomplished in linear time ($O(m)$ time). The reader is referred to [18] for further details. Any matching we later attempt to verify that involves one of the vertices or edges removed during this preprocessing step can be immediately classified as unstable.

Rotations. As with many algorithmic results for the stable matching problems (e.g., [21, 22, 23]), our methods in this chapter make use of an alternative characterization of the set of all stable matchings in terms of *rotations*, cyclic augmenting structures that allow us to move from one stable matching to another. Rotations were first introduced by Irving as "all-or-nothing cycles" [21]. This term was originally used in reference to the stable roommates problem, which differs from stable marriage in that all people in the

problem belong to one set instead of being separated into men and women. Unlike stable marriage problems, an instance of the stable roommates problem can possibly have zero solutions [16]. This concept was first used in relation to the stable marriage problem in [22] in which rotations were used to give a characterization of the set of stable matchings for a given instance. More recently, Baiou and Balinski [4] and Dean and Munshi [9] have generalized the concept of rotations to solve and improve algorithms for the stable allocation problem, which is a stable bipartite matching problem with both sides having non-unit size, in which the assignments between elements are nonnegative real numbers.

For our purposes, we define a rotation $\pi = (\pi_M, \pi_W)$ as a set of men $\pi_M \subseteq M$ and a set of women $\pi_W \subseteq W$. We say that a rotation π is *exposed* in a stable matching $\mathscr{M}$ if we obtain another stable matching by advancing the assignments in π_M and retreating those in π_W, where by "advancing" we mean assigning each man $i \in \pi_M$ to an entry in his preference list at some point following his partner in $\mathscr{M}$, and by "retreating" we mean assigning each woman $j \in \pi_W$ to an entry in her preference list somewhere before her partner in $\mathscr{M}$. The process of moving from $\mathscr{M}$ to another stable matching by advancing π_M and retreating π_W is known as *eliminating* π from M. For example, in Figure 2.1, boxes around entries in preference lists indicate stable partners, and the matching currently shown (given by the left-most box in each man's preference list and the rightmost box in each woman's list) has only the light blue rotation exposed. If we were to eliminate this rotation, man one would change his partner from woman 6 to woman 5, and man 6 would change his partner from woman 5 to woman 6. Elimination of the light blue rotation also exposes the magenta rotation, whose elimination would then expose the dark blue rotation.

As mentioned before, when the men propose we get a solution where the men enjoy their best possible partners while the women end up with their worst. Every stable matching instance admits two special stable solutions, a "man-optimal, woman-pessimal" matching $\mathscr{M}^m$ where every man is assigned his first stable choice and every woman her last stable choice, and a "woman-optimal, man-pessimal" matching $\mathscr{M}^w$ where every woman is assigned her first stable choice and every man his last stable choice. Note that it is possible for these two solutions to be the same in the event that there is only one stable solution.

As shown in [25, 23], we can find rotations by breaking up one match and then continuing the Gale-Shapley algorithm until it once again stabilizes (or does not if no more solutions exist.) This process can lead, depending on the marriage that is broken, to multiple rotations being combined together in one. That is, it appears that one rotation was found, but in reality, part of that rotation could have taken place without the rest and still provided a stable solution. This occurs when a marriage is broken that is not part of the next exposed rotation. Nothing actually prevents the algorithm from finding another stable matching, provided one exists,

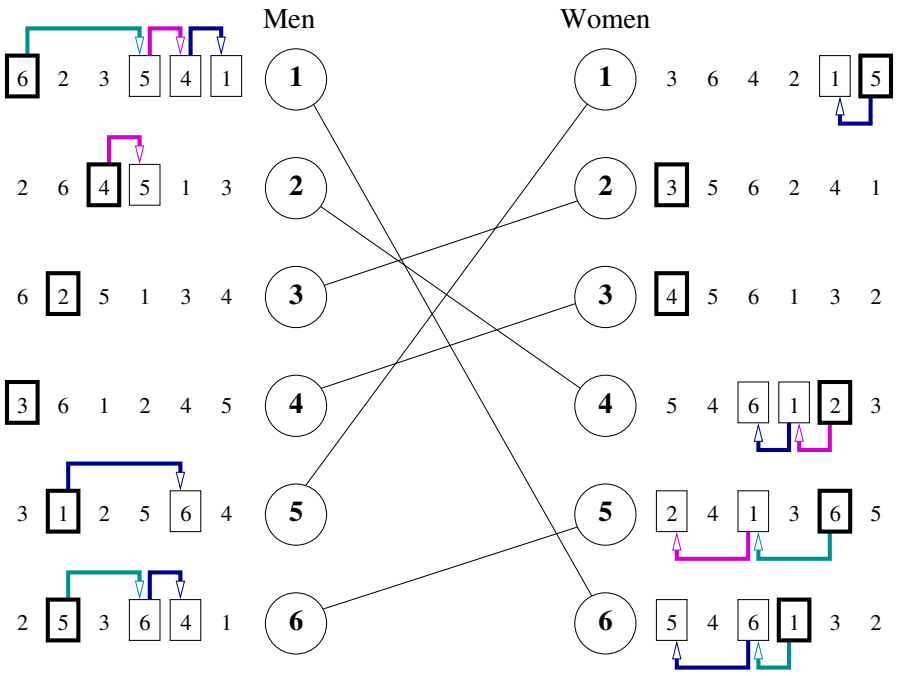

Figure 2.1: An example of the stable marriage problem with solutions and rotations marked. Men and their preference lists appear on the left-hand side, and women and their preference lists appear on the right-hand side. Note that the man-optimal solution (shown with boxes highlighted in bold, as well as with lines matching each man with his corresponding partner) is the leftmost marked elements on the men's side (man 1 with woman 6, man 2 with woman 4, man 3 with woman 2, and so on) and the woman-optimal solution is the rightmost marked elements on the men's side. The arrows mark rotations moving from one stable solution to the next.

but any rotations that would have to precede the intended one would take place as well to reach the solution. As shown in Figure 2.1, certain rotations happen after others. The same rotation (dark blue) on the first man's preference list is also on the fifth man's list. The key point to note is that on the first man's list there are two other rotations that should happen first before the dark blue rotation while that same dark blue rotation is the first (and only) one to affect the fifth man's list. If we started with the man-optimal solutions and decided to break the 5th man's marriage up and continue Gale-Shapley until it re-stabilized, we would end up with the woman-optimal solution. That would make it appear that only one rotation and 2 stable solutions exist, but as the figure shows, there are 3 possible rotations and 4 stable solutions. There are at least two ways to find an actual "next" rotation to follow in order to see all rotations. First, it can be done by breaking up each man's marriage and each time keeping track of the difference in total movement by each man down his preference list from the broken matching to the next stable solution that is found. The smallest difference is the next

rotation. Another way to find the next is to unassign an arbitrary man and restart Gale-Shapley if we are not already at the woman optimal solution. Eventually, everyone will be assigned. The last man assigned comes from the rotation with no prerequisites, so we can start over again by breaking up that man's marriage to find the next rotation.

In this chapter we have eliminated men, women, and edges not involved in any stable matching, so our definitions of rotations now change slightly compared. We obtain another stable matching from an exposed rotation by advancing the assignments in π_M and retreating those in π_W, but we now "advance" by assigning each man $i \in \pi_M$ to the entry in his preference list now **immediately following** his partner in $\mathscr{M}$, and "retreat" by assigning each woman $j \in \pi_W$ to the entry in her preference list **immediately preceding** her partner in $\mathscr{M}$, *eliminating* π from $\mathscr{M}$. Note that now our definition of a rotation is somewhat simpler than the standard definition of a rotation in the literature, owing to the fact that we have already pruned away useless entries from our preference lists. For a more detailed introduction to rotations and all of their properties we introduce in this chapter, see [21, 22, 23].

Suppose we start with the man-optimal assignment $\mathscr{M}^m$ and repeatedly move to a new stable matching by eliminating an arbitrary exposed rotation, stopping when we reach $\mathscr{M}^w$ (note that every stable matching except $\mathscr{M}^w$ exposes some rotation). The sequence of rotations we apply during this process is called a rotation elimination ordering. Remarkably, it turns out that every rotation elimination ordering involves the exact same set Π of rotations. Moreover, for each man i and each woman j in i's preference list (i.e., for each edge $(i, j) \in E$), there is a unique rotation $\pi^-(i, j) \in \Pi$ whose elimination advances i's assignment to j (unless j is the first entry on i's preference list), and there is a unique rotation $\pi^+(i, j) \in \Pi$ whose elimination advances i's assignment past j (unless j is the last entry on i's preference list). For example, Figure 2.2(a) shows an example stable matching instance with 8 rotations labeled $A \ldots H$, where $\pi^-(1,6) = C$ and $\pi^+(1,6) = D$. In order to ensure that $\pi^-(i, j)$ and $\pi^+(i, j)$ are clearly defined for every edge $(i, j) \in E$, we include two dummy rotations π^- and π^+ in Π, with $\pi^-(i, j) = \pi^-$ if j is the first woman on i's preference list, and $\pi^+(i, j) = \pi^+$ if j is the last woman on i's preference list.

The set Π of all rotations adheres to a natural partial ordering: for any two rotations $\pi, \pi' \in \Pi$, we say $\pi \prec \pi'$ if π' cannot be exposed until π is eliminated (i.e., if π precedes π' in every possible elimination ordering). Observe from Figure 2.2(a) that two rotations π and π' share a precedence relationship if and only if $\pi_M \cap \pi'_M \neq \emptyset$, in which case their relative ordering amongst the common preference lists of π_M and π'_M determines whether $\pi \prec \pi'$ or $\pi' \prec \pi$. Also note that $\pi^- \preceq \pi$ and $\pi^+ \succeq \pi$ for all $\pi \in \Pi$. We commonly represent the set of all rotations using a directed acyclic graph (DAG), D, where $V(D) = \Pi$ and $(\pi, \pi') \in E(D)$

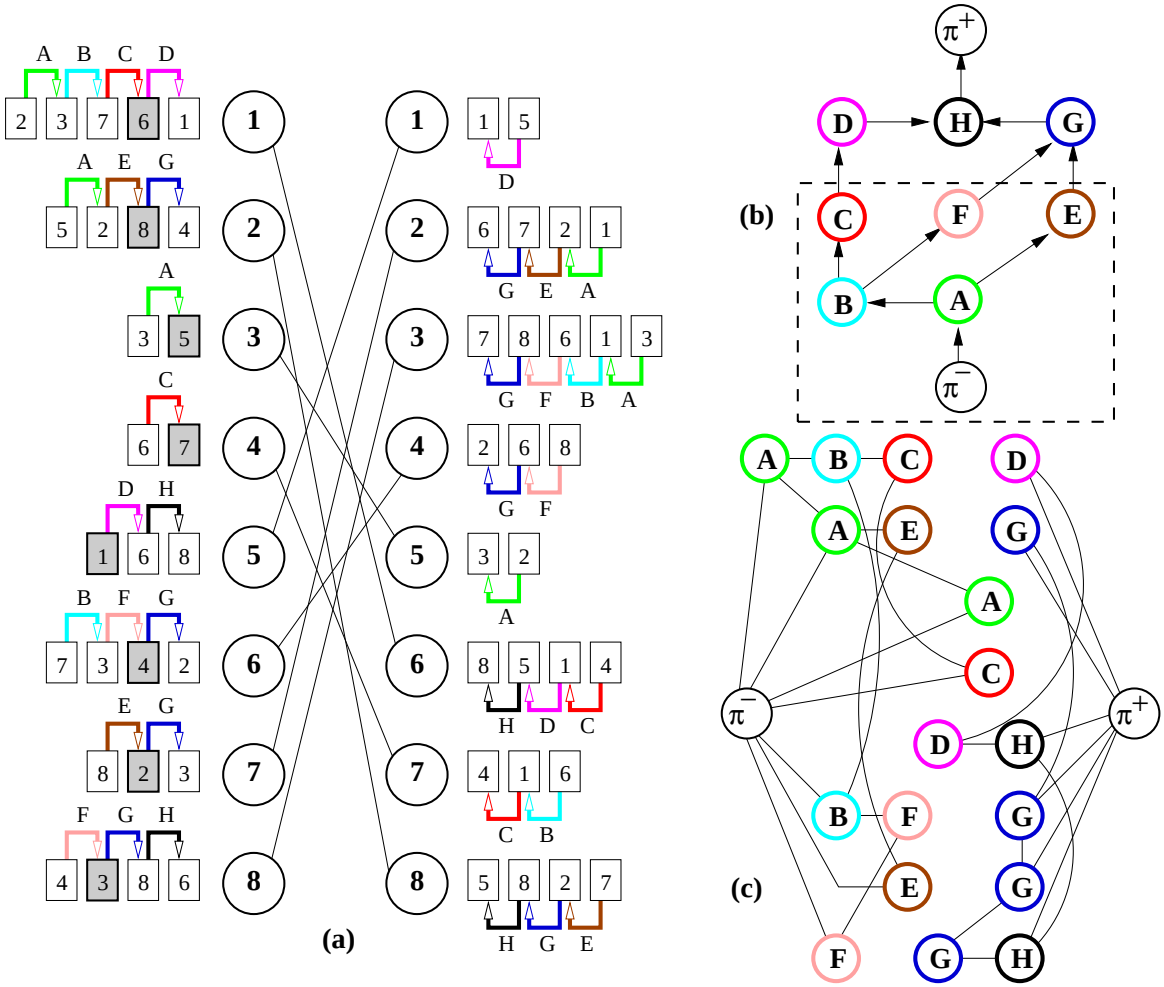

Figure 2.2: An example instance of the stable matching problem. In (a), the men and their preference lists are shown on the left of the bipartite matching graph G, and the women on the right. The eight rotations (labeled $A \ldots H$) are shown overlaid on the preference lists, and the reduced rotation DAG is shown in (b) to the right of the matching instance. A stable matching $\mathscr{M}$ is indicated with shaded entries in the men's preference lists, and its corresponding closure is outlined in the rotation DAG. The expanded rotation graph H' is shown in (c). Note that the removal of the edges $P(i,j)$ corresponding to $(i,j) \in \mathscr{M}$ leaves H' with two connected components, one containing π^- and the other containing π^+.

if and only if $\pi \prec \pi'$. From an algorithmic standpoint, it typically suffices to omit a large number of edges implied by transitivity and consider a reduced DAG D' with $V(D') = \Pi$ and $(\pi, \pi') \in E(D')$ if and only if $\pi = \pi^-(i,j)$ and $\pi' = \pi^+(i,j)$ for some edge $(i,j) \in E$. An example of such a reduced rotation DAG is shown in Figure 2.2(b). Letting $n = |M| = |W|$ and $m = |E|$ describe the size of our (pruned) bipartite input graph G, it is easy to see that $|V(D')| = |\Pi| \leq m$ and $|E(D')| = m$. One can show that D is the transitive closure of

D', and one can also compute in $O(m)$ time the values $\pi^-(i,j)$ and $\pi^+(i,j)$ for all $(i,j) \in E$, the structure of all rotations in Π, and the reduced rotation DAG D' [18].

For any subset $S \subseteq \Pi$, let S^* be the set of all rotations π such that $\pi \preceq \pi'$ for some $\pi' \in S$ (i.e., the set of all rotations π from which there exists a directed path in D (equivalently, D') to some rotation $\pi' \in S$). We say S is a *closure* if $S = S^*$. We now arrive at a key structural connection between the rotation DAG D' and stable matchings, a proof of which appears in [22].

Lemma 1. *There is a 1-to-1 correspondence between stable matchings in G and closures S in D' such that $\pi^- \in S$ and $\pi^+ \notin S$.*

The closure $S \subseteq \Pi$ corresponding to a stable matching $\mathscr{M}$ contains precisely the set of rotations that must be applied starting from $\mathscr{M}^m$, according to some valid elimination ordering (a topological ordering of S), in order to reach $\mathscr{M}$. The condition $\pi^- \in S$ and $\pi^+ \notin S$ has been added to ensure that $\mathscr{M}^m$ and $\mathscr{M}^w$ both correspond to a distinct closure; this condition holds automatically for the closure corresponding to any other stable matching $\mathscr{M}$.

2.2 A Connectivity-Based Characterization

Given an instance of the stable marriage problem, we build its *expanded rotation graph* H (an undirected graph) by including in $V(H)$ a vertex representing π^-, a vertex representing π^+, and a vertex for every pair (π, i) where $\pi \in \Pi - \{\pi^-, \pi^+\}$ and $i \in \pi_M$. The edge set $E(H)$ is given by $P \cup R$, where $P = \{P(i,j) : (i,j) \in E\}$ is a set of *precedence-related* edges, with edge $P(i,j)$ connecting vertex $(\pi^-(i,j), i)$ to vertex $(\pi^+(i,j), i)$; that is, each edge $P(i,j)$ connects the two rotation instances surrounding the edge (i,j) in i's preference list, reflecting the precedence relationship between these two rotations. The set R contains *rotation-related* edges, and consists of an an arbitrarily-chosen forest in which we have a spanning tree on the vertices originating from each distinct source rotation $\pi \in \Pi - \{\pi^-, \pi^+\}$ (i.e., the vertices (π, i) with $i \in \pi_M$). Note that $P \cap R = \emptyset$, since each rotation $\pi \in \Pi$ appears at most once in each preference list. One can show that $|V(H)| = m - n + 2 = O(m)$ and $|E(H)| = 2m - n + 2 - |\Pi| = O(m)$, and one can build H in $O(m)$ time by first computing the structure of all rotations in Π.

We now arrive at a new characterization of stability in terms of the connectivity of H:

Lemma 2. *A matching $\mathscr{M}$ is stable if and only if π^- and π^+ lie in different connected components in H, after the edge set $P(\mathscr{M}) = \{P(i,j) : (i,j) \in \mathscr{M}\}$ is deleted.*

Proof. Let H' denote H with edge set $P(\mathscr{M})$ deleted, as shown in Figure 2.2(c). Since we never delete edges from the forest R, note that each rotation $\pi \in \Pi - \{\pi^-, \pi^+\}$ remains "intact", with all its corresponding vertices $(\pi, \cdot)$ in the same connected component of H'. Also note that there is a path of precedence-related edges from π^- to the vertex $(\pi^-(i,j), i)$, and from the vertex $(\pi^+(i,j), i)$ to π^+, for every $(i,j) \in \mathscr{M}$, which implies that H' either consists of one large connected component, or two connected components, with one containing π^- and the other π^+.

Suppose $\mathscr{M}$ is stable but that H' consists of just one single component. Let $S \subseteq \Pi$ be the closure corresponding to $\mathscr{M}$ according to Lemma 1. Since $\pi^- \in S$ and $\pi^+ \notin S$, if we follow any path p from π^- to π^+ in H', we must at some point traverse a precedence-related edge $P(i,j)$ where $\pi^-(i,j) \in S$ and $\pi^+(i,j) \notin S$. But then $(i,j) \in \mathscr{M}$, so $P(i,j)$ would have been removed when constructing H', a contradiction. Suppose now that H' consists of two components. Here, we obtain a valid closure S by taking all rotations lying in the π^- component, so the corresponding matching $\mathscr{M}$ must be stable. □

2.3 An Efficient Batch Algorithm for Verifying Stability

Given the characterization from Lemma 2, let us now build the dynamic connectivity data structure of [19] on H. This structure supports the insertion and deletion of edges in an r-vertex graph in $O(\log^2 r)$ amortized time, and it can determine whether or not an arbitrary pair of vertices lie in the same connected component in $O(\log r / \log\log r)$ time. In our case, since $|V(H)| = O(m)$, we obtain amortized running times of $O(\log^2 m) = O(\log^2 n)$ and $O(\log m) = O(\log n)$ respectively. We can build the initial structure in a batch fashion (without inserting edges one by one) in only $O(m)$ time, although amortization requires that we charge $O(m \log^2 n)$ for this step, giving us a credit for future work to be redeemed later at query time.

To verify the stability of a candidate matching $\mathscr{M}$, we now simply delete the n edges $P(i,j)$ for all $(i,j) \in \mathscr{M}$ from H in $O(n \log^2 n)$ amortized time and issue a connectivity query to test whether π^- and π^+ lie in the same component. Our matching $\mathscr{M}$ is stable if and only if the answer to this query is negative. We then restore the edges $P(i,j)$ to H in anticipation of future queries. Our time for verifying k queries is therefore $O((m+kn)\log^2 n)$, which is effectively $O(n \log^2 n)$ amortized time per stability query after spending $O(m \log^2 n)$ preprocessing time.

2.4 Concluding Remarks

Note that we cannot claim an $O(n \log^2 n)$ *worst-case* bound (after preprocessing) for verifying a single matching, since due to amortization, some of the $O(m \log^2 n)$ "preprocessing time" is actually spent during queries instead of during preprocessing. It is an interesting open question to see if one can obtain a reasonable worst-case bound for queries. Presently, the best-known dynamic graph connectivity data structures with worst-case bounds [13, 11] seem insufficient to yield a stability test with worst-case running time sub-quadratic in n.

The running time of our algorithm can be easily improved if one obtains a stronger dynamic connectivity data structure. For example, Thorup [33] uses randomization to obtain an expected amortized update time of $O(\log n(\log\log n)^3)$, which gives an $O((m+kn)\log n(\log\log n)^3)$ total expected running time (whereas our original running time is deterministic). Perhaps a faster time can possibly be obtained by exploiting the special structure of H combined with the fact that we really only need to accommodate edge deletions, rather than deletions plus insertions.

Chapter 3

Adaptive Stable Marriage Algorithms

In the second branch of our research on stable matching, we introduce the concept of adaptive algorithms to the stable marriage problem. There are limits as to how fast it is possible to perform certain tasks in the worst case. For instance, $\Omega(n \log n)$ time is necessary in the worst case to run a comparison-based sorting algorithm. Recall that the worst case input for solving a stable marriage problem with n men and n women will lead to an $\Theta(n^2)$ running time. Estivill-Castro and Wood, however, present a survey of what Mehlhorn called an adaptive algorithm, one that, with certain input, possibly achieves an improvement by utilizing a running time that is based on both the size of and the "disorder" present in the input [12, 26]. They show that Burge in 1958 had seen that the order of the input would affect the choice of the best algorithm for sorting [5]. For that problem, one method of measuring disorder is the number of inversions present in the input. [12] They also provide references that show that "nearly sorted sequences are common in practice," giving motivation for the use of these adaptive sorting algorithms [12]. The running time of an adaptive sorting algorithm scales not only based on the size of input but also on the amount of disorder present in that input. Recall the example of insertion sort that scales gracefully between $O(n)$ and $O(n^2)$ based on the number of inversions in the input. This kind of analysis makes practical sense because of the common occurrence of mostly-sorted data in the real world.

In a stable marriage problem, it makes sense that preference lists would also in some cases tend to have less disorder, in this case represented by changes that exist from one list to another. If one woman is particularly attractive, it stands to reason that many men will place her high on their lists. For another example, we refer back to the NRMP that is used to match med-school graduates with hospitals. Although the preference lists submitted to this system are (for obvious reasons) confidential, one might expect highly

reputable or geographically advantageous hospitals to appear near the top of a large number of graduate preference lists, and likewise, we might see the strongest students near the top of a large number of hospital preference lists. We seek to take advantage of this behavior to create an adaptive algorithm for stable marriage problems that would improve running time in practice.

3.1 Preliminaries

Although the Gale-Shapley algorithm runs quite fast in practice, often in nearly linear time, it carries a worst-case running time of $\Theta(n^2)$ (and recall that the stable marriage problem in general involves a worst-case lower bound of $\Omega(n^2)$ [18], mostly due to the fact that its input has size $\Theta(n^2)$). By contrast, suppose we know in advance that each man in our input instance has the same preference list, and that the women also have identical preference lists; this reduces the effective size of the input to only $O(n)$, since we only need to write down one copy of each preference list. In this case, Gusfield and Irving also describe a simple inductive proof that there is a unique stable matching: for simplicity, let us assume the women are ordered according to the preference list of the men, and vice-versa. Hence, man 1 and woman 1 are both their own respective first choices, and therefore both must be matched together in any stable assignment (otherwise they would form a blocking pair). Continuing in an inductive fashion, we see that man 2 and woman 2 must be matched, and so on, leading to a trivial $O(n)$ algorithm for the special case where we are told in advance that all men share a common preference list, and all women share a common preference list. For notational simplicity, let us say that a stable marriage instance is M-consistent if it contains only one common preference list shared by all men, W-consistent if it contains one common preference list shared by all women, and MW-consistent if both of these cases apply.

Obviously, we can not expect full MW-consistency for many instances of the problem in practice. We then need to introduce disorder through a number of small changes in the preference lists which will complicate the algorithm while making it more interesting and practical. Our contribution in this chapter is an $O(n+k)$ adaptive algorithm for W-consistent instances of the stable marriage problem. This result is weaker than one that works for fully MW-consistent lists, but it does actually turn out to be similar to the behavior of some sports drafts such as that used by the NFL. The women's side with the uniform preference lists can represent an order of draft picks while the women themselves become the players. This behavior is realistic in that the players themselves don't (or at least shouldn't) have a say in by whom they are drafted, instead agreeing to play for whoever drafts them in a predetermined order based on records from the previous

season (and trades of draft picks which we will ignore for now). The men become the teams in the draft, each choosing the highest player in his list when his turn comes. Because all the women have the same order and because at this point the current man is the highest unmatched in the women's list, he will always be accepted. The fact that the men, or teams, start off with the same list and then have a few differences in their rankings of women, or players, also mimics the draft. Players with the most potential will be highly valued by every team, but different organizations will have different experiences scouting out the incoming talent, leading to slight differences. Obviously, this model is somewhat simplified from the actual draft. It actually does present a few other options for problems to study using this framework, though. We discuss these areas in the final part of this section.

There are a number of options for how to measure and encode disorder in our problem. We considered inversions, which have the problem of the possibility of a quadratic number of inversions when comparing two lists, leading to a cubic number overall and an $O(n^3)$ running time. A different method was to consider the changes necessary to move from one list to another. This method can keep the running time lower, but it is slightly less informative when comparing how similar two lists are. In a list of size n for instance, changing woman 1 with woman n could be considered a more significant change than changing woman 1 with woman 2, but both appear the same using this method. For the sake of speed, we used the second method for our algorithm, but it may be interesting to consider a measurement of disorder for preference lists that can combine both without disproportionately valuing either.

In order to prevent us from having to examine an input of size $\Omega(n^2)$, we must encode our input concisely. We can encode a nearly-MW-consistent instance as follows. For the men, we first specify a global "consensus" preference list, and then for each individual, we list only the entries in his preference list that differ from the consensus list (e.g., "entry 7 is changed to women 2, and entry 9 is changed to woman 5"). Preference lists of the women are encoded the same way. If we let k denote the total number of differing entries over all men and women, this entire encoding requires $\Theta(n+k)$ space, ranging from $\Theta(n)$ if we have complete MW-consistency, up to $\Theta(n^2)$ if preference lists are highly inconsistent.

The succinct encoding above raises the question of whether or not one can design an adaptive algorithm for the stable marriage problem running in only $O(n+k)$ time. Note that this type of adaptive algorithm result would be slightly weaker than, say, an adaptive sorting algorithm, since we still need to explicitly encode the input of the problem as above in order for the algorithm to be able to exploit partial MW consistency (in contrast to an adaptive sorting algorithm, which is not told any extra structural information beyond just the array to be sorted). However, with the stable marriage problem, succinct encoding is an unavoidable

requirement — even if we have complete MW consistency, an algorithm cannot leverage this fact unless it is informed about it in advance; otherwise, it may still need to examine essentially all of its $\Theta(n^2)$-size input. Hence, a partially MW-consistent instance is only of use to us if it is succinctly encoded[1].

Consider a succinctly-encoded W-consistent instance of the stable marriage problem with n men and n women. We number the women $1 \ldots n$ in the order of the consensus preference list of the men, and likewise we number the men $1 \ldots n$ in the order specified by the consensus preference list of the women (if the men and/or women are not ordered this way initially in our input, then it only takes $O(n)$ time to re-order them accordingly). The consensus preference list of the men now lists all women in the order $1, 2, \ldots n$, and the consensus preference list for the women now lists all men in the order $1, 2, \ldots, n$. Since our instance is W-consistent, only the men can have changes in their preference lists deviating from the consensus list. We encode the changes away from the consensus list for man m using a list C_m of ordered pairs (i, w), where each such pair indicates that index i in m's preference list is changed to woman w (note that $i \neq w$, since $i = w$ in the consensus list). We denote the total amount of change in all of the men's preferences by $k = \sum_m |C_m|$.

A matching $\mathscr{M}$ is a subset of n man-woman pairs (m, w) such that no two pairs share a common man or woman. We write $w_{\mathscr{M}}(m)$ to denote the woman partnered with m in $\mathscr{M}$, and similarly we write $m_{\mathscr{M}}(w)$ to denote the man partnered with woman w (when clear from context, we often omit the $\mathscr{M}$ subscript). A matching $\mathscr{M}$ is stable if it admits no blocking pair $(m, w) \notin \mathscr{M}$ where m prefers w to $w_{\mathscr{M}}(m)$ and w prefers m to $m_{\mathscr{M}}(w)$.

3.2 The Algorithm

The key idea of our algorithm is to speed up the implementation of the simplified Gale-Shapley algorithm above by keeping track of the set of women who are not yet assigned, but who *should have* been chosen already if the men had identical preference lists. If all men shared the same preference list $1, 2, \ldots, n$, then by the time our algorithm considered man m, we would have already assigned man i with woman i for $i = 1 \ldots m-1$. The two ways that another woman would have been preferred at the time of choosing for man i are for disorder to exist on list i or for it to exist on a previous list, leaving a woman $< i$ still available at the time of his choosing.

Hence, we keep track of a list R, stored in a doubly-linked list, which in iteration m keeps a sorted

[1] As a brief side note, observe that the related problem of determining the most succinct encoding for a partially MW-consistent instance (an encoding that minimizes k) is solvable in polynomial time by a straightforward reduction to a weighted bipartite matching problem.

```
AdaptiveStableMarriage:
 1  for each woman w = 1...n:
 2      S[w] = True
 3  for each man m = 1...n:
 4      Append m to the end of R
 5      min = n + 1
 6      for each pair (i, w) in C_m with S[w] = True:
 7          T[w] = True
 8          if i < min:
 9              w(m) = w
10              min = i
11      for all women w starting from the front of R:
12          if S[w] = False:
13              Delete w from R, Continue
14          if T[w] = False
15              if w < min:
16                  w(m) = w
17              Break
18      for each pair (i, w) in C_m:
19          T[w] = False
20      S[w(m)] = False
```

Figure 3.1: Pseudocode for our algorithm.

list of all the women in the set $\{1,\ldots,m\}$ who are still single (the array R may also end up containing some women who are assigned, but these are cleaned out of from R whenever they are encountered). When considering the partner to whom man m should propose, this will either be one of the women in C_m, or it will be the first single woman in R who is not also in C_m.

The pseudocode in Figure 3.1 illustrates the operation of the algorithm. We make use of a boolean array $S[1\ldots n]$, where $S[w] = \text{True}$ if and only if woman w is still single, and we also use a temporary boolean array $T[1\ldots n]$, where $T[w] = \text{True}$ if woman w appears in the change list C_m for the current iteration. To further clarify the structure of the algorithm, a detailed example is shown in Figure 3.2.

Observe that the algorithm assigns each man m to a woman either from his change list C_m in line 9, or alternatively he is assigned in line 16 to his most-preferred woman in R (who is not also in C_m). It is important to note that at least one of these two cases always applies, so each man must end up assigned a partner. If C_m is empty, then line 16 will definitely be triggered for at least one woman in R, since by the time we process man m, we will have added m entries to R but deleted at most $m-1$ of them, leaving at least one single woman still present in R.

Theorem 3. *Our algorithm runs in $\Theta(n+k)$ time.*

Proof. Discounting the "for" loop starting on line 11, it is clear that for each man m, we spend only linear time checking the entries in C_m, for a total of $O(n+\sum_m |C_m|) = O(n+k)$ time. To analyze the "for" loop starting in line 11, we note that the amount of time spent deleting entries from R (line 13) will be at most $O(n)$ over the entire algorithm, since each woman is added and hence also deleted at most once from R. Not counting the time spent on entries of R that are deleted, the "for" loop on line 11 spends $O(1+q)$ time, where q denotes the number of entries w in R with $T[w] =$ True (since we break the loop immediately when we encounter an entry w with $T[w] =$ False). However, since $q \leq |C_m|$, the work spent in the for loop is dominated by the running time of the rest of the algorithm. □

It is crucial that we break the "for" loop starting in line 11 immediately upon reaching an entry w in R with $T[w] =$ False, since the list R can grow quite long; for example, in an instance where C_m contains $(1, n-m+1)$ for all $m = 1 \ldots n$, R grows each iteration until $m = n/2$.

Theorem 4. *Our algorithm computes a stable matching.*

Proof. To prove this, we need only show that every man m proposes to the first single woman in his preference list; stability then follows from our earlier inductive argument. Consider now the point in time when our algorithm processes some man m who is at position m on the women's consistent preference list, and let L denote the first m women on m's preference list. We claim that at least one of the women in L must still be single, since at most $m-1$ women have already been assigned. The women in L with $S[w] =$ True are either entries from C_m (all of which are explicitly checked by our algorithm), or they belong to $\{1, 2, \ldots, m\}$, all of which have previously been added to the list R by line 4; moreover, every women $w \in \{1, 2, \ldots, m\}$ with $S[w] =$ True must still remain in R, since only women w with $S[w] =$ False are deleted from R. Among these, our algorithm explicitly examines the one most highly preferred by m. As a result, while processing m, our algorithm must consider (and issue a proposal to) the most preferred single woman $w \in L$. □

3.3 Future directions

Expanding the algorithm to related problems

The adaptive stable marriage model introduced in this paper leads to a number of interesting directions for future research. Foremost, there is the question of whether an algorithm with complexity close to $O(n+k)$ can be developed for the fully-general adaptive stable marriage problem, not assuming W-consistency as we have above. This problem seems to be quite challenging, since when changes appear

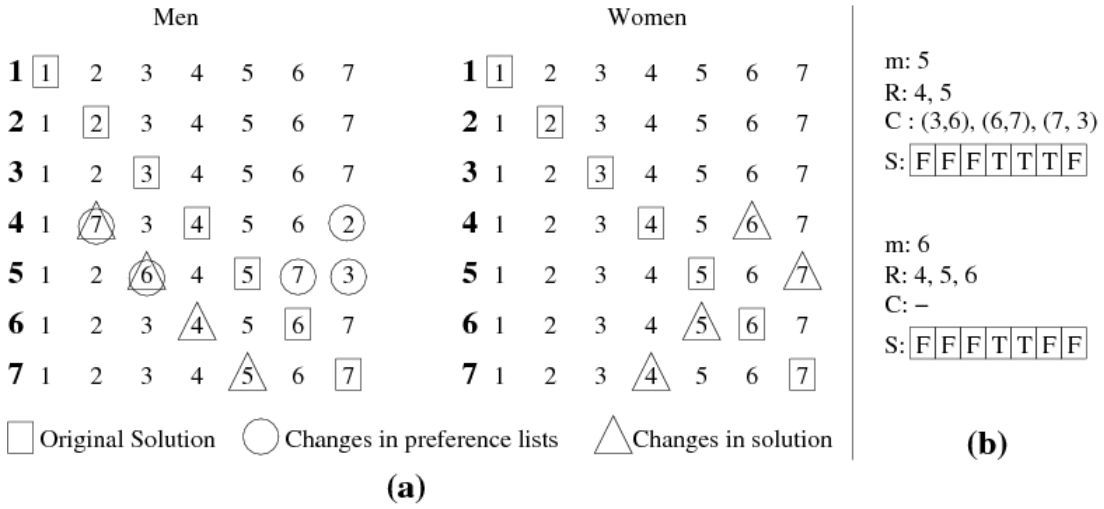

Figure 3.2: An instance of a W-consistent stable marriage problem is shown in (a). Rectangular boxes show what the solution would have been if all men had the identical preference list $1, 2, \ldots, n$. The circles show which entries in the men's preference lists have changed from these identical lists, and the triangles show the new stable solution for individuals for whom it is different from the original. In (b), we see two snapshots showing information at different times in the execution of the algorithm on the provided instance. They show the list R of women who remain single but would not have, if preference lists had not been changed. Also shown are the list of changes C_m for the current man m and the boolean array S indicating which women are still single.

in preference lists on both sides of the instance, we can lose much of the useful structure that our algorithm manages to exploit. If it seems that an efficient algorithm is possible for the general case, then a number of other problems stand to be addressed. In particular, there may no longer be a single unique stable solution, so one might want to see if it is possible to compute the *rotation DAG* that implicitly describes the set of all stable solutions in an efficient manner (see [21, 22, 23] for additional detail on the underlying structure of the set of all stable solutions).

Another interesting direction (which we believe also seems quite challenging) might be to approach the problem from a "data structure" perspective. For example, can one encode the state of a computation in an appropriate fashion so that after making a single new change (say, a single swap in one of the preference lists), a new stable solution can be recomputed more quickly than by solving the problem again from scratch? It is worth noting that a single swap in one preference list has the potential to change the assignment for every single man and woman, so a data structure solving this problem will likely need to maintain some sort of implicit encoding of the solution, which we can query to determine the current state of the assignment.

An example of this change can be seen in figure 3.3. Notice that the only change from the top instance to the bottom instance is a swap on the last man's preference list of woman 1 and woman 9, yet every man and woman end up with a different partner in the new solution. The picture itself comes from screen shots of a visualization program that we implemented to study the effects of these changes on solutions. Note that it also includes a count of rotations for situations where there can be more than one solution (not the W-consistent case). We also developed another version that displays all solutions and rotations for an instance of the problem.

Stable marriage problems with incomplete preference lists, or with preference lists containing ties, have been well-studied in the literature (see, e.g., [18]). Perhaps results for these variants can also be extended to work in the adaptive case. Another direction to consider could be expanding our algorithm to work in the case where instead of one consensus preference list we have a constant number of them.

An expanded view of the draft model

As mentioned before, a W-consistent case of the stable marriage problem can resemble a draft. This leads to a few other models that would also make sense in the context of a draft and that could be studied in the future. One problem would involve a matching in which a team is able to choose multiple players similar to the multiple rounds in a draft. This problem actually would be similar to the college admissions problem presented by Gale and Shapley, where the colleges actually try to fill a quota of applicants instead of only one. [16] To make things more interesting, in this model teams would be allowed to trade picks

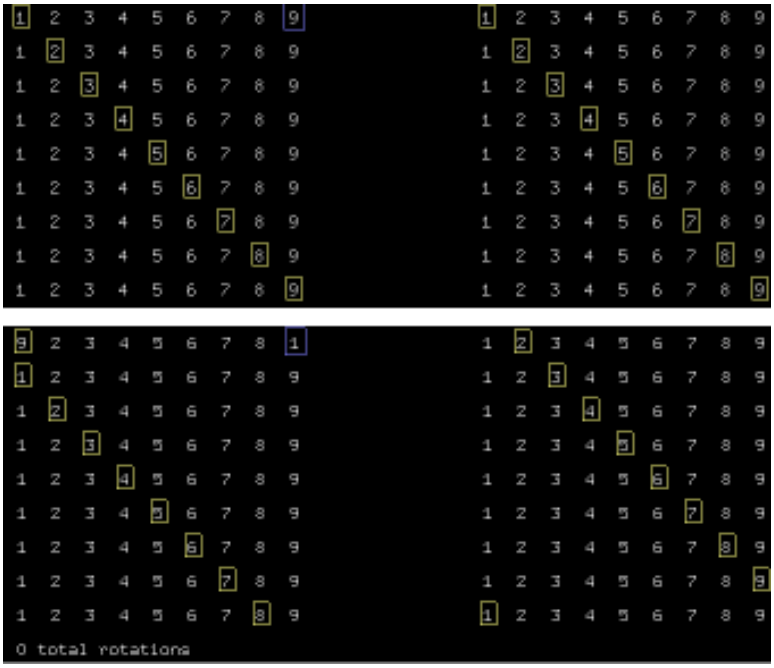

Figure 3.3: An example of a simple change in preference lists causing an $\Theta(n)$ change in solution.

when they feel it would benefit them overall. For instance, a team could have a cycle of non-uniformity in its list that moves one player much higher than the teams choosing after the current one would value him. In this case, the team would know that it could trade for a lower pick and still get the valued player while gaining added benefit in another pick. The other team would have to be willing to give up something later in exchange for moving up in the selection order to pick a player before the competition does. This trade would only make sense if both teams felt they would benefit from the transaction. This model could make even more sense with a value-based matching instead of with preference lists, where a value could be created based on where the picks take place in the draft and distance along preference lists. Another way to make this problem more complex would be adding different categories for the players in the same way that they play different positions. A team might value 3 players in the same position highly, but once one is chosen the value of the remaining players to that team could decrease. Additionally, since some teams are closer rivals than others, it can be valuable for one team to choose a player in order to prevent another team from getting him. All of these problems could be interesting from a computational standpoint. There are many experts in this field who would propose different drafting strategies, and it would be interesting to see how an algorithm would differ from their decisions. Unfortunately, the strategies that are actually used by teams

are likely kept fairly secret because of the large amounts of money that a sport represents and the competition between teams. Finally, how does the strategy of one team change when it is not aware of the preference lists of its competitors? Interestingly, in the stable marriage problem, the men all have their best matches in the same solution and then move together down their preference lists toward the woman-optimal solution. Does this suggest that teams would be better off being open about their preferences? How does this compare to previous work on dishonesty in preference lists? [20, 18]

Chapter 4

Cost-Based Versus Ordinal Matching

Several authors (e.g., [23]) have considered adding costs to the edges of an ordinal (preference list based) matching problem, in order to find solutions that are not only, say, stable, but also low in cost. In other words, these results aim to improve the quality of the solution to an ordinal matching problem by incorporating aspects of matching problems with explicit costs. In this chapter, we explore what is in some sense the opposite approach: how ordinal matching can be used as a heuristic for quickly obtaining good solutions to cost-based matching problems. We discuss, implement, and test different methods of translating costs to preference lists and compare their results. We also explore "bicriteria" approaches that trade off between stability and cost.

Bipartite matching problems with explicit edge costs have received substantial attention in the literature, and are quite prevalent in practice. The simplest example is perhaps the minimum-cost *bipartite assignment* problem, which we can write as a linear program as

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c(i,j)x(i,j) \\ \text{Subject to} \quad & \sum_{j=1}^{n} x(i,j) = 1 \quad \forall i \\ & \sum_{i=1}^{n} x(i,j) = 1 \quad \forall j \\ & x(i,j) \in \{0,1\} \quad \forall (i,j), \end{aligned}$$

where $x(i,j)$ is a decision variable that is 1 if man i is matched with woman j, and where $c(i,j)$ is the cost of matching man i with woman j. In this chapter, we focus on a generalization of this problem allowing for non-unit sized elements on the left and right side of the matching problem:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m} c(i,j)x(i,j) \\
\text{Subject to} \quad & \sum_{j=1}^{m} x(i,j) = 1 \quad \forall i \\
& \sum_{i=1}^{n} p(i)x(i,j) \leq c(j) \quad \forall j \\
& x(i,j) \in [0,1] \quad \forall (i,j).
\end{aligned}
$$

This problem appears in several domains of application; for example, it is known as the *transportation* problem in the context of optimally routing goods from factories (supply nodes) to warehouses (demand nodes). It can also serve as a good model for a common scheduling problem: given n jobs with processing times $p(1) \ldots p(n)$ and m machines with capacities $c(1) \ldots c(m)$, as well as a cost $c(i,j)$ for assigning job i to machine j, compute a minimum-cost *preemptive*[1] schedule that fully assigns every job[2], and that does not fill any machine beyond its capacity. We adopt this scheduling viewpoint for this chapter, so that we have more concrete terminology at our disposal (i.e., we can refer to "jobs" and "machines" instead of "left hand side elements" and "right hand side elements"). The "many to many" nature of this problem also necessitates that we move away from "men" and "women", to avoid any awkwardness involved with multiple-partner matchings.

Part of the motivation for our studying heuristics for the problems above is that their best-known running times are not realistic to apply to large-scale instances in practice. For example, the currently strongest running times known for minimum-cost assignment problem are $O(mn + n^2 \log n)$, due to Gabow [14], and $O(\sqrt{n \log n \alpha(m,n)} m \log(nC))$, due to Gabow and Tarjan [15], where $\alpha(\cdot)$ denotes the inverse Ackermann function. For the high-multiplicity variant of this problem (the scheduling problem above), these bounds become even worse, as the problem becomes essentially a minimum-cost flow problem. As a result, large-scale instances of these problems can be quite challenging to solve to optimality. A good case study is the company Akamai Technologies, which performs load balancing of traffic from a large number of source regions on the Internet across thousands of servers spread around the world, a problem naturally modeled as an instance of bipartite matching. Although Akamai initially relied on cost-based algorithms for its load balancing infrastructure, the increasing size of its network eventually forced the company to consider other alternatives (based on ordinal matching heuristics, as we discuss in this chapter) in order to meet its real-time performance

[1] A non-preemptive assignment is required to schedule every job in its entirety during a contiguous block of time on a single machine; a preemptive schedule has the flexibility to terminate a job before its completion and resume it later on the same machine or on a different machine. Our linear program allows preemptive solutions since our decision variables $x(i,j)$ are allowed to take fractional values in the range $[0,1]$.

[2] To ensure the existence of a feasible solution, we assume our instance has been augmented with a "dummy" machine capable with very high capacity (and high assignment cost) to which every job could be mapped; assignment of part of a job to the dummy machine corresponds to that part of the job being unassigned.

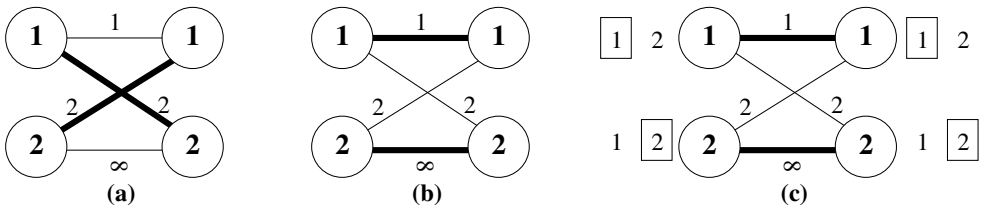

Figure 4.1: Trivial example that demonstrates how bad the greedy approach can perform in the worst case. Illustration (a) shows a bipartite assignment problem with optimal solution in bold, and (b) shows the greedy solution to this problem, whose cost is arbitrarily bad compared to the cost of the optimal solution. In (c), we see that the solution of a stable marriage instance obtained by "symmetric" construction of preference lists is the same as the greedy solution.

demands.

Cost-based matching problems that have special properties often can be solved or approximated more efficiently. Examples include input that satisfies the triangle inequality, the Monge property, and bipartite and non-bipartite problems using points in a Euclidean plane [17, 1, 6, 34]. Researchers have also considered various approximation algorithms for cost-based matching problems. For the maximum-weight matching problem in general graphs, linear-time algorithms with performance guarantees $1/2$ and $2/3-\varepsilon$ were proposed by Drake and Hougardy [10, 35], and a 2-approximation algorithm for minimum-cost perfect matching in general graphs with edge costs forming a metric (i.e., satisfying the triangle inequality) was developed by Goemans and Williamson [17]. Numerous other authors have also considered approximation approaches for cost-based matching problems, with perhaps the heuristics given by Kurtzberg in 1962 [24] being the first rigorous study of such methods. See the survey of Avis [2] for further references.

In contrast to the excessive running times required to optimally solve cost-based problems, ordinal matching problems (based on ranked preference lists) can typically be solved much faster. For example, the Gale-Shapley algorithm can solve the stable marriage problem (the ordinal analog of the bipartite assignment problem) in a bipartite graph with m edges in only $O(m) = O(n^2)$ time, and it often runs in closer to $O(n)$ time if most of the men end up with highly-ranked partners (since in this case it does not explore far down many of the preference lists). A recent result of Dean and Munshi [9] shows how to solve the *stable allocation* problem (the ordinal analog of the high-multiplicity scheduling problem above) in only $O(m \log n)$ time, which is also nearly linear in the size of the input, and also much faster than algorithms for the analogous cost-based problem.

4.1 Greedy Approaches

Perhaps the most natural heuristic for quickly obtaining a reasonable solution to a cost-based matching problem is a greedy one: repeatedly select the edge (i, j) of minimum cost from those remaining, increase $x(i, j)$ as much as possible, and then delete from consideration any nodes that are fully assigned. Another outlook on this process is that we repeatedly select the node i on the left-hand side whose best available match is the best overall, and then we match i with its best possible partner j by increasing $x(i, j)$ as much as possible. The running time of this approach is only $O(m \log n)$ – just the time needed to sort the edges in our problem by cost.

From a worst-case perspective, unfortunately, the greedy approach above can perform quite poorly. For example, in Figure 4.1, we see a simple instance on which the greedy algorithm is forced to deliver a solution whose cost is arbitrarily worse than that of an optimal solution. Moreover, even if our instance satisfies certain nice properties, the greedy approach can still perform poorly. For instance, even if edge costs satisfy the triangle inequality (even if these edges represent distances between nodes embedded on a one-dimensional number line), Reingold and Tarjan show the (tight) bound that a greedy assignment for the general matching problem can be as bad as $(4/3)n^{\log(3/2)} - 1 \approx (4/3)n^{0.585} - 1$ times the cost of an optimal assignment [29].

Even though the greedy approach can perform poorly in the worst case, its performance in practice tends to be somewhat more reasonable. To see this, we have performed a number of computational experiments to compare the quality of a greedy solution with that of an optimal solution on various randomly-generated instances. Since we will be using the same underlying computational testing framework throughout this chapter, we now briefly describe how our random inputs are generated, and then continue our discussion of the quality of the greedy heuristic.

We use three methods of constructing random bipartite instances on which to test the different approaches outlined in this chapter, as well as the performance of the greedy approach above. Each randomly assigns costs to edges, with slightly different distributions to allow us to measure which of our methods are effective under different common scenarios:

- **Geometric.** Here, we randomly generate n orange points and n purple points in a unit square in the 2D plane. The weight of an edge between an orange and a purple point corresponds to Euclidean distance. Geometric bipartite matching instances of this sort arise commonly in practice in applications such as tracking objects through multiple frames of a video. According to [3], the non-bipartite variant of this

random input distribution leads to some rather interesting asymptotic characteristics in optimal and greedy solutions; for example, as n increases, the ratio of quality between the optimal and a greedy solution tends to a fixed constant.

- **Random-1.** Here, we simply assign random real weights in the range $[0, M]$ to every edge in a bipartite graph with n nodes on the left and n nodes on the right, where we currently use $M = 100$ (although changing M should not effect the relative performance of any heuristic compared to the quality of an optimal solution). For such instances, Kurtzberg [24] showed the remarkable result that the expected cost of an optimal assignment is at most $3M$, independent of n, and Walkup [36] showed that the expected performance of the greedy heuristic is $\ln n + O(1)$.

- **Random-2.** In this case, we randomly assign real weights in $[0, M]$ to all nodes in a bipartite graph with n nodes on the left and n nodes on the right. We then set the weight of an edge (i, j) to be the sum of the weight of i plus the weight of j plus a zero-mean Gaussian-distributed random number with standard deviation σ (we set $M = 100$ and $\sigma = 10$ in our tests). The instance in this case is intended to resemble real-world instances such as those we anticipated in Chapter 3, in that distances start out somewhat "uniform" in the sense that there is a clear underlying ordering of both sides, but this is obscured slightly by having random noise added.

We also in each case give all jobs and machines a random size (*processing time*, in the case of jobs, and *capacity*, in the case of machines) in the range $[0, 100]$, giving us a high-multiplicity matching problem. We do not take any steps to ensure equality of the total multiplicities of the two sides. Since the resulting imbalance may affect how well certain approaches we try perform as heuristics, we need to keep track of this characteristic when reporting our results. If the processing times of all jobs total more than the aggregate capacity of all machines, we say that an instance is *job-heavy*, while *machine-heavy* instances will have an abundance of capacity. The amount by which an instance is job-heavy ($\sum_i p_i - \sum_j c_j$) we will call *job-overflow*. The fact that one side will be "heavier" than the other means that that side will not be completely matched. For simplicity, we assume that both sides of our instance are fully assigned by introducing a dummy job and dummy machine to be assigned an amount equal to the job-overflow (or negative job-overflow in the case of a machine-heavy instance). We have found that the magnitude and sign of the job-overflow can dramatically affect how different methods of generating preference lists compare to a simple greedy approach, as discussed throughout the rest of the chapter.

Since the heuristics introduced in the rest of this chapter are intended to serve as stronger approxi-

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: 19.13% StdDev: 5.32% Min: 5.18% Max: 29.73%	Average: 19% StdDev: 5.03% Min: 9.87% Max: 32.4%
Random-1	Average: 65.29% StdDev: 23.48% Min: 24.52% Max: 155.57%	Average: 73.93% StdDev: 28.23% Min: 33.85% Max: 134.69%
Random-2	Average: 18.57% StdDev: 3.24% Min: 12.35% Max: 28.67%	Average: 18.49% StdDev: 2.47% Min: 13.35% Max: 24.17%

Table 4.1: A comparison of greedy and optimal solutions. Percentage values are (greedy-optimal)/optimal * 100.

mation to an optimal assignment than the greedy algorithm, it will help to first have some idea how far from optimal a greedy solution tends to be in practice. In Figure 4.1, we see a comparison between greedy and optimal for our three different types of random instances, using CPLEX 10.1 to compute optimal solutions. As with all computational experiments throughout the chapter, we run each approach on 100 random instances of each random input type. The percentages shown in the table indicate improvement in cost relative to the cost of an optimal solution. Note that random-1, the most random instance, shows by far the largest gap relative to an optimal solution, while the other two methods of generating instances show on average a gap of around 20%. In these two cases, we have "less room to work with", which can explain why our methods will have somewhat more difficulty generating a high percentage improvement for them. The more randomness in the input, however, the harder it seems for greedy to compete with an optimal cost-based solution.

4.2 "Symmetric" Preference List Construction and Greedy Solutions

Our goal in this chapter is to develop heuristics based on stable matching approaches that improve on the greedy solution while still maintaining a relatively fast running time. To do this, we must convert the problem into a stable allocation problem with preference lists in such a way that we save on running time while maintaining a nearly-optimal solution. An obvious first approach to consider would be to create preference lists based solely on the cost to match the jobs to the machines. With complete preference lists, this

takes $O(n^2 \log n)$ time based on the necessary sorting of $O(n)$ lists of size n. Recall that we do lose information in this process, which is one way in which we expect our solution to gain in cost. A transformation such as this makes sense to try because the cost of the solution is directly affected by the cost of the chosen edges. Unfortunately, a trivial example can show how this could result in an arbitrarily bad solution in the worst case, and as we will see later, there turns out to be no benefit over a greedy solution.

In the example shown in Figure 4.1, the min-cost solution for the instance in (a) would be to use both edges that have cost 2. In the converted problem (c), though, the difference between 1 and 2 becomes indistinguishable from the difference between 2 and $+\infty$ — both are one location on a preference list apart. Because the edge of cost 1 is the cheapest, it is the highest on the preference lists for the nodes on both sides, and so will be included in the solution, forcing the worst edge also into play. Otherwise stated, the cost 1 edge becomes a blocking pair if not used. So with an adversary setting up the problem, it can lead to an arbitrarily bad solution, compared with the cost of an optimal solution. Our work tests different kinds of instances to see in which we can accomplish more reasonable solutions. Figure 4.1 (b) shows the greedy solution that is in this case equivalent to the stable matching. It turns out that this will happen every time we construct preference lists in this manner, leading us to the following lemmas.

Lemma 5. *When a cost-based matching problem with no ties is converted to a stable allocation problem with preference lists where lower cost means earlier in the preference list for both sides of the problem, there is a unique stable solution that is the same as the greedy solution.*

Proof. Because there are no ties, there will always be one lowest cost edge overall for the problem. This edge, the first chosen by the greedy algorithm, would consequently be the lowest cost edge between the job and machine it connects and will then be in the first position on each's preference list when converted. If either the job or machine is matched with any other without this edge being fully saturated, then both the job and machine would prefer to be matched together and would therefore be a blocking pair, preventing the solution from being stable. Once the edge in question is completely filled, a new lowest cost edge (among the remaining jobs and machines) exists and would be chosen by the greedy algorithm. This edge must then also be chosen for a stable matching as it is the highest remaining edge on both affected preference lists, and by induction this process will therefore lead to the one unique stable solution. □

If we expand our view to a more general outlook, we can see that this is actually the case for any instance in which preference lists are constructed *symmetrically* based on the costs ranked in order for each job and for each machine. When we say "symmetrically," we do not mean that the lists are the same or mirror

images but instead that they are created using the same cost or the same value to rank preferences for both sides. That is, when every edge can be ranked in some way and both sides base their lists on these rankings and only these rankings, we say that we have symmetrically constructed lists.

Lemma 6. *Any matching problem with preferences based on a "symmetric" measurement between jobs and machines will have a single unique solution equal to that of the greedy algorithm.*

Proof. This is proved in a very similar manner to Lemma 1. The greedy algorithm will always start with a job i on the left hand side with the best possible partner opportunity. Job i is then matched greedily with its best possible partner. At the same time we can, through a proposal based algorithm, create a stable solution. Because the lists are created symmetrically based on the same costs, what is best overall from one side will also be best overall for the other. Therefore when the job i proposes to machine j because j is most preferred over all machines for all jobs, the machine j will also have job i first in its preference list for the stable matching, creating a pair that will either be matched or blocking. i would propose to j and since i is the best remaining choice, it would be accepted and never rejected later in the algorithm since any future proposals will happen farther down j's list. This holds throughout the algorithm, allowing only one possible stable pair to be added at each step, the same pair chosen by the greedy algorithm. Both problems have the same sizes and capacities, so each would terminate at the same time with the same solution. □

Another prominent example fitting this mold is that of maximum-cardinality bipartite matching. Here, our goal is to compute a matching containing as many edges as possible in a bipartite graph. Perhaps the most natural greedy heuristic for this problem is to define D_i, for each left-hand node i, as the minimum degree among i's neighbors. We then repeatedly select a left-hand node with minimum D_i and match it to its minimum-degree neighbor; the intuition is that high-degree nodes are left for the end of the process, since they presumably have more flexibility in their choices for the matching. As we see in Lemma 6, the solution obtained by this greedy algorithm will be equivalent to the solution we would obtain by solving a stable marriage problem, where every node's preference list is sorted in decreasing order by degree.

As a consequence of this behavior, our research follows two main directions. First, because basing both left and right sides of the preference lists on the same cost ranking will always result in nothing more than the greedy solution, we explore methods in which one side's lists are based on costs but the other's adopt a more "long term" approach by attempting to avoid future penalties. In other words, we move into "asymmetrically" generated preference lists to make the problem more interesting and to open a path toward improving cost over greedy. We try different means of creating these new lists, compare and combine the

methods, and test the output to judge how they fare against a greedy solution. Secondly, we notice that since the one solution provided by the greedy algorithm and the stable allocation algorithm is unique and therefore in the context of stable matchings is both job and machine optimal, the optimal solution overall must, if it is different, be unstable. This makes sense if you look at the bad example in Figure 4.1 and notice that allowing instability gives the option of moving to the better solution and avoiding the trap that we illustrate. We examine different ways to measure the "instablity" of a solution and also discuss the tradeoff between stability and solution cost. Section 4.3 focuses on the asymetric methods of generating preference lists and the experiments that test them while Section 4.4 details our study of instability.

4.3 Asymmetric Preference List Construction

Because the symmetrically created preference lists end up giving us results identical to those obtained by the greedy algorithm, we move on to new classes of lists constructed with different methods of rankings for the left and right hand side lists. In our experiments, the left hand side will always use the cost based approach to ranking. For the right hand side we utilize different approaches based on how connected nodes are and how costs rise in the future in order to hopefully order choices in such a way that we avoid forcing ourselves into a bad choice.

4.3.1 "Pagerank"-Based Preference Lists

Our first attempt involves a system similar to *Pagerank* algorithms used by Google to index the Web. We seek to generate our order based on how connected a job is, a factor that leads us to believe that it would have sensible fallback choices in the event that it is not immediately chosen.

Let $I = \{1,2,\ldots,n\}$ and $J = \{1,2,\ldots,m\}$ denote the sets of jobs and machines respectively, let $c(i,j)$ denote the cost of assigning one unit of job $i \in I$ on the left with machine $j \in J$ on the right, and let $c^{-1}(i,j)$ denote $1/c(i,j)$. For notational shorthand, we use set notation to indicate summations; for example $c(I,j) = \sum_{i \in I} c(i,j)$ and $c^{-1}(I,J) = \sum_{i \in I} \sum_{j \in J} c^{-1}(i,j)$.

Let us now define a random walk on our bipartite matching graph, where the probability of transitioning from $i \in I$ to $j \in J$ is $p_{ij} = c^{-1}(i,j)/c^{-1}(i,J)$, and the probability of transitioning from $j \in J$ to $i \in I$ is $p_{ji} = c^{-1}(i,j)/c^{-1}(I,j)$. According to these transition probabilities, it is more likely to transition outward from a node along an edge of lower cost than it is along an edge of higher cost. The $n \times m$ matrix $P = [p_{ij}]$ encodes all of our transition probabilities. In terms of this random walk, we can now define a random walk

on just the jobs by starting at some $i \in I$, and then by taking consecutive pairs of steps from the walk above, each taking a step to a machine $j \in J$ and then back to another job $i' \in I$. The transition probability matrix for this job-oriented walk is the $n \times n$ matrix PP^T, known as a *Markov* matrix since its rows all sum to 1.

A well-known fact about random walks is that any random walk on a connected non-bipartite graph must converge eventually to a "stationary" probability distribution over all vertices in the graph, and this distribution is given by the principal eigenvector[3] of W^T, where W is the Markov matrix defining the transition probabilities of the walk. Vertices with high weight in the stationary distribution are visited frequently during a random walk, and hence must be both "centrally located" and "well connected" in our graph. These vertices are sometimes said to have high *eigenvector centrality* in our graph, and this premise is the central idea underlying the Google "Pagerank" algorithm for quantifying the importance of a web page based on how well-connected it is in the context of the larger graph [28].

Computation of the principal eigenvector of a random walk on a graph G can be done very efficiently using the so-called *power method*, where we start with some arbitrary vector x and repeatedly multiply it by W^T, normalizing after each step. Each multiplication takes time linear in the number of edges in the graph, and convergence to a reasonable approximation of the principal eigenvector usually only takes a small constant number of iterations (as evidence that this process can be done efficiently, Google frequently computes the principal eigenvector of a random walk of the entire Web graph, which has billions of vertices and edges). In our case, we are using PP^T as our transition matrix, so each step of the power method involves computing $(PP^T)^T x = PP^T x = P(P^T x)$, which requires two matrix-vector multiplications both taking time linear in the number of edges in our bipartite matching instance.

In our application of cost-based matching, we suggest that jobs $i \in I$ with high weight in the principal eigenvector of PP^T are those who have the most flexibility in terms of the availability of good edges usable for matching in their local neighborhoods. Jobs having low weight are the most highly constrained, having few options for good matching edges in their local neighborhoods. This motivates the heuristic of allowing vertices with low weight to choose their partners first, after which the high-weight vertices (presumably with more good choices to begin with) have their turn. We therefore define preference lists for the machines by the sorted ordering of all jobs according to their weight in the principal eigenvector of our random walk.

Recall how in Chapter 3 the adaptive case had a similar composition with a global priority ordering of the nodes on the left hand side based on identical preference lists. In this case, the global priority is

[3]The Markov matrix corresponding to a random walk in a connected, non-bipartite always has a single distinguished eigenvalue (known as the principal eigenvalue) equal to one; its corresponding eigenvector is called the principal eigenvector. All other eigenvalues have magnitude strictly less than one).

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: -4.06% StdDev: 5.51% Min: -22.41% Max: 5.12% Average Benefit: 0.5% StdDev Benefit: 1.19%	Average: 0.78% StdDev: 3.7% Min: -10.03% Max: 6.17% Average Benefit: 1.1% StdDev Benefit: 1.61%
Random-1	Average: -2.59% StdDev: 9.37% Min: -28.88% Max: 12.77% Average Benefit: 2.55% StdDev Benefit: 3.97%	Average: -0.65% StdDev: 8.04% Min: -15.91% Max: 14.64% Average Benefit: 3.09% StdDev Benefit: 4.43%
Random-2	Average: 0.82% StdDev: 0.83% Min: -1.11% Max: 2.7% Average Benefit: 0.87% StdDev Benefit: 0.74%	Average: 0.67% StdDev: 0.7% Min: -1.35% Max: 2.32% Average Benefit: 0.73% StdDev Benefit: 0.61%

Table 4.2: Percent improvements over greedy while using pagerank to construct preference lists.

based on how "well connected" the jobs are in the instance. After preliminary tests, we found that a pure "pagerank" based approach generally was not as successful as we would like in generating good solutions and at times was quite bad. This is logical because we are completely abandoning the costs in determining machine preference lists. We modify the algorithm to normalize the costs and the pageranks before then combining the two, giving a score of x times cost plus $(1-x)$ times pagerank, for some mixing ratio x (we use $x = 0.1$ and $x = 0.925$ in our tests, after noting that these seem to give the best results). At this point, we are no longer using a global priority ordering but are back to different lists with the costs again as a factor. After testing we determined that the split should vary based on the way the instance was created with different styles of input varying widely in how pagerank performed.

Pagerank in general does not seem to perform very well in creating lists. As there are a few similar charts in the following pages, here we describe the format. We separate the input into job-heavy and machine-heavy instances because sometimes this difference can produce a noticeable effect. For each method of randomly generating an instance, labeled on the left, we include the average, min, max, and standard deviation for all types of inputs. The percentages are percent improvement over greedy. In this case we take percentage

improvement over 100 instances of each type of input and compute our statistics on these results. The other two categories, those dealing with benefits, are what we get when we use the minimum cost (so maximum improvement) of the method the chart is based on and the greedy method, so we basically are only looking at the positive numbers or 0 since greedy is the baseline. Interesting to note about the pagerank results is that random-2 seems to show a slight improvement with only a small change due to removing negative improvement values (average to average benefit in the tables) while random-1 shows bad results in general but individual runs are far enough on either side of 0 so we do see a larger improvement when we remove the negative values.

4.3.2 Regret-Based Preference Lists

Another method we consider for defining preference lists on the right-hand side is based on a notion of "regret"[4]. Suppose a machine $j \in J$ must decide between two jobs i and i', with $c(i,j) = c(i',j)$. Both may look equally attractive in terms of their immediate matching costs, but it could be the case that i's next alternative is not much worse than $c(i,j)$ in cost, while the next alternative of i' is significantly worse. In this case, rejecting i does not worry us much, since i has other good alternatives on which to fall back. Rejection of i' might be a decision we would regret, however, since i' is now condemned to make do with a set of much costlier alternatives.

Consider any pair (i,j) with $i \in I$ and $j \in J$. Let $\gamma_{ij}(p)$ denote the pth entry in i's preference list after j. For example, $\gamma_{ij}(0) = j$, and $\gamma_{ij}(1)$ is the element i would propose to next if it were rejected by j. We define the *additive regret* of j rejecting i as

$$R_a(i,j) = c(i,\gamma_{ij}(1)) - c(i,j),$$

and the *multiplicative regret* of this rejection as

$$R_m(i,j) = c(i,\gamma_{ij}(1))/c(i,j).$$

Both expressions reflect some measure of how much worse i's immediate prospects become as a consequence of rejection from j. It therefore seems reasonable to consider assigning the preference list of every machine $j \in J$ in decreasing order of either $R_a(i,j)$ or $R_m(i,j)$.

[4]Some of the original ideas from this section are based on discussions with M. Goemans and others previously working in the load balancing group at Akamai Technologies.

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: -105.54% StdDev: 55.07% Min: -241.7% Max: -22.56% Average Benefit: 0% StdDev Benefit: 0%	Average: -9.96% StdDev: 7.05% Min: -31.13% Max: 1.54% Average Benefit: 0.05% StdDev Benefit: 0.24%
Random-1	Average: -849.95% StdDev: 15.38% Min: -1690.85% Max: -317.22% Average Benefit: 0% StdDev Benefit: 0%	Average: 5.34% StdDev: 342.97% Min: -48.64% Max: 30.45% Average Benefit: 9.25% StdDev Benefit: 8.76%
Random-2	Average: -44.05% StdDev: 10.63% Min: -66.15% Max: -21.27% Average Benefit: 0% StdDev Benefit: 0.0%	Average: -6.35% StdDev: 3.83% Min: -18.76% Max: 0.69% Average Benefit: 0.02% StdDev Benefit: 0.1%

Table 4.3: Percent improvements over greedy while using additive regret to construct preference lists.

There is the special case to consider regarding the final position on a regret based preference list. Obviously this cannot work directly with the expressions above because the next entry in this case does not exist. The initial solution considered for this situation was to move this last entry to the front of the preference list. The reasoning behind this move was that this is the last possible choice before the job becomes completely unassigned, inspiring some kind of dramatic effort at preventing the unassignment from happening. Unfortunately, initial testing using this method shows it to result in extremely poor solutions for job-heavy instances. Some jobs remain unassigned, as would be expected in this case, but many more receive their worst possible choice. A closer examination of the process by which the jobs are matched shows why this occurs. First, it is important to realize that when the machines cannot handle work of all the jobs, job-overflow processing time will not be assigned except to dummy. We want to make sure that the jobs that don't get assigned are the ones that would end up in bad matches if they were assigned. Unfortunately, if we are not careful the opposite occurs. Once a job, through proposals and rejections, arrives as its final choice in its preference list (a point in which it already knows it is a relatively bad solution), this job gets

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: -13.72% StdDev: 8.64% Min: -39.61% Max: -2.42% Average Benefit: 0% StdDev Benefit: 0%	Average: -1.12% StdDev: 3.84% Min: -9.99% Max: 5.03% Average Benefit: 0.96% StdDev Benefit: 1.49%
Random-1	Average: -28.17% StdDev: 22.35% Min: -80.06% Max: 9.85% Average Benefit: 0.23% StdDev Benefit: 1.52%	Average: 5.97% StdDev: 9.81% Min: -22.28% Max: 26.7% Average Benefit: 7.67% StdDev Benefit: 7.14%
Random-2	Average: -26.56axs% StdDev: 6.77% Min: -40.66% Max: -13.49% Average Benefit: 0% StdDev Benefit: 0%	Average: -3.68% StdDev: 2.48% Min: -8.72% Max: 1.25% Average Benefit: 0.03% StdDev Benefit: 0.17%

Table 4.4: Percent improvement over greedy while using multiplicative regret to construct preference lists.

priority over any other job that could (and likely does) have a better solution. Even worse, this triggers a chain reaction moving jobs down their preference lists until each hits the worst choice and is then accepted. The algorithm cannot terminate until job-overflow weight is assigned to dummy, but this pattern continues to accept bad choices before they can be completely rejected and unassigned by normal machines. A better idea then would be to say that once a job is at the end of its preference list it is also at the end of the machine's, so if someone is already a bad choice and is not assigned already, then it will be the first to be unassigned. To avoid ties and ensure that the worst solution is thrown out first, we can modify this to sort any jobs with the machine in question as a last choice by order of increasing cost to that machine. Another possible solution that could make sense in future work is to have an actual cost for each job to remain unassigned. Maybe some jobs are expensive to schedule but are very important so it is actually worse for the overall solution to have them unassigned than another job that costs little but also has little penalty for not being completed.

One important thing to notice about simple regret based matching is that it just cannot seem to handle job-heavy input well at all. In only one case do we get any positive result at all, and still that case averages

about 28% worse than greedy. The reasoning we see for this result is that the comparison with the next job in the list is a bad way to move a bad solution all the way down a list, and in job heavy lists this is necessary because some jobs will be assigned to dummy, the last choice of everyone's list.

Even on machine-heavy instances, only in the very rare cases is simple regret based construction useful. An explanation could be that it is hard to improve on greedy with close points or similar lists. More random lists, though, where greedy tested poorly against optimal, give this approach room to improve the solution but again only in machine heavy instances.

4.3.3 A More Refined "Longer Look" Approach

Part of the reason preference lists based on purely additive or multiplicative regret (as defined above) seem not to work well, is that they disregard "immediate" information about the cost of proposals. For example, if $j \in J$ is considering two alternatives i and i' with $c(i,j) >> c(i',j)$, then j does not consider this fact while deciding which to reject. To remedy this, it seems that perhaps a measure combining regret with the immediate assignment cost is more appropriate. Furthermore, it seems that our notion of regret for (i,j) may be somewhat "short-sighted" in only comparing with the cost of the next alternative for i after rejection, and not the alternatives further down the road in i's preference list. We therefore use least squares regression to fit a degree-2 function to entire set of remaining costs on i's preference list after j. That is, we take the set of points $(x, c(i, \gamma_{ij}(x)))$ and fit a degree-2 polynomial $a_2x^2 + a_1x + a_0$ so as to minimize the error

$$e(a_2,a_1,a_0) = \sum_{x=0,1,2,\ldots} \alpha^x |a_2x^2 + a_1x + a_0 - c(i,\gamma_{ij}(x))|^2,$$

where we chose $\alpha < 1$ to place exponentially-decreasing emphasis on errors arising far down i's preference list (so the fit should be best for the entries in the preference list right after j). We use $\alpha = 0.1$ in our testing; after experimenting with several values of α, this value seems to give good performance. By setting $\nabla e = 0$, we can compute the optimal coefficients a_2, a_1, and a_0 by solving the linear system

$$\begin{bmatrix} \sum_x \lambda^x x^4 & \sum_x \lambda^x x^3 & \sum_x \lambda^x x^2 \\ \sum_x \lambda^x x^3 & \sum_x \lambda^x x^2 & \sum_x \lambda^x x^1 \\ \sum_x \lambda^x x^2 & \sum_x \lambda^x x^1 & \sum_x \lambda^x \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_x \lambda^x x^2 c(i,\gamma_{ij}(x)) \\ \sum_x \lambda^x x c(i,\gamma_{ij}(x)) \\ \sum_x \lambda^x c(i,\gamma_{ij}(x)) \end{bmatrix}.$$

To ensure the system has a unique solution, we replicate the last entry of each preference list (and its cost) twice, thereby giving at least 3 distinct points to use for our fit; this is necessary since we are fitting 3

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: 0.57% StdDev: 1.83% Min: -2.24% Max: 5.93% Average Benefit: 0.99% StdDev Benefit: 1.41%	Average: 1.23% StdDev: 1.92% Min: -3.09% Max: 6.44% Average Benefit: 1.51% StdDev Benefit: 1.49%
Random-1	Average: 8.67% StdDev: 8.26% Min: -8.47% Max: 24.06% Average Benefit: 9.31% StdDev Benefit: 7.23%	Average: 9.38% StdDev: 8.44% Min: -6.49% Max: 29.49% Average Benefit: 9.85% StdDev Benefit: 7.75%
Random-2	Average: 2.96% StdDev: 1.68% Min: -0.56% Max: 7.13% Average Benefit: 2.98% StdDev Benefit: 1.65%	Average: 2.79% StdDev: 1.05% Min: 0.82% Max: 5.64% Average Benefit: 2.79% StdDev Benefit: 1.05%

Table 4.5: Percent improvement over greedy while using a more long term look at regret to construct preference lists.

parameters.

Despite its complicated appearance, this actually only takes $O(1)$ time to solve for each (i, j), for a total of linear time over the entire stable matching process, so it does not impact our total asymptotic running time. Due to the geometric nature of the summations in the matrices above, each entry can be updated in only $O(1)$ time when updating the system after computing the previous entry, so that we are considering $(i, \gamma_{ij}(1))$ instead of (i, j).

After solving the system above for a particular (i, j), we set $a_{ij}(x) = -a_2x^2 - a_1x + a_0$. This reflects the positive "immediate" cost a_0 of matching (i, j) (which is not exactly $c(i, j)$, but a more forward-looking analog of this cost based on our curve fit). The linear and quadratic terms are negated, since both of these indicate forms of "regret": the linear term is a more sophisticated analog of the additive regret we formulated in the previous section, and the quadratic term indicates a higher-order form of regret in terms of the projected rate at which regret will be increasing as i keeps moving down its preference list. To form preference lists for each machine j, we now simply sort all jobs $i \in I$ in increasing order by $a_{ij}(x)$, for some specified value x.

	Job-heavy inputs	Machine-heavy inputs
Geometric	Average: 1.49% StdDev: 1.51% Min: 0% Max: 5.03%	Average: 2.1% StdDev: 1.96% Min: 0% Max: 6.44%
Random-1	Average: 9.97% StdDev: 6.77% Min: 0% Max: 24.06%	Average: 13.85% StdDev: 7.51% Min: 0% Max: 30.45%
Random-2	Average: 2.98% StdDev: 1.64% Min: 0% Max: 7.13%	Average: 2.79% StdDev: 1.05% Min: 0.82% Max: 5.64%

Table 4.6: Percent improvement over greedy while using a combination of all methods to construct preference lists.

Our choice of x indicates how "forward-looking" j should be. By setting $x = 0$, we select only the term a_0 that corresponds to immediate cost (with some forward-looking bias based on the curve fit). As we increase x, we place less emphasis on immediate cost and more and more emphasis on the higher-order measures of regret. More experimentation with x determined that x of about 2.5 is a good choice for the multiple types of random instances.

The longer look approach seems to be by far the best both because it can provide good improvements for different kinds of random (but still best on fully random) and also because it does not have the same massive penalties when using job heavy instances. Also, the improvement seems to come from actually better solutions instead of relying on greedy to set a lower bound for which other methods sometimes take advantage of to add to their solutions.

4.3.4 Combining Different Methods

Finally, we take all of these methods, the longer look approach generally being the best overall but others showing promise in certain instances, and combine them to determine how much of an increase we can show over greedy. We took 100 of each random type of instance, solved them using preference lists created by all previously discussed approaches, and took the maximum gain over greedy (including 0 for greedy itself) and find that we do see a significant improvement overall. This improvement does seem to vary quite a bit, however, based on instance type. For example, the fully random distances (random-1) show somewhat

larger improvements, but this is to be expected with the differences seen when comparing greedy and optimal for that case. Interestingly, we did see noticeable improvements over the longer look method by itself for both geometric and random-1 instances, even when they turn out to be job-heavy. This can happen when individual executions turn out much better but are hidden by a large number of bad tests.

4.4 Bicriteria Approaches

The other main direction we consider in this chapter is the relationship between stability and cost. The optimal solution to a cost based matching problem is generally not the same as the stable solution of an ordinal problem to which it is converted. Multiple factors account for this, for example the loss of information (although normally much less extreme) as seen in the bad example in Figure 4.1. Accordingly, a stable allocation algorithm loses the ability to measure "how much worse" is one position on a preference list than earlier positions. However, even though the optimal solution to a cost-based matching problem may not be stable, we would still expect it to be "somewhat" stable, in the sense that many of the edges not in the optimal solution should not be blocking pairs. This leads to the interesting question of how one can quantify the amount of "instability" present in a particular solution.

Perhaps the most obvious measurement of instability for a particular solution is the number of blocking pairs it admits (or perhaps a more reasonable measurement is the fraction of all edges that are blocking pairs). However, this measurement fails to capture the "degree" to which a blocking pair might be blocking. For instance, an edge (i, j) might be a blocking pair if its cost is only 1% lower than edges used by i or j in an optimal solution. In such a case, one could argue that this only constitutes a minor infraction against stability, and that i and j may still be sufficiently content to stay put with their current assignments (i.e., the "cost of divorce" might be high enough that it is not worth considering a switch to alternative solutions that are only marginally better).

In order to quantify the degree to which an edge (i, j) not in an assignment is blocking, we introduce two new measures, which we call *additive* and *multiplicative* instability. Let a_i denote the maximum cost of all edges $(i, \cdot)$ used by i in the assignment in question, and let b_j denote the maximum cost of all edges $(\cdot, j)$ used by j. We then define the additive instability of (i, j) as

$$\varepsilon(i, j) = \max(0, \min(a_i, b_j) - c_{ij}),$$

	Alpha	Epsilon
Geom	Max: 78.7 Avg: 1.57	Max: 0.34 Avg: 0.04
	Max unstable: 3.72%	Avg unstable: 1.45%
Rand-1	Max: 3473.86 Avg: 5.63	Max: 4.94 Avg: 0.81
	Max unstable: 0.99%	Avg unstable: 0.53%
Rand-2	Max: 727.68 Avg: 28.12	Max: 74.73 Avg: 16.08
	Max unstable: 11.38%	Avg unstable: 8.23%

Table 4.7: Statistics measuring instability based on number of unstable pairs and how far the most extreme blocking pairs are in each instance.

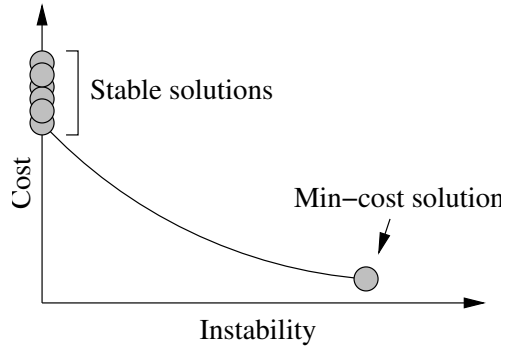

Figure 4.2: Trade-off curve between cost and stability of solutions.

where the maximum with zero ensures that $\varepsilon(i,j) \geq 0$ always. We similarly define the multiplicative instability of (i,j) as

$$\alpha(i,j) = \max(1, \min(a_i, b_j)/c_{ij}),$$

where again the maximum with one ensures that $\alpha(i,j) \geq 1$.

To understand what "reasonable" values of these measurements are in practice, we measured the maximum and average of α and ε over all blocking pairs over 100 examples of optimal solutions to randomly-generated instances (using the same methods for random generation as before). Results are shown in Figure 4.7.

A potentially interesting direction for future research is a study of "bicriteria" approximation algorithms that consider a sliding scale between stability and cost. In general, the lower we want to make cost,

the higher we must allow for instability (measured according to one of our metrics above), giving rise to a trade-off curve of the form shown in Figure 4.2. This curve is interesting since we can know how to easily compute solutions at its endpoints – either a cost-optimal solution that has no regard for stability, or a stable solution that has no regard for cost[5]. Moreover, stable solutions can be computed much faster than cost-optimal solutions.

This motivates the algorithmic question of whether one might be able to solve for a solution not just at endpoints. For example, we could ask for a minimum-cost solution subject to a constraint limiting the maximum amount of instability present in any edge, or we could ask for a solution minimizing the maximum instability of an edge, subject to a constraint limiting the cost of the solution by a specific threshold. While we currently do not know if it is possible to solve either of these questions efficiently, this might be an interesting direction for future study, since this might allow for a continuous trade-off between cost and stability that, from an algorithmic perspective, allows us to achieve as much speedup as possible through the use of ordinal techniques.

[5]Note that we can compute a minimum-cost stable solution in polynomial time [23], but that the best-known methods for doing so are currently slower than the best-known methods for solving a minimum-cost assignment problem.

Chapter 5

Conclusions

Over the course of this dissertation, we have developed results in three main areas of study:

- We resolved the open question of Gusfield and Irving on the possibility of efficient batch testing of stable marriage solutions. We presented an algorithm that accomplishes this job by utilizing a characterization of matching based on rotations and a dynamic connectivity data structure. It will check the validity of k queries in $O((m+kn)\log^2 n)$ time, effectively $O(n\log^2 n)$ amortized time per query after $O(m\log^2 n)$ preprocessing. The results of this research have been published in Algorithmica [8].

- We have also created an algorithm to solve instances of stable marriage with disorder restricted to the men's preference lists. This work takes advantage of properties of "adaptive" algorithms to allow the running time to scale with both the size of input and the disorder present within it. It produces the stable solution in $O(n+k)$ time where n is the size of the input and k is the total amount of change from uniform preference lists on the men's side. This work has been presented at the Association for Computing Machinery (ACM) Southeast Conference [7].

- Finally, we have experimented with ordinal matching using as a heuristic for cost-based matching problems. Using three different methods of creating random cost-based instances, we converted them to problems with preference lists created by a variety of "asymmetric" techniques. We tested how well these ideas performed and demonstrated that stable allocation performs better than a greedy algorithm on average using our techniques in these situations. Additionally, we explore just how unstable an optimal solution to these problems can be, and we present ideas on where this viewpoint can lead in the future.

5.1 Future Work

The research in this dissertation can lead to other problems to be studied both as direct improvements to the results in this work, and also as more open-ended exploration based on the ideas we present. For our results in batch testing, the main open question yet remaining is whether these results can be achieved in a worst-case, rather than amortized setting. In the domain of adaptive stable matching algorithms, the largest remaining open question is whether it is possible to construct an efficient algorithm for non-W-consistent inputs. Other interesting ideas include an expansion of the "draft" model of this problem, and a "data structure" viewpoint where we try to re-compute a stable solution quickly after a single change is made to one of the preference lists.

Regarding our work on ordinal matching as a heuristic for cost-based matching, we bring up the idea of a tradeoff between stability and solution cost: could one design algorithms that exploit this tradeoff, producing bicriteria approximation solutions that are reasonably good both in terms of stability and cost? To do so, one may need to address another longstanding open question of Gusfield and Irving [18] as to how one can characterize the set of all stable solutions in an instance with ties in preference lists. One could also explore the creation of new heuristics, or the fine-tuning of our existing heuristics further, for example by tuning the parameters underlying these approaches in an adaptive fashion. For job-heavy instances, perhaps one can apply a preprocessing heuristic of some sort to eliminate all job-overflow, since this seems to have a dramatic effect on the quality of many of our heuristics.

Bibliography

[1] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber. Efficient minimum cost matching using quadrangle inequality. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:583–592, 1992.

[2] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[3] D. Avis, B. Davis, and J. Steele. Probabilistic analysis of a greedy heuristic for euclidean matching. *Probability in the Engineering and Informational Sciences*, 2:143–156, 1988.

[4] M. Baiou and M. Balinski. Erratum: The stable allocation (or ordinal transportation) problem. *Mathematics of Operations Research*, 27(4):662–680, 2002.

[5] W. H. Burge. Sorting, trees and measures of order. *Inf. Contr.*, 1:181 197, 1958.

[6] R. Burkard, B. Klinz, and R. Rudolf. Perspectives of monge properties in optimization. *Discrete Appl. Math.*, 70(2):95–161, 1996.

[7] J. Dabney and B.C. Dean. Adaptive stable marriage algorithms. In *Proceedings of the 48th Annual ACM Southeast Regional Conference (ACMSE)*, 2010.

[8] J. Dabney and B.C. Dean. An efficient algorithm for batch stability testing. *Algorithmica*, 58(1):52–58, 2010.

[9] B. Dean and S. Munshi. Faster algorithms for stable allocation problems. In *Proceedings of the MATCH-UP (Matching Under Preferences) workshop at ICALP*, pages 133–144, 2008.

[10] D. Drake and S. Hougardy. Linear time local improvements for weighted matchings in graphs. In *IN INTERNATIONAL WORKSHOP ON EXPERIMENTAL AND ECIENT ALGORITHMS (WEA), LNCS 2647*, pages 107–119. Springer-Verlag, 2003.

[11] D. Eppstein, Z. Galil, G. Italiano, and A. Nissenzweig. Sparsification – a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.

[12] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM*, 24 (4):441–476, 1992.

[13] G.N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.

[14] H. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443. SIAM, 1990.

[15] H. Gabow and R. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, 1991.

[16] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[17] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[18] D. Gusfield and R. Ivring. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.

[19] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.

[20] N. Immorlica and M. Mahdian. Marriage, honesty, and stability. In *Proceedings of 16th ACM Symposium on Discrete Algorithms*, pages 53–62, 2005.

[21] R. Irving. An efficient algorithm for the "stable room-mates" problem. *Journal of Algorithms*, 6:577–595, 1985.

[22] R. Irving and P. Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15:655–667, 1986.

[23] R.W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM*, 34(3):532–543, 1987.

[24] J. Kurtzberg. On approximation methods for the assignment problem. *Journal of the ACM*, 9(4):419–439, October 1962.

[25] D.G. McVitie and L.B. Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–492, 1971.

[26] K. Mehlhorn, W. Brauer, G. Rozenberg, and A. Salomaa. *Data structures and algorithms. Volume 1 : Sorting and searching*. Springer, 1984.

[27] C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.

[28] L. Page, S. Brin, R. Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Computer Science Department, Stanford University, 1998.

[29] E. Reingold and R. Tarjan. On a greedy heuristic for complete matching. *SIAM J. Comput*, 10:676–681, 1981.

[30] A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92:991–1016, 1984.

[31] A.E. Roth. The national residency matching program as a labor market. *Journal of the American Medical Association*, 275(13):1054–1056, 1996.

[32] A.E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American Economic Review*, 89:748–780, 1999.

[33] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.

[34] K. Varadarajan and P. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *In SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms*, pages 805–814, 1999.

[35] D. Vinkemeier and S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Trans. Algorithms*, 1(1):107–122, 2005.

[36] D. Walkup. On the expected value of a random assignment problem. *SIAM Journal on Computing*, 8(3):440–442, August 1979.