The Stable Roommates Problem with Globally-Ranked Pairs

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Emmanuel Kobina Arthur

May, 2022

Approved for the Division
(Mathematics)

Dylan McNamee

Marcus Robinson

Acknowledgements

I want to thank my advisors Marcus Robinson, Dylan McNamee, and former advisor David Ramirez for their contributions in the form of feedback and insightful discussions about the direction of my thesis writing.

Contents

List of Tables

List of Figures

Abstract

This writing aims to introduce a variant of the stable matching problem in matching theory called the stable roommates problem with globally-ranked pairs (SR-GRP). There are three chapters in this writing.

The first chapter introduces the reader to mathematical and computer science fundamentals needed to understand SR-GRP such as graphs, augmenting, path, matching, reduction, and runtime complexity. SR-GRP is also formally defined here using these fundamentals.

The second chapter delves into how the SR-GRP is solved by an $O(min(n + R, R\sqrt{n})m)$ algorithm which finds a rank-maximal matching. n is the number of items to be paired up, R is the worst rank a pair could have, and m is the number of graphs created with respect to each rank. The chapter ends with a description of how the stable activities problem, relevant in peer-to-peer networking can be reduced to SR-GRP.

The last chapter delves mainly into the two common applications of SR-GRP: peer-to-peer networking and kidney exchange. In kidney exchange there is emphasis on how SR-GRP is used in the priority mechanism of optimizing the exchange. In peer-to-peer networking, there is a deeper dive into acyclic preferences with the property of acyclicity that allows its matching problems to be modeled as SR-GRP. The reduction described in chapter 2 plays a key role here.

This brief abstract provides a summary of what to expect in this thesis writing. Matching theory and SR-GRP are very young and very practical, and this is what will make this writing more interesting to readers with diverse backgrounds.

Dedication

I would briefly like to dedicate my work so far to two crucial helpers. My former thesis advisor David Ramirez and my current co-advisor Marcus Robinson. Your help with editing and other insightful recommendations have formed the heart of the shape of this work. A big thank you guys for aiding me in this entire writing process.

Introduction

Pairing items from a group is a seemingly simple task on a fundamental level. You can take one item from a group and randomly pick its partner and can continue to do this until everyone gets a partner. Alternatively, you could divide the items into two groups each containing half of the total number of items, then pair people up by taking each person in the first group and randomly selecting a partner from the unpaired items within the second group until no one is without a partner.

However, things become complicated when items in consideration have the vital power of preference, especially when the items are living humans or organizations run by humans. Using the almost trivial methods of pairing described above does not guarantee optimization in terms of happiness of the people in each pair. No one would want to be paired with his worst conceivable partner. This brings out the complexity of the pairing problem and that was thoroughly analyzed, surprisingly, very recently in in the late 20th century by David Gale and Lloyd Shapely.

These two mathematicians were investigating how they could optimally assign college applicants to colleges of their choice possible. In each college-student pair, no college and its paired partner students are mutually well off with different partners. To resolve this issue, the problem was simplified to what is known as the stable marriage problem, where a restriction of only one student admitted was imposed on colleges such that we have a problem similar to pairing men and women up with a pair consisting of one man and one woman who marry each other. By doing so Gale and Shapely came up with a quadratic runtime algorithm for solving this problem. This blew open many questions about various variations of matching problems involving stable pairs later. In this introductory chapter the aim will be to introduce briefly the variant of the stable matching problem introduce SR-GRP, and the structure of this thesis writing in terms of chapter content.

The Stable Roommates Problem with Globally Ranked Pairs (SR-GRP) is a generalized version of the stable roommates problem(SR) which together with its algorithm was introduced in a related paper Abraham et al. (2008) by David J. Abraham, Ariel Levavi,David F. Manlove, and Gregg O'Malley Abraham et al. (2008). In this version each conceivable pair of agents is put into fixed ordered ranks. There are two details that distinguish SR-GRP from the original stable roommates problem, and all other popular original versions of the other variants for that matter. First is that the problem takes indifference into account. Indifference is where an agent A cannot clearly state who they would prefer as their partner amongst multiple choices of other agents. The second is that we can have strongly stable matchings where everyone strictly prefers their partner to anyone else or weakly stable matchings where we can have some agents who are indifferent between their partner and some other agent(s).

It turns out that SR-GRP is really important in the field of health, more specifically in the pairwise kidney exchange process. Here, we have a situation where patient-donor pairs are matched with one other such pair for quick and emergent swapping. This happens because each pair consists of donor whose kidney isn't compatible with their partner patient. The crucial aspect of kidney exchange is in its optimization. Compatibility is dependent on crucial factors such as blood type. The famous optimization mechanism this writing focuses on is the priority mechanism. This mechanism optimizes the exchange process in such a way that patients are ranked in times of their likelihood of getting into a matching. The patients lower probabilities are given the highest priorities to make sure they matched. This is where SR-GRP comes in if the ranking of the patients are used to rank pairs of patients involved in a swap.

The second major real-world application of SR-GRP is in peer-to-peer networking systems. The network has a master preference list in which all peers for sharing transactions are ranked based on parameters such as bandwidth, latency and storage capacity. A peculiar added feature on which this is modeled as SR-GRP is the fact that there is also a ranking of pairs such that the rank is the sum of the ranks of the individual members of each pair within the master list. From this preference tables are created for each peer with each peer ranking other peers based off of the ranks of the pair involving that particular peer. From here the pairing process is modeled as SR-GRP. Details of these important real-world applications will be discussed in second to last chapter of this writing.

To conclude this introductory chapter, what follows is an explanation of the structure of this thesis. The first chapter is dedicated to familiarising the reader with essential mathematical terms and concepts needed to understand and appreciate SR-GRP. Essential terms and concepts in graph theory here will mostly be introduced here. Formal general and graph theoretic introductions to matchings and stability is at the heart of the this chapter. Stability is introduced in a general sense that covers the first two variants and in the more specific context of SR-GRP. At the end some essential terms in concepts of time complexity and reduction in Computer Science are introduced.

The second chapter is dedicated to discussing in detail the stable roommates problem more formally and then transitioning into a deeper dive into the SR-GRP model (including its algorithmic solution(s), and associated formal proofs) which is at the heart of this chapter. The third chapter is dedicated to discussing in detail how pairwise kidney exchange works using the SR-GRP model, emphasizing the importance of this application and the influence of the model on it. Following this in the same chapter is an in-depth description of the modelling of P2P as SR-GRP. In some parts of this writing, there are challenges for the reader to engage in. These challenges aim to enhance understanding of the material presented.

Chapter 1

Fundamental Concepts

This chapter formally introduces fundamental mathematical and computer science related concepts of basic relevance to the understanding of the stable matching problem and more specifically SR-GRP. Basic graph theory concepts will be introduced and this will lead into introductions to SR-GRP concepts in a graph theoretical sense.

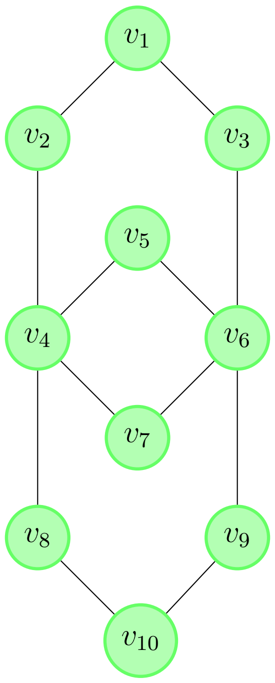


Figure 1.1: Illustration of a graph

1.1 The basics: What is a graph?

Definition 1.1.1. We define a graph to be a pair consisting of sets E and V. V is a set of vertices, which are endpoints of edges. They are normally depicted pictorially as labeled dots. E is a set of edges, each depicted as a straight lines drawn from one vertex to the another vertex.

In figure 1.1 below the vertices are represented as the green colored circles. So set $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$. The edge set is shown in figure 1.1 as well. Each edge is a set of two vertices, one being the *initial vertex* and the other being the *terminal vertex.* Eg in Fig 1.1 $\{v_1, v_2\}$ is an edge.

There are two types of edges: directed edges and undirected edges. A directed edge is an edge with an arrow head close to a vertex at one end, but no arrow heads at the other end. An undirected edge is an edge with no arrow heads close to vertices at either end. A graph that contains only directed edges are called *directed graphs*, whiles graphs that contain no directed edges are called *undirected graphs.* Examples of directed and undirected graphs are shown in Fig 1.3 and Fig 1.4 respectively. In directed graphs the sets representing each of the edges are ordered: the first element of the set is strictly the initial vertex and the second element is strictly the terminal vertex.

1.1.1 Fundamental terminology

Below is the essential terminology on basic graph theory:

- Two edges $A = \{u, v\}$ and $B = \{s, t\}$ are said to be *adjacent* if $A \cap B \neq \emptyset$ In other words there is at least one vertex that both A and B share. Also two vertices u and v are also said to be *adjacent* if there is at least one edge $e \in E$ such that e joins u and v. For example in fig 1.1 $\{v_1, v_2\}$ and $\{v_1, v_3\}$ are adjacent edges but $\{v_1, v_2\}$ and $\{v_3, v_4\}$ are not adjacent. Also v_1 and v_2 are adjacent vertices but v_2 and v_5 aren't.

- A vertex u is said to be *incident* to an edge $\alpha \in E$ if $u \in \alpha$. In Fig 1.1 for example, v_1 and v_2 are incident vertices to the edge $\{v_1, v_2\}$.

- A *path* is a series of vertices and edges such that neither edges nor vertices are repeated. We represent a path by a sequence $a_1 a_2 \cdots a_k$ where if $i, j \in \{1, \cdots, k\}$ and $i \neq j$, then $a_i \neq a_j$ and $a_i, a_j \in V$. In a directed graph all edges must be oriented in the same direction. For example in Fig 1.3 $d_1 d_2 d_3 d_5$ is a directed path. An example of this in Fig 1.1 is $v_1 v_2 v_4 v_5$.

- A *cycle* is a series of vertices and edges such that edges are not repeated and vertices are not repeated except for two specific ones: the starting vertex and the ending vertex,which are one and the same. The representation is similar to the one in the description of a path just that $a_1 = a_k$. An example of this in Fig 1.1 is $v_4 v_5 v_6 v_7 v_4$. The latter definition relates generally to undirected graphs. For directed graphs the definition is the same but we want all the edges in the cycle to move in same direction just like in the definition of a path above. An example of this in Fig 1.3 $d_1 d_2 d_3 d_1$. A graph is said to be *cyclic* if it is itself a cycle or contains a cycle. The graph in Fig 1.3 for example is a cyclic graph.

- Given a graph $G = (V, E)$ and $G' = (V', E')$, G' is a *subgraph* of G if $V \subseteq V'$ and $E \subseteq E'$. An example of a subgraph is $(V' = \{v_4, v_5, v_6, v_7\}, E' = \{\{v_4, v_5\}, \{v_4, v_7\}, \{v_5, v_6\}, \{v_6, v_7\}\})$.

- A graph is said to be *connected* if there is a path between any two vertices in the graph. Fig 1.1 is a connected graph.

- A graph $G = (V, E)$ is said to be *bipartite* if we can split V into two disjoint subsets say A and B such that $A \cup B = V$ and any edge $e \in E$ is a pair $\{u, v\}$ such that $u \in A$ and $v \in B$

- A *component* of a graph $G = (V, E)$ is a subgraph $G' = (V', E')$ such that G' is connected and that there is no edge $u, v \in E$ such that $u \in V'$ and $v \notin V'$. Consider the graph in Fig1.2 it has two components: First is one with vertex set $\{a, b, c, d, e, f\}$ and another with the vertex set $\{g, h, i\}$. The former is said to be an *even component* because it has an even number of vertices and the latter is said to be an *odd component* because it has an odd number of vertices.

- A graph is said to be *simple* if it contains no loops or multiple edges (more than one edge between two nodes). One can view a *loop* as a cyclic subgraph of the original graph. Figure 1.2 is an example of a simple graph

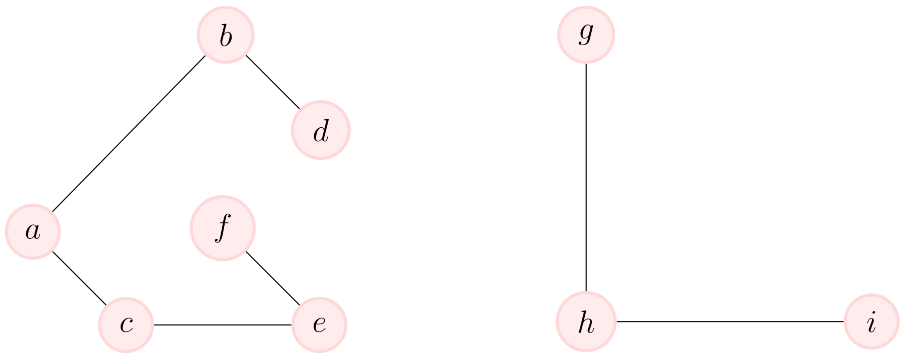


Figure 1.2: A disconnected graph with two components

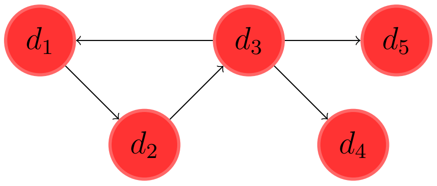


Figure 1.3: A directed graph

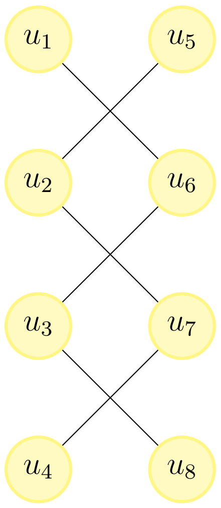


Figure 1.4: A bipartite graph

1.1.2 What are Matchings

Definition 1.1.2. Given a graph $G = (V, E)$ a matching M is a subset of E such that no two edges $e, e' \in M$ are adjacent. A vertex $u \in V$ is said to be *matched* if there is an edge $e \in M$ such that $u \in e$. The partner of this matched vertex u is denoted as $M(u)$. On the other hand a vertex $v \in V$ is said to be *exposed* if there is no $e \in M$ such that $v \in e$.

An example of a matching from fig 1.2 is $\{\{u_1, u_6\}, \{u_2, u_7\}, \{u_3, u_8\}\}$. In reference to the following matching in Fig1.1: $\{\{v_2, v_1\}, \{v_4, v_5\}, \{v_6, v_7\}, \{v_9, v_{10}\}\}$, v_1 is a matched vertex and v_8 is an exposed vertex.

Exercise 1.1 How many matchings are there in a graph?
Answer: There isn't a generalized method of counting the number of matchings in any graph. So the answer here is that it would depend on the type of graph we're dealing with. What follows is a definition for two relevant types of matching.

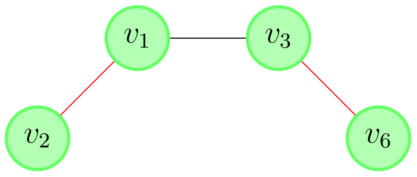
Definition 1.1.3. Let $\mathcal{F}$ be the set of matchings in a graph G. A *maximum matching* is an $M \in \mathcal{F}$ such that $|M|$ is maximized. This means the maximum matching includes as many vertices in V and edges in E as possible A matching M is perfect if every vertex in G is incident to an edge in M.

An example of a maximum matching from the graph in Fig 1.2 is $\{\{b, d\}, \{a, c\}, \{f, e\}, \{g, h\}\}$. From the bipartite graph in Fig 1.3, the set $\{\{u_1, u_6\}, \{u_2, u_5\}, \{u_3, u_8\}, \{u_4, u_7\}\}$ is a perfect matching. There are two types of paths that are essential to understanding matchings and SR-GRP. First is an *alternating path* with respect to a matching M which is a path that contains edges in M and not in M alternately. The other is an *augmenting path* with respect to M which is an alternating path where both the starting and ending vertices are unmatched. Consider the matching,$\{\{u_2, u_7\}\}$ from Fig 1.3. Also take a look at the path $u_5u_2u_7$. This path is alternating because

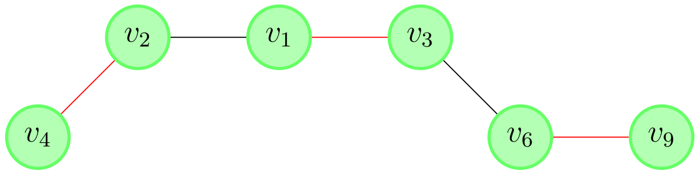
$\{u_2, u_7\}$ is in our considered matching and $\{u_5, u_2\}$ isn't. Additionally $u_5u_2u_7u_4$ is an augmenting path because u_4 and u_5 are exposed relative to our considered matching. One may ask if an augmenting path can consist of just a single edge? The answer is yes and by definition of an augmenting path, the edge must not be part of the reference matching. It is possible to build a maximum matching using alternating and augmenting paths. The process is described as follows:

1. Consider the graph in Fig 1.1 We assigned the following labels to its edges:$A = \{v_4, v_2\}$, $B = \{v_3, v_6\}$, $C = \{v_9, v_{10}\}$, $D = \{v_1, v_2\}$,$E = \{v_1, v_3\}$, $F = \{v_4, v_5\}$, $G = \{v_5, v_6\}$, $H = \{v_4, v_7\}$, $I = \{v_7, v_6\}$, $J = \{v_8, v_4\}$, $K = \{v_6, v_9\}$, $L = \{v_8, v_{10}\}$

2. Say we had the matching $\{E\}$. When we add the neighboring edges on both ends of E, D and B to form a path DEB, we get an augmenting path with respect to E. We non-augment by making our reference matching $\{D, B\}$ instead of $\{E\}$ as shown below.Note that non-augmenting means changing the reference matching, which turns an augmenting path into a non-augmenting one (ie an ordinary alternating path).

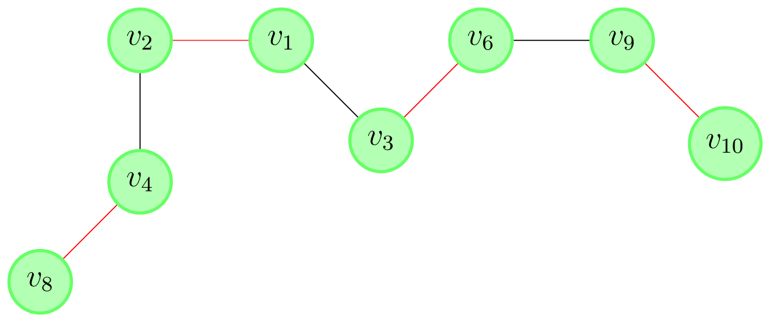


3. Adding edges A and K to left and right sides of DE respectively we get the augmenting path $ADEBK$ still with our reference matching being $\{D, B\}$. We non-augment by changing our reference matching to $\{A, E, K\}$ as shown below.



4. Adding J and C to the left and right sides of DE respectively, we have another augmenting path $JADEBKC$ with our reference matching still $\{A, E, K\}$. We non-augment by changing the reference matching to $\{J, D, B, C\}$ as shown below.

5. Now we can't create another augmenting path by adding the last adjacent edge to C, L, so we end here and the maximum matching we've found is our reference matching $\{J, D, B, C\}$. Therefore the length of the maximum matching is 4.



Challenge 1.2 Generate an algorithm for the steps outline above for find a maximum matching and give a runtime analysis of this algorithm **Challenge 1.3** Write working code in Python or C++ that mimics the algorithm for exercise 1.4 above

1.1.3 Gallai-Edmond's Decomposition (GED)

Given any maximum matching the set of all maximum matchings, Ω in a graph $G(V, E)$, we can use Ω to partition the vertex set V of the graph into three disjoint sets. To get these three disjoint sets, we first find the set of all the vertices that occur in every maximum matching in Ω. Call this set κ. Within κ, find all of the vertices u such that $M(u) \in \kappa$ for all $M \in \Omega$. Such us are called *perfectly demanded vertices* with the set of such vertices, which becomes the first partition, is denoted as $GED - P[G]$. Note that for any $u \in GED - P[G]$, $M(u) \in GED - P[G]$. Next, we get the second partition which is the set of vertices in κ but not in $GED - P[G]$. This set is denoted as $GED - O[G]$, the set of *overdemanded vertices*. Now the last partition, if not obvious at this point, is the set of vertices that are not in κ. Denote this set as $GED - U[G]$, the set of *underdemanded vertices*. This partitioning of the vertex set, V just described is known as the *Gallai-Edmonds Decomposition*. Consider the graph in Fig 1.4 below. If we consider every maximum matching in this graph it happens that $GED - P[G] = \{w_8, w_9, w_{10}, w_{11}\}$; $GED - O[G] = \{w_2, w_4\}$; $GED - U[G] = \{w_1, w_3, w_5, w_6, w_7\}$

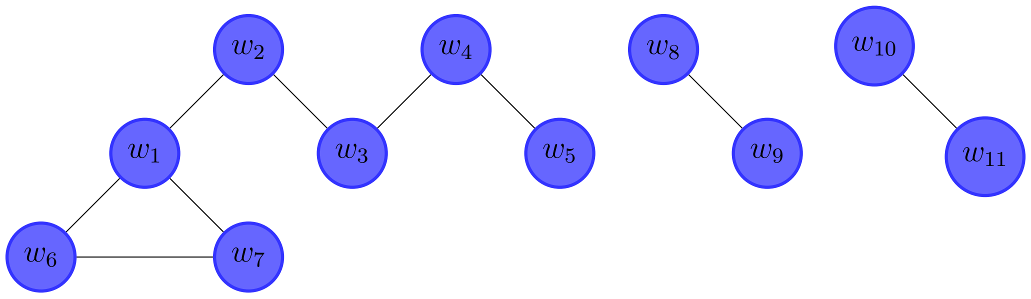


Figure 1.5: A graph to illustrate GED

Based on this decomposition, the lemma given as follows was derived to give some useful properties.

Lemma 1. In any maximum matching M of G

(1) For all u in $GED-O[G]$, $M(u)$ is in $GED-U[G]$
Explanation: Say $M(u)$ was in $GED-P[G]$. Then it must be that $u \in GED-P[G]$, which is a contradiction. Also say $M(u) \in GED-O[G]$ then it must be that both u and $M(u)$ are in $GED-P[G]$ which is also a contradiction. Hence the claim above is valid. Remember that $GED-O[G]$,$GED-U[G]$, $GED-P[G]$ are partitions of V and are by definition disjoint. In the graph in Fig 1.4 for every possible maximum matching the $M(w_2)$ is either w_1 or w_3, both of which are underdemanded. Same can be said of $M(w_4)$ which could be either w_3 or w_5.

(2) For all even components C_e of $G \setminus GED-O[G]$
(i) $C_e \subseteq GED-P[G]$ and (ii) $M(u) \in C_e$, for all u in C_e.
Explanation: The proof of this centers around some definitions that are beyond the scope of this writing. Details are found in Bry & Las Vergnas (1982) Consider the graph in Fig 1.5 which is G without $GED-O[G]$ vertices. We have these even components with these vertex sets:$\{w_8, w_9\}$ and $\{w_{10}, w_{11}\}$, with $w_8, w_9, w_{10}, w_{11} \in GED-P[G]$.

(3) For all odd (cardinality) components C_o of $G \setminus GED-O[G]$
(i) $C_o \subseteq GED-U[G]$ (ii) $M(u) \in C_o$, for all u in C_o except one say, v and (iii) either v is unmatched in M or $M(v)$ is in $GED-O[G]$
Explanation: The proof of this just like (2) centers around some definitions that are beyond the scope of this writing. Details are found in Bry & Las Vergnas (1982) Consider the graph in Fig 1.5 which is G without $GED-O[G]$ vertices. We have odd components with these vertex sets: $V' = \{w_1, w_6, w_7\}$, $V'' = \{w_3\}$,$V''' = \{w_5\}$, with $w_1, w_6, w_7, w_3, w_5 \in GED-U[G]$. Consider the matching $M = \{\{w_1, w_6\}, \{w_2, w_3\}, \{w_4, w_5\}\}, \{w_8, w_9\}, \{w_{10}, w_{11}\}\}$. In V', w_7 is unmatched, w_3 in V'' is matched with an overdemanded vertex (w_2), and w_5 in V''' is matched with an overdemanded vertex (w_4).

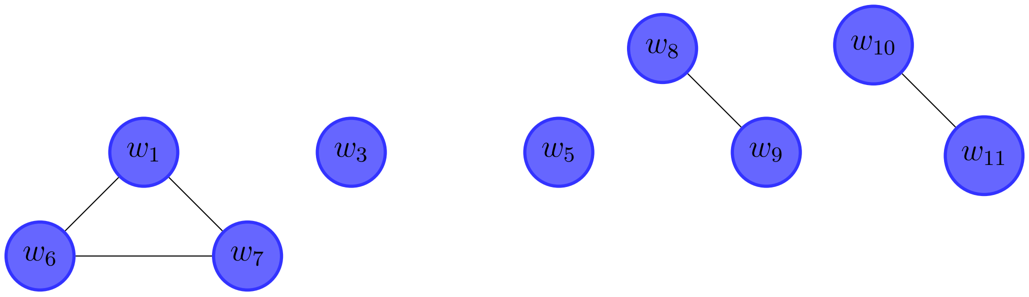


Figure 1.6: A graph in Fig 1.4 but with GED-O[G] vertices removed

1.2 The basics in SR-GRP context

Before we define SR-GRP formally, there these key terms defined below:

1. An *agent* is any member that is being paired in matching. In graph theory, agents are represented by vertices.

2. A *preference list* of an agent is an ordered set of other distinct agents. If the agents *don't* each have a preference list as just defined then the preference list would be just an ordered set of possible pairs of distinct agents. The SR-GRP model has the latter kind of preference list. It is possible though to convert the preference satisfying the former definition into one satisfying the latter. This conversion is very relevant in peer-to-peer networking.

3. A *preference table* is a set of preference lists of each of the given agents. This is normally illustrated as an n by n table where n is the number of agents. If each agent does not have a preference list then the preference list with ordered pairs becomes the preference table.

Since SR-GRP is the generalization of the stable roommates problem SR we give a definition that describes SR below:

Definition 1.2.1. The stable roommates problem (SR) is the matching problem where each agent has a preference list of all other distinct agents, with a strict order imposed in each preference list. In graph theory, SR can be modeled as a *non-bipartite* directed graph with each edge weight being the position assigned to each agent by a particular agent in their preference lists. The graph of which SR is modeled as a non-bipartite because every agent can be possibly paired with every other agent.

The algorithm that solves SR is called Irving's algorithm and the explanation of the details of how it works can be found in Irving (1985).In the context of SR, the emphasis on strict order in the preference lists is based on feasibility because if we include indifference in any of the preference lists, finding a matching becomes a hard problem.Normally with preference tables and rankings, every other agent is ranked in a particular agent's preference list but in a more general sense, we could have a situation where some other agent other than the agent owning the preference list in the table is not included at all. This means that for some agent A, he cannot fathom pairing up with some other agent B and so does not include B in his preference list. Therefore we call (A, B) an *unacceptable* pair and we call (A, C) where C is in A's preference list as an *acceptable* pair. When we imagine the preference list of A as the set of possible pairings that include A, the pair (A, B) will not be a member of this set, but (A, C) would be a member. It would be very convenient that the preference list of each agent in a preference table is thought of as a set of acceptable pairs to easily model the *mutually acceptable* pairs as edges. A pair $\{a, b\}$ would be mutually acceptable if $\{a, b\}$ is in both a and b's preference lists. Below is a formal definition of the SR-GRP model:

Definition 1.2.2. SR-GRP modeled as a graph $G(V, E)$ where V, the vertices represent all agents. Now, if we let γ_i be the preference list of acceptable pairs for agent i where $i \in \{1, 2, ..., n\}$, then the set of edges in the graph E would be $E = \bigcup_{i=1}^{n} \gamma_i$. A formal statement of the derivation of preferences in SR-GRP is with a ranking function $rank : E \to \mathbb{N}$. So an agent u prefers agent v to agent w if $e = \{u, v\}$ and $e' = \{u, w\}$ and $rank(e) > rank(e')$. On the other hand still assuming $e = \{u, v\}$ and $e' = \{u, w\}$, u is *indifferent* between v and w if $rank(e) = rank(e')$.

Now let's introduce some notation here that would be relevant to Chapter 2. We define E_i to be the set of edges with rank i and $E_{\leq i}$ to be the set of edges $\bigcup_{j=1}^{i} E_i$. Additionally let $n = |V|$ be the number of agents in consideration and $m = |E|$ be the number of mutually acceptable pairs. Without loss of generality, we can say here that the largest possible rank for any edge in a graph is m.

1.2.1 Matchings in SR-GRP context

We can define a matching, $M \subseteq E$ as a set of edges (i.e., mutually acceptable pairs) such that if $x, y \in V$, and $x \in E$ and $y \in E$, then $x \cap y = \emptyset$. Here below are some points to note about matchings in SR-GRP context:

1. We say u is matched in M *iff* there exists an edge, $e \in E \cap M$ such that $u \in E$. If $\{u, v\}$ is the pair in M then define $M(u) = v$ and $M(v) = u$.

2. An agent u prefers one matching M' to another matching M if:
(i) u is matched in M' and not in M, and
(ii) u prefers $M'(u)$ to $M(u)$.

3. An agent u on the other hand is *indifferent* between M and M':
(i) $u \notin M'$ and $u \notin M$
(ii) $M'(u) = M(u)$.

4. A pair $\{u, v\}$ is said to be *strongly blocking* for M if $\{u, v\} \notin M$ and both u and v prefer the matching M' over M such that $\{u, v\} \in M'$. Consider the Table 1.1. Also let the matchings $M = \{\{b, c\}, \{a, d\}\}$ and $M' = \{\{a, b\}, \{c, d\}\}$. $\{a, b\}$ is a strongly blocking pair of M because both a and b *strictly* prefer M' to M.

5. According to Abraham et al. (2008), a matching M is weakly stable if there's no M' such that there exists a pair $\{u, v\} \in M$ such that u and v prefer M' ie M does not *admit* any strongly blocking pair, $\{u, v\}$. For example from Table 1.1,$\{\{a, b\}, \{c, d\}\}$ is a weakly stable matching. The algorithm of focus in Chapter 2 finds weakly stable matchings in polynomial time.

6. According to Abraham et al. (2008), a pair $\{u, v\}$ is said to be *weakly blocking* for the matching M if $\{u, v\} \notin M$ and u prefers $M' = u, v$ to M while v either prefers M' to M or is indifferent between them. Consider the Table 1.1. Also let the matchings $M = \{\{b, c\}, \{a, d\}\}$ and $M' = \{\{b, d\}, \{a, c\}\}$. $\{b, d\}$ is a

Preference Table with ranks for each pair				
Agents	a	b	c	d
a	N/A	1	3	3
b	1	N/A	2	2
c	3	2	N/A	1
d	3	2	1	N/A

Table 1.1: Preference Table with ranks for each pair

strongly blocking pair of M because both d *strictly* prefers M' to M, and b is indifferent between M and M'

7. According to Abraham et al. (2008), a matching M is *strongly stable* if there's no M' such that there exists a pair $\{u, v\} \in M$ such that any of u and v prefer M' to M or is indifferent between M' and M ie M does not *admit* any weakly blocking pair, $\{u, v\}$. For example from Table 1.1,$\{\{a, b\}, \{c, d\}\}$ is a strongly stable matching.

Note that from the examples given in points 5 and 7 above, the matching $\{\{a, b\}, \{c, d\}\}$ is strongly stable and weakly stable. That's because a closer look at the definitions reveal that strongly stable matchings are weakly stable matchings but not vice-versa.

1.3 The basics of time complexity and reduction

1.3.1 Intro to time complexity

Time is an important physical quantity in any scientific endeavor. Especially in physics the study of time has helped shape our understanding of our physical reality. Time in the physical sciences is measured in seconds, and efficiency in most science and engineering endeavour have subtle change in smaller magnitudes of seconds. However time in computer science is perceived and measured quiet differently. In computer science time is generally measured in steps which could take a time of a few seconds. That's because computer scientist care alot about feasibility of programs and not necessarily accuracy in terms of speed. Another way time is perceived differently in computer science is that the number of steps is actually measured as a function of the size of an input fed into a computer program.

To get a sense of the feasibility of the program in these time measurements, we use other functions as bounds and there are two cases of time we normally get from these bounds. They are average-case time,and the worst-case time. In this subsection, we look at how the worst case time is expressed as what is called the big-O or big-Oh notations of functions used as bounds.

Big-Oh Notation

Before we give the formal definition of the big-Oh notation, these steps below summarize given an algorithm for a program, how a computer scientist will measure its running time complexity expressing it as a big-O notation.

1. Express the number of steps the algorithm takes to run its input as a function of the size of the input. For example if the input size is n, the number of steps will be $f(n)$ where f is a function.

2. Find the minimum known function we could use as an upper bound if increased by a positive integer factor

3. This runtime complexity is expressed as big-O of this minimum known function. Eg. $O(n^3)$, where n^3 is the known function from the previous step.

Below is the formal definition of big-O

Definition 1.3.1. If f and g are partial functions from $\mathbb{N}$ to $\mathbb{R}^+$, we say that $f = O(g)$, (pronounced as *big-Oh* of g) if for some positive numbers C and N, $f(n) \leq Cg(n)$ for every $n \geq N$.

One can see that this definition is connected with all steps outlined above. Now we take a look at popular types of time complexities that are each distinguished by their distinct functions used as upper bounds. The elaboration of these types are taken from Team (2016) and here they are as follows.

1. **Constant Time:** An algorithm is said to have constant time if the number of steps is generally not dependent on its input size. It's big-O notation is $O(1)$.

2. **Linear Time:** An algorithm is said to have a linear time complexity when the running time increases linearly with the length of the input. Its function involves just checking all input values given. It's big-O notation is $O(n)$.

3. **Quadratic Time:** Here an algorithm is said to have a non – linear time complexity where the running time increases non-linearly (n^2) with the length of the input. Generally, nested loops come under this time complexity order where for one loop takes $O(n)$ and if the function involves a loop within a loop, then it goes for $O(n) * O(n) = O(n^2)$ order.

4. **Logarithmic Time:** An algorithm is said to have a logarithmic time complexity when the size of the input data is reduced for every constant number of steps taken. This indicates that the number of operations is not the same as the input size. The number of operations gets reduced as the input size increases. Algorithms with Logarithmic time complexity are found in binary trees or binary search functions. This involves the search of a given value in an array by splitting the array into two and starting searching in one split. This ensures the operation is not done on every element of the data.

5. **Exponential Time:** This is the time-complexity where the output of the function of steps increases exponentially by some constant integer factor. Exponential time complexity is usually seen in Brute-Force algorithms. These algorithms solve a problems through exhaustion: they go through all possible choices until a solution is found. Although they are generally consistent, brute-force algorithms are usually not optimal ways of solving problems. Brute-Force algorithms are used in cryptography as attacking methods to defeat password protection by trying random strings until they find the correct password that unlocks the system.

6. **Factorial Time:** This is the time complexity in which the function depicting the number of steps for execution of an algorithm is the factorial of the input size. This is considered the worst time-complexity in terms of feasibility and the famous Travelling Salesman problem is an example of a problem whose solution has such a time complexity. Translating this time complexity to real time execution by a computer should indicate astronomical magnitude of years to run.

Examples of how the runtime is derived from sample programming examples can be found at Team (2016) and Karumanchi (2015). The algorithms that have constant, linear, and quadratic time complexities generally fall under the general time complexity category whose big-O notation is of the form $O(n^k)$ where k is an integer. These time-complexities are known as *polynomial* time complexities because n^k is a polynomial. Such time-complexities for algorithms are a sign of the feasibility of the algorithms in general. Evidence of this is that when a real computer runs the algorithm as a program it would take a reasonable amount of real time to execute. Logarithmic time complexities are even much more ideal in terms of feasibility because it can take seconds or even maximum minutes for a computer to execute in real time. However the exponential and factorial time complexities are not feasible time complexities for algorithms. An algorithm that has exponential or factorial time complexities could take, for a reasonable input size, thousands of years to execute by a computer in real time. For example the algorithm for the Travelling Salesman problem could take an estimated 1,928 years for a computer to execute on input size 20. For an input size 60 it would take 13.9×10^{67}years which is greater than the estimated age of the universe.

1.3.2 Reduction

Reduction is an essential problem solving method in Computer Science, Mathematics, Engineering and the Natural Sciences. It is the process of transforming a given problem into an easier problem whose solution is known, such that the solution of that other problem induces a solution to the original problem. Take a very practical example like this: You have arrived at the Reagan International Airport in Washington D.C and wish to visit the White House. However since this is your first time visiting the city, you don't know the directions to your destination. Fortunately for you, you have your phone with cellular internet connection and the Google maps app.

Now you can reduce the problem of finding the right directions to the White House by looking up the right direction on Google maps. This is possible by the definition of reduction given because you already know how to find the directions between any two locations.

In computer science, lots of problems are solved by using reduction but reduction is also very central in also determining the time complexity of an algorithm and how feasible that time-complexity is. If a problem is reduced to one whose solution is not feasible in terms of time complexity, then the original problem is most likely not feasible. Another crucial component of determining feasibility in reductions is the feasibility of the reduction itself. If the reduction itself takes polynomial time it helps in making the accrued time complexity of the original problem feasible. Same cannot be said when the reduction itself has exponential and factorial time complexities. Below are summary steps of reduction taken from Radford (2022):

1. To solve an instance of problem A:
   - i Transform the instance of problem A into an instance of problem B
   - ii Solve the instance of problem B
   - iii Transform the solution to problem B into a solution of problem A
2. We say that problem A reduces to problem B

These two examples from Radford (2022) demonstrate reduction

1. Finding the Least Common Multiple (LCM) of 2 integers. Eg LCM(24, 36) = 72.
   Solution technique: reduce LCM to Greatest Common Divisor (GCD):
   - (a) Instance of Problem A: LCM(24, 36)
   - (b) Instance of Problem B: GCD(24, 36)
   - (c) Solution of B: GCD(24, 36) = 12 (by Euclid's algorithm)
   - (d) Transform solution of B into solution of A: LCM(24, 36) $= \frac{24\times36}{GCD(24,36)} = \frac{24\times36}{12} = 72$
2. Solve Element Uniqueness problem: given a set of two dimensional points, are all elements unique?
   Solution technique:
   - (a) Reduce Element Uniqueness to Closest Pair
   - (b) Solve Closest Pair
   - (c) Transform solution of Closest Pair to solution of Uniqueness: Answer to Element Uniqueness is yes if Closest Pair distance > 0

Chapter 2

The Stable Roommates Problem with Globally-Ranked Pairs

2.1 Introduction

In this chapter focus will be mainly on SR-GRP but we start here with a brief description of the original stable roommates problem (SR). Here we again have a set of agents and a preference table containing preference lists of each agent. By running Irving's algorithm, we can find an $O(N^2)$ running time to either determine if a stable matching does not exist or, if it exists, produce one. One key known feature here of an SR model is that there are no ties in any preference list within the preference table. Now consider a situation where some agent is indifferent between multiple other agents in his preference list. If we wanted to use Irving's algorithm to solve this problem we would have to consider multiple instances of breaking the ties such that we have a strict order of preference in the preference lists as before. The importance of considering multiple instances is the fact that for one instance, we may have no stable matching existing which may not guarantee this same result in the original problem with ties. From an intuitive mathematical perspective there are exponential ways of breaking such ties and this will result in an exponential runtime for an alteration of Irving's algorithm to cater for all these instances.

This is not ideal from an optimization standpoint, and it turns out that the stable roommates problem with ties is in general a computationally hard problem. An article which most of the content here will be based of off presents a polynomial time algorithm for finding a rank-maximal weakly stable matching. This algorithm nicely generalizes the Irving's algorithm without increasing complexity. In this chapter we talk about some essential results that makes it possible to model our SR problem with ties as SR-GRP. We continue by talking about the polynomial time rank-maximal matching algorithm and proofs of its correctness. Lastly we conclude by talking about the b-matching problem which is actually relevant in peer-to-peer networking.

Preference Table with preference lists for each agent			
Agents/Rank	1st	2nd	3rd
1	4	2	3
2	1	3	4
3	2	4	1
4	2	1	3

Table 2.1: Preference Table with preference list for each agent

2.2 What makes a matching problem model SR-GRP

To find out if a graph of a matching problem can be modeled as SR-GRP, another model called the SR-GAP model. An SR problem can be modeled as SR-GAP if the possible pairs can be put in an ordered sequence such that loops or cycles are not formed. Here are the steps for finding out if an instance I of SR with $G = (V, E)$ can be modeled as SR-GAP:

1. Construct a digraph (in short for directed graph) $P(G)$ with each vertex representing an edge, $e = \{u, v\} \in E$
2. Draw an arc arrow from e to $e' = \{u, w\}$ if u prefers w to v
3. Now for each $e, e' \in E$, merge them into one vertex if u is indifferent between v and w
4. I therefore belongs to SR-GAP iff P(G) is acyclic.

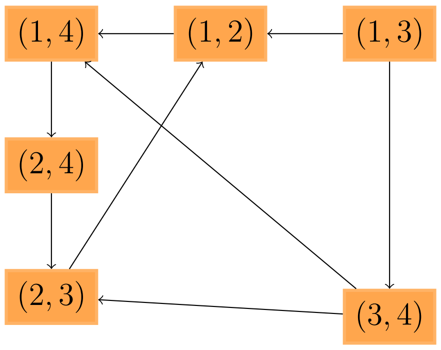


Figure 2.1: A directed graph with pairs as vertices

Consider the SR instance given as preference table in Table 2.1. Going by the steps above gives the digraph in Fig 2.1. Notice that there is the cycle $(1,4)(2,4)(2,3)(1,2)(1,4)$. Therefore this SR instance is not SR-GAP and hence not SR-GRP. Based on the fact that there must be an absence of cycles in P(G) as a requirement for SR-GAP, we can produce a master preference list based off of all of the individual preference lists with monotonically improving ranks. This master preference list can be generated using a suitable rank function from a running reverse topological sort algorithm on P(G), i.e., rank(e) < rank(e') if and only if e appears before e'. The following proposition holds: 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.

There are two desirable properties that the SR-GRP restriction of SR has that the original SR does not have here. One is the fact that SR-GRP admits weakly stable matchings to account for indifference. Second is as was mentioned in the chapter 1, finding a weakly stable matching or showing that it does not exist is a polynomial time problem and not NP-Hard as is the case for SR. Let's prove two useful Lemmas for later on. However first let $G = (V, E_1 \cup, \cdots, \cup E_m)$ such that E_i is all edges of rank $i \in \{1, \cdots, m\}$,and $E_{\leq i} = E_1 \cup E_2 \cup \cdots \cup E_n$. Now the lemma:

Lemma 2. 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. $\implies$ Assume for the sake of contradiction that $M \cap E_{\leq i}$ is not a maximal matching of $E_{\leq i}$ for all i. That means that there exists an $i \in \{1 \cdot n\}$ such that $M \cap E_{\leq i}$ is not a maximum matching. That means there is an edge $e = \{u, v\} \in E_{\leq i}$ such that both u and v are not matched in M. This makes u and v a strongly blocking pair for M and hence M is not weakly stable. Therein lies the contradiction.

$\impliedby$ Assume for the sake of contradiction that M is not weakly stable. This means there is some weakly blocking pair $e = \{u, v\}$ which form an edge of some rank r. This means neither u nor v is matched in $M \cap E_{\leq r}$ and so $M \cap E_{\leq r}$ is not weakly stable and therefore not maximum since $r < n$. Here lies the contradiction.

□

Lemma 3. M is a strongly stable matching of G if and only if $M \cap E_i$ is a perfect matching of $E^\alpha = \{e \in E_i : e \text{ is not adjacent to any } e' \in E_{<i}\}$ for all i.

Proof. $\impliedby$ For the sake of contradiction, assume that M is not strongly stable. That means it admits a weakly blocking pair $\{u, v\} \in M$. Let r be the rank of $\{u, v\}$. There are two cases to consider here: First case is when both u and v prefer each other to their partners in M. This means no edge in $M \cap E_{<i}$ is incident on either u or v, otherwise $\{u, v\}$ wouldn't be a blocking pair in the first place. Therefore $\{u, v\} \in E^\alpha$ and hence $M \cap E_r$ is not a perfect matching because both u and v are unmatched. Therein lies the contradiction. Consider the case where u prefers v to $M(u)$ but v is indifferent between u and $M(v)$. Once again we get a situation where $\{u, v\} \in E^\alpha$ and $M \cap E_r$ is not a perfect matching because u is unmatched. This is also a contradiction.

$\implies$ For the sake of contradiction, assume that $M \cap E_i$ is not a perfect matching of E^α. In the first case, v is indifferent between u and $M(v)$, and u strictly prefers v to $M(u)$. In the second case u strictly prefers v to $M(v)$ and v strictly prefers

u to $M(u)$. Therefore by definition, in both cases, $\{u, v\}$ is a weakly blocking pair and so M is not strongly stable which is a contradiction.

$\square$

It is very difficult to find a strongly stable matching because E_i along the way can produce no perfect matchings. However a strongly stable matching can be found in $O(m\sqrt{n})$ time using Micali and Vazirani's algorithm for finding the maximum matching in a non-bipartite graph. This improves on the best known running time of $O(m^2)$ for general SR, according to Abraham et al. (2008). Due to this reason and the fact that weak stable matchings are used mainly in kidney exchange, this chapter will focus on weakly stable matchings and its algorithmic construction. Looking back at $SR - GAP$ sufficiency for $SR - GRP$, Chung Chung (2000) showed that the no "odd ring" condition of preferences is sufficient for the existence of weakly stable matchings which makes the acyclic condition for $SR - GAP$ more restrictive in the sense that no odd or even rings are permitted. Also from Lemma 4 we can see that a weakly stable matching could be found in $O(m + n)$ time, where n is the number of vertices and m is the number of ranks, by these steps:

1. Finding a maximal matching using the rank-1 edges.
2. Remove edges with matched vertices from rank 1 in rank 2.
3. Repeat steps 1 and 2 for rank 3 and so on.

In this chapter however, we don't focus on finding just any maximum matching, we look for maximal matchings that are rank maximal. Here's the definition below:

Definition 2.2.1. A rank maximal matching is a matching that includes the maximum possible number of rank-1 edges and subject to the maximum number of rank-2 edges and so on.

The above definition brings out the fact that in finding the rank-maximal matching for every rank i some matched vertices in rank $i - 1$ could be removed. We could more formally define the *signature* of a matching M as $\langle s_1, s_2, \cdots, s_m \rangle$, where s_i is the number of rank-i edges in M. Matching is rank-maximal *iff* it has the lexicographic maximal signature amongst all matchings. Another thing to note in relation to Lemma 5 is that a rank-maximal matching is strongly stable if it is perfect for each rank. An illustration of how matching can transition to instability to weak stability to strong stability.

Matching, $M \xrightarrow{rank-maximal}$ Weak $\xrightarrow{perfect}$Strong.

A rank-maximal matching is by definition a weakly stable matching so it works well as a good replacement for strongly stable matching in terms of optimization. *Irving et al 06* gives an $O(min(n+R, R\sqrt{n})m)$ algorithm for the problem of finding a rank-maximal matching in a bipartite graph. R in the runtime complexity just given is the rank of the worst ranked edge in the matching. With regards to optimization there are two cases which would be considered in this chapter for weakly stable matchings:

1. Egalitarian: The aim here is to minimize the sum of the ranks of the edges within a weakly stable matching.

2. Minimum regret: The aim here is to minimize the number of weakly blocking pairs in a weakly stable matching.

For a general instance I of SR, each of 1 and 2 above have been found to be NP-Hard problems. The time-complexity of the problem related to 2 is until now an open problem. There is an obvious greedy away of finding rank-maximal matchings which involves finding rank-maximal matchings at a based-level and exponentially improving upon that with each rank. This is captured by the algorithm of Gabow and Tarjan in Gabow & Tarjan (1991) which takes $O(K_2\sqrt{n\alpha(m,n)log(m)log(n)})$ time and this is worse than the improved runtime of $O(min(n+R, R\sqrt{n})m)$ of our combinatorial algorithm that avoids this exponentiation in improving weights.

2.3 Rank-maximal matching algorithm

In this section we explore the combinatorial algorithm that runs in $O(min(n+R, R\sqrt{n})m)$ time as described in Abraham et al. (2008). To reiterate, R is the rank of the worst ranked edge. To summarize the whole algorithmic process, according to Abraham et al. (2008), first of all we let $G_i = (V, E_{\leq i})$. Then we construct a maximum matching M_1 on G_1. Automatically M_1 is rank maximal over G_1 by definition. Now recursively given the matching M_{i-1} which is rank maximal over G_{i-1}, we augment M_{i-1} with edges in E_i to construct a rank-maximal matching over G_i. Remember augmentation means constructing new augmenting paths to find new matchings with increased number of edges. To ensure rank-maximality though, certain edges need to be deleted before augmentation. These edges are called *safe edges* as they would not take away rank-maximality in previous graphs in the looping in any way but may interfere with rank-maximality in the current G_i. The augmentation after the safe edge deletion is done by a fast maximum matching algorithm by Micali and Vazirani Micali & Vazirani (1980). It is worth noting however that in a non-bipartite setting, an extra deletion occurs. After this shrinking certain components into supervertices is done before augmentation. The steps taking in each looping step for building the rank-maximal matching as depicted by the algorithm in consideration are below:

1. Safe edge deletion: Deleting the edges that are redundant or may interfere with rank maximality.

2. Shrinking: Encapsulating certain components into *supervertices.*

3. Augmentation: Constructing rank-maximal matchings for current i^{th} subgraphs by adding edges into M_{i-1} to form new augmented paths.

2.3.1 Safe edge deletion

Consider the graph $G_2 = (V, E_{\leq 2})$ which we aim to find the rank maximal matching for as our first inductive step in the algorithm. Here the rank-maximal matching M_1

of $G_1 = (V, E_1)$ is available and as our first step we need to delete safe edges. An edge $e = \{u_1, v_1\}$ is considered to be *safe* in this scenario if it has the following properties:

1. $u \in GED - O[G_1]$ and $v_1 \in GED - O[G_1] \cup GED - P[G_1]$. This edge is safe because by the decomposition lemma for all maximal matchings in G_1, all $GED - O[G_1]$ vertices are matching only with $GED - U[G_1]$ edges. Hence any edges that connect vertices in $GED - O[G_1]$ with either vertices in $GED - O[G_1]$ or those in $GED - P[G_1]$ have become redundant and can safely be deleted.

2. $e \in E_{\geq 2}$ and $u \in GED - O[G_1] \cup GED - P[G_1]$. This is also a valid safe edge because by performing the deletions in 1 above and securing a rank-maximal matching for G_1, all edges that the $GED - O[G_1]$ vertices are connected to that aren't in G_1 will not be part of our subsequent maximal matching constructed at the higher rank graphs. Therefore they are redundant and need to be deleted. This is similar for all edges with $u \in GED - P[G_1]$ because they are already matched with some $v \in GED - P[G_1]$ by the decomposition lemma. This deletion also optimizes our rank-maximal matching because if we wanted to maintain these edges in consideration we may have to delete the edges in G_1 with $u \in GED - O[G_1] \cup GED - P[G_1]$ instead to create an overall maximal matching. This not only ruins the the rank-maximality of M_1 but also moves more edges in our overall maximal matching for G into the higher valued ranks which are less preferred.

3. $e \in E_{\geq 2}$ and both u and v belong to the same odd component of G_1. These are the edges not in G_1 and by part3 of the Lemma u and v are underdemanded vertices matched with each other or some other underdemanded vertices. It is therefore clear to see why such an edge between u and v needs to be deleted both for optimization of the preservation of the rank-maximal matching of G_1. Note that this third deletion type is required for non-bipartite graphs, since only one vertex in each odd component is unmatched internally. This makes sense because u and v would be in the same set in the set duo of the graph. Because of this there is no edge between them by definition of the bipartite graph and so they wont be matched in the odd component of G_1.

Warning about safety of edges: One may ask why we do not just delete all other edges in G_1 that are not part of our rank-maximal matching in consideration? The problem is that edges such that $u \in GED - O[G_1] \cup GED - U[G_1]$ and $v \in GED - U[G_1]$ can be part of other valid rank-maximal matchings in G_1. In addition recall that maximal matchings change during the process augmentation and so to accommodate for this change, these edges in consideration here need to be maintained and not deleted. Also unsafe edges that are much more obvious are the ones with $u \in GED - P[G_1]$ and $v \in GED - P[G_1]$. That's because these edges are automatically part of the maximal matching in G_1 and no alternating path through G_1 would remove such edges from the maximal matching. We now move on to the next step of shrinking.

2.3.2 Shrinking

At this stage we shrink each odd component, C of G_1 into a *supervertex.* By shrinking we mean perceiving each component as a giant vertex hence the name "supervertex". Recall from the third part of the decomposition lemma and the safe edge deletion criteria given in the previous subsection that there is one vertex in each odd component that is unmatched internally. We call this vertex the *root* of C. Relative to an external matching, a supervertex is matched iff its root is matched. This makes sense because by part3 of the GED Lemma the root could either be unmatched or matched with an overdemanded vertex. If the root is matched with an overdemanded vertex then the entire odd component (ie the supervertex) is matched . Now adding $e = \{u, v\} \in E_{\geq 2}$ to G where $u \in C$ and $v \notin C$, we replace e with an edge between v and C's supervertex. It's as though the node u has been replaced by the supervertex. This perceptual view of the supervertex is essential for understanding augmentation. Note that along the way in the running of the algorithm we may be dealing with graphs with supervertices which in turn contain their own supervertices in a kind of recursive sense. 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. The *top-level vertices* are those underdemanded vertices in the each odd component that were matched internally originally with each other. Consider the graph below in Fig 2.2. The two supervertices are ones with the vertex sets: $V_1 = \{a_9, a_2, a_4\}$, and $V_2 = \{a_6, a_7, a_8\}$. Also take the matching $M = \{\{a_9, a_4\}, \{a_1, a_2\}, \{a_3, a_6\}, \{a_7, a_8\}\}$. M is a legal matching because the roots a_2 and a_6 of the supervertices V_1 and V_2 respectively are unmatched internally ie. they are unmatched *within* their supervertices relative to M. The thick edges in Fig 2.2 are the edges matched in M. Relative to the matching $M = \{u, v\}$ above

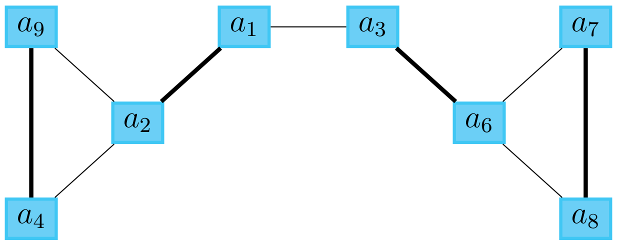


Figure 2.2: A graph demonstrating a legal matching

we could trivially augment by removing $\{u, v\}$ and include the two rank-2 edges. However this destroys the rank-maximal matching. Remember that an augmenting path is an alternating path with a unmatched vertices at both ends. By shrinking the triangle into a supervertex, O, we get the alternating paths O_y and O_x. Since y and x are unmatched we have our augmenting paths. Remember that augmenting a graph means using an augmenting path P you delete all $e \in P \cap M$ and replace it with $f \in P/M$. These augmenting paths Ox and Oy are beautifully expanded within the supervertex O by removing and adding one rank 1 edge. Hence we have a *legal* augmenting path in the original graph without altering the number of matched edges

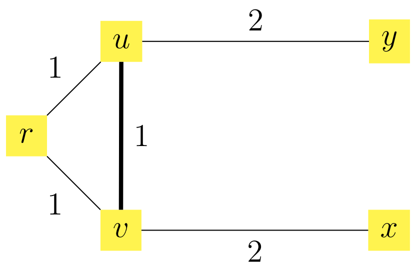


Figure 2.3: A supervertex, O containing rank 1 edges with rank two edges connected to it

in the supervertex. Quick note that G'_i i the graph we get after deleting safe edges in G_i and including relevant edges just before and during augmentation to create a maximum matching that has edges in $E_{\leq i}$ (ie the edges in rank j where $j \leq i$). Onward now to the final step of augmentation.

Algorithm 1: Rank-Maximal-Matching

```
**Data:** $G = (V, E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_m)$
**Result:** $M_m$, such that $M_m$ is rank-maximal
Set $G'_1$ to $G_1$;
Let $M_1$ be any maximum matching of $G_1$;
**for** $i = 2$ *to* $m$ **do**
    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 $E_{\leq i}$ where (i) $e$ is incident on a $GED - O[G'_{i-1}]$ 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 $\{u, v\}$ 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**
**return** $M_m$
```

2.3.3 Augmentation

The approach used in this step is to find an augmenting path P while regarding each top-level supervertex in G'_i as a regular vertex. Let's let this top supervertex be C. In Fig 2.4 and Fig 2.5, C has the vertex set $\{a, v, r\}$. To make recognizing C as a supervertex easier, one can imagine a circle drawn around C with only the vertices touch the circumference of the circle. After finding P, we expand P through C in the following way:

1. Find a vertex u in C through which a path enters via an unmatched edge relative to the maximum matching in C. Let's call the maximum matching M_i. Note though that if we have multiple augmenting paths through the supervertex, we choose the one which enters the supervertex through an unmatched edge.

2. Check if u is the root, r. If so then C is unmatched because r is unmatched. This can be seen in Fig 2.4. Now since C is unmatched we replace C by $\approxeq = r$. One can view this replacement as collapsing all of the supervertex (its edges and vertices) into r.

3. Otherwise if $u \neq r$ the we have two cases:
   (i) C is unmatched. This can be seen in Figure 2.4 where P enters via $\{a, y\}$ or $\{v, x\}$
   (ii) P leaves C via a matched edge incident on r. This can be seen in Fig 2.5 where P enters via $\{a, y\}$ or $\{v, x\}$ and leaves via a matched edge $\{z_1, r\}$
   There is a lemma that says there is an even length alternating path between u and root r. In case, we augment P by replacing C with the even-length alternating path.

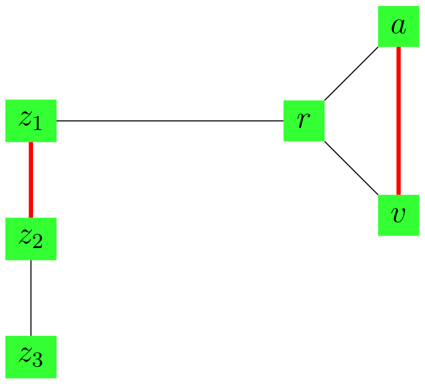


Figure 2.4: A supervertex, C with an augmenting path $P = z_3 z_{21} z_1 C$ entering via an unmatched edge through r

Challenge 2.3: Write recursive pseudocode that simulates this augmentation of M_i in G_i'

2.3.4 Correctness lemmas

We prove some lemmas that follow.

Lemma 4. Let M be a legal matching on some vertex C with root r. Let u be any other node in C. Then there is an even-length alternating path from u to r.

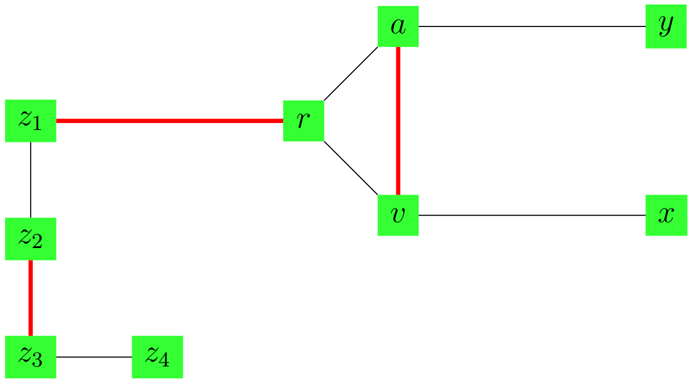


Figure 2.5: A supervertex, C with the path $P = z_4z_3z_2z_1Cy$ entering via an unmatched edge through u

Proof. Consider decomposition M be a legal matching of C in which u is unmatched (such a matching is guaranteed by the decomposition lemma). u and r cannot both be in one of M and M' because relative to M, r is unmatched and relative to M', u is unmatched. So if we have a path from u to r, each vertex encountered would be in both M and M'. Also if one edge incident on such a vertex is in M, the other edge incident on such a vertex would be in M'. Since there are an odd number of vertices between u and r, the path from u to v is even in the number of edges. □

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 establish the correctness of the algorithm. Lemma 5 proves that no rank-maximal matching contains a deleted edge. Lemma 6 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 7 proves that the final matching is rank-maximal on the original graph G.

Lemma 5. 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. $M \cap E_{\leq i-1}$ is rank-maximal for G_{i-1} and a legal maximal matching from G'_{i-1}. In constructing G'_i, the edges deleted do not belong to any maximal matching of G'_{i-1} by the GED-Lemma. Since $M = (M \cap E_{\leq i-1}) \cup (M \cap E_i)$, we prove that $M \cap E_i$ has no deleted edges. To prove this assume there is an $e \in (M \cap E_i)$ that is deleted by the algorithm. We delete any edge, e which is incident on $GED - O[G'_{i-1}]$ vertex or incident a $GED - P[G_{i-1}]$ vertex ,or e has both of its vertices in the same odd component of G'_{i-1}. In each of these cases, $e \in M \cap E_i$ means that $E \cap E_{\leq i-1}$ cannot be a rank-maximal matching of G_{i-1}, which gives the required contradiction. □

Lemma 6. Let M_i and M_j be 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 will have edges within the top-level vertices of G' and between top-level vertices of G'_i. We already know that augmenting through a supervertex doesn't change the number of edges in the supervertex, and this is true in a recursive manner. Hence M, the matching we're building, contains the same number of edges as M_i.By the GED Lemma, there are edges in M_i incident on $GED - O[G'_i]$ and $GED - P[G'_i]$ supervertices. Since these vertices are matched in M_i, they will be matched in M_j, and still augmenting won't affect the matching status. Also no edges of rank worse than i will be incident on such vertices matched M_i due to deletions. Hence $|M_j| \leq |M_j \cap E_{\leq i}|$. But $|M_j \cap E_{\leq i}| \leq |M_i|$ since all edges from $E_{\leq i}$ in G_j are also in G'_i (the first step within the for loop of the algorithm justifies this) and so M_i is a maximal legal matching of G_i. □

Lemma 7. For every i, the following statements hold:

(i) Every rank maximal matching of G_i is a legal maximum matching of G'_i

(ii) M_i is a rank-maximal matching G_i

Proof. (i) Using induction, for $i = 1$, $G_1 = G'_1$, so with the rank-1 edges any maximum matching on G_i is also legal and hence maximum matching of G_i is a legal one of G'. This proves the base case. We now have the inductive hypothesis to be that every rank-maximal matching of G_{i-1} is a legal maximum matching of G'. From the hypothesis and Lemma 5, every rank-maximal matching of G_i is contained in G'_i. Call a rank-maximal matching of G_i, N_i and the legal maximum matching of G'_i, M'_i. Since the signature of N_i is $\langle s_1 \cdot s_i \rangle$ is the same as that of M_i, then the two matchings N_i and M_i must be legal maximum matchings.

(ii) This has the same base case as (i). By the inductive hypothesis and lemma 5, every rank maximal matching in G_i is contained in G'_i. By Lemma 6, M_i has the same signature as M_{i-1} up to rank $i - 1$ so the signature of $M_i = \langle s_1 \cdots s_{i-1}, t_i \rangle$ where $t_i \leq s_i$, since M_{i-1} is a rank-maximal matching of G_{i-1}. From (i), $\underline{M_i \text{ is a maximal legal matching of } G'_i \text{ and } t_i = s_i}$ and M_i is a rank-maximal matching of G_i. The underlined part of the latter statement is true because t_i is the number of edges we add from G_i to M_{i-1}. After this addition we get M_i the rank-maximal matching of G_i. Hence $s_i = t_i$.

□

2.3.5 Runtime analysis

1. In each iteration computing the decomposition(given a maximum matching), deleting edges, and shrinking components all take $O(m)$ time. That's because in doing these three activities we go through some portion of the number of ranks, m. The worst case is when we go through all of the ranks, after which the portion we go through is the entirety of m.

2. Constructing M_i from M_{i-1} requires $|M_i| - |M_{i-1}| + 1$ augmentations. That's because to create M_i we need to add the $|M_i| - |M_{i-1}|$ edges to M_{i-1}. This is done during augmentation. However we end our augmentation by adding an extra edge which makes our augmenting path non-augmenting. Therefore the total number of augmentations is $|M_i| - |M_{i-1}| + 1$. Remember here that an augmentation is the addition of consecutive matched and unmatched edges to increase the size of our augmenting path as much as possible.

3. At the top-level of augmenting (when supervertices are now vertices after shrinking) we can use Micali-Vazirani(MV) algorithm to find the maximum matching in every supervertex.

4. Using the MV algorithm at 3 yields a runtime of O(min($\sqrt{n}$, ($|M_i| - |M_{i-1}| + 1)m$)). The proof of this is found in Micali & Vazirani (1980).

5. Now we need to expand each augmenting path P through the supervertices. Letting u be the vertex through which an unmatched edge in P enters a supervertex C. The expansion can be done in linear time written as the size of C by appending a dummy unmatched vertex d to u and the looking for an augmenting path from d to the root r in C. Each supervertex will belong to at most one such augmenting path in each round of the MV-algorithm,this expansion does not affect the runtime. The reason for this can be found in Micali & Vazirani (1980). According to Abraham et al. (2008), it follows that after R iterations runtime is at most $O(min(n + R, R\sqrt{n})m)$.

2.4 B-matching generalization of SR

Here we study a generalization of the stable roommates problem where each agent has a non-acyclic integer quota on the number of possible partnerships. This normally called the stable multiple activities problem (SMA) and it is relevant in the study of the peer-to-peer application of SR-GRP. This relevance exists because it happens that SMA problems can be reduced to an instance of the stable roommates (SR) problem. Once this happens, the guarantee of acyclicity of preferences will allows us to then model it as SR-GRP. Below is a description of SMA:

- An instance of the stable multiple activities problem is a triple (G, O, b) where $G = (V, E)$ is a finite multigraph, O is a set of linear orders $\prec_v$ for $v \in V$, $\prec_v$ is an order on the set $E(v)$ of edges incident with v, and $b : V \longrightarrow N$ is a quota function on the vertices. A *multigraph* is a graph where there can be multiple edges connecting two nodes. With regards to the linear orders here, we're basically saying that every vertex places an ordering (or ranking) on every edge it is incident on and these rankings are then placed in set O. O can then be said to be equivalent to a preference table for SR.

- We say that subset F of E *b-dominates* edge e of E if there is a vertex, v of e and different elements $f_1, f_2, \cdots, f_{b(v)}$ of F such that $f_i \prec_v e$ for $i = 1, 2, \cdots, b(v)$. In other words there at least the quota number of edges that v prefers to e.

- We denote by $D^b(F)$ the set of edges that are b-dominated by F. A subset M of E is a stable matching of SMA instance (G, O, b) if $D^b(M) = E \backslash M$. This stable b-matching definition implies that the only set of edges that are b-dominated by M are the set of edges that are not in M.

- The linear order $<_v$ on the edges incident on v is such that v prefers edge a to edge b if $a <_v b$.

- Note that the fact that edges are ranked for each vertex makes it possible to reduce SMA to SR-GRP.

Also by definition of set equality (two sets are equal if they're subsets of each other) we can say that M is a stable b-matching if $D^b(M) \subseteq E \backslash M$ and $E \backslash M \subseteq D^b(M)$. The former condition means that each vertex must be incident with at most $b(v)$ edges of M, while the latter condition says that for any edge e outside M, there is a vertex $m_1, m_2, \cdots, m_{b(v)}$ of M such that $m_i \prec_v e$ for $i = 1, 2, \cdots, b(v)$.

To understand the reduction to SR we need to understand a stable matching problem similar to SMA called the stable activity problem (SA). Here is a description of it as follows:

- Let's have $G = (V, E)$ a finite multigraph. For each vertex v to V, we have linear order $\prec_v$ on the set of edges $E(v)$ that are incident on v.

- If O denotes the set of linear orders $\prec_v$ (for $v \in V$) the (G, O) is called an instance of the stable activities problem. Note the similarity between this and SR in terms of the order of neighbors of each vertex.

- We say that a subset F of E *dominates* edge e of E if there is a vertex v of V and an edge f of F such that $f \prec_v e$.

- Let $D(F)$ denote the set of edges that are dominated by F. A stable matching SA instance (G, O) is a subset M of E with the property that $D(M) = E \backslash M$.

- Each SR instance (G, O) can be considered to be an instance (G, O^*) of the SA, if we let $<_v$ on the neighbors of v induce linear order $<_v$ on $E(v)$ for each vertex v. This is a useful clue of the possibility of performing reductions from SA to SR and perhaps viceversa.

- First consider $D(M) = E \backslash M$. Then M is a set of disjoint edges otherwise some edge of M would be dominated by M. So M is a matching and each edge outside M is dominated by M showing that M is a stable matching of SR instance (G, O). If M is a stable matching of SR instance (G, O), then there is no edge that blocks any edge in M which means every edge in M is preferred to any edge outside M. Hence $D(M) = E \backslash M$

The theorem below states the fact that SA can be reduced to SR.

Theorem 8. Let (G, O) be an instance of the SA, and let SR instance (G', O') be constructed from (G, O) as described below. There is a stable matching of (G, O) if and only if there is a stable matching of (G', O'). Moreover any stable matching of (G', O') induces a stable matching of (G, O) and each stable matching (G, O) can be induced by some matching of (G', O').

According to Cechlárová & Fleiner (2005), the G' for SR is constructed by substituting each edge $e = \{u, v\}$ of G by a 6-cycle with two hanging edges. The proof of the theorem is also found in Cechlárová & Fleiner (2005).

It is well known that there's always an existing stable matching of the SMA instance (G, O, b) whenever G is finite, bipartite and simple. The Gale-Shapely algorithm has an extension that finds one. This extension was more generally supposed to solve the college admission problem whose bipartite graph had a many-to-one property. Generalizing this feature in the SMA context, we say SMA instance (G, O, b) has the many-to-one property if $b(u) = 1$ or $b(v) = 1$ for each edge $e = \{u, v\}$ of $E(G)$. Define the instance (G^b, O^b) SA by $V(G^b) = \{v^i : v \in V(G) \text{ and } i = \{1, 2, \cdots, b(v)\}\}$; $E(G^b) = \{e^{i,j} = \{u^i, v^j\} : e = \{u, v\} \in E(G) \text{ and } u^i, v^i \in V(G^b)\}$; $O^b = \{\prec_{v^i} : v^i \in V(G^b)\}$, where for $e = \{u, v\}$, $f = \{u, w\}$

$$e^{i,j} <_{u^i} f^{i,k} \Longleftrightarrow \begin{cases} e <_u f \text{ or} \\ v = w \text{ and } j < k \end{cases}$$

That is instead of vertex v of G, we introduce $b(v)$ equivalent copies and the linear orders are essentially inherited from SMA instance. Consider a simple diagram of SMA to SA below. $V(G) = \{u, v, w\}$ and $V(G^b) = \{u^1, u^2, v, w\}$. Also for the ordering take $e = \{u, v\}$ and $f = \{u, w\}$. It can be seen that $e^{1,1} <_{v^1} f^{1,2}$ and $e^{1,1} <_{u^1} f^{1,1}$ are true based on the definition of the orderings.

The following lemma states the fact that we can reduce a many-to-one version

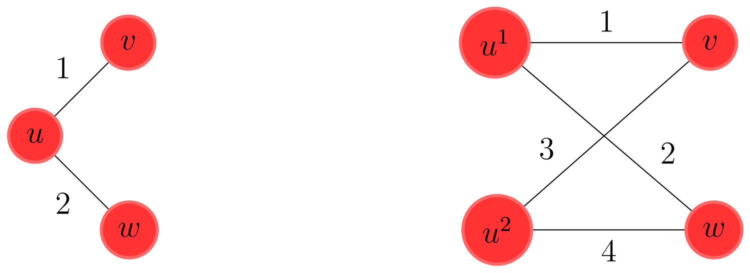


Figure 2.6: An instance of SMA(left) with its equivalent instance in SA(right)

.

of SMA to SA.

Lemma 9. Let (G, O, b) be an instance of the SMA with the many-to-one property. There is a stable b-matching of (G, O, b) if and only if there is a stable matching of (G^b, O^b). Moreover, any stable matching of (G^b, O^b) induces a stable b-matching of (G, O, b) and each stable b-matching of (G, O, b) can be induced by a stable matching of (G^b, O^b)

Cechlárová & Fleiner (2005) provides the proof of this. According to Cechlárová & Fleiner (2005), we can reduce the SMA with the many-to-one property to SA by lemma 9. However we can't use this same method of reduction for reducing the general SMA to SA. This reason is that not all stable matchings of (G^b, O^b) would correspond to a stable b-matching of (G, O, b). This is brought about by disjoint copies of a single edge of (G, O, b) may appear in a stable matching of (G^b, O^b). We would then have a situation where a certain edge is selected more than once in a stable b-matching which is not possible. Consider the following definition

Definition 2.4.1. For SMA instance (G, O, b), define SMA instance (G', O', b') by defining G' and O' just as was defined for a new SR instance used in showing the reduction of SA to SR. Let $b'(v) := b(v)$ if v is a vertex of G and let $b'(v) := 1$ otherwise.

As different vertices of G are not adjacent in G', we have the following observation: the SMA instance (G', O', b') in the latter definition above has the many-to-one property. The next lemma gives the reduction from SMA to SMA with many-to-one property.

Lemma 10. Let (G, O, b) be an instance of the SMA and (G', O', b') be the instance of the SMA in definition 2.4.1 above. There is a stable b-matching of (G, O, b) if and only if there is a stable b'-matching of (G', O', b'). Moreover any b'-matching of (G', O', b') induces a stable b-matching of (G, O, b), and each stable b-matching of (G, O, b) can be induced by some stable b'-matching of (G', O', b').

The proof of the lemma above found in Cechlárová & Fleiner (2005) involves using the same proof for theorem 8 but just perform the following semantic replacements: SR $\mapsto$ SMA with the many-to-one property; SA $\mapsto$ SMA; (G, O) $\mapsto$ (G, O, b); (G', O') $\mapsto$ (G', O', b'); and matching $\mapsto$ b-matching, or b'-matching, whichever applies. Combining Lemma 9, Lemma 10 and the observation from definition 2.4.1, we get a reduction of the SMA to SR. The theorem below formally summarizes that.

Theorem 11. Let (G, O, b) be an instance of the SMA. There is a stable b-matching of instance (G, O, b) if and only if there is a stable matching of instance $((G')^{b'}, (O')^{b'})$ of the SR. Moreover, any stable matching of $((G')^{b'}, (O')^{b'})$ induces a stable b-matching of (G, O, b), and each stable b-matching of (G, O, b) can be induced by a stable matching of $((G')^{b'}, (O')^{b'})$.

The proof of this theorem according to Cechlárová & Fleiner (2005) involves checking that $(G')^{b'}$ is simple. However this follows from showing the simplicity of G'. Now that it is confirmed that SMA can be reduced to SR, we can reduce SR to SR-GRP if the SR reduction result satisfies the SR-GAP check. The diagram below shows the reduction transitions from SR to SR-GRP

$$\begin{tikzcd}
\boxed{\text{SMA}} \arrow[r] & \boxed{\text{SMA (Many-to-one)}} \arrow[r] & \boxed{\text{SA}} \arrow[r] & \boxed{\text{SR}} \arrow[r, "Acyclity"] & \boxed{\text{SR-GRP}}
\end{tikzcd}$$

Figure 2.7: Reduction transitions from SMA to SR-GRP

.

2.5 Conclusion

In this chapter, we explored SR-GRP in detail which at its core held the description of an algorithm that found the rank-maximal matching. After that we explored the stable multiple activities problem (SMA) which is on the surface, the problem of matching systems in peer-to-peer networks. This connects to SR-GRP in the sense that SMA can be reduced to SR-GRP. Due to the reduction possibility we can in a sense view SMA as a generalization of not just SR but also of SR-GRP.

Chapter 3

Applications of SR-GRP

3.1 Introduction

In this chapter we explore two common real-world applications of SR-GRP: Kidney exchange and peer-to-peer networking. Certain concepts from chapter 2 will be found to be essential in these applications described.

3.2 Kidney exchange

3.2.1 Introduction

Many people suffer from kidney failure and need healthy kidneys as replacement for their defective ones. In the US more than a 100,000 people are on the waiting list for transplants every year. One idea of getting healthy kidneys available for these patients is to used deceased ones but maintenance of such deceased kidneys can be difficult because of the gradual decay. A better idea will be to get kidneys of healthy living patients as most of the cells are active and can readily be available for patients. This idea is also feasible because any living person can still survive with one kidney instead of two. Unfortunately though having a living donor is not always enough. The donor kidney must be compatible with a patient especially in terms of blood type. If the blood types of the patient and donor don't match, we say the patient-donor pair is incompatible. An example of compatibility and incompatibility is that a patient with O blood type is incompatible with every other blood type except O. Another examples is that a donor with blood type AB can only donate to a patient with blood type AB. From the above diagram, suppose that P_1 of blood type A is paired is paired with donor D_1 of blood type B and P_2 of paired with D_2 of blood type A. Since they are aware of each other's incompatibilities with their donor partners, P_1 and P_2 can exchange donors as shown above. This is what the system of *kidney exchange* is about. In this section we look at the two ways in which kidney exchange can be done with special focus on the kidney exchange through applying matching algorithms. For the coverage of the exchange through matching algorithms we look at how this is modeled as a maximum matching problem. Then we finally cover the

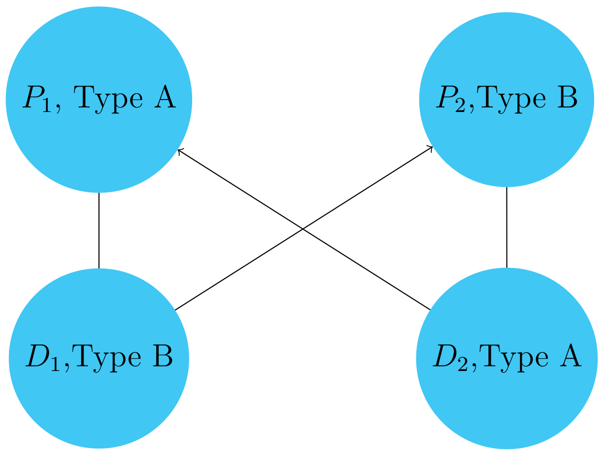


Figure 3.1: An illustration of a kidney exchange between two patient-donor pairs

optimizations of the exchange by matching with focus on the priority mechanism. Priority mechanisms enable the kidney exchange problem to be modeled as SR-GRP.

3.2.2 Kidney Exchange by TTC method

According to Roughgarden (2013), in this method,we model kidney exchange as a house allocation problem treating patient-donor pairs as agents and incompatible living donor as a house. A patient's total ordering over donors is defined according to the probability of transplant success based of off factors such as blood type, tissue type etc. The aim is to produce cycles like this in the figure below.

Reallocating donors according to this two-length cycle corresponds to what happens

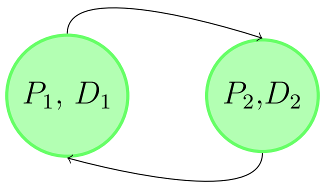


Figure 3.2: An illustration of a two cycle that occurs in the TTC process

in the Fig 3.1 above. This allocation also improves probability of a successful transplant. The deal breaker here is that TTC will sometimes produce cycles that aren't of optimum length two in Fig 3.2 above. Instead it produces a cycle like this:

$$(P_1, D_1) \longrightarrow (P_1, D_2) \longrightarrow \cdots \longrightarrow (P_n, D_n) \longrightarrow (P_1, D_1)$$

The problem with poducing cycles greater than length 2 is that. According to Roughgarden (2013),a cycle of length two (Fig 3.1) already corresponds to four surgeries—two to extract donors' kidneys, and two to implant them in the patients. Moreover, these four surgeries must happen simultaneously. In addition, from Roughgarden (2013) Incentives are the reason: in the example in Fig 3.1, if the surgeries for P1 and D2 happen first, then there is a risk that D1 will renege on her offer to donate her kidney to P2. One problem is that P1 unfairly got a kidney for free. The much more serious problem is that P2 is as sick as before and, since her donor D2 donated her kidney, P2 can no longer participate in a kidney exchange. Because of this risk, non-simultaneous surgeries are almost never used in kidney exchange. The constraint of simultaneous surgeries, with each surgery needing its own operating room and surgical team, motivates keeping reallocation cycles as short as possible. A final critique of the TTC approach ,according to Roughgarden (2013), is that modeling preferences as a total ordering over the set of living donors is overkill: empirically, patients don't really care which kidney they get as long as it is compatible with them. Binary preferences over donors, which is covered in the next subsection, is therefore more appropriate.

3.2.3 Kidney Exchange by applying Matching Algorithms

Using matching algorithms for kidney exchanges has proven to be more effective than TTC approaches. This has been for reasons given above in the description of the nature of TTC. What follows is an exploration in some points how kidney exchanges are setup to be solved as matching problems.

- Let $N = \{1, 2, \cdots, n\}$ denote the set of patients each of whom have one or more incompatible donors.
- Each patient is indifferent between all compatible donors but strictly prefers compatible donors to incompatible ones.
- $\succsim_i$ is the preference relation for patient i over the set of patients N.
- $j \succ_i i$ means that patient j has a compatible donor for patient i.
- $i \succ_i j$ means patient i has no compatible donor for patient j.
- $j \sim_i h$ both j and h each have compatible donors for patient i
- $\succ_i$ denotes strict preference and $\sim_i$ denotes indifference relation induced by $\succsim_i$
- Pairwise kidney exchange problem is a pair $(N, \succsim)$ where $\succsim = (\succsim_i)_{i \in N}$ denotes list of patient preferences.
- Patients $i, j \in N$ are mutually compatible if $i \sim_j j$ and $j \sim_i i$.
- A matching $\mu : N \longrightarrow N$ is a function such that $\mu(i) = j$ if and only if $\mu(j) = i$ for any pair of patients $i, j \in N$.

- A matching μ is individually rational if for any patient $i \in N$, $\mu(i) \neq i$ implies $\mu(i) \sim_i i$. Another way to describe this definition is that in an individually rational matching, an individual pairs up with another individual who has a compatible donor.

- Let $\mathbb{M}$ be the set of individually rational matchings for the problem $(N, \succsim_i)$.

- For any matching $\mu \in \mathbb{M}$, (i) $\mu(i) = i \Longrightarrow$ patient i is unmatched.
  (ii) For $j \in N$, $\mu(i) = j$ means the matching μ makes patient i recieve a compatible kidney from donor of patient j and patient j receives a compatible kidney from donor of patient i. This means $\mu(i) = j \Longleftrightarrow \mu(j) = i$

- A mutual compatibility matrix $R = [r_{i,j}]_{i \in N, j \in N}$ with dimensions $|N| \times |N|$ is defined as:

$$r_{i,j} = \begin{cases} 1 & \text{if } j \succ_i \text{ and } i \succ_j j \\ 0 & \text{if } i \succ_j \text{ or } j \succ_j i \end{cases} \quad \text{for any } i, j \in N \text{ and } i \neq j$$

- (N, R) is referred to as the reduced problem of $(N, \succsim)$.

- We can make a graph $G = (N, R)$ where there is an edge $\{i, j\} \in R$ if and only if $r_{i,j} = 1$.

- A matching μ of G is a subset of edges such that each vertex in μ is in at most one edge.

- If vertex i is not any edge in μ is unmatched.

- If we were interested in just maximizing the kidney transplants then all we have to do is to have an algorithm the simply returns the maximum matching of a graph.

It is worth noting some assumptions here. First, every patient credibly reports which other patient they're compatible with for matching. Every patient also prefers an outcome where they are matched to one where they are unmatched. There is a generalized mechanism that is followed in performing the exchanges by matching. This is shown by the steps below:

1. Collect a report F_i from each agent i.

2. Form a graph $G = (V, E)$, where V corresponds to patient-donor pairs and E is the compatibility matrix such that $E[i][j] = 1$ or $(i, j) \in E$ if and only if patients corresponding to i and j report as compatible the donors corresponding to j and i respectively.

3. Use Micali-Vazirani algorithm in Micali & Vazirani (1980) to return maximum cardinality matching of graph G.

Note that we could use the Micali-Vazirani algorithm because edges in G are not ranked. Define a matching $\mu \in M$ to be *Pareto-efficient* if and only if there exists no other matching $\eta \in \mathbb{M}$ such that $\eta(i) \succsim_i \mu(i)$ for all $i \in N$ and $\eta(i) \succ_i \mu(i)$ for some $i \in N$. In other words μ is Pareto efficient if there is no other matching such that for some agent i, $\eta(i)$ is a preferred compatible donor to $\mu(i)$. In addition for every agent i, $\eta(i)$ either has a better donor or an equally compatible donor to $\mu(i)$. This implies from how compatibility is defined that μ is Pareto efficient if and only if there is no agent i such that $\mu(i) = i$.

3.2.4 Optimization Mechanisms of Kidney Exchanges

Kidney exchanges through stable matchings are done through what are called *mechanisms*. A mechanism is a systematic procedure that selects a matching for each problem. The main mechanism of focus is the priority mechanism but there others that are briefly explained following the priority mechanism.

Priority Mechanism

- A *priority ordering* is a permutation of patients such that the kth patient in the permutation is the patient with the kth priority.

- The priority ordering is the natural ordering $(1, \cdots, n)$ with patient k being the kth priority patient.

- Priorities depend on patient characteristics such as percent reactive antibodies which correlate to the difficulty of finding a compatible kidney for the patient.

- In general,a priority function is $\pi : N \longrightarrow R_+$ and it is increasing if $\pi(i) \geq \pi(i+1)$.

- With the aim of finding the set of exchanges that maximizes a preference defined over matchings. $\succ_T$ is a priority preference if it is responsive to the priority ordering.ie $\mu \succ_T v$ whenever $M_v \subset M_\mu$ differ in only one patient($M_\mu \backslash M_v = \{i\}, M_v \backslash M_\mu = \{j\}$, for some $i, j \in N$ and $i < j$). So when M_μ and M_v differ by one patient then the patient with the higher priority is preferred making $\mu \succ_T v$.

- A priority mechanism produces a matching as follows for any problem (N, R) and priority ordering $(1, 2, \cdots, n)$ among n patients: Let $\mathbb{E}^0 = \mathbb{M}$ ie the set of all maximum matchings. In general $k \leq n$ let $\mathbb{E}_k \subseteq \mathbb{E}_{k-1}$ be such that:

$$\mathbb{E}_k = \begin{cases} \{\mu \in \mathbb{E}_{k-1} : \mu(k) \neq k\} \text{ if } \exists \mu \in \mathbb{E}_{k-1} \text{s.t } \mu(k) \neq k \\ \mathbb{E}_{k-1} \text{ otherwise} \end{cases}$$

  The algorithmic representation of this mathematical construction of the priority matching above is shown in algorithm

- This construction of the priority matching above shows how to get $\mathbb{E}_k$, the set of matchings with the highest priority being k: Check all matchings in $\mathbb{E}_{k-1}$ such that k is matched (i.e $\mu(k) = k$). If there aren't any, set $\mathbb{E}_k = \mathbb{E}_{k-1}$.
- A priority matching matches as many patients as possible starting with the patients with the highest priority and never sacrificing a higher priority patient with a lower priority patient. Consider to patients A and B. Patient A has a higher priority than patient B if their A assigned priority is lower in magnitude compared to B. For example if patient A is of 2nd priority and patient B is of 4th priority then patient A is of a higher priority compared to patient B.
- Priority mechanism algorithm is by nature a greedy one, since we greedily select the highest priority remaining patient at each stage.

Algorithm 2: Priority Mechanism for Pairwise Kidney Exchange

```
Data: $V = \{1 \cdots n\}$: set of patients; $\mathbb{M}$: set of Pareto-efficient matchings
Result: a Pareto-efficient matching that matches vertex with $n^{th}$ (lowest)
        priority
Initialize $M_0$ to $\mathbb{M}$;
for $k = 1$ to $n$ do
    Let $\mathbb{E}_k$ denote the set of matchings in $M_{k-1}$ that match vertex $k$ ;
    if $\mathbb{E}_k \neq \emptyset$ then
        set $M_k = \mathbb{E}_k$;
    else
        set $M_i = M_{i-1}$;
    end
end
return an arbitrary matching of $M_n$
```

Let's now take a look at why priority mechanism is necessary. Sometimes we can have ties between edges in terms of including an edge out of a possible edge choices to include in our matching. An example of this situation is shown below.
We know that P_1D_1 is in every maximum matching but how do we decide between P_2D_2, P_3D_3, P_4D_4 as the other vertex in our selected edge. This is where priority comes in.

Now let's look at how the GED Lemma influences kidney exchange. Since each patient represents a graph vertex, we have the partitions of the vertex set to be the set of underdemanded patients, the set of overdemanded patients, and the set of perfectly demanded patients. Underdemanded patients are the patients for whom there is at least one Pareto efficient (maximum matching) that leaves each such patient unmatched. Overdemanded patients are the set of patients each of whom is not underdemanded but is mutually compatible with at least one underdemanded patient. Perfectly demanded patients are the patients who are matched with another patient in each possible Pareto-efficient matching and who are not mutually compatible with

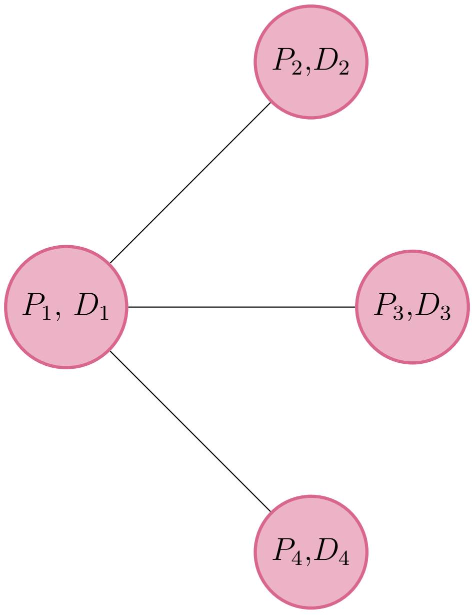


Figure 3.3: Deciding between three possible edges to include in a Pareto efficient matching

any underdemanded patients. Based off of the GED lemma we can make these claims about these partitions of the patients:

1. A Pareto efficient matching matches each perfectly matched patient with another perfectly matched patient in the same even component.

2. Each overdemanded patient is matched with an underdemanded patient.

3. One patient in each odd component is either matched with an overdemanded patient or remains unmatched whereas the remaining underdemanded patients on the same component are matched with one another.

So each even component is self-sufficient as opposed to the odd component patients who have to compete for the overdemanded patients. Now let $\mathbb{D} = \{D_1, D_2, \cdots, D_p\}$ be the partition of the set of underdemanded patients. Each D_k for $k \in \{1, 2, \cdots, p\}$ is an odd component of the graph with the overdemanded vertices absent. We get an implication as follows.

Corollary 11.1. Let N^O be the set of overdemanded patients and N^U be the set of underdemanded vertices. $|\mathbb{D}| > |N^O|$ whenever N^U is nonempty, and $|\mu| = |\mathbb{D}| - |N^O|$. μ is a Pareto efficient matching.

Taking a look at the corollary above, it makes sense that $|\mathbb{D}| > |N^O|$ because overdemanded patients according to GED lemma must be competed for by some set of underdemanded patients, call it $\daleth$, belongs to a distinct odd component. The result about $|\mu|$ in the corollary above follows from the $|D| > |N^O|$ explanation. That's because each element in $\daleth$ is competes for the overdemanded vertices and thus the number of vertices that will be unmatched after the competition is $|D| - |N^O|$. So the number of matched vertices is $|N| - (|D| - |N^O|)$.

Now we consider how the GED-Lemma allows us to see in more detail how the competition for compatible kidneys plays out in priority mechanisms. The Pareto efficient matching that comes out of the priority mechanism guarantees, by the GED lemma, matches for the perfectly demanded and overdemanded patients. It is some underdemanded patients that are unmatched in their odd components and have to compete for the overdemanded patients. This competition ensued is due to the fact that there are more odd components than overdemanded patients as observed in the previous corollary. Which underdemanded patients, each of unique odd components, gets the overdemanded patients is decided by the priority ordering. Since by the GED lemma we have all but one of the patients unmatched in each odd component, the ordering of the odd components will the depend will depend on the rank of each unmatched vertex as per the priority ordering. The odd component with the highest rank has the unmatched patient with the highest rank. This said gets to be matched one of the remaining overdemanded patients and this process repeats until all the overdemanded patients are matched. The aftermath of this is that we have some underdemanded vertices of low rank unmatched in end and they are the vertices that are unmatched in the resulting Pareto efficient matching.

At this moment we consider how the rank maximal matching algorithm from Chapter 2 could perform a priority mechanism matching. Contrary to the algorithm given for the priority mechanism involving having a set of maximum matchings to choose from, the rank maximal matching algorithm needs edges to be ranked and not vertices. So how can we rank the edges based off of the priority ordering of the vertices? Well we could use an obvious way which is to put all edges incident on the highest priority vertex in rank 1 and so on till the lowest priority vertex. Now here are the parts of the algorithm where we would have to add algorithmic steps that actually select vertices of higher priority to be matched first.

(i) Augmentation step - In the augmentation, we need to augment M_i in G_i to get a maximal matching. This is done by selecting the right augmenting path through G_i. The right augmenting path would be one where we have edges incident on vertices of higher priority than ones of lower priority. Therefore some step in the augmentation process should emphasizes selecting the augmenting path with more higher priority vertices.

(ii) Third deletion made in $E_{\geq i}$. Here is where the priority mechanism shows up implicitly. We get rid of edges in lower ranks, thereby getting rid of vertices of lower priority that may have been competing with a higher priority vertex for a higher rank edge. This is seen mostly in part(i) of this deletion. Part(ii) of this deletion guarantees the sole matching of underdemanded vertices of higher priority in the first rank. Of course there may be some underdemanded vertices unmatched that may be competing for overdemanded vertices of lower rank.

Other Optimization Mechanisms out there

There are other optimization mechanisms used in kidney exchange and they are briefly outlined as follows:

1. Stochastic exchange : In this mechanism we have a set of what are called *lotteries*. Each lottery is a probability distribution over the set of maximum matchings. When a lottery is selected, the patients are matched based on the aggregate probability of being matched across all of the maximum matchings in the lottery.

2. Egalitarian mechanism: This is similar to the stochastic exchange in terms of the usage of probabilities. However there is emphasis here on equalizing these probabilities of receiving transplants across patients, especially the underdemanded ones. This is to help prevent any biases in selection of patients for matching. The GED lemma also plays a crucial role here.

3.3 Peer-to-Peer Networking

3.3.1 Introduction

Most computer networks have two main components a client and a server. A client is a network element that requests for and uses services from a server, and a server is a network element that responds to a data request from the client. Where there are multiple devices in a network, there is usually one huge main device or set of devices that acts as the server, to which all other devices, the clients connect to. These servers acts as the provide of information in the communication between two clients. However for peer-to-peer (P2P) networks, such main servers do not exist. Every device in a peer-to-peer network can act as a client or the server depending on the nature of its connections. This means if two devices are interacting in this network there is no third party device mediating the interaction. Each device within a peer-to-peer network is called a *peer*, hence the name of this network.

Just like most networks, peer to peer networks have protocols that run them. These protocols put a restriction on the nature of how most these networks operate in order to prevent network failure. However there is one important restriction these protocols place is on the number of peers a particular peer in the network can connect and interact with at any given time. In this regard every peer would want to optimize its choices of which other peers to connect with and this motivates the creation of *preference systems* either in general for all peers or for every peer in the network. These preference systems rank all peers or other peers than the peer in question based off of many parameters such as download bandwidth,latency,storage capacity or even other manual caveats for defining preference. For example, in Bit Torrent Gai et al. (2007b) upload bandwidth is a major parameter in the selection of collaborators(peers that connect with each other at a given time) in a P2P network. The use of these preferences to select peers that optimally connect with each other under the bounded number of connections for each peer is what gives us an instance of a stable matching problem for P2P networks.

In this section, we first take a look at how the optimal connections between peers is modeled as a problem in the matching theory paradigm. Then we move on to describe the most used preference systems, acyclic preference systems and their various types which could allow us to model the connection between peers as more specifically SR-GAP and hence also SR-GRP. After that we move on to describe and prove characteristics of preference systems that enable them to be considered acyclic. We then move ahead to describe the stable nature of these paired connections in P2P. Finally we conclude with a summary of stable matching problem in P2P networks and possible future work.

3.3.2 Relationship between P2P and Matching Theory

A peer to peer network system is established by establishing overlay networks between peers. Each peer as already mentioned is bounded in number of connections by protocols. These protocols continuously search for new and better partners for those peers. The algorithms these protocols use are those which solve a type of stable matching problem called the *b-matching problem.* Such problems are a generalization of the stable roommates problem where each peer in P2P are allowed to have up to certain number of partners. It may seem like this has no relationship with SR or SR-GRP where we allow each peer to have only one partner. However in Cechlárová & Fleiner (2005), it is proved that a b-matching problem where $b > 1$ can be transformed into an equivalent 1-matching problem that is modeled as SR, SR-GRP, or SR-GAP. Hence if we were to apply the algorithm in Chapter 2 to P2P, it is the case that the matching problem of P2P is first transformed into a 1-matching problem being modeled as SR-GRP and then applying the algorithm to form the weakly stable matchings. Here below are some term descriptions that describe the model of P2P networks as a Stable roommates problem in general:

1. *Acceptance graph:* For a set P of n peers, not all possible connections are desired connections. This are undesired connections because some peers may be aware of the presence of some other peers in the network or some peers may dismiss the possibility of even connecting with some other peers for various reasons. Such criteria forms the basis of an *acceptance graph* $G(V, E)$ similar to the one defined for SR-GRP, where the neighbours of a peer $p \in V$ (vertex) are those peers that p can possibly collaborate (connect) with.

2. *Quota:* We've mentioned previously that the protocol places a bound on the highest number of connections possible for a peer, p at any instant. This number is called the *quota* and is denoted by $b(p)$.

3. *Marks:* It is assumed the peers use, based on criteria such as bandwidth,latency etc, *marks* to rank their neighbors in an accepted graph. These marks are normally in the form of non-zero real numbers or non-zero integers. The set of all marks set by all peers is normally represented as a matrix. For a peer

p, we say that p prefers peer i to peer j if and only if $m(p,i) < m(p,j)$ in p's preference list. We assume for convenience that p has a different mark for each of its neighbors. However in the SR-GRP standpoint having neighbors with the same mark may not pose much of a problem.

4. *Preference system:* A matrix of marks M creates an instance L of a preference system. We denote the *preference list* of each peer as $L(p)$ and in a preference list of every peer p, p ranks its neighbors. We say p prefers r to s iff $L(p,r) < L(p,s)$. One other assumption we make for a preference system is that we can't have a pair such that one member is not in his partners preference list. In other words it should be that $p \in L(q)$ iff $q \in L(p)$. This explains why acceptance graphs are undirected instead of directed.

5. *Configuration:* When a partnership is established between two peers say p and q, we say that each one is a *mate* of the other.A *configuration* denoted as C is the set of all formed pairs $\{p,q\}$ such that each pair p has at most $b(p)$ mates. In a set of all configurations we could trivially have a configuration $C_{\emptyset}$, where no pair is obtained. In the case of having a configuration contain only stable pairs, we know that forming a stable matching was impossible if $C = C_{\emptyset}$. In a configuration C, we say that p is *under-mated* if it has less than $b(p)$ mates in C

6. *Blocking pairs:* So far we have assumed that peers aim to link with their most preferred neighbors based on criteria for ranking. The best way to do so is for them to form blocking pairs. A blocking pair of a configuration C is a pair $\{p,q\} \in E-C$ and both prefer to change the configuration and to link with each other. It is assumed that this system evolves by discrete steps. At any step, two nodes can be linked together if and only if they form a blocking pair. At any point in time as peers form new pairs they drop their worst performing peers to maintain their quota. In this light we could also define a blocking pair as a pair $\{p,q\} \in E - C$ such that p and q prefer each other to their worst partners in their current b-pairings. A configuration is said to be *stable* if no such blocking pairs exist. In this sense a stable configuration is likened to a stable matching. The reason why we want to think of form more blocking pairs to establish a stable matching is that forming block pairs forces us to pick matches that are closer to being stable. We could think of it has coming with a matching seeing whether blocking pairs can be formed. If so pick a better matching involving those blocking pairs. We continue to do so until we find a matching where we cannot form any more blocking pairs. This final matching then becomes stable.

7. *Loving pairs:* Peers p,q form a loving pair if p prefers q to all its neighbors and q, in its turn, prefers p to all other neighbors. It implies a strong link which cannot be destroyed in the given preference system.

3.3.3 Types of acyclic preference systems

Protocols have several criteria upon which preference systems are created. These criteria though are normally categorised into these three:

1. *Proximity:* This refers to distances in the physical network, in a virtual space or similarities according to some characteristics.

2. *Capacity related:* network bandwidth, computing capacity, storage capacity.

3. *Distinction:* Differences in type and number of resources owned by different peers.

These criteria correspond to the three main types of preference systems as we shall see shortly. Now consider a $k > 3$ number of peers $p_1 \cdot p_k$. A *preference cycle* occurs if p_i prefers p_{i+1} to p_{i-1} (modulo k) if the original general preference pattern was that p_i is generally preferred to p_{i+1} where i is in modulo k.

A preference instance is *acyclic* if it contains no preference cycle. Acyclic preference systems are the most accurate descriptor of preference systems in P2P networks based on the criteria above. There are three main acyclic preference systems to consider here and they are outlined as follows:

1. *Global preferences:* Consider P2P networks where mates are selected according to the criteria that fall under the capacity category such as network bandwidth,computing capacity or storage capacity. In these types of systems the easiest way to mark each peer is to have a global preference list where all peers are given a mark each. Since the higher values of the capacity criteria such us bandwidth is most preferred in any network, peers with higher marks are most preferred. A mathematical depiction of marks for global preferences is this: $m(i,p) = m(j,p) = m(p)$. If we wanted to rank edges in their acceptance graphs such we could easily model pairings as SR-GRP, we could have the rank of the pair $\{u, v\}$ to be simply the average of $m(u)$ and $m(v)$. Preference lists that a borne from such a policy as described are called *global preferences*. Due to the nature of global preferences, we can construct a chain of peers such that their marks decrease or increase along the chain. Therefore global preferences are always acyclic. A common illustration of a global preference system is the "Tit-for-Tat" strategy used in BitTorrent, where a peer will only connect with and exchange data with peers with the highest upload capacity. Those neighbors with the best upload capacity usually appear as those peers are known for uploading the highest chunks of data as soon as a steady sequence of chunk exchanges is initiated.

2. *Symmetric preferences:* This is a preference system where a peer p gives a mark to his neighbour q in a symmetric manner such that $m(p, q) = m(q, p)$ for all p, q. Each peer in this case would like to pair with peers with the highest marks. This preference system is normally generated off of the second protocol criteria

of proximity. This makes sense intuitively because if say peer A is in close proximity to peer B, then B is also in close proximity to A in the same magnitude. The marks again increase along a preference chain, preventing the existence of a preference cycle. A common example of a proximity based criterion is the smallest round trip time (RTT) in the physical network. In for example the Pastry network connections we have peers exactly optimize pairings based on smallest RTT. This is actually inspired by latency optimization and because RTTs are generally symmetric in measure we then have a symmetric preference system. Another common example of a symmetric preference situation is in massively multiplayer online games (MMOG)which require connection between players (peers) with nearby coordinates in a virtual space. The preference system is presented as symmetric because players give marks to each other based off of distance within the virtual space. Some authors also have a preference for connecting with other participant authors with similar interests. This similarity of interests is a symmetry and hence induce a symmetric preference system.

3. *Complementary preferences:* This preference system relates to the third category of criteria mentioned earlier which which is based on complementary character of resources owned by different peers. This criteria is imposed by protocol in a situation where peers try to get the same resources. In order to allow for fair distribution of resources peers prefer its neighbors who have the highest number of resources it is missing. Therefore the peers with the highest rank in a peer, p's preference list are those peers who have the highest number of missing resource blocks. Consider two peers i and j. If we let $v(j)$ be the number of resources possessed by j and $c(i,j)$ be the number of resources that i and j have common, the mark i gives to j in his preference list is given as $m(i,j) = v(j) - c(i,j)$. These marks obviously change as new resources are acquired through downloads so the preference lists change from time to time. Also the peers for which the gap in resource numbers is highest have the longest exchange sessions.

Although some networks of which a few were mentioned above can be straightforwardly modeled as one of the three preference systems, most P2P networks as technology advances combine the usage of all three from time to time based on services required.

3.3.4 Acyclic preferences equivalence

Before we get into some proofs related to equivalence of preference systems, we will define two mark matrices we'll be dealing with:

1. *Symmetric mark matrix* Let's we denote this matrix by m. m is a matrix such that given an acceptance graph $G(V,E)$, its dimensions are $|V| \times |V|$ and $m_{ij} = mark(i,j) = m_{ji} = mark(j,i)$.

2. *Complementary mark matrix*. Once again denote this matrix by m. m is a matrix such that given an acceptance graph $G(V,E)$, its dimensions are also

$|V| \times |V|$. However $m_{ij} = v(j) - c(i,j)$, where $v(j)$ is the resources possessed by j and $c(i,j)$ are the resources i and j have in common.

Here are some theorems and lemmas about acyclic preference equivalence.

Theorem 12. Let P be a set of n peers, A be the set of all possible acyclic preference instances on P, S be the set of of all possible symmetric preference instances on P, and G be the set of all possible global preference instances, then $G \subsetneqq A = S$

The implication of this theorem is that any acyclic preference instance can be described by the mean of the symmetric marks. As a special case we can say that intuitively global mark instances are emulated by symmetric instances which are more generic in the sense of having an added quality of symmetry between the marks of any two peers. To prove this theorem, we need to show that $S \subset A$ and $G \subset A$, then $A \subset S$, then $G \neq A$.

Lemma 13. Global and symmetric preference systems are acyclic

Proof. Assume the contrary that the list of peers $p_1, ..., p_k$ is cyclic and each peer prefers its successor to its predecessor. In the form of marks it means $m(p_i, p_{i+1}) < m(p_i, p_{i-1})$ for all i modulo k. Take the sum of all possible i to get

$$\Sigma_{i=1}^{k} m(p_i, p_{i+1}) < \Sigma_{i=1}^{k} m(p_i, p_{i-1})$$

If the marks are global we can rewrite the formula above as

$$\Sigma_{i=1}^{k} m(p_i) < \Sigma_{i=1}^{k} m(p_i)$$

This is not possible and so it must be that global preferences are acyclic. We get a similar situation for symmetric because we can rewrite the former formula as

$$\Sigma_{i=1}^{k} m(p_i, p_{i+1}) < \Sigma_{i=1}^{k} m(p_i, p_{i+1})$$

This is also not possible and so symmetric preferences must be acyclic as well. □

Lemma 14. Any non-trivial acyclic preference instance always has at least one loving pair.

Proof. Consider a non-trivial acyclic preference instance. In this instance there exists 2 peers p_0, p_1 such that $L(p_0, p_1) = 1$. If $L(p_1, p_0) = 1$, then we have $\{p_0, p_1\}$ as our loving pair. If not we find another peer $p_2 \neq p_1$ and check if $L(p_1, p_2) = 1$ and $L(p_2, p_1) = 1$. If not we repeat this process until we find a loving pair p_i, p_{i+1} such that $L_(p_i, p_i + 1) = 1$. If this is not possible, then we will have a sequence of peers $p_0, \cdots, p_i, \cdots$ such that $L(p_{i-1}, p_i) = L(p_i, p_{i+1}) = 1$, where $p_{i-1} \neq p_{i+1}$. Such a sequence, with the assumption that it is infinite, will have loops in it. This contradicts the fact that we had an acyclic instance in the first place. □

One immediate usefulness of the loving pair lemma above is that a symmetric mark matrix can be constructed in polynomial time given a preference instance.

A proof of the correctness of this algorithm is in Gai et al. (2007a). One thing to note though about acyclic preferences is that not all acyclic preferences can be considered global. A counter example is with 4 peers p_1, p_2, p_3 and p_4 with the following preference lists: $L(p_1) : p_2, p_3, p_4, L(p_2) : p_1, p_3, p_4, L(p_3) : p_4, p_1, p_2$. L is acyclic but p_1 prefers p_2 to p_3 whereas p_4 prefers p_3 to p_2. p_1 and p_4 rate p_2 and p_3 differently. Thus the instance is not global.

In our discussion of complementary preference systems we said that the complementary preferences are deduced using this formula $m(p, q) = v(q) - c(p, q)$. $v(q)$ is the resources possessed by q which can be seen as a global mark. $c(p, q)$ on the other hand is the resources that p and q have in common which can be seen as symmetric mark. Therefore the complementary mark in a complementary preference system could be viewed as a linear combination of a global mark and a symmetric mark. The following theorem and its proof from Gai et al. (2007a) generalizes this observation.

Theorem 15. Let m_1 and m_2 be global or symmetric marks. Any linear combination $\lambda m_1 + \mu m_2$ is acyclic.

Proof. Suppose that the preference system induced by $m = \lambda m_1 + \mu m_2$ contains a preference cycle $p_1, p_2, \cdots, p_k, p_{k+1}, p_1$. Assume without loss of generality that m_1 is global, m_2 is symmetric and that marks of higher values are preferred for m. Then $m(p_i, p_{i-1})$ for all i modulo k. Taking the sum over all possible i

$$\Sigma_{i=1}^{k} m(p_i, p_{i+1}) > \Sigma_{i=1}^{k} m(p_i) = \Sigma_{i=1}^{k} \lambda m_1(p_i, p_{i-1}) + \mu m_2(p_i, p_{i-1})$$

but

$$\begin{aligned}\Sigma_{i=1}^{k} \lambda m_1(p_i, p_{i-1}) + \mu m_2(p_i, p_{i-1}) &= \lambda \Sigma_{i=1}^{k} m_1(p_{i-1}) + \mu \Sigma_{i=1}^{k} m_2(p_{i-1}, p_i) \\ &= \lambda \Sigma_{i=1}^{k} m_1(p_{i+1}) + \mu \Sigma_{i=1}^{k} m_2(p_i, p_{i+1}) \\ &= \Sigma_{i=1}^{k} m_2(p_i, p_{i+1})\end{aligned}$$

This latter result is a contradiction that proves the theorem □

In Gai et al. (2007a), a counterexample is given out the fact that the linear combination of acyclic preferences expressed as a mark matrix isn't necessary acyclic. Another thing worth noting is that ties may arise as a result of duplication of marks in the preference system result from the linear configuration. Although ties won't affect the existence of a stable matching configuration. They definitely affect the consequences of the stable configuration. The problem with not having uniqueness is that time is wasted sorting through the same set of edges to include in the stable matching. As a result Gai et al. (2007a) indicates that a way to generate a tieless acyclic instance is that the relevant parameter for preference construction be first converted to integer symmetric marks using the algorithm(for construction a symmetric note matrix). Then a linear combination using $\mathbb{Q}$-independent scalars produces distinct acyclic nodes.

3.3.5 Nature of Stability in P2P systems

This section seeks to give the reader the nature of some stability results in general. There is also the description of stability nature in the types of preference systems which could make them individually preferable in different situations. One important fact here is about the existence of stable matchings in the P2P. In Lebedev et al. (2007), Theorem 1 says that an acyclic b-matching instance always has a unique stable configuration (ie unique stable b-matching). It is proved by proving two facts. First is to prove that there can be at most one stable solution when preferences are acyclic. Second is to prove that any long sequence of active initiative s leads to a stable configuration. An *active initiative* are pairing actions in the process of getting a stable maximum matching that affect the nature of the pairs involved the matching. An example could be getting rid of one partner (ie a vertex) or getting rid of both partners(ie an edge). Another interesting result about stability, closely related to the latter result about existence,is in Theorem 3 of Lebedev et al. (2007) which says that given any acyclic preferences instance, when we start with any initial configuration C, there exists a sequence of at most $B/2$ initiatives leading to the stable solution where $B = \Sigma_{p \in P} b(p)$. The proof involves showing that such a stable state is reached using loving pairs.

In Gai et al. (2007a), some interesting results about stable configurations are revealed in the section on graph properties of stable configurations. These graphs used are small world graphs and a key component of the graphical data used here is the *clustering coefficient*. A small world graph is a type of mathematical graph in which most nodes are not neighbors of one another, but the neighbors of any given node are likely to be neighbors of each other and most nodes can be reached from every other node by a small number of hops or steps. The clustering coefficient is the probability that two vertices(ie two peers), say x and y, are linked given that x and y have at least one common neighbor. Global marks have been shown to have stable configurations that produce graphs with a high clustering coefficient. Even when it is lessened by using alternate acceptance graphs the clustering coefficient is still quite high. This is high clustering for global marks is as a result of what is called the *stratification effect*: peers only link to peers that have marks similar to them. Random symmetric matrices on the other hand have lower clustering coefficients. The advantage of having higher clustering coefficients is that the stable configurations will have nice routing properties.

To conclude this subsection we discuss the following important areas surrounding effective stable configurations in P2P systems:

1. *Stability:* Whether a stable configuration is a good or not depends on the characteristics and needs of a practical applications. If continuous link alteration has high cost or if the stable configuration has certain appealing properties like some of the small world graph properties, the it would be best to allow the stable configuration to converge and be established. *Convergence* can be said to be the series matchings you get by constantly adding or removing edges in the graph till the final stable matching is produced. On the other hand, if we have global marks that result in stable configuration with diameter, or some

gossip protocols constantly alter the acceptance graph, it would be best to stop the convergence and find a better configuration.

2. *Convergence speed:* The speed of the convergence also plays an important role in deciding whether a stable solution is a desired one or not. A stable configuration with a faster convergence rate will be preferred to one with a slower convergence rate. Convergence speed according to Gai et al. (2007a) depends on several factors such as the acceptance graph, the preference system used, the activity of the peers, the quotas, etc. Learning about convergence speeds could really help with understanding protocols and optimizing them.

3. *Dynamics of preference systems:* In our discussions so far we've considered fixed preference systems and acceptance graph. However as mentioned in Gai et al. (2007b), in real applications, arrivals and departures modify the acceptance graph, along with the discovery of new contacts (a toy example is BitTorrent, where a tracker periodically gives new contacts to the clients). The preference system itself could also change over time. Among the three preference systems, the complementary system is one that is closest to being dynamic in nature. That's because as a peer gets resources from a complementary peer, a mark in this complementary system decreases.

All these factors impact the nature of the stable configuration of the system and they provide areas for possible work on optimization research.

3.4 Conclusion

In this chapter we discussed two main applications of SR-GRP: Kidney exchange and Peer-to-peer networking. In kidney exchange we discussed how using matching algorithms are preferred to using the TTC method of exchanging in a sequence. We also got to understand how the SR-GRP algorithm is mainly used in the priority mechanism of optimizing kidney exchange. This is where crucial role of GED Lemma in generating Pareto efficient matchings was seen. In P2P we mainly saw how mandatory acyclicity of preferences are by examining the three types acyclic preference systems. We examined their properties and hinted on how SR-GRP can be used in developing stable configurations in P2P systems. The interesting aspect of these acyclic systems is was how they could be converted into one another or could combine to create more acyclic systems, depending on the networking issue in consideration. These two applications are very practical and essential to our modern lives. In this regard more developments in research about SR-GRP could both further solidify its relevance in these applications or reveal its relevance in other real-world applications as well.

Bibliography

Abraham, D. J., Blum, A., & Sandholm, T. (2007). Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, (pp. 295–304).

Abraham, D. J., Levavi, A., Manlove, D. F., & O'Malley, G. (2008). The stable roommates problem with globally ranked pairs. *Internet Mathematics*, *5*(4), 493–515.

Borbel'ová, V., & Cechlárová, K. (2008). On the stable b-matching problem in multigraphs. *Discrete applied mathematics*, *156*(5), 673–684.

Bry, F., & Las Vergnas, M. (1982). The edmonds-gallai decomposition for matchings in locally finite graphs. *Combinatorica*, (3), 229–235.

Cechlárová, K., & Fleiner, T. (2005). On a generalization of the stable roommates problem. *ACM Transactions on Algorithms (TALG)*, *1*(1), 143–156.

Chung, K.-S. (2000). On the existence of stable roommate matchings. *Games and economic behavior*, *33*(2), 206–230.

Gabow, H. N., Kaplan, H., & Tarjan, R. E. (2001). Unique maximum matching algorithms. *Journal of Algorithms*, *40*(2), 159–183.

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

Gai, A.-T., Lebedev, D., Mathieu, F., De Montgolfier, F., Reynier, J., & Viennot, L. (2007a). Acyclic preference systems in p2p networks. In *European Conference on Parallel Processing*, (pp. 825–834). Springer.

Gai, A.-T., Mathieu, F., De Montgolfier, F., & Reynier, J. (2007b). Stratification in p2p networks, application to bittorrent. In *ICDCS'07, International Conference on Distributed Computing Systems 2007*. IEEE Computer Society.

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

Gusfield, D., & Irving, R. W. (1989). *The Stable Marriage Problem: structure and algorithms*. MIT press.

Irving, R. W. (1985). An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, *6*(4), 577–595.

Irving, R. W. (2007). The cycle roommates problem: a hard case of kidney exchange. *Information Processing Letters*, *103*(1), 1–4.

Karumanchi, N. (2015). *Data Structure and Algorithmic Thinking with Python: Data Structure and Algorithmic Puzzles*. CareerMonk Publications.

Lebedev, D., Mathieu, F., Viennot, L., Gai, A.-T., Reynier, J., & De Montgolfier, F. (2007). On using matching theory to understand p2p network design. In *INOC 2007, International Network Optimization Conference*.

Manlove, D. (2013). *Algorithmics of Matching under preferences*. 27 Warren Street, Suite 401-402, Hackensack, NJ, USA 07601: World Scientific Publishing Co. Pte. Ltd.

Martin, J. C. (1991). *Introduction to Languages and the Theory of Computation*, vol. 4. McGraw-Hill NY.

McVitie, D. G., & Wilson, L. B. (1971). The stable marriage problem. *Communications of the ACM*, *14*(7), 486–490.

Micali, S., & Vazirani, V. V. (1980). An o (v— v— c— e—) algoithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*, (pp. 17–27). IEEE.

Radford (2022). why-is-time-comlexity-essential. `https://sites.radford.edu/~nokie/classes/360/reduce.html`

Roth, A. E., Sönmez, T., & Ünver, M. U. (2005). Pairwise kidney exchange. *Journal of Economic theory*, *125*(2), 151–188.

Roughgarden, T. (2013). Cs364a: Algorithmic game theory lecture# 10: Kidney exchange and stable matching. *blood*, *1*, P2.

Team, G. L. (2016). why-is-time-comlexity-essential. `https://www.mygreatlearning.com/blog/why-is-time-complexity-essential`