Matching Markets: Design and Analysis

David John Abraham

CMU-CS-09-167

September, 2009

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
R. Ravi, Chair
Alan Frieze
David Manlove (University of Glasgow)
Luis von Ahn

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers IIS-0121678, CCF-0514922, CCR-0313148 and IIS-0427858. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Mechanism Design, Algorithms, Matching, Stable Marriage, Kidney Exchange

Abstract

A market consists of buyers and sellers of some commodity, say a DVD movie. In this thesis, we assume the role of market operator. Our goal is to ensure that the market has certain desirable properties, including truthfulness, fairness and stability. We explore how to achieve these properties by designing rules for how the participants (buyers and sellers) can interact. We also explore how to efficiently compute the outcome of large numbers of participants interacting at once.

The main type of market we study is called a matching market. We study several particular matching markets, including keyword auction, kidney exchange and stable roommate markets. In each of these cases, the aim is to match the participants to each other, somehow taking into account their preferences for one another. Our results focus on properties of the matching process and the design of efficient algorithms for finding various types of matchings. In particular, we present new polynomial time algorithms for finding matchings that have one of the following properties: popularity, rank-maximality and fairness. We also give an efficient algorithm for clearing large swap markets such as kidney exchanges. Finally, we present a new decomposition technique for designing keyword auctions.

Acknowledgements

I would like to thank my advisor R. Ravi. Ravi has a well-deserved reputation as an outstanding advisor. He is patient, energetic, insightful and always able to give good advice. In short, I've learned a lot and had a great time working with him. I would like to give Ravi special thanks for guiding me through the most difficult period of my PhD, and also giving me the freedom to follow my interests.

Thanks also to Luis von Ahn for giving me the opportunity to join his startup, reCAPTCHA. It has been an amazing ride so far, something that few people get to experience. I hope reCAPTCHA's success continues and that one day, we achieve its goal of digitizing all the books, newspapers and important historical documents in the world.

I would also like to thank David Manlove, my MSc advisor and main collaborator. David trained me in the field of matching markets and research in general. Without David's influence, I would not have come across the two areas of research I enjoyed the most: popular matching and kidney exchange markets. Also, I would like to give him special thanks for his understanding and help in getting me into CMU, to be with my partner, Liz.

Thanks also to my committee in general. I appreciate your guidance, and the work involved in providing it.

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

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

A market consists of buyers and sellers of some commodity, say a DVD movie. In this thesis, we assume the role of market operator. Our goal is to ensure that the market has certain desirable properties, which may include fairness, revenue maximization, and so on. We achieve these properties by designing rules for how buyers and sellers can perform transactions. Our main focus will be on matching markets.

An example of a matching market is Netflix [69], the mail-based DVD rental business. Each day, Netflix allocates DVDs from its warehouses to customers entitled to rent a DVD on this day. Typically, there are not enough DVDs to give all customers their first choice, so Netflix must decide on an allocation that somehow takes into account the preferences of customers over DVDs. This is an example of a bipartite market, since there are two disjoint sets of participants, namely the customers and the DVDs/Netflix, and each set of participants only interacts with the other set. Other bipartite matching markets include the allocation of doctors to internships [71], children to public schools [1], and midshipman to naval jobs [79].

More generally, a matching market can be non-bipartite, in which case participants can interact with any other participant. One type of non-bipartite matching market is a barter-exchange market. An example of a such a market is Swaptree [91], the mail-based DVD swapping business. (Swaptree also allows swapping of CDs, games, books, and so on.) Customers give Swaptree a list of DVDs they want to own, and a list of DVDs they want to trade. Whenever Swaptree finds an exchange (possibly involving three or more customers), it organizes for the customers to swap their DVDs by mail. In this market, the buyers and sellers coincide, and no money is exchanged.

There are many other examples of barter-exchange markets. For example, there is a currently

a move in the United States to establish a national kidney-exchange market [94]. Many kidney-disease patients have a willing but blood-type incompatible donor amongst their family. In the past, these potential donors have been sent home, leaving the patients to wait for cadaver kidneys, which are in very limited supply. However, doctors now perform exchanges in which patients swap their incompatible donors in order to obtain a compatible donor.

Matching markets have long held a central place in the game theory, mechanism design and computer science literature [29, 37, 79]. In this thesis, as well as studying the design and analysis of traditional matching markets, we explore a range of new problems that have arisen from more modern markets. These problems include special constraints on which buyers and sellers can interact. Perhaps the most significant new problem to arise out of modern matching markets is dealing with their size. For example, Netflix [69] has over 10 million subscribers and 55 million DVDs, while the upcoming UNOS kidney exchange [94] is expected to deal with hundreds of millions of possible cycle swaps. This means that the problem of constructing efficient algorithms for matching buyers and sellers is a central theme in this thesis.

1.1 Organization

In this thesis, we consider two broad classes of matching markets.

The first class consists of bipartite markets in which the aim is to match a set of agents to a set of items. Agents have preferences over which items they are paired with, but items are indifferent between all agents. Chapters 2 and 3 deal with these types of markets.

The second class consists of markets in which the aim is to match a set of agents to each other. In this case, all participants are agents, each of whom has preferences over the others. In general, these markets are non-bipartite, though when proving some hardness results, we also consider some bipartite restrictions. Chapters 4, 5 and 6 deal with these types of markets.

Each of these content chapters is based on an existing conference or journal publication, or pending submission. As such, each chapter is self-contained, and there is some repitition between chapters of key lemmas/theorems, such as the Gallai-Edmonds graph decomposition result.

In the rest of this chapter, we briefly introduce the work in each of the content chapters.

The introduction is organized according to the two classes of matching markets discussed above.

1.2 Matching Agents with Items

A bipartite matching market with one-sided preferences consists of a set of *agents* and a set of *items*. Each agent has a *preference list*, ranking the subset of items it finds *acceptable* in order of preference (first choice, second choice, and so on). The *outcome* we seek is a *matching* of agents to acceptable items in which no agent is matched with more than one item, and no item is matched with more than one agent.

Preferences over items extend naturally to preferences over outcomes / matchings: an agent prefers all matchings in which it is allocated an acceptable item. Given two such matchings, an agent prefers the one in which it obtains the better-ranked item. If an agent does not prefer one matching over another, it is *indifferent* between the two.

A *matching algorithm* selects a matching based on the preferences of agents over outcomes. One area of research involves trying to find computationally-efficient algorithms that produce optimal matchings according to some social welfare objective.

1.2.1 Popular Matching

There are many possible social welfare objectives. For example, we might assign weights to possible allocations, with better-ranked allocations having more weight. The aim then is to find a *maximum-weight* matching, where the weight of a matching is the weight sum of all the allocations it makes. This setting is itself quite general, as many weight functions are possible.

For any social welfare objective though, we need a way to compare two matchings (for example, by weight). This comparison can be made democratic by allowing the agents to vote between the matchings. A matching M' is *more popular than* another M if the number of agents preferring M' to M exceeds the number of agents preferring M to M'. We say that M is *popular* if there is no matching M' more popular than M. This definition of optimality extends the well-known Condorcet voting rule for presidential elections - a candidate wins only if she loses no pairwise election with any other candidate.

The notion of popular matching was originally introduced by Gärdenfors [31] in the context

of the full stable marriage problem [29, 37]. A matching is stable if there is no pair of agents who would prefer to be matched with each other than their existing partners in the matching. Since there can be an exponential number of stable matchings, Gärdenfors proposed popularity as an additional desirable property to help distinguish between all the possible stable matchings. Gardenfors showed that when preference lists are strictly ordered, every stable matching is popular. He also showed that when preference lists contain ties, there may be no popular matching.

It turns out that a popular matching may not exist, even in the restricted agent-item setting. Given an agent-item matching market, the *popular matching problem* is to find a popular matching, or prove that no such matching exists. As in the stable marriage setting, the number of matchings may be exponential in the number of agents, and so it is not immediately clear if there exists an efficient algorithm for this problem based on pairwise comparisons of matchings.

In Chapter 2, we present a novel characterization of popularity that leads to the following surprising results: the popular matching problem can be solved in *linear* time when preference lists contain no ties. More generally, when preference lists are unrestricted, we give an $O(m\sqrt{n})$ time algorithm, where n is the number of agents and items, and m is the total length of all the agent preference lists.

1.2.2 Keyword Auctions

Web search engines use keyword auctions to sell the advertising space alongside their algorithmic search results. When a user searches for a keyword (e.g. "digital camera"), merchants wanting to target this keyword (e.g. Canon, Nikon and Amazon.com) bid to have their advertisements displayed on the search results page. Based on these bids, the search engine performs a matching of merchants (agents) to advertisement slots (items). Since users are more likely to click on slots at the top of the page, merchants value these slots more highly than slots lower down the page.

Small changes in the auction mechanism can lead to big changes in both the bidding behavior of merchants, and the revenue collected by the search engine. As such, there has been significant recent interest in the design and analysis of keyword auction mechanisms (see [54] for a recent summary).

In Chapter 3, we introduce the class of *layerable* mechanisms for keyword auctions. In various settings, several existing mechanisms are layerable, including the Generalized Second Price mechanism [23], which is used by the major search engines, the laddered auction [10], and the

VCG mechanism [19, 36, 96]. Layerable mechanisms can be decomposed into a collection of layers, where each layer consists of an auction in which merchants are indifferent between the slots. This decomposition leads to a simple technique for designing and analyzing keyword auction mechanisms: instead of working in the general setting, we can work in the restricted layer setting.

The main focus of this work is to characterize the class of layerable mechanisms by giving necessary and sufficient conditions for when the decomposition technique is possible. We demonstrate our new technique by simplifying the correctness proofs of some existing mechanisms, and showing how they can be extended to more general settings. We also introduce a new mechanism that allows a search engine to truthfully bid in its own keyword auction. This is becoming increasingly common as Google, Microsoft and Yahoo compete to sell their other services, such as operating systems, video-game consoles, photo-sharing websites and so on. With existing mechanisms, this leads to a conflict of interest as the search engine may mis-represent its bids in order to make other merchants pay the search engine more money. Building on the laddered auction, our new mechanism ensures that the merchants can not be exploited in this way, whilst also guaranteeing that the search engine gains at least as much utility as in the standard laddered auction.

1.3 Matching Agents with other Agents

The setting in the previous section involved matching agents and items, where agents had preferences over items, whilst items had no preferences over agents. In this section, we consider a more general model in which the aim is to match agents with other agents. Both the stable marriage problem [29, 37] and stable roommate problem [29, 37, 45] are modelled in this setting. The stable marriage problem is a bipartite market in which we match, for example men with women. The stable roommate problem is a non-bipartite market in which we match, for example, college students to be dormitory roommates. As well as considering matching markets, we will also consider the following generalization.

A *barter-exchange market* consists of a set of agents, each of whom owns an item. Each agent has a preference list over the set of items it finds acceptable. These preferences extend naturally to the set of agents who own these items. An *exchange* is a disjoint collection of cycles of agents, where each agent wants the item owned by the next agent in its cycle.

Our main focus has been on kidney-exchange markets, where patients with kidney disease trade their incompatible donors. In the United States, more than 70,000 patients are on the waiting list for the transplant of a cadaver kidney. Because demand for cadaver kidneys far

outstrips supply, thousands of patients die each year waiting for their transplant opportunity. Since it is illegal to buy and sell human organs, barter-exchange markets, which are almost universally regarded as ethical and legal, present the only real options for patients unlikely to survive the wait for a cadaver kidney.

1.3.1 Finding a Maximum Exchange

The problem of finding an exchange involving the maximum number of agents, also known as the directed cycle cover problem, is polynomial-time solvable. However, if a fixed limit (more than 2) is placed on the maximum allowable cycle lenght, we show the problem becomes NP-hard [4]. Contraints on the maximum cycle length arise naturally in kidney-exchange markets. In a k-way kidney exchange, all $2k$ operations have to be done *simultaneously* (and typically in the same transplant center, which only has a fixed number of operating rooms, doctors, nurses and so on). The reason that simultaneous operations are required is that the incompatible donor for a patient could back out of an exchange once his/her patient has recieved a kidney from another donor. This leaves some other patient without a new kidney and also without their incompatible donor, which is their bargaining chip in the market.

Natural approaches to dealing with NP-hard problems include looking for good heuristics or approximation algorithms. However, such approaches are unacceptable in this context - any loss in optimality may lead to patients dying unnecessarily. As such, several attempts have been made at devising an exact algorithm, however all have had trouble in scaling up to solve markets of the size expected by the upcoming national kidney-exchange market.

In Chapter 4, we present a new algorithm that exceeds what is required by this market. The algorithm uses column generation techniques from linear programming theory, as well as novel upper and lower bounding techniques.

1.3.2 Stable Roommates with Globally-Ranked Pairs

The stable roommates problem [29, 45] involves pairing up a set of agents, each of whom ranks the others in order of preference. In Chapter 5, we introduce a restriction of the roommate problem in which preferences are derived from a global ranking function on the agent pairs. This restriction is motivated by the following application:

When two (patient,donor) pairs are matched with each other in order to swap donors, we are not certain if the swap can occur until expensive last-minute compatibility tests are per-

formed on the donors and patients. If either potential transplant in the swap is incompatible, the swap is cancelled and the two patients must wait for a future match run. Doctors can rank potential swaps by their chance of success. This ranking induces patient preferences, since patients prefer to be involved in swaps with better chances of success.

In Chapter 5, we give a polynomial-time algorithm to find a *rank-maximal* matching in the globally-ranked pairs restriction. A *rank-maximal* matching includes the maximum number of rank-1 pairs possible, and subject to this, the maximum number of rank-2 pairs possible, and so on. Every rank-maximal matching is stable. Also, every rank-maximal matching is *strongly stable*, whenever a strongly stable matching exists. This is the first generalization of the rank-maximal matching algorithm due to Irving et al. [48] to a non-bipartite setting. Also, we prove several hardness results in an even more restricted setting, including for example the problem of finding a stable matching with the minimum number of weakly blocking pairs.

1.3.3 Egalitarian Matchings

In Chapter 6, we consider a further restriction of the stable roommates problem. Agent preferences must now be binary, meaning that agents either find each other acceptable or unacceptable. Even in a maximum matching of agents, it may not be possible to find partners for all the agents simultaneously. This exposes the market operator to charges of bias - why was one matching selected (in which a particular agent is not matched) over another (in which the agent is matched)? Can we be fair to the agents, even though some will not end up in the final matching?

The egalitarian matching problem is to find a distribution over maximum exchanges, so that, as much as possible, the *probability* of each agent being matched is equalized. Recently, it was shown that there always exists a distribution with very strong equality and incentive properties [78]. Building on the work of [78], in Chapter 6, we derive an alternative, simpler proof of the existence of this distribution. Our alternative proof is based on a local search approach where we start with an arbitrary distribution and then repeatedly make it more egalitarian. This local search based approach leads to a new structural characterization of an egalitarian distribution. Our main result uses this structural characterization as the basis of the first polynomial time algorithm for sampling from an egalitarian distribution in a general graph.

Chapter 2

Popular Matching

Declaration

The material in this chapter is joint work with Robert W. Irving, Telikepalli Kavitha, and Kurt Mehlhorn. The exposition is based on the paper describing our results [6].

Abstract

We consider the problem of matching a set of *applicants* to a set of *posts*, where each applicant has a *preference list*, ranking a non-empty subset of posts in order of preference, possibly involving ties. We say that a matching M is *popular* if there is no matching M' such that the number of applicants preferring M' to M exceeds the number of applicants preferring M to M'. In this chapter, we give the first polynomial-time algorithms to determine if an instance admits a popular matching, and to find a largest such matching, if one exists. For the special case in which every preference list is strictly ordered (i.e. contains no ties), we give an $O(n+m)$ time algorithm, where n is the total number of applicants and posts, and m is the total length of all the preference lists. For the general case in which preference lists may contain ties, we give an $O(\sqrt{n}m)$ time algorithm, and show that the problem has equivalent time complexity to the maximum-cardinality bipartite matching problem.

2.1 Introduction

An instance of the *popular matching problem* is a bipartite graph $G = (\mathcal{A} \cup \mathcal{P}, E)$ and a partition $E = E_1 \dot\cup E_2 \ldots \dot\cup E_r$ of the edge set. For exposition purposes, we call the nodes in $\mathcal{A}$ *applicants*, the nodes in $\mathcal{P}$ *posts*, and the edges in E_i the edges of rank i. If $(a, p) \in E_i$ and $(a, p') \in E_j$ with $i < j$, we say that a *prefers* p to p'. If $i = j$, we say that a is *indifferent* between p and p'. This ordering of posts adjacent to a is called a's *preference list*. We say that preference lists are *strictly ordered* if no applicant is indifferent between any two posts on his/her preference list. More generally, if applicants can be indifferent between posts, we say that preference lists contain *ties*.

A *matching* M of G is a set of edges no two of which share an endpoint. A node $u \in \mathcal{A} \cup \mathcal{P}$ is either *unmatched* in M, or *matched* to some node, denoted by $M(u)$ (i.e. $(u, M(u)) \in M$). We say that an applicant a *prefers* matching M' to M if (i) a is matched in M' and unmatched in M, or (ii) a is matched in both M' and M, and a prefers $M'(a)$ to $M(a)$. M' is *more popular than* M, denoted by $M' \succ M$, if the number of applicants that prefer M' to M exceeds the number of applicants that prefer M to M'.

Definition 2.1.1. *A matching M is* popular *if and only if there is no matching M' that is more popular than M.*

Example 2.1.2. *Figure 2.1 shows the preference lists for an example instance in which $\mathcal{A} = \{a_1, a_2, a_3\}$, $\mathcal{P} = \{p_1, p_2, p_3\}$, and each applicant prefers p_1 to p_2, and p_2 to p_3. Consider the three symmetrical matchings $M_1 = \{(a_1, p_1), (a_2, p_2), (a_3, p_3)\}$, $M_2 = \{(a_1, p_3), (a_2, p_1), (a_3, p_2)\}$ and $M_3 = \{(a_1, p_2), (a_2, p_3), (a_3, p_1)\}$. It is easy to verify that none of these matchings is popular, since $M_1 \prec M_2$, $M_2 \prec M_3$, and $M_3 \prec M_1$. In fact, this instance admits no popular matching, the problem being, of course, that the* more popular than *relation is not acyclic.*

$$\begin{array}{lcccc} a_1 & : & p_1 & p_2 & p_3 \\ a_2 & : & p_1 & p_2 & p_3 \\ a_3 & : & p_1 & p_2 & p_3 \end{array}$$

Figure 2.1: An instance for which there is no popular matching.

The *popular matching problem* is to determine if a given collection of preferences admits a popular matching, and to find such a matching, if one exists. We remark that popular matchings may have different sizes, and a largest such matching may be smaller than a maximum-cardinality matching. The *maximum-cardinality popular matching problem* then

is to determine if a given instance admits a popular matching, and to find a *largest* such matching, if one exists.

In this chapter, we use a novel characterization of popular matchings to give an $O(\sqrt{n}m)$ time algorithm for the maximum-cardinality popular matching problem, where n is the number of nodes, and m is the number of edges. For instances with strictly-ordered preference lists, we give an $O(n+m)$ time algorithm. No polynomial time algorithms were known previously.

2.1.1 Previous Work

The bipartite matching problem with a graded edge set is well-studied in the economics literature; see for example [2,77,99]. It models some important real-world markets, including the allocation of graduates to training positions [43], and families to government-owned housing [98]. Instances of these markets are restrictions of stable marriage instances [29,37], in which members of one side of the market (posts) are indifferent between members of the other side of the market (applicants).

The notion of popular matching was originally introduced by Gärdenfors [31] in the context of the full stable marriage problem. Every stable marriage instance admits a weakly stable matching (one for which there is no pair who strictly prefer each other to their partners in the matching) [37]. In fact, there can be an exponential number of weakly stable matchings [37], and so Gärdenfors considered the problem of finding one with additional desirable properties, such as popularity. Gärdenfors showed that when preference lists are strictly ordered, every stable matching is popular. He also showed that when preference lists contain ties, there may be no popular matching.

For the problem setup considered in this chapter, various other definitions of optimality have been studied. For example, a matching M is *Pareto optimal* [2,5,77] if there is no matching M' such that (i) some applicant prefers M' to M, and (ii) no applicant prefers M to M'. In particular, such a matching has the property that no coalition of applicants can collectively improve their allocation (say by exchanging posts with one another) without requiring some other applicant to be worse off. This is the weakest reasonable definition of optimality - see [5] for an algorithmically oriented exposition. Stronger definitions exist: a matching is *rank-maximal* [48] if it allocates the maximum number of applicants to their first choice, and then subject to this, the maximum number to their second choice, and so on. Rank-maximal matchings always exist and may be found in time $O(min(n, C\sqrt{n})m)$ [48], where C is the maximum edge rank used in the matching. Finally, we mention *maximum-utility* matchings, which maximize $\sum_{(a,p)\in M} u_{a,p}$, where $u_{a,p}$ is the utility of allocating post p to

applicant a. Maximum-utility matchings can be found through an obvious transformation to the maximum-weight matching problem. Popular matchings are Pareto optimal, though Pareto optimal matchings are not always popular [31].

2.1.2 Preliminaries

For exposition purposes, we create a unique *last resort* post $l(a)$ for each applicant a and assign the edge $(a, l(a))$ higher rank than any edge incident on a. In this way, we can assume that every applicant is matched, since any unmatched applicant can be allocated to his/her last resort. Matchings with the property that every applicant is matched are called *applicant-complete*. From now on then, matchings are applicant-complete, and the size of a matching is just the number of applicants not matched to their last resort. Also, without loss of generality, we assume that instances have no *gaps*, meaning that if an applicant a is incident to a rank i edge, then a is also incident to edges of all smaller ranks than i.

2.1.3 Chapter Outline

In Section 2.2, we develop an alternative characterization of popular matchings, under the assumption that preference lists are strictly ordered. We then use this characterization as the basis of a linear-time algorithm to solve the maximum-cardinality popular matching problem. In Section 2.3, we consider preference lists with ties, giving an $O(\sqrt{n}m)$ time algorithm for the maximum-cardinality popular matching problem. In Section 2.4 we give some empirical results on the probability that a popular matching exists. We conclude in Section 2.5 by summarizing recent work on popular matching.

2.2 Strictly-ordered Preference Lists

In this section, we restrict our attention to strictly-ordered preference lists, both to provide some intuition for the more general case, and because we can solve the popular matching problem in only linear-time. This last claim is not immediately clear, since Definition 2.1.1 potentially requires an exponential number of comparisons to even check that a given matching is popular. We begin this section then by developing an equivalent (though efficiently-

checkable) characterization of popular matchings.

2.2.1 Characterizing Popular Matchings

For each applicant a, let $f(a)$ denote the first-ranked post on a's preference list (i.e. $(a, f(a)) \in E_1$). We call any such post p an *f-post*, and denote by $f(p)$ the set of applicants a for which $f(a) = p$.

Example 2.2.1. *Figure 2.2 gives the preference lists for an instance with six applicants and six posts that we shall use to illustrate the results in this section. Note that we use l_i as an abbreviation for $l(a_i)$. The f-posts for this instance are p_1, p_2 and p_3, and $f(p_1) = \{a_1, a_2\}$, $f(p_2) = \{a_3, a_4, a_5\}$, $f(p_3) = \{a_6\}$.*

$$
\begin{array}{llllll}
a_1 & : & p_1 & p_2 & p_3 & l_1 \\
a_2 & : & p_1 & p_5 & p_4 & l_2 \\
a_3 & : & p_2 & p_1 & p_3 & l_3 \\
a_4 & : & p_2 & p_3 & p_6 & l_4 \\
a_5 & : & p_2 & p_6 & p_4 & l_5 \\
a_6 & : & p_3 & p_2 & p_5 & l_6
\end{array}
$$

Figure 2.2: An illustrative example.

The following lemma gives the first of three conditions necessarily satisfied by a popular matching.

Lemma 2.2.2. *Let M be any popular matching. Then for every f-post p, (i) p is matched in M, and (ii) $M(p) \in f(p)$.*

Proof. Every f-post p must be matched in M, for otherwise we can promote any $a \in f(p)$ to p, thereby constructing a matching more popular than M. Suppose for a contradiction then that p is matched to some $M(p) \notin f(p)$. Select any $a_1 \in f(p)$, let $a_2 = M(p)$, and since all f-posts are matched in M, let $a_3 = M(f(a_2))$. We can again construct a matching more popular than M, this time by (i) demoting a_3 to l_3, (ii) promoting a_2 to $f(a_2)$, and then (iii) promoting a_1 to p. □

Example 2.2.3. *According to Lemma 2.2.2, we can be sure that, if a popular matching exists for our example instance, then posts p_1, p_2 and p_3 are matched, and $M(p_1) \in \{a_1, a_2\}$, $M(p_2) \in \{a_3, a_4, a_5\}$, $M(p_3) = a_6$.*

For each applicant a, let $s(a)$ denote the first non-f-post on a's preference list (note that $s(a)$ must exist, due to the introduction of $l(a)$). We call any such post p an *s-post*, and remark that f-posts are disjoint from s-posts.

Example 2.2.4. *Figure 2.3 shows the preference lists for our example instance with the f-posts and s-posts highlighted. The bold entry in each preference list is the f-post and the underlined entry is the s-post.*

$$
\begin{array}{lcllll}
a_1 & : & \mathbf{p_1} & p_2 & p_3 & \underline{l_1} \\
a_2 & : & \mathbf{p_1} & \underline{p_5} & p_4 & l_2 \\
a_3 & : & \mathbf{p_2} & p_1 & p_3 & \underline{l_3} \\
a_4 & : & \mathbf{p_2} & p_3 & \underline{p_6} & l_4 \\
a_5 & : & \mathbf{p_2} & \underline{p_6} & p_4 & l_5 \\
a_6 & : & \mathbf{p_3} & p_2 & \underline{p_5} & l_6
\end{array}
$$

Figure 2.3: The f-posts and s-posts for the example instance.

In the next two lemmas, we show that a popular matching can only allocate an applicant a to either $f(a)$ or $s(a)$.

Lemma 2.2.5. *Let M be any popular matching. Then for every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list.*

Proof. Suppose for a contradiction that $M(a)$ is strictly between $f(a)$ and $s(a)$. Since a prefers $M(a)$ to $s(a)$, we have that $M(a)$ is an f-post. Furthermore, M is a popular matching, so a belongs to $f(M(a))$ (by Lemma 2.2.2), thereby contradicting the assumption that a prefers $f(a)$ to $M(a)$. □

Lemma 2.2.6. *Let M be a popular matching. Then for every applicant a, $M(a)$ is never worse than $s(a)$ on a's preference list.*

Proof. Suppose for a contradiction that a_1 prefers $s(a_1)$ to $M(a_1)$. If $s(a_1)$ is unmatched in M, we can promote a_1 to $s(a_1)$, thereby constructing a matching more popular than M. Otherwise, let $a_2 = M(s(a_1))$, and let $a_3 = M(f(a_2))$ (note that $a_2 \neq a_3$, since f-posts and s-posts are disjoint). We can again construct a matching more popular than M, this time by (i) demoting a_3 to l_3, (ii) promoting a_2 to $f(a_2)$, and then (iii) promoting a_1 to $s(a_1)$. □

The three necessary conditions we have just derived form the basis of the following preliminary characterization.

Lemma 2.2.7. *A matching M is popular if and only if*

(i) every f-post is matched in M, and

(ii) for each applicant a, $M(a) \in \{f(a), s(a)\}$.

Proof. Any popular matching necessarily satisfies conditions (i) and (ii) (by Lemmas 2.2.2 - 2.2.6). It remains to show that together, these conditions are sufficient.

Let M be any matching satisfying (i) and (ii), and suppose for a contradiction that there is some matching M' that is more popular than M. Let a be any applicant that prefers M' to M, and let $p' = M'(a)$ (note that p' is distinct for each such a). Now, since a prefers p' to $M(a)$, it follows from condition (ii) that $M(a) = s(a)$. So, p' is an f-post, which by condition (i), must be matched in M, say to a'. But then $p' = f(a')$ (by condition (ii) and since f-posts and s-posts are disjoint), and so a' prefers M to M'.

Therefore, for every applicant a that prefers M' to M, there is a distinct corresponding applicant a' that prefers M to M'. Hence, M' is not more popular than M, giving the required contradiction. □

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, we define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing two edges for each applicant a: one to $f(a)$, the other to $s(a)$. We remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq l(a)$.

Example 2.2.8. *Figure 2.4 shows the reduced graph for our example instance.*

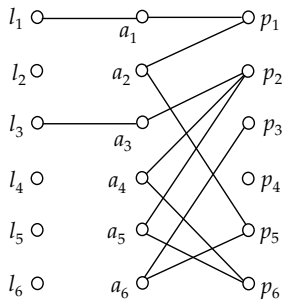


Figure 2.4: The reduced graph G' for the example instance

Lemma 2.2.7 gives us that M is a popular matching of G if and only if every f-post is matched in M, and M belongs to the graph G'. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 2.2.9. *M is a popular matching of G if and only if*

(i) every f-post is matched in M, and

(ii) M is an applicant-complete matching of the reduced graph G'.

Example 2.2.10. *By applying Theorem 2.2.9 to the reduced graph of Figure 2.4, it may be verified that our example instance admits four popular matchings, two of size 5 and two of size 4, as listed below. (Clearly, in the matchings of size 5, a_3 is matched with his last resort in the reduced graph, and in those of size 4, a_1 is also matched with his last resort.)*

$$M_1 = \{(a_1, p_1), (a_2, p_5), (a_4, p_2), (a_5, p_6), (a_6, p_3)\}$$

$$M_2 = \{(a_1, p_1), (a_2, p_5), (a_4, p_6), (a_5, p_2), (a_6, p_3)\}$$

$$M_3 = \{(a_2, p_1), (a_4, p_2), (a_5, p_6), (a_6, p_3)\}$$

$$M_4 = \{(a_2, p_1), (a_4, p_6), (a_5, p_2), (a_6, p_3)\}$$

2.2.2 Algorithmic Results

Figure 2.5 contains an algorithm for solving the popular matching problem. The correctness of this algorithm follows immediately from the characterization in Theorem 2.2.9. We only remark that at the termination of the loop, every f-post must be matched, since $f(a)$ is unique for each applicant a, and f-posts are disjoint from s-posts. We now show a linear-time implementation of this algorithm.

It is clear that the reduced graph G' of G can be constructed in $O(n + m)$ time. G' has $O(n)$ edges, since each applicant has degree 2, and so it is also clear that the loop phase requires only $O(n)$ time. It remains to show how we can efficiently find an applicant-complete matching of G', or determine that no such matching exists.

One approach involves computing a maximum-cardinality matching M of G', and then testing if M is applicant-complete. However, using the Hopcroft-Karp algorithm for maximum-

```
Popular-Matching(G = (A ∪ P, E))
    G' := reduced graph of G;
    if G' admits an applicant-complete matching M then
        for each f-post p unmatched in M
            let a be any applicant in f(p);
            promote a to p in M;
        return M;
    else
        return "no popular matching";
```

Figure 2.5: Linear-time popular matching algorithm for instances with strictly-ordered preference lists

cardinality matching [41], this would take $O(n^{3/2})$ time, which is super-linear, whenever m is $o(n^{3/2})$. Consider then the algorithm in Figure 2.6.

```
Applicant-Complete-Matching(G' = (A ∪ P, E'))
    M := ∅;
    while some post p has degree 1
        a := unique applicant adjacent to p;
        M := M ∪ {(a, p)};
        G' := G' − {a, p};    // remove a and p from G'
    while some post p has degree 0
        G' := G' − {p};
    // Every post now has degree at least 2
    // Every applicant still has degree 2
    if |P| < |A| then
        return "no applicant-complete matching";
    else
        // G' decomposes into a family of disjoint cycles
        M' := any maximum-cardinality matching of G';
        return M ∪ M';
```

Figure 2.6: Linear-time algorithm for finding an applicant-complete matching in G'

This algorithm begins by repeatedly matching a degree 1 post p with the unique applicant a adjacent to p. No subsequent augmenting path can include p (since it is matched and has degree 1), so we can remove both a and p from consideration. It is not hard to see that this loop can be implemented to run in $O(n)$ time, using for example, degree counters and lazy deletion. After this, we remove any degree 0 posts, so that all remaining posts have degree at least 2, while all remaining applicants still have degree exactly 2. Now, if $|\mathcal{P}| < |\mathcal{A}|$, G'

cannot admit an applicant-complete matching by Hall's Marriage Theorem [38]. Otherwise, we have that $|\mathcal{P}| \geq |\mathcal{A}|$, and $2|\mathcal{P}| \leq \sum_{p \in \mathcal{P}} \deg(p) = 2|\mathcal{A}|$. Hence, it must be the case that $|\mathcal{A}| = |\mathcal{P}|$, and every post has degree exactly 2. G' therefore decomposes into a family of disjoint cycles, and we only need to walk over these cycles, choosing every second edge.

We summarize the preceding discussion in the following lemma.

Lemma 2.2.11. *We can find a popular matching, or determine that no such matching exists, in $O(n + m)$ time.*

We now consider the maximum-cardinality popular matching problem. Let $\mathcal{A}_1$ be the set of all applicants a with $s(a) = l(a)$, and let $\mathcal{A}_2 = \mathcal{A} - \mathcal{A}_1$. Our target matching must satisfy conditions (i) and (ii) of Theorem 2.2.9, and among all such matchings, allocate the fewest $\mathcal{A}_1$-applicants to their last resort.

We begin by constructing G' and testing for the existence of an applicant-complete matching M of $\mathcal{A}_2$-applicants to posts (using the Applicant-Complete-Matching algorithm in Figure 2.6). If no such M exists, then G admits no popular matching by Theorem 2.2.9. Otherwise, we remove all edges from G' that are incident on a last resort post, and exhaustively augment M, each time matching an additional $\mathcal{A}_1$-applicant with his/her first-ranked post. If any $\mathcal{A}_1$-applicants are unmatched at this point, we simply allocate them to their last resort. Finally, we ensure that every f-post is matched, as in the Popular-Matching algorithm in Figure 2.5. It is clear that the resulting matching is a maximum-cardinality popular matching, and so we only comment on the time complexity of augmenting M.

Note that an alternating path Q from an unmatched applicant a is completely determined (since applicants have degree 2). If we are able to augment along this path, then no subsequent augmenting path can contain a node in Q, since such a path would necessarily terminate at a, who is already matched. Otherwise, if there is no augmenting path from a, then it is not hard to see that again, no subsequent augmenting path can contain a node in Q. This means we only need to visit and mark each node at most once, leading to the following result.

Theorem 2.2.12. *For instances with strictly-ordered preference lists, we can find a maximum-cardinality popular matching, or determine that no such matching exists, in $O(n + m)$ time.*

2.3 Preference Lists with Ties

In this section, we relax our assumption that preference lists are strictly ordered, and consider problem instances with ties. We begin by developing a generalization of the popular matching characterization, similar to Theorem 2.2.9. Using this characterization, we then go on to give a $O(\sqrt{n}m)$ time algorithm for solving the maximum-cardinality popular matching problem. For the special case in which all edges have rank one, the problem of finding a popular matching reduces to the problem of finding a maximum-cardinality bipartite matching. Hence, we cannot hope for a faster popular matching algorithm without also improving on Hopcroft and Karp's $O(\sqrt{n}m)$ time algorithm for finding a maximum-cardinality matching in a bipartite graph [41].

2.3.1 Characterizing Popular Matchings

For each applicant a, let $f(a)$ denote the *set* of first-ranked posts on a's preference list. Again, we refer to all such posts p as *f-posts*, and denote by $f(p)$ the set of applicants a for which $p \in f(a)$.

It may no longer be possible to match *every* f-post p with an applicant in $f(p)$ (as in Lemma 2.2.2), since, for example, there may now be more f-posts than applicants. Below then, we work towards generalizing this key lemma.

Let M be a popular matching of some instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$. We define the *first-choice graph* of G as $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$, where E_1 is the set of all rank-one edges.

Example 2.3.1. *Figure 2.7 gives an example instance that we use as an illustration in this section. Ties in the preference lists are indicated by parentheses.*

$$
\begin{array}{lcllll}
a_1 & : & (p_1 & p_2) & p_4 & l_1 \\
a_2 & : & p_1 & (p_2 & p_5) & l_2 \\
a_3 & : & p_2 & (p_4 & p_6) & l_3 \\
a_4 & : & p_2 & p_1 & p_3 & l_4 \\
a_5 & : & p_4 & p_3 & p_2 & l_5 \\
a_6 & : & (p_5 & p_6) & p_1 & l_6
\end{array}
$$

Figure 2.7: An example with ties in the preference lists.

The graph G_1 for this instance is shown in Figure 2.8.

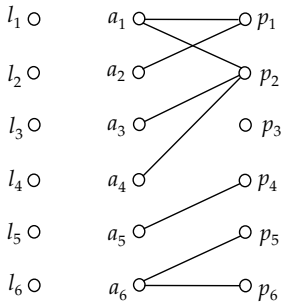


Figure 2.8: The graph G_1 for the example instance with ties

For instances with strictly-ordered preference lists, Lemma 2.2.2 is equivalent to requiring that every f-post is matched in $M \cap E_1$ (note that f-posts are the only posts with non-zero degree in G_1). But since applicants have a unique first choice in this context, Lemma 2.2.2 is also equivalent to the weaker condition that $M \cap E_1$ is a maximum matching of G_1. The next lemma shows that this weaker condition must also be satisfied when ties are permitted.

Lemma 2.3.2. *Let M be a popular matching. Then $M \cap E_1$ is a maximum matching of G_1.*

Proof. Suppose for a contradiction that $M_1 = M \cap E_1$ is not a maximum matching of G_1. Then M_1 admits an augmenting path $Q = \langle a_1, p_1, \ldots, p_k \rangle$ with respect to G_1. It follows that a_1 is unmatched in M or $M(a_1) \notin f(a_1)$, and p_k is either unmatched in M, or $M(p_k) \notin f(p_k)$. We now show how to use Q to construct a matching M' that is more popular than M. Begin with $M' = M \setminus \{(a_1, M(a_1))\}$. There are two cases:

(i) *p_k is unmatched in M'.*
Since both a_1 and p_k are unmatched in M', we augment M' with Q.

In this new matching, a_1 is matched with p_1 (where $p_1 \in f(a_1)$), while all other applicants in Q remain matched to one of their first-ranked posts. Hence M' is more popular than M.

(ii) *p_k is matched in M'.*
Let $a_{k+1} = M'(p_k)$ and note that $p_k \notin f(a_{k+1})$. Remove (a_{k+1}, p_k) from M' and then augment M' with Q. Select any $p_{k+1} \in f(a_{k+1})$. If p_{k+1} is unmatched in M', we promote a_{k+1} to p_{k+1}. Otherwise, we demote $a = M'(p_{k+1})$ to either $l(a)$ (if $a \neq a_1$), or back to $M(a_1)$ (if $a = a_1$), after which we can promote a_{k+1} to p_{k+1}. It is clear from this that at least one of a_1 and a_{k+1} prefers M' to M. Also, at most one applicant (that is a) prefers M to M', though in this case *both* a_1 and a_{k+1} prefer M'. Hence, M' is more popular than M.

□

Example 2.3.3. *In our example, we see from Figure 2.8 and Lemma 2.3.2 that posts p_1, p_2 and p_4, and applicants a_5 and a_6 must be matched in any popular matching M. Furthermore, we deduce that $M(p_1) \in \{a_1, a_2\}$, $M(p_2) \in \{a_1, a_3, a_4\}$, $M(p_4) = a_5$, and $M(a_6) \in \{p_5, p_6\}$.*

We now begin working towards a generalized definition of $s(a)$. For instances with strictly-ordered preference lists, $s(a)$ is equivalent to the first post in a's preference list that has degree 0 in G_1. However, since Lemma 2.2.2 no longer holds, $s(a)$ may now contain any number of surplus f-posts. It will help us to know which f-posts *cannot* be included in $s(a)$, and for this we use the following well-known ideas from bipartite matching theory.

Let M_1 be a maximum matching of some bipartite graph $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$. (Note that we are using notation that matches our use of this theory - so $M_1 = M \cap E_1$, and G_1 is the graph G restricted to rank-one edges.) Using M_1, we can partition $A \cup \mathcal{P}$ into three disjoint sets: A node v is *even* (respectively *odd*) if there is an even (respectively odd) length alternating path (with respect to M_1) from an unmatched node to v. Similarly, a node v is *unreachable* if there is no alternating path from an unmatched node to v. Denote by $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$ the sets of even, odd, and unreachable nodes, respectively. The Gallai-Edmonds Decomposition Lemma, covered in detail in [56], gives some fundamental relationships between maximum matchings and this type of node partition.

Lemma 2.3.4. *[Gallai-Edmonds Decomposition] Let $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$ be the node sets defined by G_1 and M_1 above. Then*

(a) $\mathcal{E}$, $\mathcal{O}$ and $\mathcal{U}$ are pairwise disjoint. Every maximum matching in G_1 partitions the node set into the same partition of even, odd, and unreachable nodes.

(b) In any maximum-cardinality matching of G_1, every node in $\mathcal{O}$ is matched with some node in $\mathcal{E}$, and every node in $\mathcal{U}$ is matched with another node in $\mathcal{U}$. The size of a maximum-cardinality matching is $|\mathcal{O}| + |\mathcal{U}|/2$.

(c) No maximum-cardinality matching of G_1 contains an edge between two nodes in $\mathcal{O}$, or a node in $\mathcal{O}$ and a node in $\mathcal{U}$. And there is no edge in G_1 connecting a node in $\mathcal{E}$ with a node in $\mathcal{U}$.

Example 2.3.5. *In our example, it may be verified from a maximum matching, say $\{(a_1, p_2), (a_2, p_1), (a_5, p_4$ in Figure 2.8, that $\mathcal{E} = \{a_1, a_2, a_3, a_4, p_3, p_5, p_6, l_1, l_2, l_3, l_4, l_5, l_6\}$, $\mathcal{O} = \{a_6, p_1, p_2\}$ and $\mathcal{U} = \{a_5, p_4\}$.*

Now, since M_1 is a maximum-cardinality matching of G_1, Lemma 2.3.4(b) gives us that every odd or unreachable post p in G_1 must be matched in M to some applicant in $f(p)$. For a given applicant a, such posts cannot be members of $s(a)$, and so we define $s(a)$ to be the set of top-ranked posts in a's preference list that are *even* in G_1 (note that $s(a) \neq \emptyset$, since $l(a)$ is always even in G_1). This definition coincides with the one in Section 2.2, since degree 0 posts are even, and whenever every applicant has a unique first choice, posts with non-zero degree (i.e. f-posts) are odd or unreachable.

Example 2.3.6. *Figure 2.9 displays the preference lists for our example instance, annotated as before, with the f-posts in bold and the s-posts underlined. Note that, when ties are present, f-posts and s-posts may coincide, as occurs here for applicant a_6.*

$$
\begin{array}{lcllll}
a_1 & : & (\mathbf{p_1} & \mathbf{p_2}) & p_4 & \underline{l_1} \\
a_2 & : & \mathbf{p_1} & (p_2 & \underline{p_5}) & l_2 \\
a_3 & : & \mathbf{p_2} & (p_4 & \underline{p_6}) & l_3 \\
a_4 & : & \mathbf{p_2} & p_1 & \underline{p_3} & l_4 \\
a_5 & : & \mathbf{p_4} & \underline{p_3} & p_2 & l_5 \\
a_6 & : & (\underline{\mathbf{p_5}} & \underline{\mathbf{p_6}}) & p_1 & l_6
\end{array}
$$

Figure 2.9: An example with ties in the preference lists.

Recall that our original definition of $s(a)$ led to Lemmas 2.2.5 and 2.2.6, which restrict the set of posts to which an applicant can be matched in a popular matching. We now show that the generalized definition leads to analogous results here.

Lemma 2.3.7. *Let M be a popular matching. Then for every applicant a, $M(a)$ can never be strictly between $f(a)$ and $s(a)$ on a's preference list.*

Proof. Suppose for a contradiction that $M(a)$ is strictly between $f(a)$ and $s(a)$. Then since a prefers $M(a)$ to any post in $s(a)$ and because posts in $s(a)$ are the top-ranked even nodes in G_1, it follows that $M(a)$ must be an odd or unreachable node of G_1. By Lemma 2.3.4(b), odd and unreachable nodes are matched in every maximum matching of G_1. But since $M(a) \notin f(a)$, $M(a)$ is unmatched in $M \cap E_1$. Hence M is not a maximum matching on rank-one edges and so by Lemma 2.3.2, M is not a popular matching. □

Lemma 2.3.8. *Let M be a popular matching. Then for every applicant a, $M(a)$ is never worse than $s(a)$ on a's preference list.*

Proof. Suppose for a contradiction that $M(a_1)$ is strictly worse than $s(a_1)$. Let p_1 be any post in $s(a_1)$. If p_1 is unmatched in M, we can promote a_1 to p_1, thereby constructing a matching more popular than M. Otherwise, let $a_2 = M(p_1)$. There are two cases:

(a) $p_1 \notin f(a_2)$:
Select any post $p_2 \in f(a_2)$, and let $a_3 = M(p_2)$ (note that p_2 must be matched in M, for otherwise Lemma 2.3.2 is contradicted). We can again construct a matching more popular than M, this time by (i) demoting a_3 to l_3, (ii) promoting a_2 to p_2, and then (iii) promoting a_1 to p_1.

(b) $p_1 \in f(a_2)$:
Now, since $p_1 \in s(a_1)$ as well, it must be the case that p_1 is an even post in G_1. It follows then that G_1 contains (with respect to $M \cap E_1$) an even length alternating path $Q' = \langle p_1, a_2, ..., p_k \rangle$, where p_k is unmatched in $M \cap E_1$ (note that p_k may be matched in M though). Now, let $Q = \langle a_1, p_1, a_2, ..., p_k \rangle$ (i.e. a_1 followed by Q'), and let $M' = M \setminus \{(a_1, M(a_1)\}$.

The remaining argument follows the proof of Lemma 2.3.2. If p_k is unmatched in M', $M' \oplus Q$ is more popular then M. Otherwise, p_k is matched in M'. Let $a_{k+1} = M'(p_k)$ and note that $p_k \notin f(a_{k+1})$. Remove (a_{k+1}, p_k) from M' and then augment M' with Q. Select any $p_{k+1} \in f(a_{k+1})$. If p_{k+1} is unmatched in M', we promote a_{k+1} to p_{k+1} and M' is more popular than M since a_1 and a_{k+1} prefer it. Otherwise, p_{k+1} is matched in M' to some applicant a. If $a \neq a_1$, we can demote a to $l(a)$, and then promote a_{k+1} to p_{k+1} so that M' is more popular than M, since it is preferred by a_1 and a_{k+1}, and only a prefers M to M'. Alternatively, if $a = a_1$, we can reassign a_1 back to $M(a_1)$, and then promote a_{k+1} to p_{k+1}. In this case, all applicants are indifferent between M and M', except for a_{k+1}, who prefers M'.

□

The three necessary conditions we have just derived form the basis of the following preliminary characterization.

Lemma 2.3.9. *A matching M is popular in G if and only if*

(i) $M \cap E_1$ is a maximum matching of G_1, and

(ii) for each applicant a, $M(a) \in f(a) \cup s(a)$.

Proof. Any popular matching necessarily satisfies conditions (i) and (ii) (by Lemmas 2.3.2, 2.3.7 and 2.3.8). It remains to show that together, these conditions are sufficient.

Let M be any matching satisfying conditions (i) and (ii), and suppose for a contradiction that there is some matching M' that is more popular than M. Let a be any applicant that

prefers M' to M. We want to show that there is a distinct corresponding applicant a' that prefers M to M'.

The graph $H = (M \oplus M') \cap E_1$ consists of disjoint cycles and paths, each alternating between edges in $M \cap E_1$ and edges in $M' \cap E_1$. We claim that $M'(a)$ must be contained in a *non-empty path* Q of H. First, note that $M'(a)$ is an odd or unreachable node in G_1, since a prefers $M'(a)$ to $M(a)$, and $M(a) \in s(a)$ is a top-ranked even node of G_1 in a's preference list. So by condition (i) and Lemma 2.3.4(b), $M'(a)$ is matched in $M \cap E_1$. However, $M'(a) \neq M(a)$, so $M'(a)$ is not isolated in H. Also, $M'(a)$ cannot be in a cycle, since a is unmatched in $M \cap E_1$. Therefore, $M'(a)$ belongs to some non-empty path Q of H.

Now, one endpoint of Q must be a (if $M'(a) \in f(a)$) or $M'(a)$ (otherwise). So for each such applicant a, there is a distinct non-empty path Q. Since $M'(a)$ is odd or unreachable, every post p in Q is also odd or unreachable. It follows from Lemma 2.3.2 that all such posts must be matched in $M \cap E_1$, and so the other endpoint of Q is an applicant, say a'. It is easy to see then that a' prefers M to M', since $M(a') \in f(a')$, while $M'(a) \notin f(a')$.

Therefore, for every applicant a that prefers M' to M, there is a distinct corresponding applicant a' that prefers M to M'. Hence, M' is not more popular than M, giving the required contradiction. □

Given an instance graph $G = (\mathcal{A} \cup \mathcal{P}, E)$, we define the *reduced graph* $G' = (\mathcal{A} \cup \mathcal{P}, E')$ as the subgraph of G containing edges from each applicant a to posts in $f(a) \cup s(a)$. We remark that G' need not admit an applicant-complete matching, since $l(a)$ is now isolated whenever $s(a) \neq \{l(a)\}$.

Example 2.3.10. *Figure 2.10 shows the reduced graph for our example instance.*

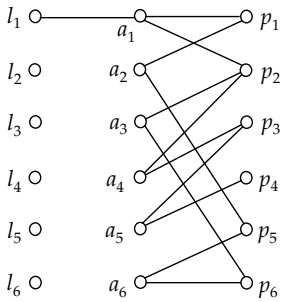


Figure 2.10: The reduced graph G' for the example instance with ties

Lemma 2.3.9 gives us that M is a popular matching of G if and only if M is a maximum matching on rank-one edges, and M belongs to the graph G'. Recall that all popular matchings are applicant-complete through the introduction of last resorts. Hence, the following characterization is immediate.

Theorem 2.3.11. *M is a popular matching of G if and only if*

(i) $M \cap E_1$ is a maximum matching of G_1, and

(ii) M is an applicant-complete matching of the reduced graph G'.

Example 2.3.12. *By applying Theorem 2.3.11 to the reduced graph of Figure 2.10, it may be verified that our example instance admits five popular matchings, two of size 6 and three of size 5, as listed below. (Clearly, in the three matchings of size 5, a_1 is matched with his last resort l_1 in the reduced graph.)*

$$M_1 = \{(a_1, p_1), (a_2, p_5), (a_3, p_2), (a_4, p_3), (a_5, p_4), (a_6, p_6)\}$$

$$M_2 = \{(a_1, p_2), (a_2, p_1), (a_3, p_6), (a_4, p_3), (a_5, p_4), (a_6, p_5)\}$$

$$M_3 = \{(a_2, p_1), (a_3, p_2), (a_4, p_3), (a_5, p_4), (a_6, p_5)\}$$

$$M_4 = \{(a_2, p_1), (a_3, p_2), (a_4, p_3), (a_5, p_4), (a_6, p_6)\}$$

$$M_5 = \{(a_2, p_1), (a_3, p_6), (a_4, p_2), (a_5, p_4), (a_6, p_5)\}$$

2.3.2 Algorithmic Results

In this section, we present algorithm Popular-Matching (see Figure 2.11) for solving the popular matching problem. This algorithm is based on the characterization given in Theorem 2.3.11, and is similar to the algorithm for computing a rank-maximal matching [48].

The following lemma is necessary for the correctness of our algorithm.

Lemma 2.3.13. *Algorithm Popular-Matching returns a maximum matching M on rank-one edges.*

Proof. Since M is obtained from M_1 by successive augmentations, every node matched by M_1 is also matched by M. Nodes in $\mathcal{O}$ and $\mathcal{U}$ are matched by M_1 (by Lemma 2.3.4(b)). Hence, nodes in $\mathcal{O}$ and $\mathcal{U}$ are matched in M.

Popular-Matching$(G = (\mathcal{A} \cup \mathcal{P}, E))$

1. Construct the graph $G' = (\mathcal{A} \cup \mathcal{P}, E')$, where $E' = \{(a, p) \mid p \in f(a) \cup s(a),\ a \in \mathcal{A}\}$.
2. Compute a maximum matching M_1 on rank-one edges i.e., M_1 is a maximum matching in $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.

   (M_1 is also a matching in G' because $E' \supseteq E_1$)
3. Delete all edges in G' connecting two nodes in the set $\mathcal{O}$ or a node in $\mathcal{O}$ with a node in $\mathcal{U}$, where $\mathcal{O}$ and $\mathcal{U}$ are the sets of odd and unreachable nodes of $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$.

   Determine a maximum matching M in the modified graph G' by augmenting M_1.
4. If M is not applicant-complete, then declare that there is no popular matching in G.
   Else return M.

Figure 2.11: $O(\sqrt{n}m)$ popular matching algorithm for preference lists with ties.

First, we claim that G', even before any potential edge deletions from Step 3 of the algorithm, has no edges of rank greater than one incident on nodes in $\mathcal{O}$ and nodes in $\mathcal{U} \cap \mathcal{P}$. Let us consider any odd or unreachable node $p \in \mathcal{P}$. This is never a candidate for $s(a)$, for any applicant a, and hence no edge of the type (a, p), where $p \in s(a)$ is incident on such a node. For odd nodes a that belong to $\mathcal{A}$, it is the case that they have first-ranked posts that are even, and so $s(a) \subseteq f(a)$. This proves our claim.

So the edges that we removed in Step 3 are rank-one edges, and these edges cannot be used by any maximum matching of G_1, by Lemma 2.3.4(c). (So no popular matching of G can use these edges.) Now the only neighbors of nodes in $\mathcal{O}$ are the even nodes of G_1 (call this set $\mathcal{E}$), and similarly, the only neighbors of nodes in $\mathcal{U} \cap \mathcal{P}$ are nodes in $\mathcal{U} \cap \mathcal{A}$ (by our edge deletions in Step 3 and Lemma 2.3.4(c)). This means that M must match all the nodes in $\mathcal{O}$ with nodes in $\mathcal{E}$ and all the nodes in $\mathcal{U} \cap \mathcal{P}$ with nodes in $\mathcal{U} \cap \mathcal{A}$.

So M has at least $|\mathcal{O}| + |\mathcal{U} \cap \mathcal{P}| = |\mathcal{O}| + |\mathcal{U}|/2$ edges of rank one. So M is a maximum matching on rank-one edges (by Lemma 2.3.4(b)). □

Thus the matching returned by the algorithm Popular-Matching is both an applicant-complete matching in G', and a maximum matching on rank-one edges. The correctness of the algorithm now follows from Theorem 2.3.11.

It is easy to see that the running time of our algorithm is $O(\sqrt{n}m)$: we use the algorithm of Hopcroft and Karp [41] to compute a maximum matching in G_1 and identify the set of edges E' and construct G' in $O(\sqrt{n}m)$ time. We then repeatedly augment M_1 (by the

Hopcroft-Karp algorithm) to obtain M. This gives us the following result.

Lemma 2.3.14. *We can find a popular matching, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

It is now a simple matter to solve the maximum-cardinality popular matching problem. Let us assume that the instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ admits a popular matching. (Otherwise, we are done.) We now want an applicant-complete matching in G' that is a maximum matching on rank-one edges and which maximizes the number of applicants not matched to their last resort.

Let M' be an arbitrary popular matching in G. We know that M' belongs to the graph G'. Remove all edges of the form $(a, l(a))$ from G' (and M'). Denote by H the resulting subgraph of G'. Note that M' is still a maximum matching on rank-one edges since no rank-one edge has been deleted from M' or G', but M' need not be a maximum matching in the graph H. Construct a maximum matching N in H by augmenting M'. N is a matching in G' that

(i) is a maximum matching on rank-one edges and

(ii) matches the maximum number of non-last-resort posts.

N need not be a popular matching. Determine a maximum matching M in G' by augmenting N. The matching M will be applicant-complete. Since M is obtained from N by successive augmentations, all posts that are matched by N are still matched by M. Hence, it follows that M is a popular matching that maximizes the number of applicants not matched to their last resort.

The following theorem is therefore immediate.

Theorem 2.3.15. *We can find a maximum-cardinality popular matching, or determine that no such matching exists, in $O(\sqrt{n}m)$ time.*

2.4 Empirical Results

In order to obtain an idea of the probability that a popular matching exists, we performed some simulations. The factors that affect this probability are the number of applicants, the number of posts, the lengths of the preference lists, and the number, size, and position of ties in these lists.

		t				
		0.0	0.2	0.4	0.6	0.8
	1	1000	1000	1000	1000	1000
	2	986	988	996	997	1000
	3	898	941	962	983	996
	4	759	846	929	979	999
	5	681	811	915	979	998
k	6	636	786	888	976	1000
	7	578	737	893	978	1000
	8	565	738	909	985	1000
	9	553	759	906	980	1000
	10	556	725	890	979	1000

Table 2.1: Proportion of instances with a popular matching for $n = 10$.

To keep this empirical investigation manageable, we restricted our attention to cases where the numbers of applicants and posts are equal, represented by n, and all preference lists have the same length k. We characterized the ties by a single parameter t, the probability that an entry in a preference list is tied with its predecessor.

Tables 2.1 and 2.2 contains the results of simulations carried out on randomly generated instances with $n = 10$ and $n = 100$ respectively. We set t to a sequence of values in the range 0.0 to 0.8. For $n = 10$ we allowed k to take all possible values $(1, ..., 10)$, and for $n = 100$ we investigated the cases $k = 1, ..., 10$ and $k = 20, 30, ..., 100$. We generated 1000 random instances in each case. In both cases, the table shows the number of instances admitting a popular matching.

These results, and others not reported in detail here, give rise to the following observations:

- When $t = 0.0$, i.e. there are no ties, the likelihood of a popular matching declines rapidly as k increases, and for large n is negligible except for very small values of k.

- Not surprisingly, increasing the value of t, and therefore the likely number and length of ties, increases the probability of a popular matching.

- For fixed n and t, increasing k initially reduces the likelihood of a popular matching, but beyond a certain range this effect all but disappears.

		t				
		0.0	0.2	0.4	0.6	0.8
	1	1000	1000	1000	1000	1000
	2	997	1000	999	1000	1000
	3	884	956	985	990	1000
	4	519	807	925	946	974
	5	204	534	806	863	879
	6	64	346	685	782	798
	7	20	192	534	705	721
	8	8	90	436	628	672
	9	3	39	309	578	670
k	10	2	28	243	531	675
	20	0	0	53	346	787
	30	0	0	37	302	776
	40	0	1	37	314	781
	50	0	0	44	291	791
	60	0	1	49	318	775
	70	0	2	36	304	780
	80	0	1	63	280	801
	90	0	0	38	306	776
	100	0	1	51	302	759

Table 2.2: Proportion of instances with a popular matching for $n = 100$.

2.5 Recent Work

Kavitha and Shah [53] found faster *randomized* algorithms for the popular matching problem and a weighted generalization of the rank-maximal matching problem. Their popular matching algorithm runs in time $O(n^\omega)$, where $\omega < 2.376$ is the best exponent for matrix multiplication.

Mahdian [57] showed that a popular matching exists with high probability, when preference lists are randomly constructed, and the number of posts is at least a factor of $\alpha \approx 1.42$ times larger than the number of applicants.

McCutchen [63] introduced two definitions of *unpopularity* and considered the problem of finding a least unpopular matching for instances in which no popular matching exists. McCutchen showed that under both definitions, whilst it is possible to compute the unpopularity of a matching in polynomial-time, the problem of finding a least unpopular matching is

NP-hard.

Huang et al. [42] studied approximation algorithms for the least unpopular matching problem, using the definitions of McCutchen. Their main algorithm generalizes the popular matching algorithm here by not only finding a first and second-choice post for each applicant, but also a third, fourth, ..., kth choice-post as well. Their algorithm stops at the first k for which every applicant can be allocated a post, and returns a matching. This matching is guaranteed to be within a factor of $(k-1)$ of the least unpopular matching according to one of McCutchen's definitions, and with a factor of $n(1-\frac{2}{k})$ according to the other.

Mestre [66] studied the weighted popular matching problem, in which each applicant has an associated weight, and a matching M' is more popular than another M if it the total weight of applicants preferring M' to M, exceeds the total weight of applicants preferring M to M' Mestre gave a $O(n+m)$-time algorithm for the special case of strictly-ordered preference lists, and a $O(\min k\sqrt{n}, nm)$-time algorithm for general preferences, where k is the number of distinct weights assigned to applicants.

Manlove and Sng [60] studied the capacitated popular matching problem, in which each post p can be allocated as many as $c(p) \geq 1$ applicants at the same time, where $c(p)$ is the *capacity* of p. Manlove and Sng showed how to find a maximum-cardinality popular matching in $O(\sqrt{C}n_1 + m)$ time when preference lists are strictly ordered, where C is the sum of the capacities of the posts, n_1 is the number of applicants, and m is the length of all the preference lists. They also gave a $O((\sqrt{C}+n_1)m)$-time algorithm for general preference lists. In a subsequent paper, Sng and Manlove [59] gave a $O(\sqrt{C}n_1 + m)$-time algorithm for finding a maximum-cardinality popular matching when applicants have weights, posts have capacities, and preferences are strictly ordered.

Abraham and Kavitha [3] studied *voting paths*, which are sequences of matchings, where each matching is more popular than its predecessor, ending in a popular matching. Voting paths are important for dynamic instances in which applicants and posts can enter or leave the instance, and applicants can change their preferences. In these situations, it is not feasible to recompute a popular matching after every change because the new popular matching may not be strictly more popular than the current unpopular matching. Abraham and Kavitha showed that, assuming a popular matching exists, it is always possible to find a 2-step voting path that starts from any matching, and that a shortest-length voting path can be found in linear time.

Kavitha and Nasre [52] give an $O(n^2 + m)$-time algorithm for finding an *optimal* popular matching when preferences are strict. An *optimal* popular matching is a maximal element amongst all popular matchings of a partial order π that satisfies two natural conditions.

Let M_1 and M_2 be two matchings with $M_1 \geq_\pi M_2$. The first condition is that, for all $e \notin M_1 \cup M_2$, $M_1 + e \geq_\pi M_2 + e$. The second condition is that, for all $e \in M_1 \cap M_2$, $M_1 - e \geq_\pi M_2 - e$. This definition of optimality is very general, encompassing particular definitions such as rank-maximality, maximum-utility, and fairness. (A fair popular matching is a popular matching that assigns the minimum number of applicants possible to their nth choice, and subject to this, the minimum number to their $(n-1)$-th choice, and so on.

McDermid and Irving [64] showed how to represent the structure of a popular matching for strictly-ordered preferences by a *switching graph*. This leads to a number of interesting results, including linear-time algorithms to count the number of popular matchings, find a random popular matching, and determine every applicant-post pair that appears in some popular matching. McDermid and Irving also used their switching graph approach to give improved algorithms for finding a rank-maximal popular matching in $O(n \log n + m)$ time, and a maximum-utility popular matching in $O(n + m)$ time.

Finally, Sng [90] studied popular matchings in the context of the stable marriage problem when preference lists can involve ties and be incomplete, as well as being symmetric. Using properties of this preference class, Sng derived a characterization of popular matchings and a polynomial-time algorithm to determine if a given matching is popular. The complexity of determining if a given instance admits a popular matching is still open.

Acknowledgment: We would like to thank David Manlove for directing us to previous work in the area, and commenting on an early draft. We would also like to thank Julian Mestre for correcting our description of Gärdenfors' original results on popular matching.

Chapter 3

Layerable Mechanisms for Keyword Auctions

Declaration

The material in this chapter is joint work with Arash Asadpour and Kamal Jain. This exposition is based on the paper describing our results.

Abstract

In this work, we introduce the class of *layerable* mechanisms for keyword auctions. A mechanism is layerable if it can be decomposed into a collection of layers, where each layer is a multi-item single-unit demand (MISUD) auction. This decomposition leads to a new technique for designing and analyzing keyword auctions: Given a set of desirable properties, for example truthfulness and individual rationality, we can construct a mechanism with these properties by simply designing them into the MISUD auction for each layer. We demonstrate this technique by designing a new mechanism that allows a search engine (auctioneer) to truthfully bid in its own keyword auction.

3.1 Introduction

Web search engines use keyword auctions to sell the advertising space alongside their algorithmic search results. When a user searches for a keyword (e.g. "digital camera"), merchants wanting to target this keyword (e.g. Canon, Nikon and Amazon.com) bid to have their advertisements displayed on the search results page. Based on these bids, the search engine allocates merchants to advertisement slots.

Small changes in the auction mechanism can lead to big changes in both the bidding behavior of merchants, and the revenue collected by the search engine. As such, there has been significant recent interest in the design and analysis of keyword auction mechanisms (see [54] for a recent summary).

In this work, we introduce the class of *layerable* mechanisms for keyword auctions. Several existing mechanisms are layerable in the classical keyword auction setting. These include the Generalized Second Price mechanism [23], which is used by the major search engines, the laddered auction [10], and the VCG mechanism [19, 36, 96].

Layerable mechanisms can be decomposed into a collection of layers, where each layer is a multi-item single-unit demand (MISUD) auction. Since designing MISUD auctions is typically easy, this decomposition leads to a simple technique for designing new keyword auction mechanisms. Also, this decomposition gives a new way to analyze existing mechanisms: instead of analyzing the general setting, we can focus on the restricted MISUD layers.

3.1.1 Preliminaries

A keyword auction consists of n merchants competing to win one of k advertising slots. Each merchant i has a private value $v_{i,j}$ for winning each slot j. We make the standard assumption that every merchant values slot j at least as highly as slot $(j+1)$. We also assume, for convenience, that $v_{i,k+1} = 0$.

In a (direct revelation) mechanism, each merchant i submits a bid $b_{i,j}$ for each slot j. (Note that $b_{i,j}$ may not equal $v_{i,j}$.) Using these bids, the mechanism allocates merchants to slots, and charges each merchant i a price p_i. A mechanism is characterized by its allocation and pricing functions.

Given a fixed collection of merchant bids, let $x_{i,j}$ be the probability that merchant i is allocated slot j. We assume utilities are quasilinear, and so merchant i's utility under this

allocation is $u_i = \sum_{j=1}^{k} x_{i,j} v_{i,j} - p_i$. Merchants select their bids in order to maximize their own utility.

In a *deterministic allocation*, each merchant is assigned to at most one slot, no slot is assigned to more than one merchant, and if slot $(j+1)$ is filled, then slot j is filled as well. A *randomized allocation* is a probability distribution over deterministic allocations. Randomized allocations are characterized by the following constraints:

1. $\sum_{j=1}^{k} x_{i,j} \leq 1$, for all merchants i.
2. $\sum_{i=1}^{n} x_{i,j} \leq 1$, for all slots j.
3. $\sum_{i=1}^{n} x_{i,j+1} > 0$ implies $\sum_{i=1}^{n} x_{i,j} = 1$, for all slots j.

The first two classes of constraints define a fractional matching. Given a fractional matching of merchants to slots, it is easy to construct an allocation that satisfies the third class of constraints: Suppose $\sum_{i=1}^{n} x_{i,j} < 1$ for some slot j, while $x_{i,j+1} > 0$. Then, since $v_{i,j} \geq v_{i,j+1}$, we can move probability from $x_{i,j+1}$ to $x_{i,j}$ such that merchant i maintains the same allocation value, and at least one of slot-j's deficit, or $x_{i,j+1}$, goes to 0. We also remark that any fractional matching can be converted in polynomial time to a distribution over deterministic allocations via the Birkhoff-von Neumann decomposition theorem [14, 97].

We are interested in keyword auction mechanisms with various desirable properties. These include:

1. **Truthfulness:** Every merchant i maximizes its utility by bidding truthfully (i.e. $\langle b_{i,j} = v_{i,j} \rangle_{j=1}^{k}$), independently of what the other merchants bid.
2. **Individual Rationality:** Every merchant i that bids truthfully obtains utility $u_i \geq 0$.
3. **Efficiency:** The allocation function maximizes social welfare, i.e. $\sum_{i=1}^{n} \sum_{j=1}^{k} x_{i,j} v_{i,j}$.
4. **Optimal / Revenue Maximizing:** Amongst all truthful mechanisms, revenue (i.e. $\sum_{i=1}^{n} p_i$) is maximized.

Keyword auctions have two important restrictions. First, when all slots are identical, each merchant has a *single* value for winning any one of the slots. This is the MISUD auction setting. And second, if there is only one slot, the setting becomes a single-item auction, and merchants again have single-parameter valuations.

We note that truthful mechanisms are well-characterized in the single-parameter setting. Let v_i (respectively b_i) be merchant i's value (respectively bid) for an item. For any fixed choice b_{-i} of bids by the other merchants, if merchant i bids b_i, let $x_i(b_i)$ be the probability merchant i has of winning an item, and let $p_i(b_i)$ be the price charged to merchant i.

Theorem 3.1.1 ([68]). *A mechanism (x,p) in the single-parameter setting is truthful if and only if, for any merchant i and any fixed choice of b_{-i}, i) $x_i(b_i)$ is monotone increasing in b_i, and ii) $p_i(b_i) = b_i x_i(b_i) - \int_0^{b_i} x_i(b)db$.*

As a concrete example of where Theorem 3.1.1 applies, consider Vickery's second-price single-item auction [96]. In this auction, the item is allocated to the merchant with the highest bid. This allocation function is monotonic, since by bidding higher, a merchant is only more likely to have the highest bid, and thus win the item. Given this allocation function, Theorem 3.1.1 prescribes the truth inducing price function for each merchant. It is easy to verify that the merchant with the highest bid pays a price equal to the second highest bid, and that all other merchants pay a price of 0.

In the general keyword auction setting, merchants do not have single parameter valuations. However, Theorem 3.1.1 can still be used to design truthful MISUD auctions in each layer of a layerable mechanism. Theorem 3.2.5 shows that a layerable mechanism constructed in this way is itself truthful.

3.1.2 Previous Work

Combinatorial auctions are a generalization of keyword auctions in which each merchant can be allocated a *bundle* of items, rather than just a single item/slot. Hence, any mechanism that applies in the combinatorial setting also applies in the keyword auction setting.

Perhaps the most famous mechanism for combinatorial auctions is the VCG mechanism [19, 36, 96]. VCG works in the following way: First, the mechanism finds the allocation that maximizes the total value of merchants for their allocated slots. Next, the mechanism charges each merchant i the difference in total value to the other merchants caused by merchant i receiving its allocation. This combination of allocation and pricing function makes VCG the unique mechanism that is individually rational, efficient and truthful. The VCG mechanism works for arbitrary merchant valuations of slots, so that in particular $v_{i,j}$ need not be at least as much as $v_{i,j+1}$.

Although VCG has three of the four main properties we are interested in, no major search engine uses VCG for their keyword auctions. Instead, most keyword auctions use a *rank-based*

mechanism, under a restricted class of merchant valuations.

We firstly remark on the restricted valuation class. In the standard keyword auction setting, each merchant i has a single private value v_i per user click-through of its advertisement. This value is independent of the slot that the click-through comes from and means that the merchant only needs to submit a single bid b_i to the mechanism. The value of merchant i for slot j is given by $v_{i,j} = CTR_{i,j}v_i$, where $CTR_{i,j}$ is the click-through rate of i's advertisement in slot j. In general, the only constraints on click-through rates are that $CTR_{i,j} \geq CTR_{i,j+1}$, since $v_{i,j} \geq v_{i,j+1}$. More typically though, click-through rates are *separable* [10], meaning that they are modelled by a merchant-specific factor, μ_i, and slot-specific factor, θ_j, so that $CTR_{i,j} = \mu_i\theta_j$, and $v_{i,j} = \mu_i\theta_j v_i$.

In a rank-based mechanism, each merchant i is assigned a weight w_i. Merchants are then ranked in non-increasing order of their weight-bid product $w_i b_i$. The mechanism then allocates merchant $i \leq k$ in the ranking to slot i.

There are many ways to assign weights to merchants. For example, the original "Overture" model sets $w_i = 1$, so that merchants are ranked directly by their bids [10]. Alternatively, the "Google" model sets w_i to the $CTR_{i,1}$, i.e. the estimated click-through rate for merchant i in slot 1. This means that merchants are ranked by their approximate revenue to the search engine.

Both the "Overture" and "Google" rank-based mechanisms have the same fundamental pricing function: each merchant $i \leq k$ in the ranking pays the minimum bid it needs in order to retain its rank. This is one way to generalize the single-item Vickery auction [96] (VCG has a different natural generalization), and so any rank-based keyword mechanism with this pricing function is called a *Generalized Second Price* (GSP) mechanism. GSP is not truthful, however, in the "Overture" model, there exists a Nash equilibrium under GSP in which the allocation is efficient, and the prices charged to the merchants are the same as the VCG prices [23]. Prior to the GSP pricing rule, Overture used a generalization of the English Auction, called the Generalized First Price (GFP) mechanism. Like GSP, GFP is rank-based, however in GFP each merchant $i \leq k$ pays exactly its bid for the slot it wins.

Our work is most closely related to Aggarwal, Goel, and Motwani's paper [10] on laddered auctions, and Roughgarden and Sundararajan's paper [84] on the trade-off between efficient and optimal auctions.

The laddered auction [10] was designed to be a truthful version of the GSP mechanism. In particular, the laddered auction is a rank-based mechanism like GSP, differing only in its pricing function. Our definition of a layerable mechanism is inspired by the following property of the laddered auction pricing function. Consider merchant i in the ranking who

wins slots i. Then for the first part of the merchant's allocation, it pays the same price as the merchant in slot k, and for the next part of its allocation it pays the same price as the merchant in slot $(k-1)$, and so on.

Roughgarden and Sundararajan [84] note that efficient auctions and optimal auctions can both be thought of as superposition of multi-item auctions. It turns out this is similar to our definition of a layerable mechanism. However, our results are set in a significantly more general setting, and our focus is on defining the class of layerable mechanisms, exploring its properties, and using the decomposition as a way to design new mechanisms.

Aggarwal et al. have a recent paper [9] that is similar in spirit to our work. In their paper, they define a general valuation class called the *max-value* model. Many variants and generalizations of the standard value-per-click model can be expressed in this class. For example, merchants can have arbitrary preferences over the slots, meaning that the slot 1 is not necessarily the most valuable slot. Aggarwal et al. show how many allocation functions over this valuation class can be recast as stable matchings in a new stable marriage model involving money. They give an algorithm for finding a merchant-optimal stable matching of merchants to slots, which guarantees truthfulness. Our work is similar in that we also define a broad class of mechanisms and show how to design a truthful mechanism in this class.

3.1.3 Chapter Outline

In Section 3.2, we define layerable mechanisms, and explore our new decomposition technique. In Section 3.3, we use our technique to design a truthful mechanism in which the auctioneer can win one of its own advertising slots. Finally, in Section 3.4, we conclude with some open problems.

3.2 Layerable Mechanisms

In this section, we introduce the class of *layerable mechanisms* for keyword auctions. Intuitively, a mechanism is *layerable* if it can be decomposed into a collection of layers, where each layer consists of a MISUD auction mechanism. We require that the collection of layers be strategically equivalent to the keyword auction setting, so that, in particular, utility-maximizing merchants i) obtain the same allocation, and ii) are charged the same price in both settings. This decomposition leads to a new technique for understanding, analyzing and designing keyword auctions.

Before formally defining the decomposition, we give some more intuition. Consider a deterministic allocation of merchants to slots, in which, without loss of generality, merchant i wins slot i for $i = 1$ to k. We can write merchant i's valuation for slot i as the telescoping sum: $v_{i,i} = \sum_{j=i}^{k}(v_{i,j} - v_{i,j+1})$, since $v_{i,k+1} = 0$. Note that each term of this sum represents the difference in merchant i's valuation between two successive slots. This difference is non-negative, since $v_{i,j} \geq v_{i,j+1}$. Figure 3.1 shows the entire allocation written in terms of these telescoping sums:

	Telescoping Sum Layer						
Value	1	2	3	...	j	...	k
$v_{1,1}$	$v_{1,1} - v_{1,2}$	$v_{1,2} - v_{1,3}$	$v_{1,3} - v_{1,4}$	$\cdots$	$v_{1,j} - v_{1,j+1}$	$\cdots$	$v_{1,k} - v_{1,k+1}$
$v_{2,2}$		$v_{2,2} - v_{2,3}$	$v_{2,3} - v_{2,4}$	$\cdots$	$v_{2,j} - v_{2,j+1}$	$\cdots$	$v_{2,k} - v_{2,k+1}$
$v_{3,3}$			$v_{3,3} - v_{3,4}$	$\cdots$	$v_{3,j} - v_{3,j+1}$	$\cdots$	$v_{3,k} - v_{3,k+1}$
$\vdots$				$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v_{j,j}$					$v_{j,j} - v_{j,j+1}$	$\cdots$	$v_{j,k} - v_{j,k+1}$
$\vdots$						$\vdots$	$\vdots$
$v_{k,k}$							$v_{k,k} - v_{k,k+1}$

Figure 3.1: Deterministic allocation written with telescoping sums

Consider layer j in Figure 3.1. Notice that j merchants win the difference between their valuations for slot-j and slot-$(j+1)$. Also, if a merchant wins in layer-j, it also wins in layer-$(j+1)$. The key observation now is that rather than sell whole slots, we can sell differences between successive slots. For example, merchant i wins the difference between its valuation for slot-i and slot-$(i+1)$, and also the difference between slot-$(i+1)$ and slot-$(i+2)$, and so on.

Define a *layer-j* mechanism M_j as one in which merchants compete to win one of j identical tokens, where each token has value $(v_{i,j} - v_{i,j+1})$ to merchant i. Note that M_j is just a MISUD auction in which merchants have single-parameter valuations.

Let $d_{i,j}$ be the bid of merchant i in M_j. (Note that $d_{i,j}$ may not equal $(v_{i,j} - v_{i,j+1})$.) Given a fixed collection of merchant bids, let $y_{i,j}$ be the probability that merchant i is allocated a token in M_j. Since there are j tokens to sell, we have the constraint $\sum_{i=1}^{n} y_{i,j} \leq j$. Let $p_{i,j}$ be the price charged to merchant i by M_j. Finally, let $u_{i,j} = y_{i,j}(v_{i,j} - v_{i,j+1}) - p_{i,j}$ be the utility of merchant i. We are now ready to define the class of layerable mechanisms.

Definition 3.2.1. *A keyword auction mechanism M is* layerable *under valuation class $\mathcal{V}$ if and only if there exist mechanisms $(M_1, M_2, \ldots, M_k)$, such that each M_j is a layer-j*

mechanism, and, for all bids $\langle b_{i,j} \rangle_{i=1,j=1}^{i=n,j=k}$ *from* $\mathcal{V}$*, if* $d_{i,j} = b_{i,j} - b_{i,j+1}$*, then for all merchants* i*:*

1. $\sum_{j=1}^{k} x_{i,j} v_{i,j} = \sum_{j=1}^{k} y_{i,j}(v_{i,j} - v_{i,j+1})$*, and*
2. $p_i = \sum_{j=1}^{k} p_{i,j}$.

In this case, we say $M = (M_1, M_2, \ldots, M_k)$*. Also, if, for all* j*,* M_j *and* M *have the same underlying allocation and pricing functions, we say that* M *is* self-layerable.

3.2.1 Designing Layerable Mechanisms

In order to design a keyword auction mechanism, we can divide the task up into layers. Unfortunately, as Example 3.2.2 demonstrates, arbitrarily combining layer-j mechanisms may not lead to a feasible outcome.

Example 3.2.2. *Consider the two-slot auction below. If we use VCG in each layer, merchants 1 and 2 win the layer-2 tokens, obtaining value 5 and 2 respectively, while merchant 3 wins the layer-1 token, obtaining value 3. (Note that* $y_{3,1} \not\leq y_{3,2}$*.) The total value of these allocations is 10. However, the maximum-value allocation (i.e. merchant 1 in slot-2, and merchant 3 in slot-1) only has value 9. Hence, for arbitrary valuation functions, VCG cannot be layered.*

	VCG Bids		*Layered Bids*	
Merchant	*Slot-1*	*Slot-2*	*Slot-1 − Slot-2*	*Slot-2 − Slot-3*
1	*6*	*5*	*1*	*5*
2	*2*	*2*	*0*	*2*
3	*4*	*1*	*3*	*1*

In Theorem 3.2.3, we give a sufficient condition for a collection of layer-j mechanisms to form a layerable mechanism. Informally, this condition requires that each merchant wins at least as much of a layer-$(j+1)$ token as it wins of a layer-j token.

Theorem 3.2.3. *Given mechanisms* $(M_1, M_2, \ldots, M_k)$ *for each layer, there exists a layerable mechanism* $M = (M_1, M_2, \ldots, M_k)$ *under valuation class* $\mathcal{V}$*, if, for all bids* $\langle b_{i,j} \rangle_{i=1,j=1}^{i=n,j=k}$ *from* $\mathcal{V}$*,* $y_{i,j} \leq y_{i,j+1}$ *when* $d_{i,j} = b_{i,j} - b_{i,j+1}$.

Proof. First we note that if M exists, we can set its price function to $p_i = \sum_{j=1}^{k} p_{i,j}$. It remains to show that there exists a feasible allocation function $x_{i,j}$ such that all merchants are allocated the same value under $x_{i,j}$ as they obtain in $(M_1, M_2, \ldots, M_k)$. Consider any merchant i:

$$\begin{aligned}\sum_{j=1}^{k} y_{i,j}(v_{i,j} - v_{i,j+1}) &= \sum_{j=1}^{k} y_{i,j}v_{i,j} - \sum_{j=1}^{k} y_{i,j}v_{i,j+1} \\ &= \sum_{j=1}^{k} y_{i,j}v_{i,j} - \sum_{j=2}^{k+1} y_{i,j-1}v_{i,j} \\ &= \sum_{j=1}^{k} v_{i,j}(y_{i,j} - y_{i,j-1}),\end{aligned}$$

where $y_{i,0} = v_{i,k+1} = 0$. Now, set $x_{i,j} = (y_{i,j} - y_{i,j-1})$. It is easy to see that merchant i obtains the same value from x as from $(M_1, M_2, \ldots, M_k)$. Also, $x_{i,j} \geq 0$, since $y_{i,j-1} \leq y_{i,j}$, by assumption. It remains to show that no merchant is fractionally allocated to more than one slot, and no slot is fractionally allocated to more than one merchant.

No merchant can be overpacked, since $\sum_{j=1}^{k} x_{i,j} = \sum_{j=1}^{k}(y_{i,j} - y_{i,j-1}) = y_{i,k} \leq 1$. It may turn out however that some slot is overpacked. Let j be the first such slot, so that $\sum_{i=1}^{n} x_{i,j} = 1+s$, for some surplus $s > 0$. In this case, we can repack the allocation to ensure feasibility. Note that the total allocation to the first j slots is $\sum_{i=1}^{n}\sum_{j'=1}^{j} x_{i,j'} = \sum_{i=1}^{n}\sum_{j'=1}^{j}(y_{i,j'} - y_{i,j'-1}) = \sum_{i=1}^{n} y_{i,j} \leq j$, where the last inequality follows from the feasibility of y. Since $\sum_{i=1}^{n} x_{i,j} = 1 + s$, the total allocation to the first $(j-1)$ slots is $\sum_{i=1}^{n}\sum_{j'=1}^{j-1} x_{i,j} \leq j - 1 - s$. Hence, we have slack of at least s to repack slot-j allocations into earlier slots. Finally, since $v_{i,j-1} \geq v_{i,j}$ for all j, we can do this whilst not overpacking any merchant, or changing its allocated value.

□

In Theorem 3.2.4, we show that even if we restrict ourselves to combining layer-j mechanisms that satisfy $y_{i,j} \leq y_{i,j+1}$, we can still construct any layerable mechanism.

Theorem 3.2.4. *Let M be a layerable mechanism under valuation class $\mathcal{V}$. Then there exist layer-j mechanisms $(M_1, M_2, \ldots, M_k)$ such that for all bids $\langle b_{i,j}\rangle_{i=1,j=1}^{i=n,j=k}$ from $\mathcal{V}$, if $d_{i,j} = b_{i,j} - b_{i,j+1}$, then $y_{i,j} \leq y_{i,j+1}$.*

Proof. Consider any merchant i. Then under M, merchant i obtains value:

$$\sum_{j=1}^{k} x_{i,j} v_{i,j} = \sum_{j=1}^{k} x_{i,j} (\sum_{j'=j}^{k} (v_{i,j'} - v_{i,j'+1})) = \sum_{j=1}^{k} (v_{i,j} - v_{i,j+1}) (\sum_{j'=1}^{j} x_{i,j'})$$

Now, set the allocation functions for $(M_1, M_2, \ldots, M_k)$ such that $y_{i,j} = \sum_{j'=1}^{j} x_{i,j'}$. Then merchant i is allocated the same value by x and y. Also, as required, $y_{i,j} \leq y_{i,j+1}$, since $x_{i,j+1} \geq 0$. It remains to show that the layer-j allocations are feasible.

No merchant is allocated more than one layer-j token, since $y_{i,j} = \sum_{j'=1}^{j} x_{i,j'} \leq 1$ (by the assumption that x is feasible). Also, layer-j sells at most j tokens, since $\sum_{i=1}^{n} y_{i,j} = \sum_{i=1}^{n} \sum_{j'=1}^{j} x_{i,j'} = \sum_{j'=1}^{j} \sum_{i=1}^{n} x_{i,j'} \leq \sum_{j'=1}^{j} 1 = j$.

Finally, we remark that we can distribute the price p_i charged to merchant i arbitrarily, so that $p_i = \sum_{j=1}^{n} p_{i,j}$. □

In Theorem 3.2.5, we show how a layerable mechanism can inherit the properties of its simpler layer-j mechanisms. Hence, if we want to design a keyword auction with certain desirable properties, we can try to build these properties into each layer.

Theorem 3.2.5. *Let $M = (M_1, M_2, \ldots, M_k)$ be a layerable mechanism under valuation class $\mathcal{V}$. Then,*

1. *M is individually rational if M_j is individually rational, for all j.*
2. *M is truthful if and only if M_j is truthful, for all j.*

Proof. The first statement is immediate. We now prove the second statement on truthfulness.

($\Leftarrow$) Suppose for a contradiction that M is not truthful. Then for some fixed choice of bids by the other merchants, some merchant i has a strategic bid such that $u_i(\langle b_{i,j} \rangle_{j=1}^{k}) > u_i(\langle v_{i,j} \rangle_{j=1}^{k})$. By Definition 3.2.1, if merchant i bids truthfully in $(M_1, \ldots, M_k)$, it obtains utility $u_i(\langle v_{i,j} \rangle_{j=1}^{k})$. However, if merchant i bids $\langle d_{i,j} = (b_{i,j} - b_{i,j+1}) \rangle_{j=1}^{k}$ instead, it obtains utility $u_i(\langle b_{i,j} \rangle_{j=1}^{k})$. Hence, some M_j must not be truthful, which is a contradiction.

($\Rightarrow$) Suppose for a contradiction that some layer is not truthful. Note that we implicitly assume bids $\langle d_{i,j} \rangle_{i=1,j=1}^{i=n,j=k}$ in $(M_1, \ldots, M_k)$ are legal only when there exist $\langle b_{i,j} \rangle_{i=1,j=1}^{i=n,j=k}$ under

$\mathcal{V}$ such that $d_{i,j} = b_{i,j} - b_{i,j+1}$. Let u_T (resp. u_S) be merchant i's utility when it bids truthfully (resp. strategically) in all layers. Then for some fixed collection of bids d_{-i} by the other merchants, $u_S > u_T$. If merchant i bids truthfully in M, it obtains utility u_T. However, if merchant i bids $b_{i,j} = \sum_{j'=j}^{k} d_{i,j'}$, it obtains utility $u_S > u_T$, which contradicts the truthfulness of M. □

As an aside, if M is individually rationaly, it may not be the case that every M_j is individually rational. For example, a mechanism M_j may take money from merchant i and give it to another i', while another mechanism M'_j takes the same amount of money from merchant i' and gives it to merchant i. Neither mechanism is individually rational by themselves, but the layerable mechanism formed from their combination is individually rational.

In order to build a truthful layerable mechanism from layers, it is necessary and sufficient to combine truthful layer-j mechanisms. Recall that Theorem 3.1.1 already characterizes truthful layer-j mechanisms. In particular, we only have to select a monotonic allocation function, and then from this, the pricing function is determined.

3.2.2 Example Layerable Mechanisms

We have already seen in Example 3.2.2 that for general valuation classes, it is not possible to layer the VCG mechanism. In particular, the problem is that the condition $y_{i,j} \leq y_{i,j+1}$ is not satisfied by merchant 3, who wins a layer-1 token, but not a layer-2 token. In this section, we consider some restricted valuation classes in which VCG and other well-known mechanisms are self-layerable.

We now switch to the standard keyword auction setting, in which each merchant i has a value v_i per click-through of its advertisement. We also assume that click-through rates are separable, so that $CTR_{i,j} = \mu_i \theta_j$, where μ_i is the merchant-specific factor, and θ_j is the slot-specific factor.

In this setting, the value of merchant i for a layer-j token is $\mu_i v_i(\theta_j - \theta_{j+1})$. It follows that the efficient allocation for layer-j can be found by a simple greedy algorithm: rank merchants in non-increasing order of $\mu_i b_i$, and allocate tokens to the first j merchants in the ranking. Since the ranking of merchants is the same across all layers, if merchant i wins a token in layer j, it also wins one in layer-$(j+1)$. Hence, by Theorem 3.2.3, we can run VCG on each layer to form a feasible keyword auction mechanism. Note that by the telescoping sum equivalence, merchant i in the ranking obtains the same total click-through rate as if it was allocated to slot i. This is equivalent to the overall VCG allocation in the keyword auction setting. Also,

the total price charged to merchant i across all layers, namely $\sum_{j=i}^{k} \mu_{j+1} b_{j+1}(\theta_j - \theta_{j+1})$, is the same price charged by the VCG mechanism in the standard keyword auction setting.

This combination of valuation class and VCG allocation function has the property that whenever a merchant wins in layer-j, it also wins in layer-$(j+1)$. This property, which satisfies the sufficient condition of Theorem 3.2.3, also holds for *rank-based* mechanisms, such as GFP and GSP. Recall that these mechanisms share the same allocation function, differing only in the prices charged to the merchants. The allocation function works as follows: each merchant i is assigned a weight multiplier w_i, where w_i is independent of i's bid. Merchants are then ranked in non-increasing order of their weight-bid product, $w_i b_i$, with merchant i in the ranking being awarded slot i. As in the VCG example, this allocation is equivalent to the one obtained by running the allocation function on each layer. Also, it is easy to verify from the mechanism-specific price functions that the merchants are charged the same price in both the keyword auction setting and the layered setting. Hence, these mechanisms are self-layerable.

We conclude this section by giving an alternative and much simpler derivation of the pricing function for the laddered auction [10]. Recall that the laddered auction was designed to have the same allocation function as GFP and GSP, whilst having the additional property of truthfulness. Rather than work in the keyword auction setting, as in [10], we use our decomposition technique to work in the simpler MISUD layer-j setting.

We begin by remarking that the allocation function is clearly monotonic, since if a merchant bids higher, it is guaranteed to appear no later in the ranking. Hence, we can use Theorem 3.1.1 to derive the pricing function that guarantees truthfulness. Let b_i be the value-per-click bid of merchant i in the ranking, so that its bid in layer-j is $d_{i,j} = (CTR_{i,j} - CTR_{i,j+1})b_i$. Merchant $i \leq j$ wins as long as $w_i b_i > w_{j+1} b_{j+1}$. So the minimum bid for i to win is $d'_{i,j} = (CTR_{i,j} - CTR_{i,j+1}) w_{j+1} b_{j+1} / w_i$. Substituting these expressions into Theorem 3.1.1, we get $p_{i,j}(d_{i,j}) = d_{i,j} - (d_{i,j} - d'_{i,j}) = d'_{i,j} = (CTR_{i,j} - CTR_{i,j+1}) w_{j+1} b_{j+1} / w_i$. Since merchant i wins in layers k through i, the total price charged to merchant i in the laddered auction is $\sum_{j=i}^{k} (CTR_{i,j} - CTR_{i,j+1}) w_{j+1} b_{j+1} / w_i$, which is equal to the expression in [10]. Also, by Theorem 3.2.5, the layerable mechanism formed by the combinations of these layers is truthful, since each layer-j mechanism is truthful.

Finally, we remark that this decomposition approach extends to valuation classes beyond the single-parameter value-per-click model. All that we require is that merchant valuations $v_{i,j}$ are such that $y_{i,j} \leq y_{i,j+1}$ in the ranking allocation function. In particular, this is true whenever there is a common ranking of merchants across all layers. Note that this common ranking does not have to be the result of the simple weight-bid product.

3.3 Selling Items to the Auctioneer

Modern search engines offer a wide array of products and services. Examples include operating systems, video-game consoles, photo-sharing websites, and instant messenger applications. Hence, like any other merchant, a search engine may want to bid for an advertising slot in a keyword auction. However, unlike other merchants, the search engine also runs the auction and collects auction payments. We will see that this can lead to a conflict of interest, even with the VCG or laddered auction mechanisms. In this section, we use our decomposition technique to design a truthful generalization of the laddered auction in which the auctioneer can retain one the advertising slots. This new mechanism also guarantees that the auctioneer obtains at least as much utility as in the standard laddered auction, and sometimes, substantially more.

3.3.1 Setting and Preliminary Discussion

We begin by extending the merchant notation to include the auctioneer. Let v_0 be the value of the auctioneer per user click-through of its advertisement. The auctioneer's value for slot j is given by $v_{0,j} = CTR_{0,j}v_0$, where $CTR_{0,j}$ is the auctioneer's click-through rate for slot j. Given allocation and pricing functions (x, p), the auctioneer's utility is $u_0 = \sum_{j=1}^{k} x_{0,j}v_{0,j} + \sum_{i=1}^{n} p_i$. Finally, as with the other merchants, the auctioneer may advertise in at most one slot at a time.

If the auctioneer has an accurate prior distribution over possible merchant valuations, Myerson's optimal auction mechanism [68] applies directly. We demonstrate our technique in the prior-free setting in which Myerson's auction does not apply. This setting is useful when it is difficult to obtain an accurate prior, or when computing and setting individual reserve prices for different keyword auctions is computationally infeasible. The prior-free setting also has better ex-post efficiency, since slots are never empty as long as there are enough merchants to fill them.

In traditional single-shot auction theory, the auctioneer has no value for the items it is selling. Hence, its utility is just the payments it receives from the merchants. In contrast to this, a search engine may prefer to keep a slot for itself, rather than sell it to a merchant. To see that this may lead to a conflict of interest, consider a single-item Vickery auction in which the merchant bids are $b_1 > b_2 > \ldots > b_n$. The merchant with the highest bid b_1 wins and pays the second-highest bid b_2. This mechanism, a special case of VCG, is truthful in the traditional setting. Suppose however the auctioneer is allowed to bid, and has value 0 for the item. By strategically raising its bid to $b_1 - \epsilon$, the auctioneer forces the winning merchant

to pay more than b_2, which directly increases the auctioneer's own utility. Hence, VCG is not truthful when the auctioneer is allowed to bid.

Before moving on to the keyword auction setting, we show how to alter the MISUD Vickery auction so that merchants can be confident that they are not being exploited in this way. Two general principles guide our design. First, for any allocation, the price charged to a merchant should be independent of the auctioneer's bid. And second, an auctioneer should be able to claim a slot if it values the slot more than it can charge a merchant. We describe the new mechanism below.

Given j-identical items to sell, the auctioneer begins by performing a preliminary run of the traditional Vickery auction, in which it does not bid. The result is that bids $b_1, b_2, \ldots,$ and b_j win, with each paying the price b_{j+1}. If the auctioneer has value $v_0 \leq b_{j+1}$ for an item, it sells all the items to obtain revenue jb_{j+1}. Otherwise, the auctioneer values an item more than the price being charged. In this case, the auctioneer selects a preliminary winner uniformly at random, retains this winner's item, and charges every other winner a price of b_{j+1} for an item. (Note that for efficiency, the auctioneer would prefer to retain b_j's item, however winning merchants would then have an incentive to overbid in order to not have the lowest winning bid.) Lemma 3.3.1 proves the truthfulness of this mechanism in the more general laddered auction setting.

3.3.2 Generalized Laddered Auction

Recall from Section 3.2.2 that the laddered auction ranks merchants by their weight-bid product. Relabel all merchants so that $w_1b_1 > w_2b_2 > \ldots > w_nb_n$. Our task is to design a truthful generalization of the laddered auction in which the auctioneer can bid. Rather than working in the general keyword auction setting, we use our decomposition technique to work in the much simpler MISUD setting. In particular, this means we will i) design a truthful layer-j mechanism for selling j identical tokens, and ii) show that there exists an allocation of merchants to slots that awards each merchant the combined value promised by each of the layers. Truthfulness of the overall mechanism follows immediately from Theorem 3.2.5. Our final mechanism applies in the restricted setting where click-through rates are separable. Also, for some combinations of merchant valuations, the mechanism may be infeasible because it allocates a merchant to two slots simultaneously. Future work will involve addressing both of these issues. We begin in the general setting by designing the following layer-j mechanism.

As in the MISUD mechanism above, the auctioneer performs a preliminary run of the laddered auction in which it does not bid. The result is that the first j merchants in the ranking

win a token, with each merchant $i \leq j$ paying $p_{i,j} = (CTR_{i,j} - CTR_{i,j+1})w_{j+1}b_{j+1}/w_i$. (See the end of Section 3.2.2 for an explanation of these prices.) Note that each winning merchant i pays a different price based on its merchant-specific ratio $(CTR_{i,j} - CTR_{i,j+1})/w_i$. Let t_j be the minimum ratio over all winning merchants, and let T_j be the set of merchants with this ratio. Since merchants in T_j are paying the least for their token, if the auctioneer wants to retain a token, it will be from one of these merchants. Specifically, if $(CTR_{0,j} - CTR_{0,j+1})v_0 > t_j w_{j+1} b_{j+1}$, the auctioneer will select a merchant from T_j uniformly at random and retain its token. Figure 3.2 contains a complete description of this mechanism.

Layer-j generalized laddered auction

1. Relabel non-auctioneer merchants so that $w_1 v_1 > w_2 v_2 > \ldots > w_n v_n$.
2. Allocate each merchant $i \leq j$ a token.
3. Charge each merchant $i \leq j$ the price $p_{i,j} = (CTR_{i,j} - CTR_{i,j+1})w_{j+1}b_{j+1}/w_i$.
4. Let $t_j = \min_{i \leq j} (CTR_{i,j} - CTR_{i,j+1})/w_i$. Let T_j consist of all merchants $i \leq j$ with ratio t_j. (T_j contains the winning merchants that pay the lowest price.)
5. If $(CTR_{0,j} - CTR_{0,j+1})v_0 > t_j w_{j+1} b_{j+1}$, select a merchant i from T_j uniformly at random, retain i's token for the auctioneer, and repay $t_j w_{j+1} b_{j+1}$ to i.

Figure 3.2: Truthful Layer-j Mechanism

Note that when the auctioneer retains a token, each merchant i in T_j obtains an expected allocation of $y_{i,j} = (1 - 1/|T_j|)$ - i.e. it obtains a $(1 - 1/|T_j|)$-fraction of a token. Also, its expected price is $p_{i,j} = (1 - 1/|T_j|)t_j w_{j+1} b_{j+1}$. Lemma 3.3.1 shows that this layer-j mechanism has the two properties we desire: individual rationality and truthfulness. It is immediate from this that the auctioneer obtains at least as much utility as from a traditional layer-j laddered auction.

Lemma 3.3.1. *The layer-j generalized laddered auction mechanism is individually rational and truthful.*

Proof. Individual rationality for the auctioneer follows from the auctioneer retaining a token when it values tokens more highly than the lowest price. Individual rationality for the other merchants i follows from the per-click price of $w_{j+1}v_{j+1}/w_i$, which is at most v_i if i wins the laddered auction.

We now prove the mechanism is truthful for the auctioneer. Suppose the auctioneer is allocated a token if it bids truthfully - i.e. $(CTR_{0,j} - CTR_{0,j+1})v_0 > t_j w_{j+1} b_{j+1}$. Overbidding does not change the outcome. Underbidding only changes the outcome if the auctioneer no longer wins a token. In this case, the change in the auctioneer's utility is $t_j w_{j+1} b_{j+1} - (CTR_{0,j} - CTR_{0,j+1})v_0 < 0$.

Suppose instead that the auctioneer does not win a token if it bids truthfully - i.e. $(CTR_{0,j} - CTR_{0,j+1})v_0 \leq t_j w_{j+1} b_{j+1}$. Underbidding does not change the outcome. Overbidding only changes the outcome if the auctioneer now wins a token. In this case, the change in the auctioneer's utility is $(CTR_{0,j} - CTR_{0,j+1})v_0 - t_j w_{j+1} b_{j+1} \leq 0$.

Finally, truthfulness for the non-auctioneer merchants follows from Theorem 3.1.1, since the merchants have single-parameter valuations in layer-j, and the allocation function is monotonic.

Next we show that the mechanism is truthful for the non-auctioneer merchants. Note that we could use Theorem 3.1.1 for these merchants, since it is a single-parameter setting, and the allocation function is monotonic. Instead, for exposition purposes, we give a direct proof. In the following discussion, $\{i \stackrel{?}{\in} T_j\}$ is 1 if $i \in T_j$, and 0 otherwise.

Suppose a merchant i is ranked outside the top j if it bids truthfully - i.e. $w_i v_i < w_j * b_j$. Underbidding does not change the outcome. Overbidding changes the outcome if the merchant wins. In this case, its utility is at most $(1 - 1/|T_j|)^{\{i \stackrel{?}{\in} T_j\}}(CTR_{i,j} - CTR_{i,j+1})(v_i - w_k b_k / w_i)$, which is no more than 0, since $v_i - w_k b_k / w_i < 0$.

Suppose instead that merchant i is ranked in the top j if it bids truthfully - i.e. $w_i v_i > w_{j+1} b_{j+1}$. Overbidding does not change the outcome. Underbidding only changes the outcome if the merchant falls outside the top j. In this case, the change in merchant i's utility is at most $0 - (1 - 1/|T_j|)^{\{i \stackrel{?}{\in} T_j\}}(CTR_{i,j} - CTR_{i,j+1})(v_i - w_{j+1} b_{j+1} / w_i)$, which is no more than 0, since $v_i > w_{j+1} b_{j+1}$. □

The next step is to prove that we can combine these layer$-j$ mechanisms to form a feasible keyword auction. In particular, this means we have to prove there is a feasible allocation x of merchants to slots (i.e. $\sum_{j=1}^{k} x_{i,j} \leq 1$ for all i, and $\sum_{i=0}^{n} x_{i,j} \leq 1$ for all j), such that each merchant i obtains value $\sum_{j=1}^{k} x_{i,j} v_{i,j} = \sum_{j=1}^{k} y_{i,j}(v_{i,j} - v_{i,j+1})$. Since $v_{i,j} = CTR_{i,j} v_i$, where v_i is independent of the slot, we only need to show that a merchant obtains the same click-through rate in x as in y.

Note that a merchant may lose a fraction of its token in layer-$(j+1)$, only to win a full

token in layer-j where tokens are too expensive for the auctioneer. As such, our main tool for this step, the sufficient condition $y_{i,j} \leq y_{i,j+1}$ of Theorem 3.2.3, does not apply. Instead, we construct x directly, though only for the following restricted setting.

We assume click-through rates are separable, i.e., $CTR_{i,j} = \mu_i \theta_j$. We also assume that $\mu_i/w_i = \mu_{i'}/w_{i'}$ for all merchants i, i'. This generalizes two standard models. The first is the "Overture" model with $w_i = 1$ and merchant independent click-through rates. The second is the "Google" model with $w_i = u_i$. These restrictions ensure that all winning non-auctioneer merchants in layer-j pay the same price $p_{i,j} = \frac{\mu_i}{w_i}(\theta_j - \theta_{j+1})w_{j+1}b_{j+1}$ in the preliminary laddered auction. Hence, if the auctioneer retains a token in layer-j, all winners are left with a $\frac{j-1}{j}$-fraction of a token. Let a be the last layer (starting from k) in which the auctioneer retains a token. Let r_i be the total click-through rate allocated to merchant i by the layered mechanisms. Figure 3.3 contains the prices charged by the mechanism and the expansion of r_i.

Merchant: i	Price: $p_i = \sum_{j=1}^{k} p_{i,j}$
$a..k$	$\sum_{j=i}^{k} \frac{j-1}{j} \frac{\mu_i}{w_i}(\theta_j - \theta_{j+1})w_{j+1}b_{j+1}$
$1..(a-1)$	$p_a + \sum_{j=i}^{a-1} \frac{\mu_i}{w_i}(\theta_j - \theta_{j+1})w_{j+1}b_{j+1}$

Merchant: i	Rate: $r_i = \sum_{j=1}^{k} y_{i,j}(\theta_j - \theta_{j+1})$
$a..k$	$\sum_{j=i}^{k} \frac{j-1}{j}(\theta_j - \theta_{j+1}) = \frac{i-1}{i}\theta_i + \sum_{j=i+1}^{k} \frac{1}{j(j-1)}\theta_j$
$1..(a-1)$	$\sum_{j=i}^{a-1}(\theta_j - \theta_{j+1}) + r_a = \theta_i - \theta_a/a + \sum_{j=a+1}^{k} \frac{1}{j(j-1)}\theta_j$
0 (auctioneer)	$\sum_{j=a}^{k}(\theta_j - \theta_{j+1}) = \theta_a$

Figure 3.3: Prices and Click-through rate allocations for generalized laddered auction

Note that for merchants $i \in [a..k]$, the coefficient of $\theta_{j>i}$ is $(\frac{j-1}{j} - \frac{j-2}{j-1}) = \frac{1}{j(j-1)}$. Also, note that for merchants $i \in [1..(a-1)]$, the first summand telescopes to $\theta_i - \theta_a$.

In Figures 3.4 and 3.5, we present an allocation function $x_{i,j}$. For the auctioneer ($i = 0$) and merchants $i > a$, we set their $x_{i,j}$ values to be the coefficients of θ_j in their rate allocation r_i. For merchants $i < a$, we allocate them to slot i with probability $\frac{r_i}{\theta_i}$. And finally, since slot a is already taken, merchant a is allocated the $(a-1)$ slots $j < a$ with probability $1 - \frac{r_j}{\theta_j}$, and also the slots from $a+1$ to k with probability $\sum_{j=a+1}^{k} \frac{a}{j(j-1)}$.

By construction, x allocates r_i click-through rate to each merchant i. It remains to show that x is feasible. First, note that no slot is overpacked. Slot a is fully allocated to the auctioneer. Slot $j < a$ is allocated to merchant j with probability $\frac{r_j}{\theta_j}$ and merchant a with

$$
\begin{aligned}
x_{i,j} &= 1, && \text{if } i = 0 \text{ (auctioneer), and} && j = a \\
&= \frac{r_i}{\theta_i}, && \text{if } 0 < i < a, \text{ and} && j < a \\
&= 1 - \frac{r_j}{\theta_j}, && \text{if } i = a, \text{ and} && j < a
\end{aligned}
$$

Figure 3.4: Allocation function for Slots $1 \ldots a$ in the generalized laddered auction

$$
\begin{aligned}
x_{i,j} &= \frac{j-1}{j}, && \text{if } i > a, \text{ and} && j = i \\
&= \frac{1}{j(j-1)}, && \text{if } i > a, \text{ and} && j > i \\
&= \frac{a}{j(j-1)}, && \text{if } i = a, \text{ and} && j > i
\end{aligned}
$$

Figure 3.5: Allocation function for Slots $(a+1) \ldots k$ in the generalized laddered auction

probability $1 - \frac{r_j}{\theta_j}$. Slot $j > a$ is allocated to merchant j with probability $\frac{j-1}{j}$, and also to merchants $a..(j-1)$ with a total probability of $\frac{a}{j(j-1)} + (j-1-a)\frac{1}{j(j-1)}$.

Next we consider the merchants. The auctioneer is fully allocated to slot a. Merchants $i > a$ have probability $\frac{i-1}{i} + \sum_{j=i+1}^{k} \frac{1}{j(j-1)}$ of obtaining a slot, where $\sum_{j=i+1}^{k} \frac{1}{j(j-1)} = \sum_{j=i+1}^{k} (\frac{j-1}{j} - \frac{j-2}{j-1}) = \frac{1}{i}\frac{k-i}{k} \leq \frac{1}{i}$. Merchants $i < a$ have probability $\frac{r_i}{\theta_i} < 1$ of obtaining a slot. Finally, merchant a has probability $\sum_{i=1}^{a-1}(1 - \frac{r_i}{\theta_i}) + \sum_{j=a+1}^{k} \frac{a}{j(j-1)}$ of obtaining a slot. In some restricted settings, this probability may exceed 1 and be as high as 2. Hence, when we convert x into a distribution over allocations via Birkhoff-von Neumann decomposition [14, 97], merchant a may appear in 0, 1 or 2 slots.

3.4 Conclusion

In this work, we introduced the class of layerable mechanisms for keyword auctions. A layerable mechanism can be decomposed into a collection layers, where each layer involves a simple MISUD mechanism. This decomposition leads to a new technique for designing and analyzing keyword auctions.

Unfortunately, we cannot combine mechanisms for each layer arbitrarily, since the overall mechanism may not be feasible. In Theorem 3.2.3, we proved a sufficient condition in order to guarantee feasibility. We also showed that any layerable mechanism can be constructed from layers that satisfy this sufficient condition.

To demonstrate our new decomposition technique, we constructed a truthful keyword auction in which the auctioneer can win one of its own advertisement slots. This setting is of independent interest, and raises several open problems, such as whether our new mechanism can be extended to the multi-unit demand setting. It would also be interesting to explore techniques for scaling down the allocation and pricing function in order to guarantee that no merchant is assigned to more than one slot at a time.

Finally, we are interested in other applications of our new decomposition design technique. We have already explored a generalization of the optimal auction setting in which merchants do not have single-parameter valuations. It turns out that we can run a Myerson auction in each layer, and interestingly, the reserve price can change from layer to layer. A related avenue for future research is considering if there are non-trivial truthful auctions in which the auctioneer can limit the number of slots after viewing the merchant bids. This problem, also mentioned in [10], might be more tractable to analyze in the MISUD setting.

Chapter 4

Clearing Algorithms for Barter Exchange Markets

Declaration

The material in this chapter is joint work with Avrim Blum and Tuomas Sandholm. The exposition is based on the paper describing our results [4].

Abstract

In barter-exchange markets, agents seek to swap their items with one another, in order to improve their own utilities. These swaps consist of cycles of agents, with each agent receiving the item of the next agent in the cycle. We focus mainly on the upcoming national kidney-exchange market, where patients with kidney disease can obtain compatible donors by swapping their own willing but incompatible donors. With over 70,000 patients already waiting for a cadaver kidney in the US, this market is seen as the only ethical way to significantly reduce the 4,000 deaths per year attributed to kidney disease.

The clearing problem involves finding a social welfare maximizing exchange when the maximum length of a cycle is fixed. Long cycles are forbidden, since, for incentive reasons, all transplants in a cycle must be performed simultaneously. Also, in barter-exchanges generally, more agents are affected if one drops out of a longer cycle. We prove that the clearing

problem with this cycle-length constraint is NP-hard. Solving it exactly is one of the main challenges in establishing a national kidney exchange.

We present the first algorithm capable of clearing these markets on a nationwide scale. The key is *incremental problem formulation*. We adapt two paradigms for the task: constraint generation and column generation. For each, we develop techniques that dramatically improve both runtime and memory usage. We conclude that column generation scales drastically better than constraint generation. Our algorithm also supports several generalizations, as demanded by real-world kidney exchanges.

Our algorithm was used for several months in 2007 by the *Alliance for Paired Donation*, one of four main regional kidney exchanges in the United States. The United Network for Organ Sharing (UNOS), which is the national body for organ donation in the United States, selected our algorithm to use for the upcoming nationwide kidney exchange.

4.1 Introduction

The role of kidneys is to filter waste from blood. Kidney failure results in accumulation of this waste, which leads to death in months. One treatment option is dialysis, in which the patient goes to a hospital to have his/her blood filtered by an external machine. Several visits are required per week, and each takes several hours. The quality of life on dialysis can be extremely low, and in fact many patients opt to withdraw from dialysis, leading to a natural death. Only 12% of dialysis patients survive 10 years [95].

Instead, the preferred treatment is a kidney transplant. Kidney transplants are the most common transplant. Unfortunately, the demand for kidneys far outstrips supply. In the United States in 2005, 4,052 people died waiting for a life-saving kidney transplant. During this time, almost 30,000 people were added to the national waiting list, while only 9,913 people left the list after receiving a deceased-donor kidney. The waiting list currently has over 70,000 people, and the median waiting time ranges from 2 to 5 years, depending on blood type.[1]

For many patients with kidney disease, the best option is to find a *living* donor, that is, a healthy person willing to donate one of his/her two kidneys. Although there are marketplaces for buying and selling living-donor kidneys, the commercialization of human organs is almost universally regarded as unethical, and the practice is often explicitly illegal, such as in the US. However, in most countries, live donation is legal, provided it occurs as a gift with no financial compensation. In 2005, there were 6,563 live donations in the US.

The number of live donations would have been much higher if it were not for the fact that, frequently, a potential donor and his intended recipient are blood-type or tissue-type incompatible. In the past, the incompatible donor was sent home, leaving the patient to wait for a deceased-donor kidney. However, there are now a few regional kidney exchanges in the United States, in which patients can swap their incompatible donors with each other, in order to each obtain a compatible donor.

These markets are examples of *barter exchanges*. In a barter-exchange market, *agents* (patients) seek to swap their *items* (incompatible donors) with each other. These swaps consist of cycles of agents, with each agent receiving the item of the next agent in the cycle. Barter exchanges are ubiquitous: examples include Peerflix (DVDs) [74], Read It Swap It (books) [75], and Intervac (holiday houses) [44]. For many years, there has even been a large shoe exchange in the United States [72]. People with different-sized feet use this to avoid having to buy two pairs of shoes. Leg amputees have a separate exchange to share the cost

[1]Data from the United Network for Organ Sharing [93].

of buying a single pair of shoes.

We can encode a barter exchange market as a directed graph $G = (V, E)$ in the following way. Construct one vertex for each agent. Add a weighted edge e from one agent v_i to another v_j, if v_i wants the item of v_j. The weight w_e of e represents the utility to v_i of obtaining v_j's item. A cycle c in this graph represents a possible swap, with each agent in the cycle obtaining the item of the next agent. The weight w_c of a cycle c is the sum of its edge weights. An *exchange* is a collection of disjoint cycles. The weight of an exchange is the sum of its cycle weights. A *social welfare* maximizing exchange is one with maximum weight.

Figure 4.1 illustrates an example market with 5 agents, $\{v_1, v_2, \ldots, v_5\}$, in which all edges have weight 1. The market has 4 cycles, $c_1 = \langle v_1, v_2 \rangle$, $c_2 = \langle v_2, v_3 \rangle$, $c_3 = \langle v_3, v_4 \rangle$ and $c_4 = \langle v_1, v_2, v_3, v_4, v_5 \rangle$, and two (inclusion) maximal exchanges, namely $M_1 = \{c_4\}$ and $M_2 = \{c_1, c_3\}$. Exchange M_1 has both maximum weight and maximum cardinality (i.e., it includes the most edges/vertices).

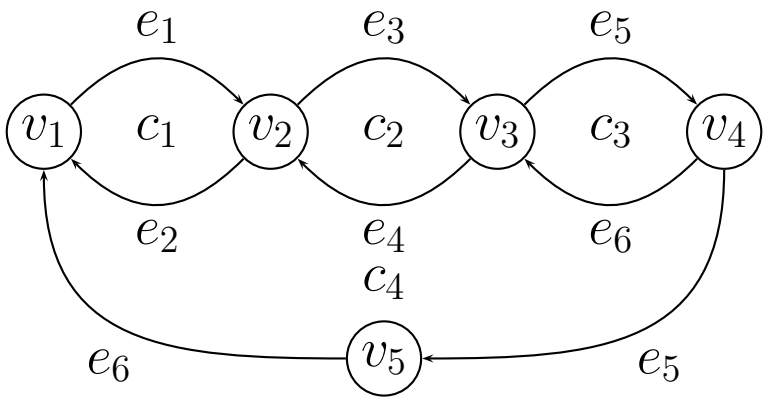


Figure 4.1: Example barter exchange market.

The *clearing problem* is to find a maximum-weight exchange consisting of cycles with length at most some small constant L. This cycle-length constraint arises naturally for several reasons. For example, in a kidney exchange, all operations in a cycle have to be performed simultaneously; otherwise a donor might back out after his incompatible partner has received a kidney. (One cannot write a binding contract to donate an organ.) This gives rise to a logistical constraint on cycle size: even if all the donors are operated on first and the same personnel and facilities are used to then operate on the patients, a k-cycle requires between $3k$ and $6k$ doctors, around $4k$ nurses, and almost k operating rooms.

Due to such resource constraints, the upcoming national kidney exchange market will likely

allow only cycles of length 2 and 3. It is worth pointing out that longer exchanges are performed. For example, in 2008, Johns Hopkins reported that its surgical teams had performed a 6-way exchange. However, another motivation for short cycles, besides the resource constraints, is that if the cycle fails to exchange, fewer agents are affected. For example, last-minute testing in a kidney exchange often reveals new incompatibilities that were not detected in the initial testing (based on which the compatibility graph was constructed). More generally, an agent may drop out of a cycle if his preferences have changed, or he/she simply fails to fulfill his/her obligations (such as sending a book to another agent in the cycle) due to forgetfulness.

In Section 4.3, we show that the decision version of the clearing problem is NP-complete when the cycle length constraint L is a constant at least 3. This contrasts with the cases where $L \geq n$ and $L = 2$, both of which allow polynomial time algorithms, as we will see in Sections 4.4 and 4.5 respectively. One approach for the NP-hard values of L might be to look for a good heuristic or approximation algorithm. However, our aim is an exact algorithm, since any loss of optimality could lead to unnecessary patient deaths. Furthermore, our aim is to use an *integer-linear program (ILP)* formulation. An attractive feature of using an ILP formulation is that it is easy to model a number of variations on the basic objective function, and to add additional constraints to the problem. For example, if 3-cycles are believed to be more likely to fail than 2-cycles, then one can simply give them a weight that is appropriately lower than 3/2 of the weight of a 2-cycle. Alternatively, if the primary aim is a maximum cardinality exchange, one can at least in a second pass find a solution (out of all maximum cardinality solutions) that has the fewest 3-cycles. Other variations that can be solved include finding various forms of "fault tolerant" (non-disjoint) collections of cycles in the event that certain pairs that were thought to be compatible turn out to be incompatible after all.

In this work, we present the first algorithm capable of clearing these markets on a nationwide scale. Straight-forward ILP encodings are too large to even *construct* on current hardware, and so solving them is intractable. The key then is incremental problem formulation. We adapt two paradigms for the task: constraint generation and column generation. For each, we develop a host of (mainly problem-specific) techniques that dramatically improve both runtime and memory usage.

4.1.1 Previous Work

Several recent papers have used simulations and market-clearing algorithms to explore the impact of a national kidney exchange [33, 80–82, 85, 89]. For example, using Edmond's

maximum-matching algorithm [24], [89] shows that a national pairwise-exchange market (using length-2 cycles only) would result in more transplants, reduced waiting time, and savings of $750 million in heath care costs over 5 years. Those results are conservative in two ways. Firstly, the simulated market contained only 4,000 initial patients, with 250 patients added every 3 months. It has been reported to us that the market could be almost double this size. Secondly, the exchanges were restricted to length-2 cycles (because that is all that can be modeled as maximum matching, and solved using Edmonds's algorithm). Allowing length-3 cycles leads to additional significant gains. This has been demonstrated on kidney exchange markets with 100 patients by using CPLEX to solve an integer-program encoding of the clearing problem [82].

Allowing cycles of length more than 3 often leads to no improvement in the size of the exchange [82]. (Furthermore, in a simplified theoretical model, *any* kidney exchange can be converted into one with cycles of length at most 4 [82].) Whilst this does not hold for general barter exchanges, or even for all kidney exchange markets, in Section 4.5.2 we make use of the observation that short cycles suffice to dramatically increase the speed of our algorithm.

At a high-level, the clearing problem for barter exchanges is similar to the clearing problem (aka winner determination problem) in combinatorial auctions. In both settings, the idea is to gather all the pertinent information about the agents into a central clearing point and to run a centralized clearing algorithm to determine the allocation. Both problems are NP-hard. For both problems, tree search has proven to be one of the most successful algorithmic approaches. Since 1999, significant work has been done in computer science and operations research on faster optimal tree search algorithms for clearing combinatorial auctions. (For a recent review, see [86].) However, the kidney exchange clearing problem (with a limit of 3 or more on cycle size) is different from the combinatorial auction clearing problem in significant ways. The most important difference is that the natural formulations of the combinatorial auction problem tend to easily fit in memory, so time is the bottleneck in practice. In contrast, the natural formulations of the kidney exchange problem (with $L = 3$) take at least cubic space in the number of patients to even model, and therefore memory becomes a bottleneck much before time does when using standard tree search, such as branch-and-cut in CPLEX, to tackle the problem. (On a 1GB computer and a realistic standard instance generator, discussed later, CPLEX 10.010 runs out of memory on five of the ten 900-patient instances and ten of the ten 1,000-patient instances that we generated.) Therefore, the approaches that have been developed for combinatorial auctions cannot handle the kidney exchange problem.

4.1.2 Chapter Outline

The rest of the chapter is organized as follows. Section 4.2 discusses the process by which we generate realistic kidney exchange market data, in order to benchmark the clearing algorithms. Section 4.3 contains a proof that the market clearing decision problem is NP-complete. Sections 4.4 and 4.5 each contain an ILP formulation of the clearing problem. We also detail in those sections our techniques used to solve those programs on large instances. Section 4.6 presents experiments on the various techniques. Finally, we present our conclusions in Section 4.7, and suggest future research directions.

4.2 Market Characteristics and Instance Generator

We test the algorithms on simulated kidney exchange markets, which are generated by a process described in Saidman *et al.* [85]. This process is based on the extensive nationwide data maintained by the United Network for Organ Sharing (UNOS) [93], so it generates a realistic instance distribution. Several papers have used variations of this process to demonstrate the effectiveness of a national kidney exchange (extrapolating from small instances or restricting the exchange to 2-cycles) [33, 80–82, 85, 89].

Briefly, the process involves generating patients with a random blood type, sex, and probability of being tissue-type incompatible with a randomly chosen donor. These probabilities are based on actual real-world population data. Each patient is assigned a potential donor with a random blood type and relation to the patient. If the patient and potential donor are incompatible, the two are entered into the market. Blood type and tissue type information is then used to decide on which patients and donors are compatible. One complication, handled by the generator, is that if the patient is female, and she has had a child with her potential donor, then the probability that the two are incompatible increases. (This is because the mother develops antibodies to her partner during pregnancy.) Finally, although our algorithms can handle more general weight functions, patients have a utility of 1 for compatible donors, since their survival probability is not affected by the choice of donor [21]. This means that the maximum-weight exchange has maximum cardinality.

Table 4.1 gives lower and upper bounds on the *size* of a maximum cardinality exchange in the kidney-exchange market, where the size is the number of patients receiving a kidney in te exchange. The lower bounds were found by clearing the market with length-2 cycles only using the polynomial-time algorithm in Section 4.5. The upper bounds were found with no restriction on cycle length using the polynomial-time algorithm in Section 4.4. For each

market size, the bounds were computed over 10 randomly generated markets. Note that there can be a large amount of variability in the markets - in one 5000 patient market, less than 1000 patients were in the maximum cardinality exchange.

	Maximum exchange size			
	Length-2 cycles only		Arbitrary cycles	
Patients	Mean	Max	Mean	Max
100	4.00e+1	4.60e+1	5.30e+1	6.10e+1
500	2.58e+2	2.80e+2	2.79e+2	2.97e+2
1000	5.35e+2	6.22e+2	5.61e+2	6.30e+2
2000	1.05e+3	1.13e+3	1.09e+3	1.16e+3
3000	1.63e+3	1.70e+3	1.68e+3	1.73e+3
4000	2.15e+3	2.22e+3	2.20e+3	2.27e+3
5000	2.53e+3	2.87e+3	2.59e+3	2.92e+3
6000	3.26e+3	3.32e+3	3.35e+3	3.39e+3
7000	3.80e+3	3.86e+3	3.89e+3	3.97e+3
8000	4.35e+3	4.45e+3	4.46e+3	4.55e+3
9000	4.90e+3	4.96e+3	5.01e+3	5.07e+3
10000	5.47e+3	5.61e+3	5.59e+3	5.73e+3

Table 4.1: Upper and lower bounds on exchange size.

Table 4.2 gives additional characteristics of the kidney-exchange market. Note that a market with 5000 patients can already have more than 450 million cycles of length 2 and 3. For more than 5000 patients, the number of length 2 and 3 cycles exceeds $2^{32} - 1$, the maximum unsigned integer size on our system.

4.3 Problem Complexity

In this section, we prove that (the decision version of) the market clearing problem with short cycles is NP-complete.

	Edges		Length 2 & 3 cycles	
Patients	Mean	Max	Mean	Max
100	2.38e+3	2.79e+3	2.76e+3	5.90e+3
500	6.19e+4	6.68e+4	3.96e+5	5.27e+5
1000	2.44e+5	2.68e+5	3.31e+6	4.57e+6
2000	9.60e+5	1.02e+6	2.50e+7	3.26e+7
3000	2.19e+6	2.28e+6	8.70e+7	9.64e+7
4000	3.86e+6	3.97e+6	1.94e+8	2.14e+8
5000	5.67e+6	6.33e+6	3.60e+8	4.59e+8
6000	8.80e+6	8.95e+6		
7000	1.19e+7	1.21e+7		
8000	1.56e+7	1.59e+7		
9000	1.98e+7	2.02e+7		
10000	2.44e+7	2.51e+7		

Table 4.2: Market characteristics.

Theorem 4.3.1. *Given a graph $G = (V, E)$ and an integer $L \geq 3$, the problem of deciding if G admits a perfect cycle cover containing cycles of length at most L is NP-complete.*

Proof. It is clear that this problem is in NP. For NP-hardness, we reduce from 3D-Matching, which is the problem of, given disjoint sets X, Y and Z of size q, and a set of triples $T \subseteq X \times Y \times Z$, deciding if there is a disjoint subset M of T with size q.

One straightforward idea is to construct a tripartite graph with vertex sets $X \cup Y \cup Z$ and directed edges (x_a, y_b), (y_b, z_c), and (z_c, x_a) for each triple $t_i = \{x_a, y_b, z_c\} \in T$. However, it is not too hard to see that this encoding fails because a perfect cycle cover may include a cycle with no corresponding triple.

Instead then, we use the following reduction. Given an instance of 3D-Matching, construct one vertex for each element in X, Y and Z. For each triple, $t_i = \{x_a, y_b, z_c\}$ construct the gadget in Figure 4.2, which is a similar to one in Garey and Johnson [32, pp 68-69]. Note that the gadgets intersect only on vertices in $X \cup Y \cup Z$. It is clear that this construction can be done in polynomial time.

Let M be a perfect 3D-Matching. We will show the construction admits a perfect cycle cover by short cycles. If $t_i = \{x_a, y_b, z_c\} \in M$, add from t_i's gadget the three length-L cycles containing x_a, y_b and z_c respectively. Also add the cycle $\langle x_a^i, y_b^i, z_c^i \rangle$. Otherwise, if $t_i \notin M$, add the three length-L cycles containing x_a^i, y_b^i and z_c^i respectively. It is clear that all vertices are covered, since M partitions $X \times Y \times Z$.

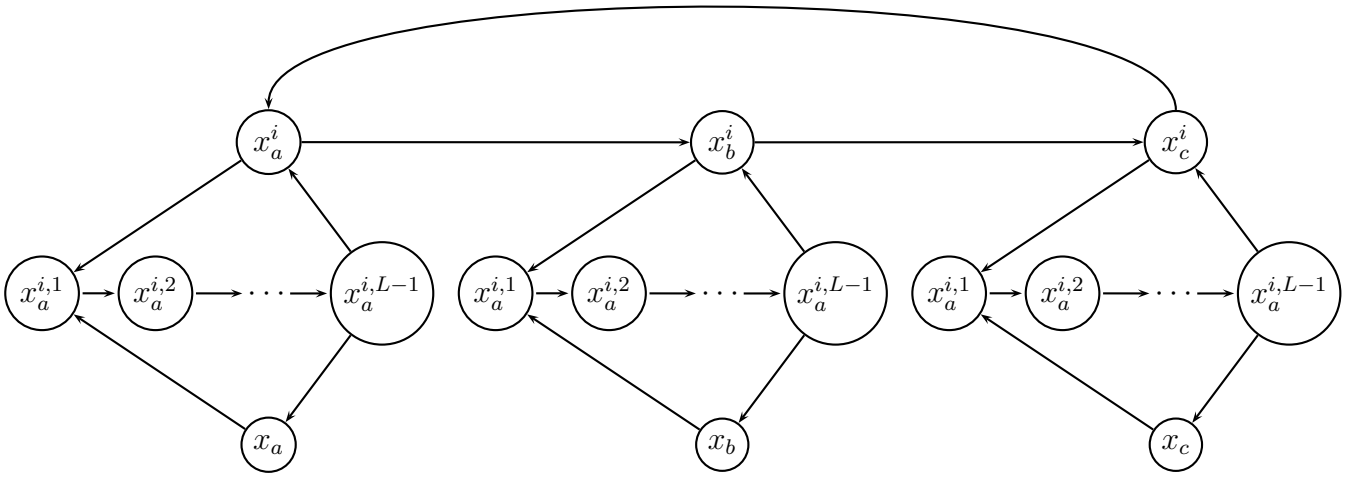


Figure 4.2: NP-completeness gadget for triple t_i and maximum cycle length L.

Conversely, suppose we have a perfect cover by cycles of length at most L. Note that the construction only has short cycles of lengths 3 and L, and no short cycle involves distinct vertices from two different gadgets. It is easy to see then that in a perfect cover, each gadget t_i contributes cycles according to the cases above: $t_i \in M$, or $t_i \notin M$. Hence, there exists a perfect 3D-Matching in the original instance. □

4.4 Solution Approaches Based on an Edge Formulation

In this section, we consider a formulation of the clearing problem as an ILP with one variable x_e for each edge e. This encoding is based on the following classical algorithm for solving the directed cycle cover problem with no cycle-length constraints.

Given a market $G = (V, E)$, construct a bipartite graph with one vertex for each agent, and one vertex for each item. Add an edge e_v with weight 0 between each agent v and its own item. At this point, the encoding is a perfect matching. Now, for each edge $e = (v_i, v_j)$ in the original market, add an edge e with weight w_e between agent v_i and the item of v_j. Perfect matchings in this encoding correspond exactly with cycle covers, since whenever an agent's item is taken, it must receive some other agent's item. It follows that the unrestricted clearing problem can be solved in polynomial time by finding a maximum-weight perfect matching. Figure 4.3 contains the bipartite graph encoding of the example market from

Figure 4.1. The weight-0 edges are encoded by dashed lines, while the market edges are in bold.

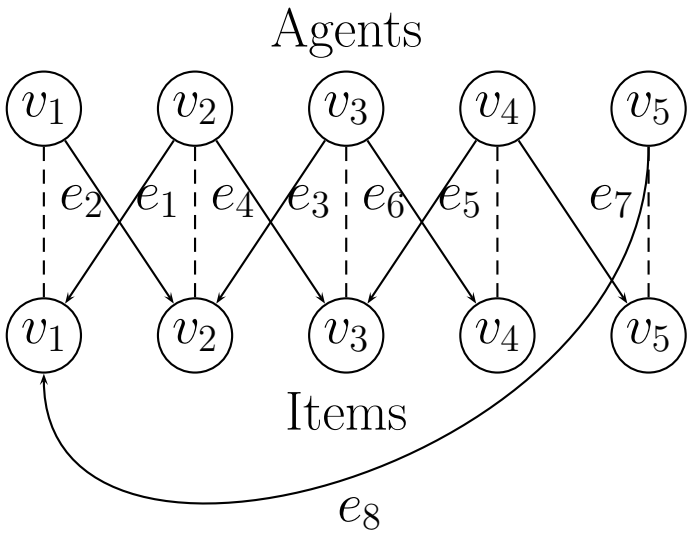


Figure 4.3: Perfect matching encoding of the market in Figure 4.1.

Alternatively, we can solve the problem by encoding it as an ILP with one variable for each edge in the *original market graph* G. This ILP, given below, has the advantage that it can be extended naturally to deal with cycle length constraints. Therefore, for the rest of this section, this is the approach we will pursue.

$$\begin{aligned} &\max \sum_{e \in E} w_e x_e \\ &\text{subject to} \sum_{e=(v_i,v_j)} x_e - \sum_{f=(v_j,v_i)} x_f = 0 \quad \forall v_i \in V(\mathit{conservation}) \\ &\qquad \sum_{e=(v_i,v_j)} x_e \leq 1 \quad \forall v_i \in V(\mathit{capacity}) \\ &\text{with } x_e \in \{0,1\} \quad \forall e \in E \end{aligned}$$

We can enforce the length-L short cycle constraint by adapting the ILP in the following way:

For each length-L path[2] $p = \langle e_1, e_2, \ldots, e_L \rangle$, add the constraint

$$x_1 + x_2 + \ldots + x_L \leq L - 1$$

Note that all cycles with length more than L contain a length-L path. Also, no cycle of length at most L contains a path of length more than $L - 1$. Hence, the constraints above forbid long cycles from being included in a feasible solution without precluding legal short cycles.

Unfortunately, in a market with only 1000 patients, the number of length-3 paths is in excess of 400 million, and so we cannot even construct this ILP without running out of memory.

Therefore, we use a tree search with an *incremental formulation* approach. Specifically, we use CPLEX, though we add constraints as cutting planes during the tree search process. We begin with only a small subset of the path constraints in the ILP. Since this ILP is small, CPLEX can solve its LP relaxation. We then check whether any of the missing constraints are violated by the fractional solution. If so, we generate a set of these constraints, add them to the ILP, and repeat. Even once all constraints are satisfied, there may be no integral solution matching the fractional upper bound, and even if there were, the LP solver might not find it.

In these cases, CPLEX branches on a variable (we used CPLEX's default branching strategy), and generates one new search node corresponding to each of the children. At each node of the search tree that is visited, this process of solving the LP and adding constraints is repeated. Clearly, this approach yields an optimal solution once the tree search finishes.

We still need to explain the details of the constraint seeder (i.e., selecting which constraints to begin with) and the constraint generation (i.e., selecting which violated constraints to include). We describe these briefly in the next two subsections, respectively.

4.4.1 Constraint Seeder

The main constraint seeder we developed forbids any path of length $L-1$ that does not have an edge closing the cycle from its head to its tail. While it is computationally expensive to find these constraints, their addition focuses the search away from paths that cannot be in the final solution. We also tried seeding the LP with a random collection of constraints from the ILP.

[2]Throughout the chapter, we follow the convention that paths are *simple*, meaning they have no repeated vertices.

4.4.2 Constraint Generation

We experimented with several constraint generators. In each, given a fractional solution, we construct the subgraph of edges with positive value. This graph is much smaller than the original graph, so we can perform the following computations efficiently.

In our first constraint generator, we simply search for length-L paths with value sum more than $L-1$. For any such path, we restrict its sum to be at most $L-1$. Note that if there is a cycle c with length $|c| > L$, it could contain as many as $|c|$ violating paths.

In our second constraint generator, we only add one constraint for such cycles: the sum of edge variable values for edges in the cycle can be at most $\lfloor |c|(L-1)/L \rfloor$.

This generator made the algorithm slower, so we went in the other direction in developing our final generator. It adds one constraint per violating path p, and furthermore, it adds a constraint for each path with the same interior vertices (not counting the endpoints) as p. This improved the overall speed.

4.4.3 Experimental performance

It turned out that even with these improvements, the edge formulation approach cannot clear a kidney exchange with 100 vertices in the time the cycle formulation (described later in Section 4.5) can clear one with 10,000 vertices. In other words, column generation based approaches appear to be drastically better than constraint generation based approaches. Therefore, in the rest of this work, we will focus on the cycle formulation and the column generation based approaches.

4.5 Solution Approaches Based on a Cycle Formulation

In this section, we consider a formulation of the clearing problem as an ILP with one variable x_c for each cycle c. This encoding is based on the following classical algorithm for solving the directed cycle cover problem when cycles have length 2.

Given a market $G = (V, E)$, construct a new graph on V with a weight w_c edge for each cycle c of length 2. It is easy to see that matchings in this new graph correspond to cycle

covers by length-2 cycles in the original market graph. Hence, the market clearing problem with $L = 2$ can be solved in polynomial time by finding a maximum-weight matching.

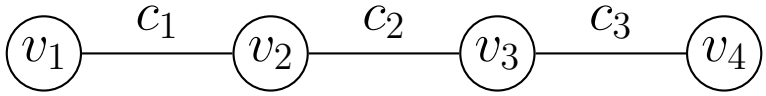


Figure 4.4: Maximum-weight matching encoding of the market in Figure 4.1.

We can generalize this encoding for arbitrary L. Let $C(L)$ be the set of all cycles of G with length at most L. Also, let $c : v_i \in c$ be the set of all cycles in $C(L)$ that cover v_i. Then the following ILP finds the maximum-weight cycle cover by $C(L)$ cycles:

$$\begin{aligned} &\max \sum_{c \in C(L)} w_c x_c \\ &\text{subject to } \sum_{c : v_i \in c} x_c \leq 1 \qquad \forall v_i \in V \\ &\text{with } x_c \in \{0, 1\} \quad \forall c \in C(L) \end{aligned}$$

4.5.1 Edge vs Cycle Formulation

In this section, we consider the merits of the edge formulation and cycle formulation. The edge formulation can be solved in polynomial time when there are no constraints on the cycle size. The cycle formulation can be solved in polynomial time when the cycle size is at most 2.

We now consider the case of short cycles of length at most L, where $L \geq 3$. Our tree search algorithms use the LP relaxation of these formulations to provide upper bounds on the optimal solution. These bounds help prune subtrees and guide the search in the usual ways.

Theorem 4.5.1. *The LP relaxation of the cycle formulation weakly dominates the LP relaxation of the edge formulation.*

Proof. Consider an optimal solution to the LP relaxation of the cycle formulation. We show how to construct an equivalent solution in the edge formulation. For each edge e in the graph, set its value x_e as the sum of values of all the cycles of which it is a member. Also,

define the value x_v of a vertex v in the same manner. Clearly, under this definition, every edge and vertex has value at most 1. Because of the cycle constraints, the conservation and capacity constraints of the edge encoding are clearly satisfied. It remains to show that none of the path constraints are violated.

Let p be any length-L path in the graph. Since p has $L-1$ interior vertices (not counting the endpoints), the total value sum (defined in the previous paragraph) of these interior vertices is at most $L-1$. Now, for any cycle c of length at most L, the number of edges it has in p, which we denote by $e_c(p)$, is at most the number of interior vertices it has in p, which we denote by $v_c(p)$. Hence, $\sum_{e \in p} x_e = \sum_{c \in C(L)} x_c * e_c(p) \leq \sum_{c \in C(L)} x_c * v_c(p) = \sum_{v \in p} x_v = L - 1$. □

The converse of this theorem is not true. Consider a graph which is simply a cycle with $n > L$ edges. Clearly, the LP relaxation of the cycle formulation has optimal value 0, since there are no cycles of size at most L. However, the edge formulation has a solution of size $n/2$, with each edge having value $1/2$.

Hence, the cycle formulation is tighter than the edge formulation. Additionally, for a graph with m edges, the edge formulation requires $O(m^3)$ constraints, while the cycle formulation requires only $O(m^2)$.

4.5.2 Column Generation for the LP

Table 4.2 shows how the number of cycles of length at most 3 grows with the size of the market. Using the cycle formulation, with one variable per cycle, our CPLEX solver was not able to clear markets with 1,000 patients before running out of memory (see Section 4.6 for a description of our computational environment and the running time graph in Figure 4.6). To address this problem, we used an incremental formulation approach.

The first step in LP-guided tree search is to solve the LP relaxation. Since the cycle formulation does not fit in memory, this LP stage would fail immediately without an incremental formulation approach. However, motivated by the observation that an optimal exchange can include only a tiny fraction of the cycles, we explored the approach of using column (i.e., cycle) generation.

The idea of column generation is to start with a restricted LP containing only a small number of columns (variables, i.e., cycles), and then to repeatedly add columns until an optimal solution to this partially formulated LP is an optimal solution to the original (aka master) LP. We explain this further by way of an example.

Consider the market in Figure 4.1 with $L = 2$. Figure 4.5 gives the corresponding master LP, P, and its dual, D.

PRIMAL P

max	$2c_1$	$+2c_2$	$+2c_3$			
s.t.	c_1			≤ 1	(v_1)
	c_1	$+c_2$		≤ 1	(v_2)
		$+c_2$	$+c_3$	≤ 1	(v_3)
			$+c_3$	≤ 1	(v_4)
with	c_1, c_2, c_3			≥ 0	

DUAL D

min	v_1	$+v_2$	$+v_3$	$+v_4$		
s.t	v_1	$+v_2$			≥ 2	(c_1)
		$+v_2$	$+v_3$		≥ 2	(c_2)
			$+v_3$	$+v_4$	≥ 2	(c_3)
with	v_1, v_2, v_3, v_4				≥ 0	

Figure 4.5: Cycle formulation.

Let P' be the restriction of P containing columns c_1 and c_3 only. Let D' be the dual of P', that is, D' is just D without the constraint c_2. Because P' and D' are small, we can solve them to obtain $OPT(P') = OPT(D') = 4$, with $c_{OPT(P')} = \langle c_1 = c_3 = 1 \rangle$ and $v_{OPT(D')} = \langle v_1 = v_2 = v_3 = v_4 = 1 \rangle$.

While $c_{OPT(P')}$ must be a feasible solution of P, it turns out (fortunately) that $v_{OPT(D')}$ is feasible for D, so that $OPT(D') \geq OPT(D)$. We can verify this by checking that $v_{OPT(D')}$ satisfies the constraints of D not already in D' — i.e. constraint c_2. It follows that $OPT(P') = OPT(D') \geq OPT(D) = OPT(P)$, and so $v_{OPT(P')}$ is provably an optimal solution for P, even though P' is contains a only strict subset of the columns of P.

Of course, it may turn out (unfortunately) that $v_{OPT(D')}$ is not feasible for D. This can happen above if $v_{OPT(D')} = \langle v_1 = 2, v_2 = 0, v_3 = 0, v_4 = 2 \rangle$. Although we can still see that $OPT(D') = OPT(D)$, in general we cannot *prove* this because D and P are too large to solve. Instead, because constraint c_2 is violated, we add column c_2 to P', update D', and repeat. The problem of finding a violated constraint is called the *pricing* problem. Here, the *price* of a column (cycle in our setting) is the difference between its weight, and the dual-value sum of the cycle's vertices. If any column of P has a positive price, its corresponding constraint is violated and we have not yet proven optimality. In this case, we must continue generating columns to add to P'.

Pricing Problem

For smaller instances, we can maintain an explicit collection of all feasible cycles. This makes the pricing problem easy and efficient to solve: we simply traverse the collection of cycles, and look for cycles with positive price. We can even find cycles with the most positive price, which are the ones most likely to improve the objective value of restricted LP [12]. This approach does not scale however. A market with 5000 patients can have as many as 400 million cycles of length at most 3 (see Table 4.2). This is too many cycles to keep in memory.

Hence, for larger instances, we have to *generate* feasible cycles while looking for one with a positive price. We do this using a depth-first search algorithm on the market graph (see Figure 4.1). In order to make this search faster, we explore vertices in non-decreasing value order, as these vertices are more likely to belong to cycles with positive weight. We also use several pruning rules to determine if the current search path can lead to a positive weight cycle. For example, at a given vertex in the search, we can prune based on the fact that every vertex we visit from this point onwards will have value at least as great the current vertex.

Even with these pruning rules, column generation is a bottleneck. Hence, we also implemented the following optimizations.

Whenever the search exhaustively proves that a vertex belongs to no positive price cycle, we mark the vertex and do not use it as the root of a depth-first search until its dual value decreases. In this way, we avoid unnecessarily repeating our computational efforts from a previous column generation iteration.

Finally, it can sometimes be beneficial for column generation to include several positive-price columns in one iteration, since it may be faster to generate a second column, once the first one is found. However, we avoid this for the following reason. If we attempt to find more positive-price columns than there are to be found, or if the columns are far apart in the search space, we end up having to generate and check a large part of the collection of feasible cycles. In our experiments, we have seen this occur in markets with hundreds of millions of cycles, resulting in prohibitively expensive computation costs.

Column Seeding

Even if there is only a small gap to the master LP relaxation, column generation requires many iterations to improve the objective value of the restricted LP. Each of these iterations is expensive, as we must solve the pricing problem, and re-solve the restricted LP. Hence,

although we could begin with no columns in the restricted LP, it is much faster to seed the LP with enough columns that the optimal objective value is not too far from the master LP. Of course, we cannot include so many columns that we run out of memory.

We experimented with several column seeders. In one class of seeder, we use a heuristic to find an exchange, and then add the cycles of that exchange to the initial restricted LP. We implemented two heuristics. The first is a greedy algorithm: for each vertex in a random order, if it is uncovered, we attempt to include a cycle containing it and other uncovered vertices. The other heuristic uses specialized maximum-weight matching code [83] to find an optimal cover by length-2 cycles.

These heuristics perform extremely well, especially taking into account the fact that they only add a small number of columns. For example, Table 4.1 shows that the size of an optimal cover by length-2 cycles is almost as the size of optimal cover by unrestricted-length cycles. However, we have enough memory to include hundreds-of-thousands of additional columns and thereby get closer still to the upper bound.

Our best column seeder constructs a random collection of feasible cycles. Since a market with 5000 patients can have as many as 400 million feasible cycles, it takes too long to generate and traverse all feasible cycles, and so we do not include a uniformly random collection. Instead, we perform a random walk on the market graph (see, for example, Figure 4.1), in which, after each step of the walk, we test whether there is an edge back onto our path that forms a feasible cycle. If we find a cycle, it is included in the restricted LP, and we start a new walk from a random vertex. In our experiments (see Section 4.6), we use this algorithm to seed the LP with 400,000 cycles.

This last approach outperforms the heuristic seeders described above. However, in our algorithm, we use a combination that takes the union of all columns from all three seeders. In Figure 4.6, we compare the performance of the combination seeder against the combination without the random collection seeder. We do not plot the performance of the algorithm without any seeder at all, because it can take hours to clear markets we can otherwise clear in a few minutes.

Proving Optimality

Recall that our aim is to find an optimal solution to the master LP relaxation. Using column generation, we can prove that a restricted-primal solution is optimal once all columns have non-positive prices. Unfortunately though, our clearing problem has the so-called *tailing-off effect* [12, Section 6.3], in which, even though the restricted primal is optimal in hindsight, a

large number of additional iterations are required in order to *prove* optimality (i.e., eliminate all positive-price columns). There is no good general solution to the tailing-off effect.

However, to mitigate this effect, we take advantage of the following problem-specific observation. Recall from Section 4.1.1 that, almost always, a maximum-weight exchange with cycles of length at most 3 has the same size as an unrestricted maximum-cardinality exchange. (This does not mean that the solver for the unrestricted case will find a solution with short cycles, however.) Furthermore, the unrestricted clearing problem can be solved in polynomial time (recall Section 4.4). Hence, we can efficiently compute an upper bound on the master LP relaxation, and, whenever the restricted primal achieves this upper bound, we have proven optimality without necessarily having to eliminate all positive-price columns!

In order for this to improve the running time of the overall algorithm, we need to be able to clear the unrestricted market in less time than it takes column generation to eliminate all the positive-price cycles. Even though the first problem is polynomial-time solvable, this is not trivial for large instances. For example, for a market with 10,000 patients and 25 million edges, specialized maximum-weight matching code [83] was too slow, and CPLEX ran out of memory on the edge formulation encoding from Section 4.4. To make this idea work then, we used column generation to solve the edge formulation of the unrestricted market (with no cycle length constraints).

This involves starting with a small random subset of the edges, and then adding positive price edges one-by-one until none remain. We conduct this secondary column generation not in the original market graph G, but in the perfect matching bipartite graph of Figure 4.3. We do this so that we only need to solve the LP, not the ILP, since the integrality gap in the perfect matching bipartite graph is 1—i.e. there always exists an integral solution that achieves the fractional upper bound, since the constraint matrix is totally unimodular.

The resulting speedup to the overall algorithm is dramatic, as can be seen in Figure 4.6.

Column Management

If the optimal value of the initial restricted LP P' is far from the the master LP P, then a large number of columns are generated before the gap is closed. This leads to memory problems on markets with as few as 4,000 patients. Also, even before memory becomes an issue, the column generation iterations become slow because of the additional overhead of solving a larger LP.

To address these issues, we implemented a column management scheme to limit the size of

the restricted LP. Whenever we add columns to the LP, we check to see if it contains more than a threshold number of columns. If this is the case, we selectively remove columns until it is again below the threshold[3]. As we discussed earlier, only a tiny fraction of all the cycles will end up in the final solution. It is unlikely that we delete such a cycle, and even if we do, it can always be generated again. Of course, we must not be too aggressive with the threshold, because doing so may offset the per-iteration performance gains by significantly increasing the number of iterations required to get a suitable column set in the LP at the same time.

There are some columns we never delete, for example those we have branched on (see Section 4.5.3), or those with a non-zero LP value. Amongst the rest, we delete those with the lowest price, since those correspond to the dual constraints that are *most* satisfied. This column management scheme works well and has enabled us to clear markets with 10,000 patients, as seen in Figure 4.6.

4.5.3 Branch-and-Price Search for the ILP

Given a large market clearing problem, we can successfully solve its LP relaxation to optimality by using the column generation enhancements described above. However, the solutions we find are usually fractional. Thus the next step involves performing a *branch-and-price* tree search [12] to find an optimal integral solution.

Briefly, this is the idea of branch-and-price. Whenever we set a fractional variable to 0 or 1 (branch), both the master LP, and the restriction we are working with, are changed (constrained). By default then, we need to perform column generation (go through the effort of pricing) at each node of the search tree to prove that the constrained restriction is optimal for the constrained master LP. (However, as discussed in Section 4.5.2, we compute the integral upper bound for the root node based on relaxing the cycle length constraint completely, and whenever any node's LP in the tree achieves that value, we do not need to continue pricing columns at that node.)

For the clearing problem with cycles of length at most 3, we have found that there is rarely a gap between the optimal integral and fractional solutions. This means we can largely avoid the expensive per node pricing step: whenever the constrained restricted LP has the same optimal value as its parent in the tree search, we can prove LP optimality, as in Section 4.5.2, without having to include any additional columns in the restricted LP.

[3]Based on memory size, we set the threshold at 400,000.

Although CPLEX can solve ILPs, it does not support branch-and-price (for example, because there can be problem-specific complications involving the interaction between the branching rule and the pricing problem). Hence, we implemented our own branch-and-price algorithm, which explores the search tree in depth-first order. We also experimented with the A* node selection order [20, 39]. However, this search strategy requires significantly more memory, which we found was better employed in making the column generation phase faster (see Section 4.5.2). The remaining major components of the algorithm are described in the next two subsections.

Primal Heuristics

Before branching on a fractional variable, we use primal heuristics to construct a feasible integral solution. These solutions are lower bounds on the final optimal integral solutions. Hence, whenever a restricted fractional solution is no better than the best integral solution found so far, we prune the current subtree. A primal heuristic is effective if it is efficient and constructs tight lower bounds.

We experimented with two primal heuristics. The first is a simple rounding algorithm [40]: include all cycles with fractional value at least 0.5, and then, ensuring feasibility, greedily add the remaining cycles. Whilst this heuristic is efficient, we found that the lower bounds it constructs rarely enable much pruning.

We also tried using CPLEX as a primal heuristic. At any given node of the search tree, we can convert the restricted LP relaxation back to an ILP by reintroducing the integrality constraints. CPLEX has several built-in primal heuristics, which we can apply to this ILP. Moreover, we can use CPLEX's own tree search to find an optimal integral solution. In general, this tree search is much faster than our own.

If CPLEX finds an integral solution that matches the fractional upper bound at the root node, we are done. Otherwise, no such integral solution exists, or we don't yet have the right combination of cycles in the restricted LP. For kidney-exchange markets, it is usually the second reason that applies (see Section 4.5). Hence, at some point in the tree search, once more columns have been generated as a result of branching, the CPLEX heuristic will find an optimal integral solution.

Although CPLEX tree search is faster than our own, it is not so fast that we can apply it to every node in our search tree. Hence, we make the following optimizations. Firstly, we add a constraint that requires the objective value of the ILP to be as large as the fractional target. If this is not the case, we want to abort and proceed to generate more columns with

our branch-and-price search. Secondly, we limit the number of nodes in CPLEX's search tree. This is because we have observed that no integral solution exists, CPLEX can take a very long time to prove that. Finally, we only apply the CPLEX heuristic at a node if it has a sufficiently different set of cycles from its parent.

Using CPLEX as a primal heuristic has a large impact because it makes the search tree smaller, so all the computationally expensive pricing work is avoided at nodes that are not generated in this smaller tree.

Cycle Brancher

We experimented with two branching strategies, both of which select one variable per node. The first strategy, branching by certainty, randomly selects a variable from those whose LP value is closest to 1. The second strategy, branching by uncertainty, randomly selects a variable whose LP value is closest to 0.5. In either case, two children of the node are generated corresponding to two subtrees, one in which the variable is set to 0, the other in which it is set to 1. Our depth-first search always chooses to explore first the subtree in which the value of the variable is closest to its fractional value.

For our clearing problem with cycles of length at most 3, we found branching by uncertainty to be superior, rarely requiring any backtracking.

4.6 Experimental Results

All our experiments were performed in Linux (Red Hat 9.0), using a Dell PC with a 3GHz Intel Pentium 4 processor, and 1GB of RAM. Wherever we used CPLEX (e.g., in solving the LP and as a primal heuristic, as discussed in the previous sections), we used CPLEX 10.010.

Figure 4.6 shows the runtime performance of four clearing algorithms. For each market size listed, we randomly generated 10 markets, and attempted to clear them using each of the algorithms.

The first algorithm is CPLEX on the full cycle formulation. This algorithm fails to clear any markets with 1000 patients or more. Also, its running time on markets smaller than this is significantly worse than the other algorithms.

The other algorithms are variations of the incremental column generation approach described

in Section 4.5. We begin with the algorithm settings described in Table 4.3 (all optimizations are switched on):

Category	Setting
Column Seeder	Combination of greedy exchange and maximum-weight matching heuristics, and random walk seeder (400,000 cycles).
Column Generation	One column at a time.
Column Management	On, with 400,000 column limit.
Optimality Prover	On.
Primal Heuristic	Rounding & CPLEX tree search.
Branching Rule	Uncertainty.

Table 4.3: Default Algorithm Configuration

The combination of these optimizations allows us to easily clear markets with over 10,000 patients. In each of the next two algorithms, we turn one of these optimizations off to highlight its effectiveness.

First, we restrict the seeder so that it only begins with 10,000 cycles. This setting is faster for smaller instances, since the LP relaxations are smaller, and faster to solve. However, at 5000 vertices, this effect starts to be offset by the additional column generation that must be performed. For larger instance, this restricted seeder is clearly worse.

Finally, we restore the seeder to its optimized setting, but this time, remove the optimality prover described in Section 4.5.2. As in many column generation problems, the tailing-off effect is substantial. By taking advantage of the properties of our problem, we manage to clear a market with 10,000 patients in about the same time it would otherwise have taken to clear a 6000 patient market.

4.7 Conclusion

In this work we have developed a scalable, exact algorithm for barter exchange with short cycles, focusing in particular on the upcoming national kidney-exchange market in which patients with kidney disease will be matched with compatible donors by swapping their own willing but incompatible donors. With over 70,000 patients already waiting for a cadaver kidney in the US, this market is seen as the only ethical way to significantly reduce the 4,000

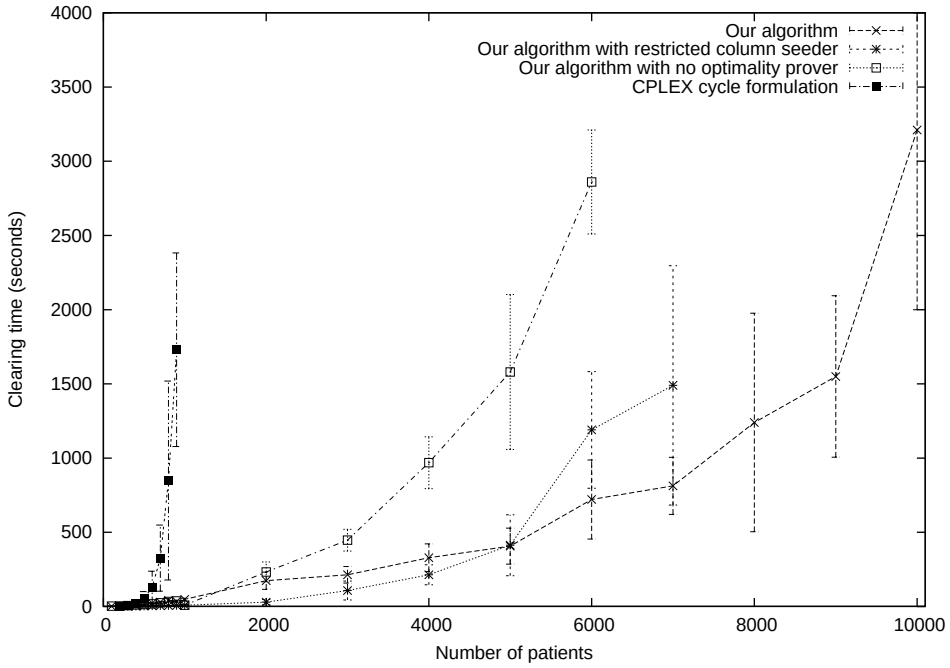


Figure 4.6: Experimental results: average runtime with standard deviation bars.

deaths per year attributed to kidney disease.

Our work presents the first algorithm capable of clearing these markets on a nationwide scale. It optimally solves the kidney exchange clearing problem with 10,000 donor-donee pairs. Thus there is no need to resort to approximate solutions. The best prior solution, which uses CPLEX on the standard column formulation, cannot clear instances with more than 1000 donor-donee pairs because it runs out of memory. The key to our improvement is *incremental problem formulation*. We adapted two paradigms for the task: constraint generation and column generation. For each, we developed a host of techniques that substantially improve both runtime and memory usage. Some of the techniques use domain-specific observations while others are domain independent. We conclude that column generation scales dramatically better than constraint generation. For column generation in the LP, our enhancements include pricing techniques, column seeding techniques, techniques for proving optimality without having to bring in all positive-price columns (and using another column-generation process in a different formulation to do so), and column removal techniques. For the branch-and-price search in the integer program that surrounds the LP, our enhancements include primal heuristics and we also compared branching strategies. Undoubtedly, further parameter tuning and perhaps additional speed improvement techniques could be used to make the algorithm even faster.

Our algorithm also supports several generalizations, as desired by real-world kidney exchanges. These include multiple alternative donors per patient, weighted edges in the market graph (to encode differences in expected life years added based on degrees of compatibility, patient age and weight, etc., as well as the probability of last-minute incompatibility), "angel-triggered chains" (chains of transplants triggered by altruistic donors who do not have patients associated with them, each chain ending with a left-over kidney), and additional issues (such as different scores for saving different altruistic donors or left-over kidneys for future match runs based on blood type, tissue type, and likelihood that the organ would not disappear from the market by the donor getting second thoughts). Because we use an ILP methodology, we can also support a variety of side constraints, which often play an important role in markets in practice [87]. We can also support forcing part of the allocation, for example, "This acutely sick teenager has to get a kidney if possible."

Acknowledgments

We thank economists Al Roth and Utku Unver, as well as kidney transplant surgeon Michael Rees, for introducing us to the clearing problem, supplying initial data sets, and discussions on details of the kidney exchange process. We also thank Don Sheehy for bringing to our attention the idea of shoe exchange.

Chapter 5

The Stable Roommates Problem with Globally-Ranked Pairs

Declaration

The material in this chapter is joint work with Ariel Levavi, David F.Manlove and Gregg O'Malley. The exposition is based on the paper describing our results [7].

Abstract

We introduce a restriction of the stable roommates problem in which roommate pairs are ranked globally. In contrast to the unrestricted problem, weakly stable matchings are guaranteed to exist, and additionally, can be found in polynomial time. However, it is still the case that strongly stable matchings may not exist, and so we consider the complexity of finding weakly stable matchings with various desirable properties. In particular, we present a polynomial-time algorithm to find a rank-maximal (weakly stable) matching. This is the first generalization of the algorithm due to Irving et al. [48] to a non-bipartite setting. Also, we describe several hardness results in an even more restricted setting for each of the problems of finding weakly stable matchings that are of maximum size, are egalitarian, have minimum regret, and admit the minimum number of weakly blocking pairs.

5.1 Introduction

The STABLE ROOMMATES problem (SR) [29, 37, 45, 46] involves pairing-up a set of *agents*, each of whom ranks the others in (not necessarily strict) order of preference. Agents can declare each other *unacceptable*, in which case they cannot be paired together. Our task is to find a pairing of mutually acceptable agents such that no two agents would prefer to partner each other over those that we prescribed for them.

We represent acceptable pairs by a graph $G = (V, E)$, with one vertex $u \in V$ for each agent, and an edge $\{u, v\} \in E$ whenever agents u and v are mutually acceptable. A pairing is just a *matching* M of G, i.e. a subset of edges in E, no two of which share a vertex. If $\{u, v\} \in M$, we say that u is *matched* in M and $M(u)$ denotes v, otherwise u is *unmatched* in M. An agent u *prefers* one matching M' over another M if i) u is matched in M' and unmatched in M, or ii) u prefers $M'(u)$ to $M(u)$. Similarly, u is *indifferent between* M' and M if i) u is unmatched in M' and M, or ii) u is matched in M and M', and u is indifferent between $M'(u)$ and $M(u)$.

A matching M is *weakly stable* if it admits no *strongly blocking pair*, which is an edge $\{u, v\} \in E \backslash M$ such that u and v prefer $\{\{u, v\}\}$ to M. A matching M is *strongly stable* if it admits no *weakly blocking pair*, which is an edge $\{u, v\} \in E \backslash M$ such that u prefers $\{\{u, v\}\}$ to M, while v either prefers $\{\{u, v\}\}$ to M, or is indifferent between them. Finally, a matching is *super stable* if it admits no edge $\{u, v\} \in E \backslash M$ such that i) u either prefers $\{\{u, v\}\}$ to M, or is indifferent between them, and ii) v also prefers $\{\{u, v\}\}$ to M, or is indifferent between them.

In this work, we introduce and study the STABLE ROOMMATES WITH GLOBALLY-RANKED PAIRS problem (SR-GRP). An instance of SR-GRP is a restriction of SR in which preferences may be derived from a ranking function $rank : E \to \mathbb{R}$. An agent u *prefers* v to w if $rank(e) < rank(e')$, where $e = \{u, v\}$ and $e' = \{u, w\}$. Similarly, agent u *is indifferent between* v and w if $rank(e) = rank(e')$.

Before giving our motivation for studying this restriction, we introduce some additional notation. We define E_i to be the set of edges with rank i, and $E_{\leq i}$ to be the set $E_1 \cup E_2 \cup \ldots \cup E_i$. Additionally, let $n = |V|$ be the number of agents, $m = |E|$ be the number of mutually acceptable pairs. Without loss of generality, we assume the maximum edge rank is at most m. Also, we make the standard assumption in the study of stable marriage problems that the adjacency list for a vertex is given in order of preference/rank.

5.1.1 Motivation

In several real-world settings, agents have restricted preferences that can be represented by the SR-GRP model. A pairwise kidney exchange market [4, 78, 80] is one such setting. Here, patients with terminal kidney-disease obtain compatible donors by swapping their own willing but incompatible donors. We can model the basic market by constructing one vertex for each patient, and an undirected edge between any two patients where the incompatible donor for one patient is compatible with the other patient, and vice versa. Of course, patients may have different preferences over donors. However, since the expected years of life gained from a transplant is similar amongst all compatible kidneys, the medical community has suggested that patient preferences should be *binary/dichotomous* [21, 34] – i.e., patients are indifferent between all compatible donors. Binary preferences are easily modelled in SR-GRP by giving all edges the same rank.

A second example also comes from pairwise kidney exchange markets. When two (patient,donor) pairs are matched with each other (in order to swap donors), we are not certain if the swap can occur until expensive last-minute compatibility tests are performed on the donors and patients. If either potential transplant in the swap is incompatible, the swap is cancelled and the two patients must wait for a future match run. Note that the probability of a swap being compatible is the same for both patients involved in the swap. Also, doctors can predict the probability of a swap being compatible. Hence patient preferences can be inferred by ranking the potential swaps by their chance of success, which is exactly the preference model of SR-GRP.

A third application arises in P2P networks [27, 28, 55, 61, 62]. For example [55], in a P2P file-sharing network, a given peer may form a preference list over other peers based on the similarity of their interests. In cooperative download applications such BitTorrent, preference functions may be derived from properties such as download / upload bandwidth, latency and storage capacity. The "Tit-for-Tat" strategy of BitTorrent can give rise to preference lists for peers that are based on a single global ranking (referred to as a *master list* in [47, 73]) of peers according to upload capacity. The presence of a global ranking of peers (as opposed to edges) gives rise to a special case of SR-GRP (this can be seen as follows: an SR-GRP instance can be obtained by defining the rank of an edge $\{u, v\}$ to be $rank(u) + rank(v)$, where $rank(w)$ is the rank of agent w in the master list). In fact, peers can have non-unitary capacity in general, so this application actually motivates a many-many variant of SR-GRP, called the stable b-matching problem [17] with globally-ranked pairs.

One final real-world setting is described in [11]. When colleges pair-up freshmen roommates, it is not feasible for students to rank each other explicitly. Instead, each student submits a form which describes him/herself in several different dimensions (e.g., bedtime preference,

cleanliness preference etc). Students can then be represented as points in a multidimensional space, and preferences over other students can be inferred by a distance function. Note that this model [11] is a restriction of SR-GRP in that it is not possible to declare another student unacceptable.

5.1.2 Preliminary Results

In order to highlight the generality of the SR-GRP model, we introduce a second restriction of SR called STABLE ROOMMATES WITH GLOBALLY-ACYCLIC PREFERENCES (SR-GAP). Instances of SR-GAP satisfy the following characterization test: given an arbitrary instance I of SR with $G = (V, E)$, construct a digraph $P(G)$, containing one vertex e for each edge in $e \in E$, and an arc from $e = \{u, v\} \in E$ to $e' = \{u, w\} \in E$ if u prefers w to v. Now, for each $e = \{u, v\}$ and $e' = \{u, w\}$ in E, if u is indifferent between v and w, merge vertices e and e'. Note that a merged vertex may contain several original edge-vertices and have self-loops. Instance I belongs to SR-GAP if and only if $P(G)$ is acyclic.

Instances of SR-GRP satisfy the SR-GAP test, since any directed path in $P(G)$ consists of arcs with monotonically improving ranks, and so no cycles are possible. In the reverse direction, given any instance of SR-GAP, we can derive a suitable rank function from a reverse topological sort on $P(G)$, i.e. $rank(e) < rank(e')$ if and only if e appears before e'. The following proposition is clear:

Proposition 5.1.1. *Let I be an instance of* SR. *Then I is an instance of* SR-GRP *if and only if I is an instance of* SR-GAP.

As well as modelling real-world problems, SR-GRP is an important theoretical restriction of SR. It is well-known that SR has two key undesirable properties. First, some instances of SR admit no weakly stable matchings (see, for example, [37, page 164]). And second, the problem of finding a weakly stable matching, or proving that no such matching exists, is NP-hard [46, 76]. It turns out that SR-GRP has neither of these undesirable properties.

Lemma 5.1.2. *Let $G = (V, E_1 \cup \ldots \cup E_m)$ be an instance of* SR-GRP. *Then M is a weakly stable matching of G if and only if $M \cap E_{\leq i}$ is a maximal matching of $E_{\leq i}$, for all i.*

Proof. Let M be a matching that is not weakly stable, and suppose for a contradiction that $M \cap E_{\leq i}$ is a maximal matching of $E_{\leq i}$ for all i. Since M is not weakly stable, it admits a blocking pair $e = \{u, v\}$ with some rank, say r. It follows that neither u nor v are matched on $M \cap E_{\leq r}$ and so $M \cap E_{\leq r}$ is not maximal on $E_{\leq r}$, a contradiction.

Let M be a weakly stable matching and suppose for a contradiction that r is the first rank for which $M \cap E_{\leq r}$ is not maximal on $E_{\leq r}$. Since $M \cap E_{\leq r}$ is not maximal, let $e = \{u, v\}$ be some edge in E_r that could be added to $M \cap E_{\leq r}$ so that $(M \cup e) \cap \mathcal{E}_{\leq r}$ is a legal matching. Since neither u nor v are matched by $M \cap E_{\leq r}$, and $\{u, v\} \in E_r$, it follows that $\{u, v\}$ blocks M, giving the required contradiction. □

So we can construct a weakly stable matching in $O(n + m)$ time by finding a maximal matching on rank-1 edges, removing the matched vertices, finding a maximal matching on rank-2 edges, and so on.

Strongly stable matchings are also easy to characterize in SR-GRP.

Lemma 5.1.3. *Let $G = (V, E_1 \cup \ldots \cup E_m)$ be an instance of* SR-GRP. *Then M is a strongly stable matching of G if and only if $M \cap E_i$ is a perfect matching of $\{e \in E_i : e$ is not adjacent to any $e' \in M \cap E_{<i}\}$, for all i.*

Proof. Let M be a matching such that $M \cap E_i$ is a perfect matching of $\{e \in E_i : e$ is not adjacent to any $e' \in M \cap E_{<i}\}$ for all i. Suppose for a contradiction that M is not strongly stable. It follows that M admits a blocking pair $\{u, v\}$, where, without loss of generality, we assume u prefers v to $M(u)$ and v either prefers u to $M(v)$ or is indifferent between them. Let $r = rank(\{u, v\})$ and consider $\{e \in E_r : e$ is not adjacent to any $e' \in M \cap E_{<r}\}$. Note that v is not incident on an edge in $E_{<r}$, otherwise v is matched in $M \cap E_{<r}$ and prefers $M(v)$ to u. Also, u is unmatched in $E_{\leq r}$, since u prefers v to $M(u)$. Hence, $\{u, v\} \in \{e \in E_r : e$ is not adjacent to any $e' \in M \cap E_{<r}\}$ and so $M \cap E_r$ is not a perfect matching, giving the required contradiction.

Let M be a strongly stable matching and suppose for a contradiction that r is the first rank for which $M \cap E_r$ is not a perfect matching of $\{e \in E_r : e$ is not adjacent to any $e' \in M \cap E_{<r}\}$. Since $M \cap E_r$ is not perfect, it leaves some vertex u unmatched, where $\{u, v\} \in E_r$, and both u and v are unmatched in $M \cap E_{<r}$. If v is also unmatched in $M \cap E_r$, then $\{u, v\}$ is a strongly blocking pair of M, giving the required contradiction. Alternative, if v is matched in $M \cap E_r$, then $\{u, v\}$ is a weakly blocking pair, contradicting the assumption that M is strongly stable. □

Of course, even E_1 may not admit a perfect matching, and so strongly stable matchings may not exist. However, we can find a strongly stable matching, or prove that no such matching exists in $O(m\sqrt{n})$ time by using the maximum matching algorithm of Micali and Vazirani for non-bipartite graphs [67] in place of the maximal matching algorithm for finding a weakly

stable matching above. This improves on the best known running time of $O(m^2)$ for general SR [88].

Lemmas 5.1.2 and 5.1.3 indicate that SR-GRP can be "more tractable" than SR. However, the possible non-existence of a strongly stable matching motivates the search for weakly stable matchings with desirable properties. A *rank-maximal* matching [48, 53] includes the maximum possible number of rank-1 edges, and subject to this, the maximum possible number of rank-2 edges, and so on. More formally, define the *signature* of a matching M as $\langle s_1, s_2, \ldots, s_m \rangle$, where s_i is the number of rank-i edges in M. Then a matching is rank-maximal if and only if it has the lexicographic-maximal signature amongst all matchings.

Recall from Lemma 5.1.3 that a strongly stable matching is perfect on rank-1 edges, and subject to removing the matched vertices, perfect on rank-2 edges, and so on. It is clear that a rank-maximal matching is strongly stable, when strong stability is possible. If no strongly stable matching exists, then a rank-maximal matching, which by Lemma 5.1.2 is always weakly stable, seems a natural substitute. Irving et al. [48] gave an $O(\min(n + R, R\sqrt{n})m)$ algorithm for the problem of finding a rank-maximal matching in a *bipartite* graph, where R is the rank of the worst-ranked edge in the matching.

Other desirable types of weakly stable matchings may be those that have maximum cardinality, are *egalitarian*, are of *minimum regret*, or admit the fewest number of weakly blocking pairs. An egalitarian (respectively minimum regret) weakly stable matching satisfies the property that the sum of the ranks (respectively the maximum rank) of the edges is minimised, taken over all weakly stable matchings. Given a general SR instance I, each of the problems of finding an egalitarian and a minimum regret weakly stable matching is NP-hard [25,58] (in the former case, even if the preference lists are complete and strictly-ordered, and in the latter case, even if the underlying graph is bipartite). However the complexity of the problem of finding a weakly stable matching with the minimum number of weakly blocking pairs in I has, until now, been open.

5.1.3 Previous work

Part of the motivation for this work is the investigation of which problems become more tractable in SR-GRP as compared to SR, and which problems maintain their hardness. Work along these lines has been done before [11,13,18,92]. In particular, in the case of SR instances where preference lists may include ties, Chung [18] shows that the "no odd ring" condition on preferences is sufficient for the existence of a weakly stable matching. The SR-GAP acyclic condition is a restriction of the "no odd ring" condition, in that neither odd nor even rings are permitted.

As previously mentioned, several recent papers have focused on instances of SR-GAP that arise from P2P networks. In particular, Lebedev et al. [55] independently proved Lemma 5.1.2 by showing that every instance of SR-GAP (and hence SR-GRP by Proposition 5.1.1) admits a weakly stable matching. Gai et al. [27] showed that every instance of SR with a master list is an instance of SR-GAP, but the converse need not be true. They also considered instances of SR with *symmetric preferences* (see Section 5.3 for the definition of this concept). See also [28, 61, 62].

Arkin et al. [11] considered the Geometric Stable Roommates problem, which is a restriction of SR-GRP in which the agents are points in $\mathbb{R}^d$, all agents are mutually acceptable, and the ranking function maps a pair of agents to the Euclidean distance between them. In this restricted context Arkin et al. proved the analogues of Lemmas 5.1.2 and 5.1.3. They also provided algorithms for finding egalitarian and super-stable matchings (if they exist).

5.1.4 Chapter Outline

In Section 5.2, we consider rank-maximal matchings, and present the first generalization of Irving et al.'s [48] algorithm to a non-bipartite setting. In Section 5.3, we report on hardness results for each of the problems of finding weakly stable matchings that are of maximum size, are egalitarian, have minimum regret, and admit the minimum number of weakly blocking pairs. We also prove that this last problem is inapproximable within a factor of $n^{1-\varepsilon}$, for any $\varepsilon > 0$, unless P = NP. These hardness results apply even in a restricted version of SR-GRP in which the graph G is bipartite, and (in the first three cases) if an agent v is incident to an edge of rank k, then v is incident to an edge of rank k', for $1 \leq k' \leq k$.

5.2 Rank-Maximal Matching

One obvious way to construct a rank-maximal matching is to find a maximum-weight matching using edge weights that increase exponentially with improving rank. However, with K distinct rank values, Gabow and Tarjan's matching algorithm [26] takes $O(K^2 \sqrt{n\alpha(m,n)\lg n}\ m \lg n)$ time[1], where α is the inverse Ackermann function. As in the bipartite restriction [48], our combinatorial algorithm avoids the problem of exponential-sized edge weights, leading to an improved runtime of $O(\min\{n + R, R\sqrt{n}\}m)$, where $R \leq K$ is the rank of the worst-ranked edge in the matching.

[1]See [65] for an explanation of the K^2 factor.

Let $G_i = (V, E_{\leq i})$. Our algorithm begins by constructing a maximum matching M_1 on G_1. Note that M_1 is rank-maximal on G_1 by definition. Then inductively, given a rank-maximal matching M_{i-1} on G_{i-1}, the algorithm exhaustively augments M_{i-1} with edges from E_i to construct a rank-maximal matching M_i on G_i. In order to ensure rank-maximality, certain types of edges are deleted before augmenting. With these edges deleted, it becomes possible to augment M_{i-1} *arbitrarily*, while still guaranteeing rank-maximality. Hence, we can perform the augmentations using Micali and Vazirani's fast maximum matching algorithm [67]. In the non-bipartite setting, we perform one additional type of edge deletion beyond the bipartite setting. Additionally, we shrink certain components into *supervertices*. Note that this shrinking is separate from any blossom-shrinking [24] that might occur in the maximum matching subroutine.

In order to understand the edge deletions and component shrinking, we begin by examining a slightly different formulation of the Gallai-Edmonds decomposition technique given in Lemma 2.3.4 [56]. Let $G = (V, E)$ be an arbitrary undirected graph. Then V can be partitioned into the following three sets, namely GED-U$[G]$, GED-O$[G]$ and GED-P$[G]$. Vertices in GED-U$[G]$ are *underdemanded*, since they are unmatched in some maximum matching of G. All other vertices that are adjacent to one in GED-U$[G]$ are *overdemanded* and belong to GED-O$[G]$. Finally, all remaining vertices are *perfectly demanded* and belong to GED-P$[G]$. The decomposition lemma gives many useful structural properties of maximum matchings. For example, in every maximum matching, vertices in GED-O$[G]$ are always matched, and their partner is in GED-U$[G]$. Similarly, vertices in GED-P$[G]$ are always matched, though their partners are also in GED-P$[G]$. We will use the properties given in Lemma 5.2.1. Note that in the following lemma, and through the rest of the exposition, the cardinality of a connected component C of a graph is the number of vertices in C.

Lemma 5.2.1 (Gallai-Edmonds Decomposition)**.** *In any maximum matching M of G,*

1. *For all u in* GED-O$[G]$*, $M(u)$ is in* GED-U$[G]$
2. *For all even (cardinality) components C of $G \setminus$* GED-O$[G]$*, i) $C \subseteq$* GED-P$[G]$*, and ii) $M(u)$ is in C, for all u in C*
3. *For all odd (cardinality) components C of $G \setminus$* GED-O$[G]$*, i) $C \subseteq$* GED-U$[G]$*, ii) $M(u)$ is in C, for all u in C except one, say v, and iii) either v is unmatched in M, or $M(v)$ is in* GED-O$[G]$

Consider the first inductive step of the algorithm, in which we are trying to construct a rank-maximal matching M_2 of $G_2 = (V, E_{\leq 2})$, given a maximum matching M_1 of $G_1 =$

(V, E_1). We do not want to *commit* to edges in M_1 at this point, because perhaps no rank-maximal matching on G_2 contains these edges. However, according to different parts of the decomposition lemma above, we can safely *delete* any edge $e = \{u, v\}$ such that either:

(i) $u \in \text{GED-O}[G_1]$ and $v \in \text{GED-O}[G_1] \cup \text{GED-P}[G_1]$ (by part 1), or

(ii) $e \in E_{\geq 2}$, and $u \in \text{GED-O}[G_1] \cup \text{GED-P}[G_1]$ (by parts 1 and 2), or

(iii) $e \in E_{\geq 2}$, and both u and v belong to the same odd component of G_1 (by part 3).

We delete all such edges to ensure they are not subsequently added to the matching when we augment. Note that the third deletion type is required for non-bipartite graphs, since only one vertex in each odd component C is unmatched internally.

After deleting edges in G_1, we *shrink* each odd component C into a supervertex. We define the *root* r of C as the one vertex in C that is unmatched within C. Note that C's supervertex is matched if and only if r is matched. Now, when we add in undeleted edges from $e = \{u, v\} \in E_{\geq 2}$ into the graph, if $u \in C$ and $v \notin C$, we replace e with an edge between v and C's supervertex. Note that during the course of the algorithm, we will be dealing with graphs containing supervertices, which themselves, recursively contain supervertices. In such graphs, we define a *legal* matching to be any collection of independent edges such that in every supervertex, all top-level vertices but the root are matched internally.

To give some intuition for why we shrink odd components, consider the graph in Figure 5.1. The triangle of rank-1 edges is an odd component O (with $\{u, v\}$ matched), and so neither rank-2 edges are deleted. One way to augment this graph is to include the two rank-2 edges and take out the rank-1 $\{u, v\}$ edge. This destroys the rank-maximal matching on G_1. If we shrink the triangle down to a supervertex O however, then O is unmatched, and so $\{O, x\}$ and $\{O, y\}$ are both valid augmenting paths. Note how these augmenting paths can be expanded *inside* the supervertex O by removing and adding one rank-1 edge to end at the root r. This expansion makes the augmenting path legal in the original graph, while not changing the number of matching edges internal to the supervertex.

Figure 5.2 contains pseudocode for our non-bipartite rank-maximal matching algorithm. One aspect that requires more explanation is how we augment M_i in G'_i. The overall approach is to find an augmenting path P while regarding each top-level supervertex in G'_i as a regular vertex. Then for each supervertex C in P, we expand P through C in the following way. Let u be the vertex in C that P enters along an unmatched edge. If u is the root r of C, then C is unmatched, and we can replace C by u in P. Otherwise, $u \neq r$, and either C is unmatched or P leaves C via the matched edge incident on r. In the next lemma, we show

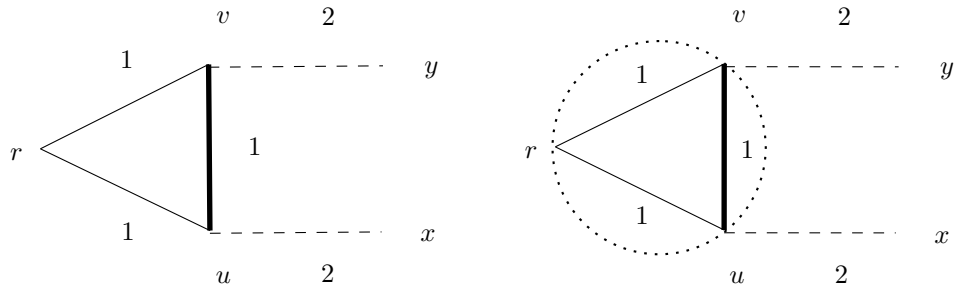


Figure 5.1: Example of shrinking operation

that there is an even-length alternating path from u to r, beginning with a matched edge. We can expand P by replacing C with this even-length alternating path.

Lemma 5.2.2. *Let M be a legal matching on some supervertex C with root r. Let u be any other node in C. Then there is an even-length alternating path from u to r beginning with a matched edge.*

Proof. Let M' be a legal matching of C in which u is unmatched (such a matching is

Rank-Maximal-Matching($G = (V, E_1 \cup E_2 \cup \ldots \cup E_m)$)
Set G'_1 to G_1;
Let M_1 be any maximum matching of G_1;
For $i = 2$ to m:
Set G'_i to G'_{i-1}, and M_i to M_{i-1};
Compute the GED of G'_{i-1} using M_{i-1};
Delete edges in G'_i between two vertices in GED-O$[G'_{i-1}]$;
Delete edges in G'_i between vertices in GED-O$[G'_{i-1}]$ and GED-P$[G'_{i-1}]$;
Delete any edge e in $E_{\geq i}$ where:
i) e is incident on a GED-O$[G'_{i-1}]$ or GED-P$[G'_{i-1}]$ vertex, or
ii) e is incident on two vertices in the same odd component of G_{i-1};
Shrink each odd component of G_{i-1} in the graph G'_i;
Add undeleted edges $\{u, v\}$ from E_i to G'_i, replacing
u or v with their supervertex, if any;
Augment M_i in G'_i until it is a maximum matching;
End For
Return M_m;

Figure 5.2: Non-bipartite rank-maximal matching algorithm

guaranteed by the decomposition lemma). Consider the symmetric difference of M and M'. Since every vertex besides u and r is matched in both matchings, there must be an even-length alternating path consisting of M and M' edges from u to r. □

In all cases of P and C, note that C has the same number of internally matched edges before and after augmentation by P, and so the matching remains legal. Also, if r was matched prior to augmentation, then it is still matched afterwards.

The next three lemmas, which generalize those in [48], establish the correctness of the algorithm. Lemma 5.2.3 proves that no rank-maximal matching contains a deleted edge. Lemma 5.2.4 proves that augmenting a rank-maximal matching M_{i-1} of G_{i-1} does not change its signature up to rank $(i-1)$. And finally, Lemma 5.2.5 proves that the final matching is rank-maximal on the original graph G.

Lemma 5.2.3. *Suppose that every rank-maximal matching of G_{i-1} is a maximum legal matching of G'_{i-1}. Then every rank-maximal matching of G_i is contained in G'_i.*

Proof. Let M be an arbitrary rank-maximal matching of G_i. Then $M \cap E_{\leq i-1}$ is a rank-maximal matching of G_{i-1}, and by assumption, a maximum legal matching of G'_{i-1}. By Lemma 5.2.1, the edges we delete when constructing G'_i belong to no maximum matching of G'_{i-1}, in particular $M \cap E_{\leq i-1}$. Now, since $M = (M \cap E_{\leq i-1}) \cup (M \cap E_i)$, it remains to show that $M \cap E_i$ contains no deleted edges.

Suppose for a contradiction that there is an edge $e \in M \cap E_i$ that is deleted by the algorithm. Note that $e \in E_i$ is deleted only if e is incident on i) a GED-O$[G'_{i-1}]$ vertex, or ii) a GED-P$[G'_{i-1}]$ vertex, or iii) two vertices in the same odd component of G'_{i-1}. In each of these cases, $e \in M \cap E_i$ means that $M \cap E_{\leq i-1}$ cannot be a rank-maximal matching of G_{i-1}, giving the required contradiction. □

Lemma 5.2.4. *Let M_i and M_j be the matchings produced by the algorithm, where $i < j$. Then M_i and M_j have the same number of edges with rank at most i.*

Proof. M_i consists of edges contained within top-level supervertices of G'_i, and edges between top-level (super)vertices of G'_i. We have already shown that augmenting through a supervertex does not change the number of matching edges internal to the supervertex. Hence, M_j contains the same number of such edges as M_i.

By Lemma 5.2.1, the remaining edges of M_i are all incident on some GED-O$[G'_i]$ or GED-P$[G'_i]$ (super)vertex. Since these vertices are matched in M_i, they are also matched in M_j, as augmenting does not affect the matched status of a vertex. Also, no edges of rank worse

than i are incident on such vertices, due to deletions, and so each must be matched along a rank-i edge or better in M_j. Hence $|M_i| \leq |M_j \cap E_{\leq i}|$. Of course, $|M_j \cap E_{\leq i}| \leq |M_i|$, since all edges from $E_{\leq i}$ in G'_j are also in G'_i, and M_i is a maximum legal matching of G'_i. □

Lemma 5.2.5. *For every i, the following statements hold: 1) Every rank-maximal matching of G_i is a maximum legal matching of G'_i, and 2) M_i is a rank-maximal matching of G_i.*

Proof. For the base case, rank-maximal matchings are maximum matchings on rank-1 edges, and so both statements hold for $i = 1$. Now, by Lemma 5.2.3 and the inductive hypothesis, every rank-maximal matching of G_i is contained in G'_i. Let $\langle s_1, s_2, .., s_i \rangle$ be the signature of such a matching. By Lemma 5.2.4, M_i has the same signature as M_{i-1} up to rank-$(i-1)$. Hence, M_i's signature is $\langle s_1, s_2, .., s_{i-1}, t_i \rangle$ for some $t_i \leq s_i$, since M_{i-1} is a rank-maximal matching of G_{i-1}. However, M_i is a maximum legal matching of G'_i, hence $t_i = s_i$ and M_i is rank-maximal matching of G_i. This proves the second statement.

Now, for the first statement, let N_i be any rank-maximal matching of G_i. By Lemma 5.2.3 and the inductive hypothesis, we know that N_i is contained in G'_i. N_i has signature $\langle s_1, s_2, ..., s_i \rangle$, which is the same signature as M_i. Hence, N_i is also a maximum legal matching of G'_i. □

We now comment on the runtime of the algorithm. In each iteration i, it is clear that computing the decomposition (given a maximum matching), deleting edges and shrinking components all take $O(m)$ time. Constructing M_i from M_{i-1} requires $|M_i| - |M_{i-1}| + 1$ augmentations. At the top-level of augmenting (when supervertices are regarded as vertices), we can use the Micali and Vazirani non-bipartite matching algorithm, which runs in time $O(min(\sqrt{n}, |M_{i+1}| - |M_i| + 1)m)$. Next, we have to expand each augmenting path P through its incident supervertices. Let u be the first vertex of some supervertex C that P enters along an unmatched edge. It is clear that we can do this expansion in time linear in the size of C by appending a dummy unmatched vertex d to u, and then looking for an augmenting path from d to r in C. Since each supervertex belongs to at most one augmenting path in each round of the Micali and Vazirani algorithm, this does not affect the asymptotic runtime. It follows that after R iterations, the running time is at most $O(\min(n + R, R\sqrt{n})m)$. Using the idea in [48], we can stop once R is the rank of the worst-ranked edge in a rank-maximal matching, because we can test in $O(m)$ time if M_R is a maximum matching of G_R together with all undeleted edges of rank worse than R (in which case M_R is rank-maximal).

Theorem 5.2.6. *Let R be the rank of the worst-ranked edge in a rank-maximal matching of $G = (V, E_1 \cup \ldots \cup E_m)$. Then a rank-maximal matching of G can be found in time $O(\min(n + R, R\sqrt{n})m)$.*

5.3 Hardness Results

In this section we describe several NP-hardness results for a special case of SR-GRP. We refer to this restriction as STABLE MARRIAGE WITH SYMMETRIC PREFERENCES (SM-SYM). An instance of SM-SYM is an instance of SR in which the underlying graph is bipartite (with *men* and *women* representing the two sets of agents in the bipartition) subject to the restriction that a woman w_j appears in the kth tie in a man m_i's list if and only if m_i appears in the kth tie in w_j's list. Clearly an instance of SM-SYM is a bipartite instance of SR-GRP in which $rank(\{m_i, w_j\}) = k$ if and only if w_j appears in the kth tie in m_i's preference list, for any man m_i and woman w_j. Indeed it will be helpful to assume subsequently that *rank* is defined implicitly in this way, given an instance of SM-SYM.

The first result, due to Manlove and O'Malley [7, 73], demonstrates the NP-completeness of COM-SM-SYM, which is the problem of deciding whether a complete weakly stable matching (i.e. a weakly stable matching in which everyone is matched) exists, given an instance of SM-SYM. The reduction begins from EXACT-MM on subdivision graphs [65], which is the problem of deciding, given a subdivision graph G and an integer K, whether G admits a maximal matching of size K.

Theorem 5.3.1. COM-SM-SYM *is NP-complete, even if each preference list comprises exactly two ties (where a tie can be of length 1) [7, 73].*

The next results, also due to Manlove and O'Malley [7, 73] concern *minimum regret* and *egalitarian* weakly stable matchings in SMC-SYM, which is the restriction of SM-SYM in which each person finds all members of the opposite sex acceptable. Let I be an instance of SMC-SYM and, let U and W be the set of men and women in I respectively. Also, let M be a weakly stable matching in I, and let p be some agent in I. Then we define the *cost* of p with respect to M, denoted by $cost_M(p)$, to be $rank(p, M(p))$. Furthermore we define the *regret* of M, denoted by $r(M)$ to be $\max_{p \in U \cup W} cost_M(p)$. M has *minimum regret* if $r(M)$ is minimised over all weakly stable matchings in I. Similarly we define the *cost* of M, denoted by $c(M)$, to be $\sum_{p \in U \cup W} cost_M(p)$. M is *egalitarian* if $c(M)$ is minimised over all weakly stable matchings in I.

We define REGRET-SMC-SYM (respectively EGAL-SMC-SYM) to be the problem of deciding, given an instance I of SMC-SYM and a positive integer K, whether I admits a weakly stable matching such that $r(M) \leq K$ (respectively $c(M) \leq K$).

Theorem 5.3.2. REGRET-SMC-SYM *and* EGAL-SMC-SYM *are NP-complete [7, 73].*

Our final hardness result, due to Abraham and Manlove, applies to SM-GRP, which is the

restriction of SR-GRP to bipartite graphs. Recall that a strongly stable matching has no weakly blocking pairs. MIN-BP-SM-GRP is the problem of finding a weakly stable matching (which by definition has no strongly blocking pairs) with the minimum number of weakly blocking pairs, given an instance of SM-GRP.

Theorem 5.3.3. MIN-BP-SM-GRP *is not approximate within a factor of* $n^{1-\varepsilon}$*, for any* $\varepsilon > 0$*, unless P=NP, where* n *is the number of men and women.*

Proof. Let $\varepsilon > 0$. We give a gap-introducing reduction from the restriction of COM-SM-SYM in which each person's list has exactly two ties, which is NP-complete by Theorem 5.3.1. Let I be an instance of this problem, where $U = \{m_1, m_2, \ldots, m_n\}$ is the set of men and $W = \{w_1, w_2, \ldots, w_n\}$ is the set of women. For each man $m_i \in U$, we let $W_{i,1}$ and $W_{i,2}$ denote the women in the first and second ties in m_i's preference list in I, respectively. Similarly, for each woman $w_j \in W$, we let $U_{j,1}$ and $U_{j,2}$ denote the men in the the first and second ties in w_j's preference list in I, respectively.

Construct an instance I' of MIN-BP-SM-GRP with $U \cup X$ as the set of men and $W \cup Y$ as the set of women, where $X = \{x_1, x_2, \ldots, x_{n^{2+k}+1}\}$ and $Y = \{y_1, y_2, \ldots, y_{n^{2+k}+1}\}$, where $k = \lceil \frac{4}{\varepsilon} \rceil$. The preferences for these people are given in Figure 5.3. It is clear from Figure 5.3 that I' is an instance of MIN-BP-SM-GRP, and additionally that I' can be constructed in polynomial time.

Suppose I admits a complete weakly stable matching M. Augment M in I' to form M' by pairing-up all people in X and Y. Note that M' is weakly stable with at most n^2 weakly blocking pairs in I', since any weakly blocking pair of M' must also weakly block M in I, and I has at most n^2 acceptable pairs.

Now suppose that I does not admit a complete weakly stable matching. Let M' be a weakly stable matching in I'. We claim that M' admits at least $n^{2+k}+1$ weakly blocking pairs in I'. For, suppose not. Then every man in both U and X must be matched in M', for otherwise he weakly blocks with each of the $n^{2+k}+1$ women in Y, giving a contradiction. It follows that M' must contain a perfect matching from X to Y, since i) every man in X is matched in M', ii) $|X| = |Y|$, and iii) men in X find only women in Y acceptable. Hence, M' contains a perfect matching M from U to W, which is clearly weakly stable in I, a contradiction. Hence the claim is established.

Thus the existence of an approximation algorithm for MIN-BP-SM-GRP with performance guarantee n^k could be used to decide in polynomial time whether I admits a complete weakly stable matching, a contradiction unless P=NP. Finally we note that I' contains a total of N men and women, where $N = 2(n+n^{2+k}+1)$. It follows that $N \leq 6n^{2+k}$ and hence

Men's preferences

	rank-1	rank-2	rank-3	
m_i :	$(W_{i,1})$	$(W_{i,2})$	(Y)	$(1 \leq i \leq n)$
x_i :			(Y)	$(1 \leq i \leq n^{2+k}+1)$

Women's preferences

	rank-1	rank-2	rank-3	
w_j :	$(U_{j,1})$	$(U_{j,2})$		$(1 \leq j \leq n)$
y_j :			$(U \cup X)$	$(1 \leq j \leq n^{2+k}+1)$

Figure 5.3: Preference lists for the constructed instance of MIN-BP-SM-GRP

$n^k \geq 6^{-\frac{k}{2+k}} N^{1-\frac{\varepsilon}{2}}$. Without loss of generality we may assume that $n \geq 3$, so that $N \geq 6^{\frac{k}{2}}$ and hence $6^{-\frac{k}{2+k}} \geq N^{-\frac{\varepsilon}{2}}$. It follows that $n^k \geq N^{1-\varepsilon}$ as required. □

5.4 Conclusion

In this work, we introduced a restriction of the stable roommates problem in which preferences are deduced from a global ranking of the roommate pairs. This restriction has the property that weakly stable matchings are guaranteed to exist – a property that does not hold in the general roommates problem, even if there are no ties in the preference lists. We derived a polynomial-time algorithm to find a rank-maximal (weakly stable) matching. This is the first non-bipartite generalization of the rank-maximal matching algorithm due to Irving et al. [48]. Also, we proved several hardness results in an even more restricted setting for each of the problems of finding weakly stable matchings that are of maximum size, are egalitarian, have minimum regret, and admit the minimum number of weakly blocking pairs.

5.5 Recent Work

O'Malley [73] showed that the problem of determining if an edge belongs to *some* weakly stable matching in an instance of SM-SYM is NP-complete. They also gave generalized algorithms for finding a strongly (resp. super) stable matching in the capacited version of SM-SYM, where agents on one side of the bipartition can be allocated to more than one agent on the other side of the bipartition.

Sng [90] studied popular matchings in the context of SM-SYM, the bipartite restriction of SR-GRP with symmetric preferences. Using properties of this preference class, Sng derived a characterization of popular matchings and a polynomial-time algorithm to determine if a given matching is popular. The complexity of determining if a given instance admits a popular matching is still open.

Finally, Ackermann et al. [8] studied two-sided stable marriage markets in which there is no central authority to find a stable matching. Agents in the market propose pairings to other agents, who may reject or tentatively accept the pairing. Ackermann et al. showed that the number of proposals required before the market reaches a stable matching (called the convergence time) can be exponential in the number of agents, even if the agents make locally optimal proposals. As a counterpart to this result, Ackermann et al. showed that for markets with *correlated* preferences, defined as SR-GRP, the convergence time is polynomial.

Acknowledgement

We would like to thank Péter Biró and Utku Ünver for helpful remarks concerning relationships between SR-GRP and SR-GAP.

Chapter 6

Egalitarian Matching

Declaration

The material in this chapter is joint work with R. Ravi.

Abstract

Given a graph G, we consider the problem of finding a distribution over maximum matchings of G that, as much as possible, equalizes all the probabilities of the vertices being matched. Building on Bogomolnaia and Moulin's work on bipartite graphs [16], and Roth et al's. work on general graphs [78], we give an alternative structural characterization of *egalitarian* distributions for general graphs. This characterization leads to the first polynomial time algorithm for sampling from an egalitarian distribution of a general graph.

6.1 Introduction

A *matching* M of an undirected graph $G = (V, E)$ is a subset of the edges E such that no two edges in M are incident on the same vertex. A vertex $v \in V$ is *matched* by M if v is incident on some edge in M, otherwise v is *unmatched*. A *maximum matching* M of G is a matching that has maximum cardinality amongst all matchings. In general, if we select one maximum matching M, some vertices are matched by M, while others are not. Alternatively, if we select some other maximum matching M', a different possibly non-disjoint set of vertices are (un)matched by M'. In this chapter, we are interested in finding a *probability distribution* over the set of all maximum matchings such that, as much as possible, we equalize the probabilities of vertices being matched. Building on the work of [16, 50, 78], we show that there exists an *egalitarian* distribution over the set of maximum matchings, and that we can sample from this distribution in polynomial time.

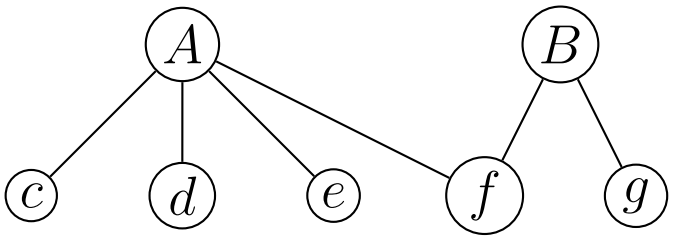
6.1.1 Egalitarianism: Lorenz Dominance

Given a distribution D over the set of maximum matchings, let $Pr_D(v)$ be the probability that vertex $v \in V$ is matched in some maximum matching of D. Define the *profile* p_D of D as the collection of vertex probabilities $[\Pr_D(v) : v \in V]$ arranged in non-decreasing order, and let $p_D(i)$ be the ith probability in p_D. As in [16], we use *Lorenz dominance* as the notion of *egalitarianism*. A distribution D *Lorenz dominates* another D' if, (i) $\sum_{i=1}^{j}(p_D(i) - p_{D'}(i)) \geq 0$ for all j, and (ii) $\sum_{i=1}^{k}(p_D(i) - p_{D'}(i)) > 0$, for some k. A distribution D is *Lorenz dominant* if it Lorenz dominates all other distributions D' with a different profile.

In order for a distribution D to be Lorenz dominant, it must allocate the maximum probability amongst all distributions to the first position of its profile. This position contains the probability of a vertex with the minimum probability of being matched. Further to this, D must allocate the maximum probability amongst all distributions to the first j positions of the profile, for all j. In particular, D is Lorenz dominant only if there is no other distribution D' that allocates a lower probability in the first $j-1$ positions of its profile in order to get a higher total probability in the first j positions of its profile. Finally, a Lorenz dominant profile is unique – for each other profile, the Lorenz dominant profile must allocate strictly more probability to the first k positions, for some k.

Perhaps a more natural notion of egalitarianism is a distribution whose profile is lexicographically maximal. From the discussion above, it is clear that Lorenz dominance is a much stronger condition in general than lexicographical dominance. For example, the *sequence* $[\frac{2}{10}, \frac{2}{10}, \frac{2}{10}, \frac{4}{10}]$ lexicographically dominates $[\frac{1}{10}, \frac{3}{10}, \frac{3}{10}, \frac{3}{10}]$, however this dominance comes a

cost, since the worst three vertices are collectively better off in the lexicographically dominated distribution. A Lorenz dominant distribution maximizes the probability of the lowest probability vertices without paying this cost. Figure 6.1.1 gives an example Lorenz dominant distribution. Because Lorenz dominance is such a strong condition, it is not immediately clear if such a distribution exists. Furthermore, since a graph can admit an exponential number of maximum matchings, even if such a distribution exists, it is not immediately clear if we can sample from such a distribution in polynomial time. We will see that in this setting, Lorenz dominance and lexicographical dominance are equivalent. Hence, at least one of the example sequences above is not the profile of a legal distribution over maximum matchings.



Matching	Probability
$(A, c), (B, f)$	1/3
$(A, d), (B, g)$	1/3
$(A, e), (B, f)$	1/6
$(A, e), (B, g)$	1/6

Figure 6.1: Example Lorenz dominant distribution.

6.1.2 Motivation

The problem of finding an Lorenz dominant distribution over the set of maximum matchings has a market-based interpretation based on the following restriction of the STABLE ROOMMATES PROBLEM (SR).

An instance of the STABLE ROOMMATES PROBLEM WITH BINARY/DICHOTOMOUS PREFERENCES (SR-BIN) consists of a set of agents, where each agent has a preference list that partitions the other agents into two groups, namely *acceptable* and *unacceptable*. Each agent a has a utility of 1 for being paired with an agent it finds acceptable, and a utility of $-\infty$ for being paired with an agent it finds unacceptable. Also, if an agent is left unpaired, it has a utility of 0. SR-BIN is a restriction of SR because each agent is indifferent amongst all its acceptable agents.

Given an instance of SR-BIN, the high-level goal is to find a pairing of agents that somehow takes into account the agent preferences for one another. A *matching* is a pairing of the agents such that no agent is member of more than one pair. We restrict ourselves to matchings

of mutually acceptable agents, since if an agent is paired with an unacceptable partner, the agent would prefer to remain unmatched.

We further restrict ourselves to matchings that are *stable*, meaning that there is no pair of agents who would prefer to be matched with each other rather than their existing partners in the matching. Let $G = (V, E)$ be a graph with one vertex for each agent, and an edge between any pair mutually acceptable agents. It is clear that for SR-Bin there is a one-to-one correspondence between stable matchings and (inclusion) maximal matchings.

The final restriction is that matchings must have maximum cardinality. The reason for this is that if we output a matching M that is maximal but not maximum, we can augment M to find a larger matching M' in which all the agents matched by M are still matched by M', but in addition, M' matches some agents that are unmatched by M. A maximum matching M is *Pareto optimal* since there is no other matching M' in which some agent prefers their partner in M' to M, and no agent prefers their partner in M to M'. Because preferences are binary, there is a one-to-one correspondence between Pareto optimal matchings and maximum matchings. Hence, we are only interested in the set of maximum cardinality matchings of G.

Even in a maximum matching, it may not be possible to match all agents. This exposes the market operator to charges of bias - why was one matching selected (in which a particular agent is not matched) over another (in which the agent is matched). Sampling from an egalitarian distribution ensures that, as much as possible, the *a priori* probability of each agent being matched is equalized.

One real world example of this problem can be found in pairwise kidney exchange markets. In these markets, patients with terminal kidney disease obtain compatible donors by swapping their own willing but incompatible donors with other patients. We can model the basic market by constructing one vertex for each patient, and an undirected edge between any two patients where the incompatible donor for one patient is compatible with the other patient, and vice versa. Of course, patients may have different preferences over donors. However, since the expected years of life gained from a transplant is similar amongst all compatible kidneys, parts of the medical community have suggested that patient preferences should be *binary/dichotomous* [21, 34] - i.e. patients should be indifferent amongst all compatible donors. This is exactly SR-Bin. Centralized kidney exchange markets have been established in many countries. Along with ensuring that as many patients get transplants as possible, one of the key aims of these markets is to ensure fairness in the matching process.

6.1.3 Previous Work

Bogomolnaia and Moulin [16] prove the existence of a Lorenz dominant distribution when G is a bipartite graph. The two key tools in their paper are i) the Gallai-Edmonds Decomposition, which gives structural information on maximum matchings, and ii) the following upper bounding technique. Let A be a subset of vertices on one side of the graph bipartition, and let $N(A)$ be the neighbors of A in G. Then, even if all vertices in $N(A)$ are matched to vertices in A, some vertex in A has probability at most $\min(1, |N(A)|/|A|)$ of being matched. Bogomolnaia and Moulin [16] give an algorithm for constructing an egalitarian distribution by repeatedly selecting a bottleneck set A with the smallest upper bound, fully allocating its neighbors to A, and then recursing on the remainder of the graph. This algorithm does not run in polynomial time however, because there are an exponential number of vertex subsets that could be a bottleneck set.

Katta and Sethuraman [50] show that there is a one to one correspondence between the sequence of bottleneck sets and the sequence of breakpoints of the min-cut capacity function in a flow network representation of the problem. Using parametric network flow algorithms [30], all the breakpoints, and hence all the bottleneck sets, can be found in $O(nm \log n)$ time, where n and m are the number of vertices and edges of G respectively.

Roth et al. [78] generalizes the work of Bogomolnaia and Moulin [16] to non-bipartite graphs. Their paper uses the Gallai-Edmonds Decomposition lemma to reduce the problem on non-bipartite graphs to a variation of the problem on bipartite graphs in which some vertices have an inherent probability of being matched, and hence don't need to be allocated as much probability by the bipartite distribution. As in the paper of Bogomolnaia and Moulin [16], Roth et al's. structural characterization algorithm does not run in polynomial time, again because there can be an exponential number of possible bottleneck sets.

In related work, Kavitha et al. [51] study popular mixed matchings. Recall from Chapter 2 that a popular agent-item matching may not exist. A *mixed* matching is a distribution over matchings. A distribution D is *popular* if there is no other distribution D' in which the expected number of agents that prefer D' to D exceeds the expected number of agents that prefer D to D'. Using linear programming techniques, [51] shows that there always exists a popular mixed matching and that, since the linear program has polynomial size, it can be found in polynomial time.

6.1.4 Chapter Outline

In Section 6.2, we give an alternative proof of the existence of an egalitarian distribution using local search techniques. We also give an alternative structural characterization for egalitarian distributions. In Section 6.3, we use the properties of the structural characterization to give the remaining details of a new polynomial time algorithm for sampling from an egalitarian distribution. Finally, in Section 6.4, we summarize the overall polynomial-time sampling algorithm.

6.2 Structural Characterization

In this section, we prove that there exists a Lorenz dominant distribution for any general graph G. Bogomolnaia and Moulin [16] proved the restriction of this theorem to bipartiate graphs. Also, Roth et al. [78] proved the theorem for general graphs [78]. The main contribution in this section then is our alternative local-search based approach. This approach leads to a new structural characterization of the Lorenz dominant distribution. We use this structural characterization in Section 6.3 to derive a polynomial time algorithm that samples from the Lorenz dominant distribution.

Briefly our approach is as follows. In Section 6.2.1, we revisit the Gallai-Edmonds Decomposition [56], which gives structural properties of maximum matchings. In Section 6.2.2, we use these structural properties to remove vertices that are never matched, or always matched in any maximum matching. In Section 6.2.4, we use more of these structural propertoes to derive a bipartite representation of the graph, in which some vertices are really supervertices that represent special components of the graph. This bipartite graph is much easier to work with than the original graph. In Section 6.2.7, we give a local search algorithm that starts with an arbitrary distribution of maximum matchings of this bipartite graph, and then repeatedly find and apply local changes that make the distribution more egalitarian. Instead of working with distributions over matchings, we work directly with probabilities on the edges, and so each of these local changes involves moving probability from one edge to an adjacent edge. Section 6.2.6 explains how to convert from edge probabilities to distributions over maximum matchings. Finally, in Section 6.2.7, once the local search has converged to a local maxima, we inspect properties of the local maxima to derive the new structural characterization.

6.2.1 Gallai-Edmonds Decomposition

We begin the full exposition of our approach with the Gallai-Edmonds Decomposition lemma [56], which gives several structural properties of maximum matchings. Let $G = (V, E)$ be an arbitrary undirected graph. The vertex set V can be partitioned into the following three sets, namely GED-U$[G]$, GED-O$[G]$ and GED-P$[G]$. Vertices in GED-U$[G]$ are called *underdemanded*, since they are unmatched in some maximum matching of G. All other vertices that are adjacent to one in GED-U$[G]$ are called *overdemanded* and belong to GED-O$[G]$. Finally, all remaining vertices are *perfectly demanded* and belong to GED-P$[G]$. We will use the following properties from the decomposition lemma: (Note that in the lemma, and through the rest of the exposition, the cardinality of a connected component C of a graph is the number of vertices in C.)

Lemma 6.2.1 (Gallai-Edmonds Decomposition)**.** *In any maximum matching M of G,*

1. *For all u in* GED-O$[G]$*, $M(u)$ is in* GED-U$[G]$
2. *For all even (cardinality) components C of $G \setminus$* GED-O$[G]$*, i) $C \subseteq$* GED-P$[G]$*, and ii) $M(u)$ is in C, for all u in C*
3. *For all odd (cardinality) components C of $G \setminus$* GED-O$[G]$*, i) $C \subseteq$* GED-U$[G]$*, ii) $M(u)$ is in C, for all u in C except one, say v, and iii) either v is unmatched in M, or $M(v)$ is in* GED-O$[G]$

where $M(v)$ is the vertex matched to v by M.

6.2.2 Removing Isolated and Perfectly Demanded Vertices

If a vertex has degree 0, it can be in no maximum matching, and so we remove it from G, thereby reducing the problem. As an aside, every vertex with degree greater than 0 is matched by some maximum matching of G. This is clearly the case for vertices in GED-O$[G]$ and GED-P$[G]$, since by definition, these vertices are matched by *every* maximum matching of G. For a vertex $u \in$ GED-U$[G]$, let M be a maximum matching which does not match u. We will show that u is matched by some other maximum matching M'. Since u has degree at least 1, it must be adjacent to another vertex v. Note that v must be matched, otherwise we can add $\{u, v\}$ to M, contradicting the assumption that M has maximum cardinality. This means that $M' = M \setminus \{\{v, M(v)\}\} \cup \{\{u, v\}\}$ is a maximum matching that matches u.

In addition to removing degree-0 vertices from G, we can also reduce the problem by removing GED-P$[G]$ vertices from G. Note that all vertices in GED-P$[G]$ are matched to each other in all maximum matchings of G. This means we can find a perfect matching on GED-P$[G]$ and include this in every maximum matching in the support of the final distribution D of maximum matchings over $G-$ GED-P$[G]$. Henceforth, we assume that G has no isolated vertices, and no GED-P$[G]$ vertices.

6.2.3 Local Search Approach

Our overall approach involves local search, i.e. start from an arbitrary distribution D_i, make a local improvement to form a new distribution D_{i+1} that Lorenz dominates D_i, and repeat until no more local improvements are possible.

Rather than work with distributions over maximum matchings though, we work with the induced *edge probabilities*, where the probability of an edge e being in a maximum matching is just the total weighted probability of all maximum matchings in D that include e. In every case, the local change involves moving probability from one edge $\{v, w\}$ to an adjacent edge $\{w, u\}$, where $Pr_D(u) < Pr_D(v)$. These local changes redistribute probability between two vertices, namely u and v, while all other vertices have the same probability of being matched in the new distribution. In Lemma 6.2.2, we show that all such local changes actually *improve* the underlying profile. Later, in Section 6.2.6, we show that these local improvements are legal in the sense that there exists some distribution over maximum matchings that induces the new edge probabilities.

Lemma 6.2.2. *Let D be a distribution over maximum matchings. Let D' be the result of moving $0 < \delta \leq (p_D(b) - p_D(a))/2$ from position b in D to position a, where $a < b$ (and hence $p_D(a) \leq p_D(b)$). Then D' Lorenz dominates D.*

Proof. Let a' and b' be the positions that the vertex probabilities at positions a and b from D end up in D'. Note that by construction, $a' \geq a$ and $b' \leq b$. Let $S(j) = \sum_{i=1}^{j}(p_{D'}(i) - p_D(i))$ be the difference between the cumulative sum of $p_{D'}$ and p_D indexed by j.

For $j < a$, $S(j) = 0$, since there is there is no difference between $p_{D'}$ and p_D. For $a \leq j < a'$, $p_{D'}(j) \geq p_D(j)$, since $p_D(a) \leq p_D(a+1) \leq \ldots \leq p_D(a'-1)$, and so $S(j) \geq 0$. At $j = a'$, $S(j) = \delta$, since every probability seen in p_D occurs in $p_{D'}$, except for $p_D(a)$ which has been increased to $p_D(a) + \delta$. For $a' < j < b'$, there is no change in $S(j)$, since both $p_{D'}$ and p_D are the same. For $b' < j < b, S(j) \geq 0$, since $p_D(b) - p_D(b') \leq \delta$. At $j = b'$, $S(j) = 0$, since every probability seen in p_D occurs in $p_{D'}$, except for $p_D(a)$ and $p_D(b)$ which have exchanged (and

conserved) δ probability. Finally, for $j > b'$, $S(j) = 0$, since there is no difference between $p_{D'}$ and p_D. □

6.2.4 Shrinking Odd Components

Recall from Lemma 6.2.1, that a maximum matching of G has at most two types of edges (assuming we have removed GED-P$[G]$ vertices). The first type of edge matches a pair of GED-U$[G]$ vertices within the same odd component, say O. The second type of edge matches a GED-U$[G]$ vertex from some odd component, say O, with a GED-O$[G]$ vertex.

In the next lemma, we show that within an odd component O of the GED decomposition of G, every vertex must be allocated the same probability of being matched by any Lorenz dominant distribution. This result will allow us to shrink O down to a single representative vertex so that we only have to focus on making local improvements to the second type of edge.

Lemma 6.2.3. *Let O be an odd component of the Gallai-Edmonds decomposition of G. A distribution D is Lorenz dominant only if $Pr_D(u) = Pr_D(v)$ for all $u, v \in O$.*

Proof. Suppose for a contradiction that D is Lorenz dominant and $Pr_D(u) < Pr_D(v)$ for some $u, v \in O$. It follows that there exists some matching M in the support[1] of D in which u is unmatched by M, while v is matched. Let p be the probability of M under D, and let $\delta = (Pr_D(v) - Pr_D(u))/2$. We will construct a new distribution D' from D by replacing M such that D' Lorenz dominates D.

Let M_O be the submatching of M induced by O. Since $O \subseteq$ GED-U$[G]$, every vertex in O is unmatched in some maximum matching of G, by Lemma 6.2.1. Also, in any maximum matching of G, all but one vertex of O are matched to each other. Hence, there exists $|O|$ maximum matchings of size $\frac{(|O|-1)}{2}$ on O, each of which leaves a different vertex of O unmatched. Let M'_O be a maximum matching on O in which u is matched and v is unmatched. Note that every vertex $w \in O - \{u, v\}$ is matched by both M_O and M'_O. If $\delta \geq p$, replace M in D' by $M - M_O + M'_O$. Otherwise, reduce the probability of M in D' by a factor of $(1 - \frac{\delta}{p})$, and add $M - M_O + M'_O$ to D' with probability δ. In either case, by Lemma 6.2.2, D' Lorenz dominates D, which gives the required contradiction. □

Let $Pr_D(O)$ be the probability each vertex of O is matched under a Lorenz dominant distribution D. Here is an easy way to convert any distribution into one in which every vertex in

[1]The *support* of a distribution consists of the set of elements in the distribution with non-zero probability.

O has the same probability of being matched. Let M be a matching in the support of D in which no vertex of O is matched to a GED-O$[G]$ vertex. Let M_O be the submatching of M induced on O. Recall from the proof above that there exist $|O|$ maximum matchings on O, each of which leaves a different single vertex of O unmatched. We can replace M_O with the uniform distribution over these maximum matchings. These maximum matchings can be found in polynomial time in the following way: Construct and output a maximum matching M_O on O. For each vertex v matched by M_O, remove v from O, find an augmenting path, output the resulting matching, and then replace v in O. Each matching that is outputed by this procedure is a maximum matching on O in which a different vertex is left unmatched.

In the next lemma, we specify the structure of $Pr_D(O)$ in all Lorenz dominant distributions.

Lemma 6.2.4. *Let D be a Lorenz dominant distribution over maximum matchings of G. Let O be an odd component of the Gallai-Edmonds decomposition of G. Finally, let $w(O)$ be the weighted fraction of matchings in D in which some vertex of O is matched to a* GED-O$[G]$ *vertex, and let $LB(O) = \frac{|O|-1}{|O|}$. Then $Pr_D(O) = LB(O) + w(O)(1 - LB(O))$*

Proof. Consider a maximum matching M in which some vertex of O is matched to a GED-O$[G]$ vertex. By Lemma 6.2.1, M matches all vertices in O. Hence, with probability $w(O)$, all vertices of O are matched.

Now consider a maximum matching M in which no vertex of O is matched to a GED-O$[G]$ vertex. By Lemma 6.2.1, M matches all vertices of O to each other, except for one, which is left unmatched. Also, by Lemma 6.2.3, every vertex $v \in O$ has the same probability $Pr_D(O)$ of being matched under D. Hence, with probability $(1-w(O))$, each vertex has a probability of $\frac{|O|-1}{|O|}$ of being matched by such matchings M.

It follows immediately that $Pr_D(O) = w(O) + (1 - w(O))LB(O) = LB(O) + w(O)(1 - LB(O))$. □

Note that the only unknown parameter in $Pr_D(O)$ at this point is $w(O)$, i.e. the fraction of time we should pair some vertex of O with a GED-O$[G]$ vertex. To solve for $w(O)$, we shrink each odd component O down to a single representative vertex. Let H be a bipartite graph with one left vertex for each odd component O, and one right vertex for each GED-O$[G]$ vertex. Add an edge between an odd component vertex and a GED-O$[G]$ vertex if the two are adjacent in G. The weight of an edge $w(e)$ will represent the fraction of matchings in which the GED-O$[G]$ endpoint is matched with some vertex in the odd component endpoint. Let $w(v)$ be the sum of edge weights incident on a vertex v. Because we are seeking a distribution over maximum matchings, we require for *feasibiliy* that $w(v) = 1$ if $v \in$ GED-O$[G]$ (recall that

GED-O$[G]$ vertices are matched in all maximum matchings). Also, we require that $w(O) < 1$ for each odd component vertex O.

6.2.5 Upper and Lower Bounds on Vertex Probabilities

Recall that every GED-O$[G]$ and GED-P$[G]$ vertex is matched in every maximum matching of G. Hence, all such vertices have probability 1 of being matched by an egalitarian distribution over maximum matchings.

Every other vertex v belongs to GED-U$[G]$ and some odd component O, and all other vertices in this odd component O have the same probability of being matched as v in an egalitarian distribution. Recall that $Pr_D(O) = LB(O) + w(O)(1 - LB(O))$, where $LB(O) = \frac{|O|-1}{|O|}$. One obvious lower bound on $Pr_D(O)$ is $LB(O)$, since $0 \leq w(O) < 1$ and $LB(O) < 1$. Similarly, an obvious upper bound on $Pr_D(O)$ is 1, which occurs if $w(O) = 1$.

Roth et al. [78] give the following improved upper bound:

Lemma 6.2.5. *Consider a collection of odd component vertices $\mathcal{O} = \{O_1, O_2, \ldots\}$. Let $N(\mathcal{O})$ be the* GED-O$[G]$ *neighbors of O in H. Then some vertex v in $\bigcup_{O \in \mathcal{O}} O$ has $Pr_D(v) \leq \min(1, \frac{\sum_{O \in \mathcal{O}} |O| - (|\mathcal{O}| - |N(\mathcal{O})|)}{\sum_{O \in \mathcal{O}} |O|})$.*

Proof. Let M be a maximum matching of G. In each odd component O, the submatching M_O of M induced on O leaves one vertex unmatched. At most $|N(\mathcal{O})|$ of the vertices unmatched by the induced matchings can be matched by the adjacent GED-O$[G]$ vertices.

Hence, in every maximum matching, at least $|\mathcal{O}| - |N(\mathcal{O})|$ vertices in some odd component are unmatched. The upper bound follows since at least one vertex $v \in \bigcup_{O \in \mathcal{O}} O$ has at most the average upper bound probability of being matched amongst all vertices in $\bigcup_{O \in \mathcal{O}} O$. □

6.2.6 From Edge Probabilities to a Feasible Distribution over Maximum Matchings

Given a feasible assignment of edge weights on H, it follows immediately from the Birkhoff-von Neumann Decomposition lemma [14, 97] that there is a distribution over maximum matchings of H that induces these edge weights. For exposition purposes, we sketch a version of the Birkhoff-von Neumann Decomposition lemma due to Dulmage and Halperin [22].

First we show that every maximum matching on positive weighted edges matches all GED-O$[G]$ vertices. Let S be any subset of GED-O$[G]$ vertices. Recall that for feasibility, $w(v) = 1$ for all $v \in$ GED-O$[G]$. It follows that $|S| = \sum_{v \in S} w(v) = \sum_{v \in S} \sum_{O \in N(v)} w(v, O)$, where $N(v)$ is the set of neighbors of v in H. Recall also that $w(O) < 1$ for all odd component vertices. It follows that $|N(S)| > \sum_{O \in N(S)} w(O) = \sum_{O \in N(S)} \sum_{v \in N(O)} w(v, O) \geq \sum_{O \in N(S)} \sum_{v \in N(O) \cap S} w(v, O) = |S|$. Hence, $|N(S)| > |S|$ for any subset S of GED-O$[G]$ vertices, and so by Hall's Marriage Theorem [38], all GED-O$[G]$ vertices are matched in any maximum matching.

The following algorithm constructs the required distribution over maximum matchings. Let M_1 be any maximum matching on positive weighted edges, and let α_1 be the smallest edge weight in M_1. Subtract α_1 from all edges in M_1, and remove any M_1 edges from H that now have weight 0. If the graph contains no more edges, stop. Otherwise, multiply all edge weights by $\frac{1}{(1-\alpha_1)}$. Note that the graph still has the property that all GED-O$[G]$ vertices have weight 1, and all odd component vertices have weight at most 1. Hence, we can recurse on the remaining graph to find M_2 and α_2, and so on.

It is easy to verify that by setting $\hat{\alpha}_i = \alpha_i * \Pi_{j=1}^{i-1}(1-\alpha_j)$, this algorithm produces a distribution over maximum matchings of H that induces the original edge weights. Also, since at least one additional edge weight becomes 0 in each iteration, the distribution contains at most m maximum matchings, where m is the number edges in H. Using the maximum matching algorithm of Hopcroft and Karp [41], the running time of this algorithm is $O(m^2 n^{\frac{1}{2}})$.

In order to sample a maximum matching of the original graph G, we can do the following. First, construct the distribution of maximum matchings over H. Then, randomly select a matching M_H from this distribution. This matching only pairs vertices of GED-O$[G]$ with odd component vertices. We also need to add in a perfect matching on GED-P$[G]$ vertices, as described in Section 6.2.2. And finally, we need to add a maximum matching on the unmatched vertices in each odd component O. For each O, if O is matched by M_H, add a perfect matching on the unmatched vertices of O. Otherwise, if O is unmatched by M_H, select a vertex from O uniformly at random, and include a maximum matching on O that does not include this vertex.

6.2.7 Local Search Algorithm

Recall from section 6.2.3 that our overall approach involves local search, i.e. start from an arbitrary distribution D_i, make a local improvement to form a new distribution D_{i+1} that Lorenz dominates D_i, and repeat until no more local improvements are possible. In this section, we consider locally optimal distributions found via this local search algorithm. We

derive a structural characterization of locally optimal distributions and use this to show that locally optimal distributions are also egalitarian distributions.

We start by making the local search algorithm concrete: Begin with a feasible distribution D over maximum matchings. For example, construct a maximum matching M of H and set the weights of all edges in M to 1, and all edges not in M to 0. Repeat the following while no change is made: If any GED-O$[G]$ vertex v has two odd component neighbors O_i and O_j with $Pr_D(O_i) < Pr_D(O_j)$ and $w(v, O_j) > 0$, move δ weight from the edge $\{v, O_j\}$ to the edge $\{v, O_i\}$. Note that the aim in this step is to construct a new distribution D' in which the probability of being matched for vertices in O_i and vertices in O_j is as equal as possible. We can get an upper bound for δ by setting $Pr_{D'}(O_i) = Pr_{D'}(O_j)$, meaning $LB(O_i) + (w(O_i) + \delta)(1 - LB(O_i)) = LB(O_j) + (w(O_j) - \delta)(1 - LB(O_j))$, which gives $\delta = \frac{Pr_D(O_j) - Pr_D(O_i)}{2 - LB(O_j) - LB(O_i)}$. Hence, $\delta = \min(w(v, O_j), \frac{\Pr_D(O_j) - Pr_D(O_i)}{2 - LB(O_j) - LB(O_i)})$.

Example 6.2.6. *Figure 6.2.6 contains part of an example execution of the local search algorithm. The first graph represents the initial distribution, which is a single maximum matching. The second graph represents the distribution after one iteration, once vertex B has redistributed probability from f to g. The final graph represents the distribution after the local search has reached a local optima. The edge weights in this graph are induced by the Lorenz dominant distribution in Figure 6.1.1.*

Lemma 6.2.7. *The local search algorithm converges to a local maxima.*

Proof. By Lemma 6.2.2, the sequence of distribution profiles monotonically increases with each local improvement. Also, the distribution profile is bounded above by the (infeasible) profile in which every vertex has probability 1 of being matched. Hence, the local search algorithm converges to a local maxima. □

Note that we are not concerned with the running time of the local search algorithm at this point, only that it converges.

We now consider some properties of local maxima found by the local search algorithm. Given a local maxima, remove all 0-weighted edges to decompose the bipartite graph into maximally connected components. For a connected component C, let ODD$[C]$ be the set of odd component vertices from H in C, and let GED-O$[C]$ be the set of GED-O$[G]$ vertices in C.

Lemma 6.2.8. *Let C be a maximally connected component of the decomposed graph H. Then every odd component vertex in* ODD$[C]$ *has the same probability of being matched.*

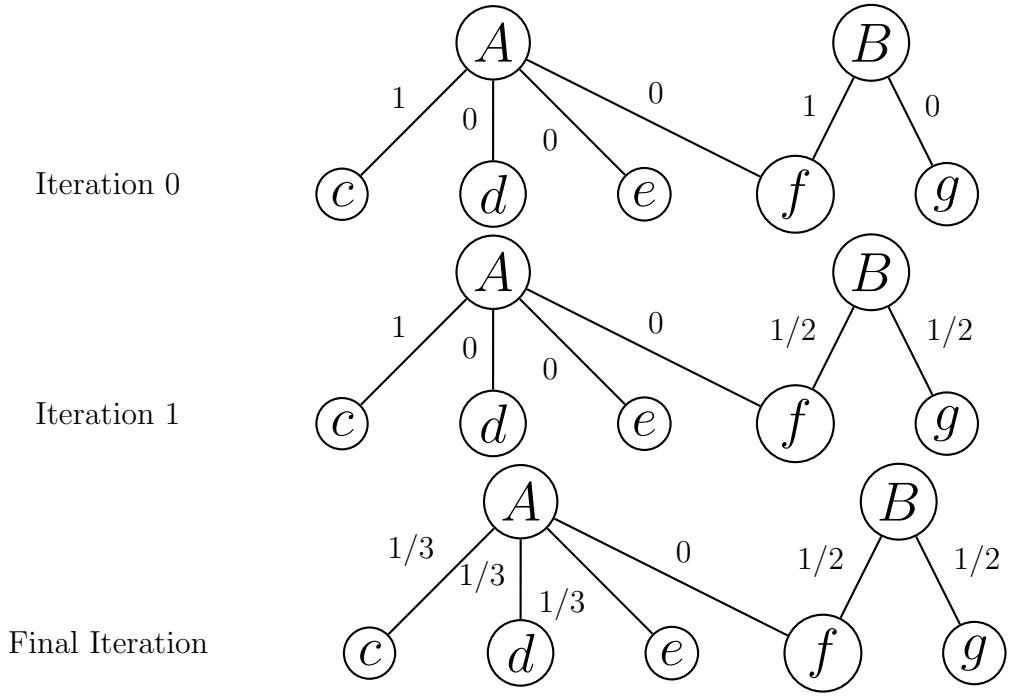


Figure 6.2: Sample execution of the local search algorithm

Proof. Suppose for a contradiction that odd component vertices in ODD$[C]$ have different probabilities of being matched under D. Since C is connected, there is a path of positive weight edges between the highest and lowest probability odd component vertices in C. At some point on this path, two odd component vertices incident on the same GED-O$[G]$ vertex have different probabilities. This contradicts the assumption that we are at a local maxima, since we can perform a local improvement by moving probability between the two odd component vertices. □

Henceforth, let $Pr_D(\text{ODD}[C])$ be the probability of being matched for all vertices in any odd component vertex in ODD$[C]$. Note that in all maximum matchings in the support of D, GED-O$[C]$ vertices are always allocated to vertices in ODD$[C]$. Also, exactly one vertex in each odd component in ODD$[C]$ is unmatched within its own odd component. It follows that $|\text{ODD}[C]| - |\text{GED-O}[C]|$ vertices in $\bigcup_{O \in \text{ODD}[C]} O$ are unmatched. But since each odd component vertex in ODD$[C]$ has the same probability of being matched, and within an odd component, all vertices also have the same probability of being matched, it follows that $Pr_D(\text{ODD}[C]) = \frac{\sum_{O \in \text{ODD}[C]} |O| - (|\text{ODD}[C]| - |\text{GED-O}[C]|)}{\sum_{O \in \text{ODD}[C]} |O|}$. We remark for later that this expression for $Pr_D(\text{ODD}[C])$ is the same as the upper bound given in Lemma 6.2.5 for ODD$[C]$ when

$N(\text{ODD}[C])$ is restricted to GED-O$[C]$.

Lemma 6.2.9. *Let C_i and C_j be maximally connected components of the decomposed graph H, with $Pr_D(\text{ODD}[C_i]) < Pr_D(\text{ODD}[C_j])$. Then there is no 0-weighted edge from an odd component vertex O of C_i to a* GED-O$[G]$ *vertex v of C_j.*

Proof. Since $C_i \neq C_j$, any edge between O and v must have weight 0 (otherwise C_i and C_j are connected). Suppose for a contradiction that there is such a 0-weighted edges. This again contradicts the assumption we are at a local maxima, since we make a local improvement by redistributing probability from some edge between v and an odd component vertex in ODD$[C_j]$ to the edge $\{O, v\}$. □

Lemma 6.2.10. *Consider an allocation of edge weights in H after 0-weighted edges have been removed. The allocation of edge weights is locally optimal in H if and only if i) for each maximally connected component C, all odd-component vertices in C have the same probability of being matched, and ii) for any pair of maximally connected components C_i and C_j with $Pr_D(\text{ODD}[C_i]) < Pr_D(\text{ODD}[C_j])$, there is no 0-weighted edge from an odd component vertex of C_i to a* GED-O$[G]$ *vertex of C_j.*

Proof. Both the first and second conditions of the lemma are necessary for an allocation to be locally optimal, as shown in Lemmas 6.2.8 and 6.2.9 respectively. It remains to show that the two conditions are sufficent.

Consider a feasible assignment of edge weights that is not locally optimal. Suppose for a contradiction that both conditions in the lemma hold. Since the assignment is not locally optimal, there is a local improvement that redistributes probability from a higher-probability odd component vertex O_j to a lower-probability vertex O_i through a GED-O$[G]$ vertex w. Now, consider the decomposition of H into maximally connected components among positive weight edges. Note that O_i and O_j cannot belong to the same maximally connected connected, otherwise the condition from Lemma 6.2.8 is violated. Hence, It must be the case that there is a 0-weighted edge from O_i to w. However, this contradicts the second condition of the lemma.

It follows that if both conditions hold, the assignment must be locally optimal. □

Now that we have characterized locally optimal solutions, we are ready to present the main result in this section – a structural characterization of Lorenz dominance in terms of the necessary and sufficient conditions for local optimality from Lemma 6.2.10.

Theorem 6.2.11. *Every locally optimal distribution is Lorenz dominant.*

Proof. Let D be a locally optimal distribution over maximum matchings. First we show that D is lexicographically maximal.

Let $C_1, C_2, \ldots$ be the maximally connected components of the decomposed bipartite graph H associated with D. With out loss of generality, assume that $Pr_D(\text{ODD}[C_1]) < Pr_D(\text{ODD}[C_2]) < \ldots$ (if $Pr_D(\text{ODD}[C_i]) = Pr_D(\text{ODD}[C_{i+1}])$ we will consider them the same component).

Consider the first component C_1. By Lemma 6.2.9, there are no 0-weighted edges from $\text{ODD}[C_1]$ vertices to vertices in $\text{GED-O}[G] - \text{GED-O}[C_1]$. It follows that there are *no* edges from $\text{ODD}[C_1]$ vertices to vertices in $\text{GED-O}[G] - \text{GED-O}[C_1]$, otherwise the $\text{GED-O}[C_1]$ endpoints of these edges would be part of C_1. Hence, $N(\text{ODD}[C_1]) \subseteq \text{GED-O}[C_1]$ and $Pr_D(\text{ODD}[C_1]) = UB(\text{ODD}[C_1])$.

Since $Pr_D(\text{ODD}[C_1]) = UB(\text{ODD}[C_1])$ and $Pr_D(\text{ODD}[C_1])$ is the smallest probability amongst all the components, no other distribution has a profile that lexicographically dominates the profile of D in the first $|\text{ODD}[C_1]|$ positions. Moreover, it is clear that the only way to achieve this upper bound is to fully allocate the vertices in $\text{GED-O}[C_1]$ to the vertices in $\text{ODD}[C_1]$.

This argument can be applied recursively to C_2 on the graph without C_1, and so on. It follows that the profile of D is lexicographically maximal. Next we show that D is Lorenz dominant.

Suppose for a contradiction that D does not Lorenz dominate some other distribution D' with a different profile. With out loss of generality, we assume that D' is locally optimal, since Lorenz dominance is transitive. By the result above, if D' is locally optimal, then it is lexicographically maximal. Hence, D' must have the same profile as D, giving the required contradiction. □

6.3 Flow Network Approach

In this section, we present a network flow based algorithm for determining the Lorenz dominant inducing edge weights in H between odd component and $\text{GED-O}[G]$ vertices. We will see in Section 6.4 how these edge weights are used in order to sample from the Lorenz dominant distribution of graph in polynomial time.

Let D be a Lorenz dominant distribution. Consider the edge weights in H induced by D, with 0-weighted edges removed from the graph. Recall that for any maximally connected component C, $Pr_D(\text{ODD}[C]) = \frac{\sum_{O \in \text{ODD}[C]} |O| - (|\text{ODD}[C]| - |\text{GED-O}[C]|)}{\sum_{O \in \text{ODD}[C]} |O|}$. Note that there are only $O(n^2)$

possible values of $Pr_D(\text{ODD}[C])$, since both the numerator and denominator have value between 1 and n, where n is the number of vertices in G. Since we do not know the maximally connected components upfront, our high-level approach involves finding the largest of these $O(n^2)$ values that each odd component can be guaranteed. We will use a network flow formulation together with the structural characterization properties in Lemma 6.2.10 to test the odd components against these $O(n^2)$ probability values.

Start by orienting H so that each edge is directed from its odd component endpoint towards its GED-O$[G]$ endpoint. Set the capacity of each of these edges to ∞. Add a source vertex s with an outgoing edge to each odd component vertex O_i. Let α_i be the capacity of the edge to O_i. Finally, add a sink vertex t with incoming edges of capacity 1 from each GED-O$[G]$ vertex.

Consider a possible value of $Pr_D\text{ODD}[C]$, say $\frac{a}{b}$. We want to test if every odd component O can be guaranteed probability at least $\frac{a}{b}$. Recall that $Pr_D(O) = LB(O) + w(O)(1 - LB(O))$, where $LB(O) = \frac{|O|-1}{|O|}$ is the minimum probability of a vertex in O being matched, even if no vertex from O is ever matched to a GED-O$[G]$ vertex. By setting $Pr_D(O) = \frac{a}{b}$ and rearranging, we get that $w(0)$ equals $\frac{|O|(a-b)|O|+b}{b}$. To test if this is achievable, we set $\alpha = \frac{(a-b)|O|+b}{b}$ and then find a maximum flow in the network. If the network has a maximum flow that saturates every edge from the source, the flow gives an allocation in which every odd component is guaranteed probability $\frac{a}{b}$.

We can work out if this value $\frac{a}{b}$ is tight for any of the odd component vertices by inspecting the flow in the network. Start by decomposing the graph into maximally connected components by removing the source and sink vertex, and also any remaining edges with flow 0. In any component C, if at least one of its GED-O$[G]$ vertices has non-unit outflow, then there is still more flow that can be pushed through C by increasing the α_i capacities into its odd component vertices. Otherwise, if all GED-O$[G]$ vertices in C have outflow of 1, we mark C as saturated. If there is an edge with 0 flow from a ODD$[C]$ vertex to GED-O$[G]$ vertex in a unsaturated component, then by Lemma 6.2.10, $\frac{a}{b}$ cannot be tight for any ODD$[C]$ vertex. In this case, we mark C as unsaturated and propagate this unsaturated label to any saturated components that have an edge with 0 flow into a GED-O$[G]$ vertex of C. It is easy to see that amongst all saturated components at the end of this procedure, the maximum probability that can be guaranteed to their odd component vertices is $\frac{a}{b}$.

Before commenting on the overall procedure, we firstly discuss some implementation issues for a single iteration in which we are trying to determine if every odd component vertex O can be guaranteed to satisfy $Pr_D(O) \geq \frac{a}{b}$. Note that for each O_i, $\alpha_i = \frac{(a-b)|O|+b}{b}$. Since every α_i is a multiple of b, we can scale all the capacities in the graph by b. This has two important consequences. Firstly, since the maximum edge capacity is $O(n)$, weakly

polynomial-time maximum flow algorithms are efficient on the network. And secondly, there are no numerical representation problems, since by the integrality theorem and properties of these flow algorithms, the final flow values when scaled back down by b are all integer multiples of $\frac{1}{b}$.

For the overall procedure, we begin by arranging the $O(n^2)$ probability values in non-decreasing order. Then we perform a binary search on the ordering to find the highest probability value in which some odd component is tight. After finding this value, we remove the saturated components whose odd component vertices are tight, and repeat on the left-over graph. The total running time of each iteration is $O(F * \log n)$, where F is the time to find a maximum flow, and $\log n$ is the number of subiterations from the binary search. Note that the component probability values strictly increase from one iteration to the next. Also note that the denominator of a tight component probability is the number of vertices belonging to a tight odd component, and that these vertices are removed for the next iteration. Since there are at most j probability values with denominator j (i.e. $\frac{1}{j}, \frac{2}{j}, \ldots, \frac{j}{j}$), and we continue until probability values have been found for all $O(n)$ odd component vertices, the maximum number of distinct denominators k in the final solution can be found by solving $\sum_{j=1}^{k} j^2 \geq n$, which gives $k = O(n^{\frac{1}{3}})$. Hence, the maximum number of iterations before all odd component vertices are assigned a probability is $\sum_{j=1}^{k} j = O(n^{\frac{2}{3}})$. The total running time of this algorithm is therefore $O(n^{\frac{2}{3}} F \log n)$, which is $O(n^{\frac{4}{3}} m \log^2 n))$, using the Goldberg-Rao algorithm [35] for network flow.

6.4 Sampling Algorithm

In this section, we summarize our overall algorithm for sampling from an egalitarian distribution of a general graph G containing n vertices and m edges.

The algorithm begins by removing degree-0 vertices, since these vertices are never matched by any maximum matching. Next the algorithm finds the Gallai-Edmonds decomposition of G, which takes $O(mn^{\frac{1}{2}})$ time using the algorithm due to Micali and Vazirani [67]. A side effect of this algorithm is that it also constructs a maximum matching M of G. In $O(n+m)$ time, the algorithm then i) extracts the perfect matching on GED-P$[G]$ vertices from the decomposition algorithm, and then ii) constructs the network on the biparte graph H, with one vertex for each GED-O$[G]$ vertex, and one vertex for each odd component from the decomposition. As in Section 6.3, the algorithm then uses a $O(n^{\frac{4}{3}} m \log^2 n))$ network flow based approach to find egalitarian probability values for all the odd components. As in Section 6.2.6, the algorithm now finds a distribution over maximum matchings of H that induces these probabilities. The

main computational step in this part of the algorithm is finding the Birkhoff-von Neumann decomposition. Using edge coloring algorithms for bipartite graphs, this can be done in time $O(m \log D)$, where D is the maximum degree of a vertex. Hence, this decomposition step can be done in time $O(n^2 \log n)$. After selecting a matching randomly from the resulting polynomial-time distribution ($O(m)$ time), the algorithm adds in the perfect matching on GED-P$[G]$ vertices to the final solution. Finally, for each odd component O, if O is matched by H, the algorithm includes a perfect matching on the remaining vertices in O, otherwise, it includes a maximum matching on H, leaving one of the vertices unmatched, uniformly at random. In either case, since we already have M from the Gallai-Edmonds decomposition, these last steps take $O(m)$ time overall.

The running time for this algorithm is dominated by network flow computations, giving an overall runtime of $O(n^{\frac{4}{3}} m \log^2 n)$.

Theorem 6.4.1. *Given a graph G with n vertices and m edges, it is possible to sample a maximum matching from an egalitarian distribution of G in $O(n^{\frac{4}{3}} m \log^2 n)$ time.*

Acknowledgment: We would like to thank Utku Ünver for pointing us towards the algorithmic work by Katta and Sethuraman [50].

Chapter 7

Conclusion

This chapter summarizes the contributions of the thesis, presents some open problems, and finishes with some concluding remarks.

7.1 Summary of Contributions

In this thesis, we studied the design and analysis of matching markets. One type of matching market is bipartite and involves matching agents to items, for example in a keyword auction. Another type of matching market is non-bipartite and involves matching agents with each other, for example in a kidney exchange. For both types of matching markets, we explored mechanisms to ensure that the market has desirable properties, such as truthfulness and stability. We also studied several notions of optimality, such as popularity, rank-maximality and egalitarianism, and derived new algorithms for efficiently computing them. Below, we give a more detailed summary of thesis contributions.

7.1.1 Popular Matching

In this work, we consider the problem of matching a set of agents to a set of items, where each agent has a preference list that ranks the items in order of preferences. We say that a matching M' is more popular than another matching M if more agents prefer their matched item in M' over their matched item M than vice versa. A matching M is popular if there are no matchings that are more popular than it. We study the time complexity of i) deciding

if a given matching is popular, and ii) constructing a popular matching, or reporting that none exists.

In order to decide if a given matching is popular, we need to show that no other matching is more popular than it. Since there can be an exponential number of matchings, it is not feasible to simply check the given matching against every other matching. Similarly, it is not feasible to find a popular matching by simply checking each matching for popularity. Both problems i) and ii) are also complicated by the *more popular than* relation, which is not transitive, or even acyclic.

Our main contribution is an algorithmic structural characterization of a popular matching. This characterization leads to $(m\sqrt{n})$ time algorithms for the problems above, where n is the number of agents and items, and m is the total length of all the agent preference lists. We also show that in the special case where no agent is indifferent between any two items, both problems can be solved in linear time.

7.1.2 Layerable Mechanisms for Keyword Auctions

In a keyword auction, merchants bid to have their advertisements displayed on the search results webpage resulting from a keyword query from a user. A keyword auction mechanism accepts bids from the merchants and then decides on an allocation of merchants to advertising slots, and a price to charge each merchant. The combination of allocation and pricing functions define the mechanism and induce the properties of the mechanism. Various desirable properties include truthfulness, individual rationality, efficiency, and revenue maximization. The problem of designing a mechanism involves selecting the desired properties and then finding an allocation and pricing function that induces these properties.

In this work, we introduce and study the class of layerable mechanisms for keyword auctions. A layerable mechanism can be decomposed into a collection of layers, where each layer involves a simple multi-item single-unit demand (MISUD) mechanism. In Theorem 3.2.5, we show how a keyword auction mechanism inherits the properties of individual MISUD mechanisms. This decomposition approach leads to a new technique for designing keyword auctions. If we want a mechanism with certain properties, we can try to build these properties into the simple layer mechanisms, and then combine these mechanisms to form a keyword auction mechanism.

Unfortunately, we cannot combine MISUD mechanisms arbitrarily, since the overall mechanism may not be feasible. However, in Theorem 3.2.3, we prove a sufficient condition in order to guarantee feasibility. We also show in Theorem 3.2.4 that any layerable mechanism

can be constructed from layers that satisfy this sufficient condition.

In various settings, many existing mechanism are layerable, including VCG, and rank-based mechanism such as GFP, GSP and the laddered auction. Rank-based auctions share the same allocation function, differing only in their pricing function. The laddered auction was designed to be a rank-based auction with the property of truthfulness. We used our decomposition technique to greatly simplify the derivation of the truth inducing pricing function. We also used our decomposition technique to design a truthful extension of the laddered auction in which the auctioneer can bid for, and win, one of its own advertising slots, all without a conflict of interest. Microsoft has applied for two patent applications based on this work.

7.1.3 Clearing Algorithms for Barter Exchange Markets

In barter exchange markets, agents seek to swap their items with one another, in order to improve their own utilities. These swaps consist of cycles of agents, with each agent receiving the item of the next agent in the cycle. Our focus is on kidney exchange markets, where patients can obtain compatible donors by swapping their own willing but incompatible donors. The key feature of kidney exchange markets, beyond generic barter exchange markets, is that long cycles are forbidden. The reason for this is that the incompatible donor for a patient could back out of an exchange once his/her patient has recieved a kidney from another donor. This leaves some other patient without a new kidney and also without their incompatibe donor, which is their bargaining chip in the market.

The clearing problem involves finding a social-welfare maximizing exchange (set of disjoint cycles). In general, when there is no constraint on the cycle length, this problem is polynomial-time solvable. Also, if the maximum cycle length is 2, this problem is polynomial-time solvable. In this work, we showed that if the cycle length is a fixed constant greater than 2, the problem is NP-hard.

Our main contribution is an exact algorithm that can clear kidney exchange markets on a nationwide scale (in excess of 10,000 patients at once). The key difficulty for an exact algorithm is limited memory, as the problem takes cubic space in the number of patients to even model. To overcome the memory problem, we used a branch-and-price algorithm to solve an integer programming encoding of the problem. As a comparison point, without using the branch-and-price algorithm, the integer programming approach could only clear markets containing 900 patients.

Our algorithm was used for several months in 2007 by the *Alliance for Paired Donation*, one

of the four main regional kidney exchanges in the United States. Our algorithm was also selected by the United Network for Organ Sharing (UNOS), which is the national body for organ donation in the United States, for use in their upcoming nationwide kidney exchange. Its use has the potential to save thousands of lives, and hundreds of millions of dollars in health care savings each year.

7.1.4 The Stable Roommates Problem with Globally-Ranked Pairs

The stable roommates problem involves pairing up a set of agents, each of whom ranks the others in order of preference. In this work, we introduce a restriction of the stable roommates problem in which preferences are derived from a global ranking of the possible agent pairs.

This restriction is motivated by kidney exchange markets in which cycles have length at most 2. When two (patient, donor) pairs are matched with each other in order to swap donors, there is a chance that the potential swap is cancelled after expensive last-minute compatibility tests have been performed. Doctors can rank potential swaps by their chance of success. This ranking induces the preferences of the patients, since patients prefer to be involved in potential swaps that have better chances of sucess.

We give a polynomial-time algorithm to find a rank-maximal matching, which maximizes the number of rank-1 pairs in the matching, and subject to this, maximizes the number of rank-2 pairs in the matching and so on. A rank-maximal matching has the property that it is always weakly stable and, additionally, it is strongly stable, whenever a strongly stable matching exists. Our algorithm is the first generalization of the rank-maximal matching due to Irving et al. [48] to a non-bipartite setting. In contrast to this algorithmic result, we report on several hardness results for the restriction of the stable roommates problem to globally-ranked pairs. For example, we consider the problem of finding a weakly stable matching amongst n agents that has the minimum number of weakly blocking pairs (these pairs prevent the matching from being strongly stable). We show that, unless $P = NP$, this problem is inapproximable within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, even when the underlying graph is bipartite.

7.1.5 Egalitarian Matching

In this work, we consider a further restriction of the stable roommates problem. Agent preferences must now be binary: agents find each other either acceptable or unacceptable. Every maximal matching of mutually acceptable agents is weakly stable However, even a

maximum matching of mutually acceptable agents, may not be strongly stable, and some agents may be left unmatched. This exposes the market operator to charges of bias – why was one maximum matching selected (in which a particular agent is not matched) over another (in which the agent is matched).

The egalitarian matching problem is to find a distribution over maximum matchings that, as much as possible, equalizes the probabilities of the agents being matched. One property of an egalitarian distribution is that it maximizes the probability of being matched for the agent that has the least probability of being matched, and subject to this, maximizes the probability for the agent with the next least probability of being matched, and so on. An egalitarian distribution is much stronger than this though because it maximizes the sum of the probabilities of the agents with the i lowest probabilities of being matched, for all i.

It is not immediately clear if a distribution with this property exists. Building on the work of [78], we give an alternative, simpler proof that there exists an egalitarian distribution in general graphs. Our alternative proof is based on a local search approach where we start with an arbitrary distribution and then repeatedly make it more egalitarian. This local search based approach leads to a new structural characterization of an egalitarian distribution. Our main result uses this structural characterization as the basis of the first polynomial time algorithm to sample from an egalitarian distribution in a general graph.

7.2 Open Problems

7.2.1 Egalitarian Matching

One problem left open in our work on egalitarian matching in non-bipartite graphs is whether the egalitarian distribution has a polynomial-sized support. We know from [16] that if the graph is bipartite, the distribution has polynomial size. However, in non-bipartite graphs, we construct one maximum matching for each vertex in each odd component of size at least 3. The probability of these matchings is determined by the size of the odd component. This means that each odd component can potentially have matchings whose probability is different from matchings in all other odd components. It is not immediately clear if we can combine matchings from all odd components with only a polynomial number of overall matchings of the graph.

We are also trying to extend our work on egalitarian matchings to settings in which the agents have more complicated preferences. In particular, our aim is to find efficient algorithms for

producing a distribution over "optimal" matchings, such that the utility of the agent with the worst expected allocation is maximized, and subject to this, the utility of the agent with the second-worst expected utility is maximized, and so on. Various definitions of optimality can be considered, including Pareto optimality, cardinality, rank-maximality and popularity.

7.2.2 Online Markets

In a kidney-exchange market, patients and donors enter and leave the market over time. Because we cannot see the future, any exchanges we make now may turn out to be suboptimal later. However, we cannot delay making exchanges, since some patients may die in the meantime, and anyway, patients will have an incentive to leave the market and organize their own exchanges.

Incentives in offline markets have been well-studied. To avoid coalitions of agents from leaving the market and organizing their own exchanges, we need to find a *core* exchange in which these coalitions are at least as well off. We have generalized this solution concept to the online setting. In the strongest generalization, whenever a change occurs in the market, we must find and immediately commit to a maximum exchange. In a weaker generalization, we must find a maximum exchange, but instead of committing to it immediately, we only need to guarantee that the patients involved will be in some exchange in the future. This gives us more control - when a new patient arrives, we may be able to include them by augmenting the existing exchange when otherwise there would be no one left for them to exchange with. Deriving competitive algorithms in these settings is an important avenue for future research. Additionally, since kidney-exchange markets are large, it may be able to augment these algorithms with knowledge of the underlying blood type and compatibility distribution of patients and donors in the general population.

7.2.3 Query-Commit Model

There is another separate online aspect to the kidney-exchange market. To determine if a patient and potential donor are compatible, there is a sequence of increasingly accurate and expensive medical tests. The spectrum goes from basic health checks for the donor, to blood-type checks, to mixing samples of the patient and donor's blood. Unfortunately, due to financial and logistical reasons, it is not feasible to perform all the pairwise patient-donor tests upfront. Instead, when finding a exchange, we must *query* the proposed transplants before we *commit* to the exchange. In the weaker core setting for pairwise exchange markets, this is not a problem - if we discover that a pair is incompatible, they simply rejoin the

market, and we can augment our possibly sub-optimal exchange. However, in the stronger core setting, we may be forced to commit to cycles that are not in any maximum exchange. We call this the *query-commit* model, and we are currently working on several different ways to approach and solve the problem, in particular for pairwise exchange markets.

In one approach, we are trying to generalize the ranking algorithm [15,49] for online bipartite matching to the non-bipartite setting. The online aspect of the bipartite matching problem involves vertices from one side of the market arriving one by one. However, this same algorithm can be reinterpreted in the query-commit model, and so this seems like a promising avenue of research. In a separate approach, we are looking at a relaxation of the problem in which we can make several simultaneous queries per patient, and only have to commit to one of these, if any queries are successful. Finally, in another approach, we consider the problem when information is given on the probability that each donor and patient are compatible. In principle, there is an optimal plan/ordering to query the edges in the graph. Similar to our work on finding large exchanges [4], we would like to explore the feasibility of efficiently producing an optimal plan for querying the edges, even though, in the worst-case, this is not polynomial-time solvable.

7.3 Concluding Remarks

As we remarked in the Chapter 1, matching markets have long held a central place in the game theory, mechanism design and computer science literature. Recently, a new field of study has emerged called *algorithmic game theory* [70], which is a combination of all three areas. On the one hand, algorithmic game theory studies the perspective of the market operator. The main problem for market operators is designing rules for the market in order to ensure desirable properties such as truthfulness, revenue maximization and fairness. On the other hand, algorithmic game theory studies the perspective of agents participating in the market. The main problem for agents is determining the best strategy for maximizing their own utility. Underlying both perspectives of algorithmic game theory is the study of algorithms, since computational issues are central to whether or not a market can achieve the desired outcome.

The study of matching markets has been central to the emergence of algorithmic game theory. One reason for this is that there are many important real-world markets, such as kidney exchanges and keyword auctions, that at their core involve matching. Because these are real markets, there has been a big drive to study and improve them.

Another reason that matching markets continue to be studied in algorithmic game theory is

that because graph matching is a well-studied area in computer science, there are already many computational tools available. Some central results in matching theory include polynomial time algorithms for finding maximum matchings, the Gale-Shapley algorithm for stable marriage, and various structural theorems, such as the Gallai-Edmonds and Birkhoff-von Neumann decomposition results. Using these tools, it is possible to sovle central problems in matching markets, problems that seem far less tractable in other classes of markets.

For these two reasons – the importance of matching markets, and the tools already available – the study of matching markets is likely to continue being one of the main areas of research in algorithmic game theory for many years to come.

Bibliography

[1] A. Abdulkadiroglu, , P. A. Pathak, A. E. Roth, and T. Sönmez. The boston public school match. *American Economic Review*, 95:368–371, 2005.

[2] A. Abdulkadiroğlu and T. Sönmez. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3):689–701, 1998.

[3] D. J. Abraham and T. Kavitha. Dynamic popular matchings and voting paths. In *Proceedings of SWAT 2006: the 10th Scandinavian Workshop on Algorithm Theory*, volume 4059 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2006.

[4] D.J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. In *Proceedings of EC'07: the 8th ACM Conference on Electronic Commerce*, pages 295–304. ACM, 2007.

[5] D.J. Abraham, K. Cechlárová, and K. Mehlhorn D.F. Manlove. Pareto-optimality in house allocation problems. In *Proceedings of ISAAC 2004: the 15th Annual International Symposium on Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2004.

[6] D.J. Abraham, R.W. Irving, K. Telikepalli, and K. Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37:1030–1045, 2007.

[7] D.J. Abraham, A. Levavi, D.F. Manlove, and G. O'Malley. The stable roommates problem with globally-ranked pairs. In *Proceedings of WINE 2007: the 3rd International Workshop On Internet and Network Economics*, volume 4858 of *Lecture Notes in Computer Science*, pages 431–444. Springer, 2007.

[8] H. Ackermann, P. W. Goldberg, V. S. Mirrokni, H. Röglin, and B. Vöcking. Uncoordinated two-sided matching markets. In *Proceedings of EC'08: the 9th ACM Conference on Electronic Commerce*, pages 256–263. ACM, 2008.

[9] G. Aggarwal, S. Muthukrishnan, D. Pal, and M. Pal. General auction mechanism for search advertising. In *WWW 2009: the 18th International World Wide Web Conference*, pages 241–250, 2009.

[10] Gagan Aggarwal, Ashish Goel, and Rajeev Motwani. Truthful auctions for pricing search keywords. In *Proceedings of EC'06: the 7th ACM Conference on Electronic Commerce*, pages 1–7, 2006.

[11] E. M. Arkin, S. W. Bae, K. Okamoto, A. Efrat, J. S. B. Mitchell, and V. Polishchuk. Geometric stable roommates. *Information Processing Letters*, 109(4):219–224, 2009.

[12] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, May-June 1998.

[13] J.J. Bartholdi and M.A. Trick. Stable matchings with preferences derived from a psychological model. *Operations Research Letters*, 5:165–169, 1986.

[14] G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[15] B. Birnbaum and C. Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.

[16] A. Bogomolnaia and H. Moulin. Random matching under dichotomous preferences. *Econometrica*, 72(1):257–279, 2004.

[17] K. Cechlárová and T. Fleiner. On a generalization of the stable roommates problem. *ACM Transactions on Algorithms*, 1(1):143–156, 2005.

[18] K.S. Chung. On the existence of stable roommate matchings. *Games and Economic Behavior*, 33(2):206–230, 2000.

[19] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, 1971.

[20] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[21] F.L. Delmonico. Exchanging kidneys - advances in living-donor transplantation. *New England Journal of Medicine*, 350:1812–1814, 2004.

[22] L. Dulmage and I. Halperin. On a theorem of frobenius-könig and j. von neumann's game of hide and seek. *Transactions of the Royal Society of Canada Section III*, 49:23–29, 1955.

[23] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[24] J. Edmonds. Path, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[25] T. Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45:233–284, 1992.

[26] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38(4):815–853, 1991.

[27] A.T. Gai, D. Lebedev, F. Mathieu, F. de Montgolfier, J. Reynier, and L. Viennot. Acyclic preference systems in P2P networks. In *Proceedings of Euro-Par 2007: the 13th International European Conference on Parallel and Distributed Computing*, volume 4641 of *Lecture Notes in Computer Science*, page 2007. Springer, 825-834.

[28] A.T. Gai, F. Mathieu, F. de Montgolfier, and J. Reynier. Stratification in P2P networks: Application to BitTorrent. In *Proceedings of ICDCS 2007: the 27th IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, 2007.

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

[30] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[31] P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Sciences*, 20:166–173, 1975.

[32] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[33] S. E. Gentry, D. L. Segev, and R. A. Montgomery. A comparison of populations served by kidney paired donation and list paired donation. *American Journal of Transplantation*, 5(8):1914–1921, August 2005.

[34] D.W. Gjertson and J.M. Cecka. Living unrelated donor kidney transplantation. *Kidney International*, 58:491–499, 2000.

[35] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *ACM*, 45(5):783–797, 1998.

[36] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

[37] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[38] P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.

[39] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[40] K. Hoffman and M. Padberg. Solving airline crew-scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993.

[41] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[42] C. Huang, T. Kavitha, D. Michail, and M. Nasre. Bounded unpopurality matchings. In *Proceedings SWAT'08: the 12th Scandinavian Workshop on Algorithm Theory*, volume 5124 of *Lecture Notes in Computer Science*, pages 127–137. Springer, 2008.

[43] A. Hylland and R. Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.

[44] Intervac. http://intervac-online.com/.

[45] R.W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6:577–595, 1985.

[46] R.W. Irving and D.F. Manlove. The Stable Roommates Problem with Ties. *Journal of Algorithms*, 43:85–105, 2002.

[47] R.W. Irving, D.F. Manlove, and S. Scott. The stable marriage problem with master preference lists. *Discrete Applied Mathematics*, 156(15):2959–2977, 2008.

[48] R.W. Irving, D. Michail, K. Mehlhorn, K. Paluch, and K. Telikepalli. Rank-maximal matchings. *ACM Transactions on Algorithms*, 2(4):602–610, 2006.

[49] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of STOC'90: the 22nd annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[50] A.-K. Katta and J. Sethuraman. A solution to the random assignment problem on the full preference domain. *Journal of Economic Theory*, 131(1):231–250, 2006.

[51] T. Kavitha, J. Mestre, and M. Nasre. Popular mixed matchings. In *Proceedings of ICALP 2009: the 36th International Colloquium on Automata, Languages and Programming*, volume 5555 of *Lecture Notes in Computer Science*, pages 574–584. Springer, 2009.

[52] T. Kavitha and M. Nasre. Optimal popular matchings. *Discrete Applied Mathematics*, 157(14):3181–3186, 2009.

[53] T. Kavitha and C. Shah. Efficient algorithms for weighted rank-maximal matchings and related problems. In *Proceedings of ISAAC '06: the 17th International Symposium on Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 153–162. Springer, 2006.

[54] S. Lahaie, D. Pennock, A. Saberi, and R. Vohra. Chapter 28: Sponsored search auctions. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[55] D. Lebedev, F. Mathieu, L. Viennot, A.-T. Gai, J. Reynier, and F. de Montgolfier. On using matching theory to understand P2P network design. In *Proceedings of INOC 2007: International Network Optimization Conference*, 2007.

[56] L. Lovász and M.D. Plummer. *Matching Theory*. Number 29 in Annals of Discrete Mathematics. North-Holland, 1986.

[57] M. Mahdian. Random popular matchings. In *Proceedings EC'06: the 7th ACM Conference on Electronic Commerce*, pages 238–242. ACM, 2006.

[58] D.F. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.

[59] D.F. Manlove and C. Sng. Popular matchings in the weighted capacitated house allocation problem. *To appear in Journal of Discrete Algorithms*.

[60] D.F. Manlove and C.T.S. Sng. Popular matchings in the capacitated house allocation problem. In *Proceedings of ESA'06: the 14th Annual European Symposium on Algorithms*, volume 4168 of *Lecture Notes in Computer Science*, pages 492–503. Springer, 2006.

[61] F. Mathieu. Self-stabilization in preference-based networks. In *Proceedings of P2P 2007: the 7th IEEE International Conference on Peer-to-Peer Computing*, pages 203–210. IEEE Computer Society, 2007.

[62] F. Mathieu. Upper bounds for stabilization in acyclic preference-based systems. In *Proceedings of SSS 2007: the 9th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, volume 4838 of *Lecture Notes in Computer Science*, pages 372–382. Springer, 2007.

[63] Richard McCutchen. The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In *Proceedings of LATIN'08: the 8th Latin American Symposium on Theoretical Informatics*, volume 4957 of *Lecture Notes in Computer Science*, pages 593–604. Springer, 2008.

[64] E. McDermid and R.W. Irving. Popular matchings: structure and algorithms. In *Proceedings of COCOON 2009: the 15th Annual International Computing and Combinatorics Conference*, volume 5609 of *Lecture Notes in Computer Science*, pages 506–515. Springer, 2009.

[65] K. Mehlhorn and D. Michail. Network problems with non-polynomial weights and applications. Unpublished manuscript, 2005.

[66] J. Mestre. Weighted popular matchings. In *Proceedings ICALP'06: the 33rd International Colloquium on Automata, Languages and Programming*, volume 4051 of *Lecture Notes in Computer Science*, pages 715–726. Springer, 2006.

[67] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|}\cdot|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of FOCS '80: the 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[68] R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[69] Netflix. http://www.netflix.com.

[70] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[71] NSW postgraduate medical internship allocation. http://www.imet.health.nsw.gov.au.

[72] National odd shoe exchange. http://www.oddshoe.org/index.php.

[73] G. O'Malley. *Algorithmic Aspects of Stable Matching Problems*. PhD thesis, University of Glasgow, Department of Computing Science, 2007.

[74] Peerflix. http://www.peerflix.com.

[75] Read it swap it. http://www.readitswapit.co.uk/.

[76] E. Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11:285–304, 1990.

[77] A. E. Roth and A. Postlewaite. Weak versus strong domination in a market with indivisible goods. *Journal of Mathematical Economics*, 4:131–137, 1977.

[78] A. E. Roth, T. Sönmez, and M. U. Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005.

[79] A. E. Roth and M. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, 1990.

[80] A.E. Roth, T. Sönmez, and M.U. Ünver. Kidney exchange. *Quarterly Journal of Economics*, 119(2):457–488, 2004.

[81] A.E. Roth, T. Sönmez, and M.U. Ünver. A kidney exchange clearinghouse in New England. *American Economic Review*, 95(2):376–380, 2005.

[82] A.E. Roth, T. Sönmez, and M.U. Ünver. Efficient kidney exchange: Coincidence of wants in a market with compatibility-based preferences. *American Economic Review*, forthcoming.

[83] E. Rothberg. Gabow's n^3 maximum-weight matching algorithm: an implementation, 1990. The First DIMACS Implementation Challenge.

[84] T. Roughgarden and M. Sundararajan. Is efficiency expensive? *3rd Workshop on Sponsored Search*, 2007.

[85] S. L. Saidman, A. E. Roth, T. Sönmez, M. U. Unver, and F. L. Delmonico. Increasing the opportunity of live kidney donation by matching for two and three way exchanges. *Transplantation*, 81(5):773–782, March 2006.

[86] T. Sandholm. Optimal winner determination algorithms. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*. The MIT Press, 2006.

[87] T. Sandholm and S. Suri. Side constraints and non-price attributes in markets. *Games and Economic Behavior*, 55(2):321–330, 2006.

[88] S. Scott. *A study of stable marriage problems with ties*. PhD thesis, University of Glasgow, Department of Computing Science, 2005.

[89] D. L. Segev, S. E. Gentry, D. S. Warren, B. Reeb, and R. A. Montgomery. Kidney paired donation and optimizing the use of live donor organs. *Journal of the American Medical Association*, 293(15):1883–1890, April 2005.

[90] C. Sng. *Efficient Algorithms for Bipartite Matching Problems with Preferences*. PhD thesis, University of Glasgow, Department of Computing Science, 2008.

[91] Swaptree barter exchange market. http://www.swaptree.com.

[92] J.J.M. Tan. A necessary and sufficient condition for the existence of a complete stable matching. *J. Algorithms*, 12(1):154–178, 1991.

[93] United Network for Organ Sharing (UNOS). http://www.unos.org/data/.

[94] UNOS pilot national kidney exchange market press release. http://www.unos.org/news/newsDetail.asp?id=1098.

[95] United States Renal Data System (USRDS). http://www.usrds.org/.

[96] W. Vickery. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[97] J. von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2:5–12, 1953.

[98] Y. Yuan. Residence exchange wanted: a stable residence exchange problem. *European Journal of Operational Research*, 90:536–546, 1996.

[99] L. Zhou. On a conjecture by Gale about one-sided matching problems. *Journal of Economic Theory*, 52(1):123–135, 1990.