

HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR INFORMATIK

Practical Applications of One-to-Many Matchings with One-Sided Preferences

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Aaron Oertel
geboren am: 18.02.1997
geboren in: Dinslaken

Gutachter/innen: Prof. Dr. Henning Meyerhenke
Prof. Dr. Timo Kehrer

eingereicht am: verteidigt am:

Abstract

Matching algorithms are of great interest due to the diverse use cases that exist in the real world. This thesis explores matching mechanisms for one-to-many matching scenarios with one-sided preferences by using a student-seminar-matching scenario as a motivating example. We first discuss several optimality criteria and four algorithms with different characteristics, before investigating the algorithms' performance characteristics in theoretical and experimental settings. The experiments show that reducing the problem to the assignment problem or performing a min-weight max-cardinality matching on the problem's induced bipartite graph yields the best results for most metrics and instances. Additionally, an interactive web system is presented that allows for managing and executing such matching scenarios by employing the presented algorithms.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Formal Definition	2
1.3	The Capacitated House Allocation Problem	2
1.4	Outline	2
1.5	Related Problems	3
1.5.1	Stable Marriage Problem	4
1.5.2	Hospitals/Residents Problem	4
1.5.3	Assignment Problem	5
1.5.4	The Student-Project Allocation Problem	5
2	Optimality criteria	6
2.1	Maximum cardinality	6
2.2	Pareto-Optimality	6
2.3	Popularity	7
2.4	Profile-based optimality	8
2.5	Strategy-Proofness	8
2.6	Application to student-seminar matching	8
3	Algorithmic Approaches	10
3.1	Greedy with serial dictatorship	10
3.1.1	Properties	10
3.1.2	Drawbacks	11
3.2	Pareto Optimal Maximal Matchings for CHA	11
3.2.1	Properties	13
3.3	Solving CHA as an assignment problem	13
3.3.1	Solving the assignment problem using flow-networks	14
3.3.2	Properties	14
3.4	Maximum Popular Matchings in CHA	15
3.4.1	An alternative characterization of popular matchings	15
3.4.2	Algorithm	16
3.4.3	Properties	17
4	Comparison of mechanisms	18
4.1	Theoretical results	18
4.1.1	Strategy-proofness and maximum cardinality	19
4.1.2	Max-PaCHA and the assignment problem	19
4.2	Practical results in the literature	19
4.2.1	Dataset	19
4.2.2	Methodology	20
4.2.3	Results	20
4.2.4	Learnings	20

5	Implementation	22
5.1	Overview	22
5.2	Algorithm Implementation	22
5.2.1	Input and Execution	22
5.2.2	RSD Implementation	24
5.2.3	Max-Pareto Implementation	24
5.2.4	Assignment Problem Implementation	24
5.2.5	Max-Popular Implementation	24
5.2.6	Modified Max-Popular Implementation	25
5.3	Web-Server Implementation	25
5.3.1	Benchmark Tools	26
5.4	Web-Interface Implementation	26
6	Evaluation	27
6.1	Research Questions	27
6.2	Sample Data	27
6.2.1	Using graph generators	27
6.2.2	Random Uniform Preference Lists	28
6.2.3	PrefLib Datasets	28
6.2.4	Random Zipf-Distributed Preference Lists	29
6.3	Methodology	29
6.3.1	Benchmark Setup	30
6.3.2	Metrics	30
6.4	Results	31
6.4.1	PrefLib1 dataset	31
6.4.2	PrefLib2 dataset	32
6.4.3	Small Zipfian datasets	33
6.4.4	Large Zipfian datasets	34
6.4.5	Large uniform dataset with incomplete preferences	35
6.4.6	Large uniform dataset with complete preferences	36
6.5	Conclusion	36
7	Extensions to the problem	39
7.1	Two-Sided preferences	39
7.1.1	Stable matchings	39
7.1.2	Algorithm	39
7.2	Many-to-Many matchings	40
7.2.1	One-sided preferences	40
7.2.2	Two-sided preferences	40
7.3	Online variant	41
7.3.1	Online max-cardinality matching	41
8	Discussion	43
8.1	Discussion of theoretical and practical results	43

8.2	Potential algorithmic improvements	44
8.3	System design recommendations	44
9	Conclusion	46

List of Figures

1	Preference distributions for first-ranked seminars for both PrefLib datasets	28
2	Two sample Zipfian preference distribution for first-ranked seminars . .	29
3	Rank distribution for PrefLib1 Dataset. Note: Popular CHA failed. . .	32
4	Rank distribution for PrefLib2 Dataset.	33
5	Rank distribution for large Zipfian dataset.	35

List of Tables

1	Instance where Serial Dictatorship admits no max cardinality matching	11
2	Comparison of different algorithmic approaches	18
3	Summary of one-sided matching mechanisms from Diebold et al. [1] . .	20
4	Summary of the results for PrefLib1	31
5	Summary of the results for PrefLib2	32
6	Average results for small Zipfian dataset with 50 runs	34
7	Average results for large Zipfian dataset (2500 Students) with 10 runs .	34
8	Average results for large uniform dataset with incomplete preferences. .	35
9	Average results for large uniform dataset with complete preferences. . .	36

1 Introduction

1.1 Motivation

Many institutions around the world use central, automated matching schemes to assign agents to resources, based on their preferences. For instance, the National Resident Matching Program (NRMP) in the United States uses such a matching mechanism to pair graduating medical students to residency positions at hospitals every year [2]. There are different goals for these matchings, which can range from efficiency to fairness, and finding a mechanism that fulfills these goals is an important task to efficiently design such markets. There are also many variants of matching problems, that include but are not limited to one-to-one, one-to-many and many-to-many matchings. The distribution of preferences need to be considered as well. For instance, in the hospital-residents problem, both the hospitals and residents supply preferences over the other set. Here it is important to consider if incomplete preference lists or ties should be allowed in the matchings.

We will formally describe, examine and analyze the problem of one-to-many matching mechanisms with one-sided preferences. While there are many use-cases for this problem, we will focus on the problem of matching students to seminars, which is of high interest for many universities, who often require their students to participate in a seminar in order to obtain their degree. These students typically have a choice between a handful of different seminars; however, there are capacity constraints that make it hard to match all students to their first choice. Let us consider the following example: 100 students have to be assigned to one of six seminars, wherein each seminar has a capacity of 20 students. The students express their preferences by supplying a strict, meaning without ties, but possibly incomplete preference list of the six seminars. The goal for the school's administration now is to find an assignment of students to seminars that fulfills their requirements, which can range from assigning as many students as possible to being as fair as possible with the assignments. We will see that there is no obvious choice in picking an algorithm, since there are plenty of trade-offs between existing mechanisms that make it necessary to prioritize the requirements. What makes this problem harder is that the students' preferences are not necessarily equally distributed. Oftentimes a majority of students prefer one seminar which conflicts with other students getting their first choice. At the same time, it can occur that students go unmatched when their preference lists are short and primarily consist of seminars that have already reached full capacity.

The goal of this thesis is to formally model the one-to-many matching problem with one-sided preferences, while presenting different algorithms for finding matchings and finally to evaluate these algorithms using certain metrics that make sense in the one-to-many case. For the purpose of this thesis, we will use the example of student-seminar matching to analyze the problem, but it is important to note that it can just as well be generalized to any other one-to-many matching problem with incomplete preferences. Lastly, an interactive system will be developed, which will allow a school's administration to find a student-seminar matching using one of the

presented algorithms.

1.2 Formal Definition

The problem of assigning students to seminars can be described as a many-to-one matching, with a set of students $S := \{s_1, s_2, \dots, s_n\}$ and a set of seminars $T := \{t_1, t_2, \dots, t_m\}$. Every student $s_i \in S$ provides a strict, i.e. tie-less, preference order over a subset of T , and every seminar $t_j \in T$ has a capacity of c_j students. The goal is to find a matching $M : S \rightarrow T$, that assigns students to one of their preferred seminars while respecting the capacity of the seminars. That means that for every seminar t_j the following is satisfied: $|M^{-1}(t_j)| \leq c_j$. Using this definition we can describe the problem as a many-to-one matching with one-sided, incomplete preferences. In the case of students and seminars this means that one student is matched to one seminar, but one seminar is matched to a number of students, while only students express their preferences over the seminars. Later, we will discuss different optimality criteria, which can be used for defining an objective function for the problem.

1.3 The Capacitated House Allocation Problem

Many economists and game theorists [3] have studied variants of the house allocation (HA) problem, wherein a set of indivisible items H , the houses, needs to be divided among a set A of applicants. Each applicant may have a strict preference order over a subset of H . Formally this means that an instance I of the problem consists of two disjoint sets, where $H := \{h_1, h_2, \dots, h_n\}$ is the set of houses and $A := \{a_1, a_2, \dots, a_m\}$ is the set of applicants. Each applicant $a_i \in A$ ranks a subset of the houses in H using a preference list. The houses, on the other hand, do not have any preferences over applicants. A matching or assignment M is map with $M : A \rightarrow H$, so that for every applicant a_i , the house $M(a_i)$ is on the applicants preference list [4].

The house allocation problem is essentially an alias for a matching problem on bipartite graphs with one-sided preferences. There are many applications including matching clients to servers, professors to offices and also students to seminars. For the latter, some generalizations have to be made to the problem; specifically, one seminar should now be matched to more than one student, whereas the houses in HA are matched to one and only one applicant. In the literature, the latter variant of the problem is also referred to as the "Capacitated House Allocation Problem", denoted by CHA [5]. The following chapters of this thesis will explore performance indicators and algorithms for finding matchings in the context of students and seminars, which is semantically equivalent to the CHA problem.

1.4 Outline

Given this problem definition, we will review the literature for optimality criteria and, based on that criteria, present algorithms that produce matchings with those

characteristics. Many of those characteristics are quite obvious such as maximum-cardinality; however, we will also review terms such as "Pareto-efficiency" or "Popularity" that are commonly used in the context of market design and game theory. Using these criteria, we will review matching algorithms that solve the problem and then analyze and compare their runtime-complexity and optimality-properties. We will see that no algorithm is known that fulfills all desirable properties, which makes it necessary to understand the trade-offs and relationships between the algorithms.

After completing a theoretical comparison of the algorithms, we will proceed to analyze matchings computed using real data to better understand the theoretical observations and trade-offs discussed before. Additionally, a web-interface will be presented that allows a university's administration to compute matchings according to their requirements using data that the system receives as input. Lastly, we will briefly explore extensions to the problem, such as the many-to-many case and finally summarize the result and make recommendations for the implementation and usage of the system, based on the current state of research on these matching mechanisms.

Therefore, the contribution of this thesis is a practically motivated survey and analysis of one-to-many matchings with one-sided preferences. We use the problem of matching students to seminars as a motivating example to understand commonly used algorithms and their performance characteristics, by performing both a theoretical and a practical analysis using real and synthesized data. This will conclude with recommendations for building such systems, based on the theoretical and practical results obtained in the previous sections.

1.5 Related Problems

There are many related matching problems that can be classified as follows:

1. Bipartite matching problems
 - a) One-sided preferences (see Section 1.3)
 - i. One-to-one (e.g. House Allocation problem)
 - ii. **One-to-many (e.g. Capacitated House Allocation problem)**
 - b) Two-sided preferences
 - i. One-to-one (e.g. Stable-Marriage problem) (see Section 1.5.1)
 - ii. One-to-many (e.g. Hospital-Residents problem) (see Sections 1.5.2, 7.1)
2. Non-bipartite matching problems
 - a) One-to-one (e.g. Stable-Roommates problem)

Even though these problems have some differences, key mechanisms (Section 3) and optimality criteria (Section 2) used for them can nonetheless be similar or even identical. For instance, in the case of one-sided preferences, mechanisms for the one-to-one case can often easily be extended to the one-to-many case. It is also important to note

that the problem can be extended to using incomplete preference lists and ties in the preference lists. In fact, real-world settings often necessitate at least incomplete preference lists, as students often want to classify a seminar as unacceptable. The following subsections will briefly present some of the listed matching problems and present their key results.

1.5.1 Stable Marriage Problem

The stable marriage problem was one of the first matching problems to be researched [6], and consequently motivated further research in the field of matching under preferences.

The problem is stated as follows: A set of men M and of women W shall be matched one-to-one, where each men and women provide a complete strict-preference order over the agents of the other set. The deferred acceptance algorithm presented in Gale and Shapley's paper [6] finds a *stable*, complete matching in polynomial time. Stability is defined as follows: given a women $w \in W$ and any man $m \in M$ that she was not matched to, w does not prefer m more than her current partner, and m does not prefer w more than his current partner. Therefore, in a stable matching, no pair of applicants (m, w) should exist, so that m and w would benefit from leaving their assigned partner to be with each other.

The deferred acceptance algorithm uses one of the sets of applicants, for example the men, and proceeds by letting every man propose to their most preferred woman. The woman will tentatively accept the proposal, if she is not engaged at that point, or prefers the new proposal over her previous one. Then the algorithm repeats this process with men proposing to their most-preferred, but not yet proposed to woman until they are either matched or have no more women to propose to [7].

It is important to note that the algorithm produces different results if the women are the proposers: It has been shown that all possible executions of the algorithm with men as proposers yield the same stable matching. That matching is men-optimal, which means that every man has the best partner that he can have in any stable matching [7]. Additionally, with men proposing, the produced matchings have also been shown to be women-pessimal, meaning that every woman is matched to the worst partner that she can have in any matching [7].

While the scenario of matching women to men like described is presumably not a common real-world occurrence, Gale and Shapley's paper inspired several new approaches to other similar problems.

1.5.2 Hospitals/Residents Problem

A few years after the publication of Gale and Shapley's original paper, it was determined that the deferred acceptance mechanism was essentially the same mechanism used by the National Resident Matching Program (NRMP) in the United States to match graduating medical students to residency positions in hospitals [7]. As a matter of fact, Gale and Shapley's paper described an algorithm for the so-called "college admissions

problem" [6], which is essentially the same problem. Just like in the stable-marriage problem, there is a solution that can be hospital-optimal or resident-pessimal.

The problem can be described as finding a one-to-many matching with two-sided, incomplete, but strict preferences. Essentially, a hospital can offer multiple spots and both parties can mark entities from the other set as unacceptable by not including them in their preference lists [8]. Therefore the main difference to the stable marriage problem is that one and only one of the sets allows their entities to have a capacity, i.e. accept more than one applicant. At the same time though, entities in both sets still express preferences, which makes this problem different from the CHA-problem (Section 1.3) where only entities from one set express preferences.

In reality, the problem solved by the NRMP is more complex, as it permits couples of residents to submit preferences together. It has been shown that a stable solution does not always exist and that finding one if it exists, or showing that it doesn't exist is NP-complete [9]. The revised algorithm used by the NRMP utilizes findings about stability and simple matching markets to find a good approximation, while minimizing opportunities for strategic manipulation, which was indeed possible before [10].

1.5.3 Assignment Problem

The problem of matching students to seminars can also be defined as an assignment problem. The goal of the assignment problem is to find a minimum-weight, perfect matching in a bipartite graph. In this case the goal of the problem is, given the set of students S , seminars T , and a cost function $W : S \times T \rightarrow \mathbb{R}$, to find a map $M : S \rightarrow T$, which minimizes the following objective function: $\sum_{s \in S} W(s, M(s))$.

One of the first algorithms used for solving this problem was the Hungarian algorithm by Munkres, which finds a minimum-weight matching in polynomial time [11]. Alternatively, the problem can be transformed into an instance of the minimum-cost flow problem to determine a minimum weight matching. Section 3.3 will further investigate using this algorithm for the problem of student-seminar matching.

1.5.4 The Student-Project Allocation Problem

A similar problem to the CHA or student-seminar assignment problem is the Student-Project Allocation problem (SPA). This problem considers three entities instead of two, which are students, projects and lecturers. In most variations of the problem, students have preferences over courses, while courses again have capacities. However, lecturers are also considered, where lecturers can have preferences over students and/or courses. Algorithmic approaches used for this problem are mostly based on the deferred acceptance mechanism and find stable matchings that can either favor the students or lecturers [5].

2 Optimality criteria

Looking at the previously presented problems, it becomes clear that there are different objective functions or optimality criteria for such matchings. For instance, in the stable marriage problem, matchings are primarily judged by the stability characteristic. However such a characteristic does not make sense in the case of the student-seminar problem, since stability assumes both-sided preferences. For that very reason, different criteria have to be used for judging the quality of a matching.

Intuitively, when thinking about matching students to seminars, it would be desirable to match as many students as possible, as well as matching the students to their first choice, but at the same time to remain fair and resistant to manipulation. To formalize these requirements, a few criteria have been discussed in the literature, which will be helpful for comparing different approaches.

2.1 Maximum cardinality

The goal of the maximum cardinality matching problem is finding a matching M on a graph $G = (V = (X, Y), E)$, so that $|M|$ is maximal [12]. Consequently, maximum cardinality as an optimality criteria means that a matching is ideal if the number of students that are matched is maximized among all possible matchings. It should be noted here that the students' preferences are not considered when computing the maximum cardinality matching. As a matter of fact, it is possible that multiple matchings of the same cardinality exist, where one of the matchings could be better in the sense of a different optimality criteria. Therefore, maximum cardinality may be desirable, but should be used in conjunction with a different criteria, as it does not consider student preferences. This criteria is also often referred to as "efficiency" in the literature, however we will use the term maximum-cardinality as it is more specific.

2.2 Pareto-Optimality

Pareto-optimality or *Pareto-efficiency* is a commonly used term in economics to describe the state of resource allocations. Intuitively an allocation, or in our case a matching, is Pareto optimal iff no improvement can be made to a single individual, without worsening the situation for other individuals. Additionally a matching, in which a subset of students S' would be better off by swapping their seminars is not Pareto optimal. In order to more formally define Pareto-optimality, we must first define student preferences more formally: Given two matchings M, M' and a student $s \in S$, the student s prefers M' over M in the following cases:

1. s is matched in M' and unmatched in M , or
2. s is matched in both M' and M , however prefers $M'(a)$ over $M(a)$

Using this definition, we now define Pareto optimality as follows: given an instance I of a matching problem and its set of possible matchings \mathcal{M} , we define a relation \succ

on \mathcal{M} , where given two matchings $M, M' \in \mathcal{M}$ the following holds true: $M' \succ M$ if no student prefers M to M' , but some student prefers M' over M . Consequently a matching $M' \in \mathcal{M}$ is called Pareto-optimal iff there exists no other matching $M \in \mathcal{M}$, such that $M' \succ M$ [5]. A Pareto-optimal matching always exists for any instance of a matching problem and can be efficiently computed using one of various algorithms. A simple greedy algorithm uses the *random serial-dictatorship* mechanism to draw each agent in random order and lets them select their most-preferred, available item from their preference list (details in Section 3.1) [13, 14]. However, the greedy algorithm does not always produce a Pareto-optimal matching of maximum cardinality, which would be desirable in student-seminar matching [15].

2.3 Popularity

In the context of matching with one-sided preference lists, using an optimality criteria like Stability, which is based on the preferences of both parties, does not apply. Instead, a commonly used criteria is a matching M 's Popularity, which indicates that more students prefer that matching M over any other possible matching [16]. Given the definition of the student-seminar matching problem, we can formally define Popularity as follows: Let $P(M', M)$ be the set of students who prefer M' over M . A matching M' is said to be more popular than M , denoted by $M' \succ M$, iff $|P(M', M)| > |P(M, M')|$. That concludes that a matching M' is popular, iff there is no other matching M that is more popular than M' , i.e. $M' \succ M$ [17, 18]. Because of that definition, this criteria is often also referred to as the majority assignment [19].

Using this definition, we can see that every popular matching also is a pareto-optimal matching. Given a popular matching M' and any matching M for an instance I of the problem, M' is pareto-optimal if $P(M, M') = 0$ and $P(M', M) \geq 1$, which obviously implies that M' is popular as well [17].

It is important to note that a popular matching's cardinality could be smaller than the maximum cardinality, meaning that, in the case of student-seminar matchings, a group of students could be left unassigned in favor of the majority of the students having a match that they prefer. Additionally, a popular matching, unlike a pareto-optimal matching, does not always exist. To illustrate this, let us consider the following instance: $S = \{s_1, s_2, s_3\}$, $T = \{t_1, t_2, t_3\}$, where each student has the same preference list, being $t_1 < t_2 < t_3$, and each seminar $t_i \in T$ has a capacity of 1. Given the following matchings, we can confirm that there exists no popular matching for the given instance:

1. $M_1 = \{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$
2. $M_2 = \{(s_1, t_3), (s_2, t_1), (s_3, t_2)\}$
3. $M_3 = \{(s_1, t_2), (s_2, t_3), (s_3, t_1)\}$

It is clear that M_2 is more popular than M_1 , M_3 is more popular than M_2 and M_1 is more popular than M_3 [18].

2.4 Profile-based optimality

Contrary to Popularity and Pareto-optimality, in which the students' satisfaction with a matching is compared, we should also examine the structure of a matching by defining the profile of a matching and comparing it. Intuitively the profile of a matching M is a vector whose i th component indicates the number of students obtaining their i th-choice seminar in M , according to their preference list.

Formally, let I be an instance and \mathcal{M} the set of its matchings. Given a matching $M \in \mathcal{M}$ with the set of students S and seminars T , we define the regret $r(M)$ of M as follows: $r(M) = \max\{rank(s_i, t_j) : (s_i, t_j) \in M, s_i \in S, t_j \in T\}$, where for every match $(s_i, t_j) \in M$, $rank(s_i, t_j)$ is defined as the position of t_j on s_i 's preference list. The profile of M is now defined as a vector $\langle p_1, \dots, p_{r^*} \rangle$, with $r^* = r(M)$ and for each $k \in [1, r^*]$, the k th component is defined as: $p_k = |\{(s_i, t_j) \in M : rank(s_i, t_j) = k\}|$ [5].

Using the definition of a matching's profile, it is now possible to define a matching as rank-maximal as follows: A matching M is rank-maximal, if its profile $p(M)$ is lexicographically maximal over all possible matchings in \mathcal{M} . That means that the number of students in M who are matched to their first choice is maximal among all $M' \in \mathcal{M}$; taking that into consideration, the number of students who are matched to their 2nd choice is maximum among all matchings, and so on.

2.5 Strategy-Proofness

The aforementioned criteria all primarily consider the structure of a matching to evaluate quality and not the properties of a matching mechanism, i.e. algorithm. An important question to consider, however, is if the agents can manipulate the outcome of an algorithm by not truthfully disclosing their preferences - indeed, in the setting of matching residents to hospitals in the US, a previously used algorithm allowed students to improve their outcome of the algorithm by not supplying their real preferences [7]. In the literature, the term for a mechanism, wherein no agent can benefit from misrepresenting their preferences, is called strategy-proof [17]. Such mechanisms are of high interest for most matching problems, since preference-based optimality criteria, like the ones previously mentioned, would certainly lose some significance if the mechanism used to compute them is not strategy-proof. For instance, it has been shown [20] that for the stable marriage problem with incomplete preferences, there exists no matching mechanism that both produces a popular matching and is strategy-proof.

2.6 Application to student-seminar matching

Given the problem description of student-seminar matching, it would be desirable to find a matching that has the following properties:

1. **Maximum Cardinality:** As few students as possible should be left unmatched.
2. **Pareto Optimality:** It should not be possible for any of the students to improve their situation without affecting other students negatively. This means that there

exists no opportunity for anyone to perform a trade or get matched to a new seminar without impacting some other student.

3. **Popularity:** The number of students who are satisfied with their match should be maximal among all possible matchings.
4. **Rank Maximality:** As many students as possible should be matched to their first choice or if not possible, their second choice, and so on.
5. **Strategy-proofness:** Students should not be able to benefit, i.e. increase their chances of being matched to their top-preference, by lying about their true-preferences. A matching mechanism should also not encourage students to supply short preference lists to improve their chances of getting matched to their.

As we have already seen, there does not always exist a popular matching or a pareto-optimal matching that is also agent complete. Additionally, the fact that students can supply incomplete preference lists can very well lead to matchings that leave a few students unassigned, even if the sum of the capacity of all seminars is greater than the number of students. Given these constraints and observations, we will see that there is no such thing as an ideal matching for all instances based on the criteria we have presented. However, it will be possible to find matchings that will optimize for most of the optimality criteria.

3 Algorithmic Approaches

In the previous section, we discussed several optimality criteria that apply to the problem of matching students to seminars. This chapter will present algorithms for computing matchings that fulfill some of those criteria, as well as evaluating them against each other. The goal of this evaluation is choosing the "ideal" algorithm for implementation. We will see that each of the algorithms has some draw-backs, which makes it necessary to consider the trade-offs when picking one for implementation. However, each of the algorithms we will examine produces pareto-optimal matchings.

3.1 Greedy with serial dictatorship

One of the simplest algorithms for the student-seminar matching problem is a greedy approach, that iterates over the set of students and assigns each of the students to their most preferred seminar that still has some capacity left. In contrast to Gale & Shapley's deferred acceptance algorithm for the stable marriage problem, this algorithm does not tentatively match students once they make their selection, but makes a final assignment for those students. Because of this, the algorithm finds a matching in $\mathcal{O}(n)$ time with n being the number of students. This mechanism of letting students successively pick their highest available preference in order is known as serial dictatorship [21]. In detail the algorithm is as follows:

Algorithm 1 Greedy serial dictatorship matching

Input: set of Students with preferences S , set of Seminars T

Output: Pareto-Optimal Matching M

function SD-MATCHING(S, T)

$M = \emptyset$

for each $s \in S$ **in random order do**

$t =$ highest ranked, available seminar on preference list of s

if $t \neq \text{null}$ **then**

$M = M \cup \{(s, t)\}$

end if

end for

return M

end function

Even though this algorithm is very simple and fast, it has some desirable properties, including one of the optimality criteria defined before:

3.1.1 Properties

Since the order, in which the students get to pick their match is pre-defined, we can easily show that the algorithm always produces a pareto-optimal matching.

Theorem 1. *A greedy algorithm that uses serial dictatorship always produces a pareto-optimal matching.*

Proof. Let M be the matching produced by the algorithm. We assume that there exists a matching N that pareto-dominates M . Now, let $s \in S$ be the first student who prefers his match in N over M . Since s prefers $N(s)$ over $M(s)$, the seminar $N(s)$ must have been unavailable when he made his pick. That means that another student $s' \in S$ exists, who picked $N(s)$ before s could. However, we required that s was matched to a better seminar in N , which means that s' gets a worse match in N . This is a contradiction, so N cannot pareto-dominate M [22]. \square

We can also easily see that the algorithm is strategy-proof, because every applicant makes his final pick once it is his turn, there is no benefit in misrepresenting preferences [17].

3.1.2 Drawbacks

When looking at the algorithm, it is clear that it has a strict preference order over students, specified by the order in which students are matched in the for-loop. Additionally, the algorithm makes no effort to match all students: if it is a student's turn to pick his match, and none of the seminars on his preference lists are free, that student will not be matched at all. This problem gets worse when we consider that our problem statement allows for incomplete preference lists, which increases the chances of having a high number of unmatched students. To illustrate this let us consider the following example in Table 1: In this example, each seminar only has a capacity of

Agent	Pref list	Seminar	Capacity
s_1	t_1, t_2	t_1	1
s_2	t_1	t_2	1

Table 1: Instance where Serial Dictatorship admits no max cardinality matching

1 and both students have seminar t_1 as their first preference. If the algorithm first gives s_1 a chance to pick, and then s_2 , s_1 will be matched to t_1 , making t_1 full and not allowing s_2 to be matched. On the other hand, matching s_2 to t_1 first and then matching s_1 to t_1 also yields a pareto optimal matching, however, in this case, all the students are matched.

To address the other problem of preference over students, a simple approach is using the random serial dictatorship mechanism, which instead creates a random order of students as the pick order.

3.2 Pareto Optimal Maximal Matchings for CHA

We have seen that serial dictatorship is an easy and time-efficient mechanism for computing Pareto-optimal matchings. A big weakness of the approach, however, is

that it finds just one of many possible Pareto-optimal matchings without making any guarantees about quality in regards to cardinality. Particularly, the example in Table 1 shows how permutations of the same instance can produce matchings of different cardinality, which motivates the search for an algorithm that produces a Pareto-Optimal matching of maximum cardinality.

Abraham et. al [15] have proposed a 3-phase algorithm for computing a maximum cardinality matching for the house allocation problem, which was extended by Sng [4] for the many-to-one case. Before presenting the algorithm, an important lemma about Pareto optimal matchings has to be shown first, which is then used to prove the correctness of the algorithm. To characterize the lemma, we need to define the terms *maximality*, *trade-in-free* and *cyclic coalition* in regards to a matching M first:

1. **Maximal:** M is maximal, if no student $s_i \in S$ and seminar $t_j \in T$ exists, so that s_i is unassigned, t_j is undersubscribed in M and t_j is on s_i 's preference list [15].
2. **Trade-in-free:** M is trade-in-free, if there are no student $s_i \in S$ and seminars $t_j, t_l \in T$, such that s_i is assigned to t_l , but prefers t_j over t_l and t_j is undersubscribed [15].
3. **Cyclic coalition:** M contains a cyclic coalition, if there exists a sequence of distinct assigned students $C = \langle s_0, s_1, \dots, s_{r-1} \rangle$ with $r \geq 2$, such that s_i prefers $M(s_{i+1 \bmod r})$ (i.e. the seminar assigned to the next student in C after s_i) over $M(s_i)$ for every i [15].

Using these definitions, Sng now presents and proofs the following lemma:

Lemma 2. *Let M be a matching of a given instance of CHA. Then M is Pareto optimal if and only if M is maximal, trade-in-free and cyclic-coalition-free [15].*

Using this lemma, Abraham et al [15] construct a 3-phased algorithm, where each phase fulfills one of the properties as described in Lemma 2, like so: Let I be an instance of CHA and G it is underlying graph, then perform the following steps:

1. **Phase 1:** In order to guarantee maximality, compute a maximum matching M in G .
2. **Phase 2:** Using the matching M produced by step 1, the algorithm then fulfills the trade-in-free criteria as follows: Search for pairs $(s_i, t_j) \in M$ with $s_i \in S$ and $t_j \in T$ and where t_j is undersubscribed in M and s_i prefers t_j over his own match $t_l := M(s_i)$. Whenever such a pair is found, remove the existing assignment (s_i, t_l) and add (s_i, t_j) to M . Consequently t_l is now undersubscribed and may be assigned to another student. Therefore, we continue the search for such pairs until no such pair can be found for every student in S .
3. **Phase 3:** The last phase of the algorithm eliminates any cyclic coalitions from M , if they exist, by using a modified version of *Gale's Top Trading Cycles* (denoted by TTC) Method [23]. Essentially, the TTC method creates a graph from the

matching M , where every student that is not matched to his most-preferred seminar, denoted by S' , is represented by a node. Next, a directed edge is created from each student $s_i \in S'$, to all students in S' who are assigned to the first seminar on s_i 's preference list. Now, there must be at least one cycle in this graph, as students may have an edge to themselves. The next step is identifying the cycles and implementing a trade among all agents of that cycle that reassigns the seminars among these students. After the trade, all students from that cycle are removed and these steps are repeated until the graph is empty. Once the graph is empty, M is coalition-free by the correctness of the TTC method [15].

3.2.1 Properties

Since all modifications to the matching in phase 1 and 2 are limited to swaps and no deletions, maximum cardinality is still guaranteed after the termination of phase 3. Additionally, the resulting matching is also trade-in-free and cyclic-coalition-free as those properties are guaranteed after performing phase 2 and 3 respectively. However, it is important to note that this algorithm, unlike the serial dictatorship mechanism, is not strategy-proof: Due to the fact that a maximum-cardinality matching is computed in step 1, students are encouraged to provide short preference lists to have a higher chance of being matched to their first preference. Forcing students to provide complete preference lists could make this mechanism strategy-proof, if we can assume that students do not have any knowledge of the other student's preferences.

The runtime of the algorithm is dominated by finding a maximum cardinality matching (phase 1), which yields a time complexity of $\mathcal{O}(E\sqrt{V})$ [15] when using the Hopcroft-Karp algorithm. Phase 1 and 2 both take $\mathcal{O}(|E|)$ of time[4], which in total yields a worst-case complexity of $\mathcal{O}(E\sqrt{V})$ for finding a maximum cardinality pareto-optimal matching given any instance I of the problem.

3.3 Solving CHA as an assignment problem

In order to find a rank-maximal matching, we will investigate a set of algorithmic methods that compute a maximum-cardinality, min-weight matching. We can easily see that such a matching must also be rank-maximal, since the min-weight property guarantees that the profile of the matching is lexicographically maximal among all matchings. In Section 1.5.3, we briefly presented the assignment problem: it is a combinatorial optimization problem, which assigns a set of agents to a set of tasks, wherein each agent-task tuple is assigned a cost, and to minimize the total cost of the assignment.

Formally, we define the assignment problem as follows: Given a set of agents A , tasks T and a map $W(a, t) = w$, with $\forall a \in A, \forall t \in T$, find a bijective map M with: $\forall a \in A, \exists t \in T : M(a) = t$, so that the following objective function is minimized: $\sum_{a \in A} W(a, M(a))$.

It is important to note here that the assignment problem tries to find a perfect matching, meaning that all agents are assigned and that a one-to-one matching is

found.

For this problem, there exist algorithms [11, 24] that find perfect, min-weight assignments in $\mathcal{O}(n^3)$ time. Using these algorithms for the one-to-many case with incomplete preference lists requires a transformation of the input, by introducing artificial edges with large weights, where no edges exist. Additionally, in the case of student-seminar matchings, seminars have to be duplicated according to their capacities to allow for one-to-many matching.

However, we can see that the assignment problem is equivalent to finding a perfect, minimum-weight matching in bipartite, weighted graph. Since the problem of matching students to seminars can simply be described as finding a minimum-weight matching on a bipartite graph, we can construct such a graph, given the set of students, preference lists and seminars and apply graph algorithms that find such matchings for us. In fact, we can simply transform the student-seminar matching problem into an instance of the minimum-cost flow problem. The goal of this algorithm will be to send $|S|$ units of flow through the network, while minimizing the cost of the flow, which is indicated by a seminar's rank on the student's preference lists.

3.3.1 Solving the assignment problem using flow-networks

The problem of matching students as seminars can be given as a bipartite graph $G = (V = (S, T), E)$, where S is the set of students and T the set of seminars. The set of edges E is defined as follows: $E := \{(s, t) \mid s \in S \wedge t \in T \wedge t \text{ is on the preference list of } s\}$. Additionally, a weight function $W : E \rightarrow \mathbb{N}$ is specified, which maps each edge to the position of the seminar on the student's preference list. In order to transform this bipartite graph into an input for the minimum-cost flow problem, a flow network has to be constructed using the bipartite graph.

To transform the bipartite graph into a flow network, we first add a source and sink vertex to the graph. Then, we add weights, capacities and edges from and to the source and sink. Specifically, we create an edge from the source to each of the student vertices with a capacity of 1 and a cost of 0. These edges indicate that a student can only be assigned once. Next, for every student we re-use the edges from the bipartite graph G , where each edge $e \in E$ is assigned a capacity of 1 and a cost of $W(e)$. The capacity, again indicates that a student can only be assigned once and the weight indicates the position of the seminar on the student's preference list. Finally, one edge is added from each seminar $s \in S$ to the sink with a capacity of $C(s)$ and a cost of 0.

3.3.2 Properties

The matching M computed by the algorithm is Pareto optimal [4] and has the minimum weight property, and therefore is rank-maximal; [4] however, a large drawback is that this mechanism is not strategy-proof: students are encouraged to provide short preference lists in order to get matched to their most-preferred seminar. The algorithm tries to match every student, which means that students with a list of just one seminar will be prioritized over students, who prefer the same seminar but also supply other preferences.

This problem could lead to having all students provide single-element preference lists, which increases the difficulty of finding perfect matchings. Similarly to the previous algorithm, requiring complete preference lists could alleviate this problem and make the mechanism strategy-proof, if it can be assumed that students' preferences are not publicly known.

3.4 Maximum Popular Matchings in CHA

Abraham et al [18] presented an algorithm for finding a popular matching in the house allocation problem (without capacities), which either finds such a popular matching or reports that none exists in $\mathcal{O}(|V| + |E|)$ time. Manlove and Sng [16] extended this algorithm for the many-to-one case, the Capacitated House Allocation problem, by developing a characterization of such popular matchings in CHA and then using it to construct an algorithm that finds a maximum popular matching for any given instance, if it exists. Intuitively, the algorithm will try to match as many applicants as possible to their most-preferred choice to fulfill the popularity criteria. Formally, Manlove and Sng [16] define and prove an alternative characterisation of Popularity in order to develop their algorithm:

3.4.1 An alternative characterization of popular matchings

Given an instance I of the CHA problem, for every student $a_i \in A$ let $h_j := f(a_i)$ be the first ranked house on a_i 's preference list. In this case, we call h_j an f-house. For each house $h_j \in H$, define the set of applicants, who named h_j as their first choice, as $f(h_j) = \{a_i \in A : f(a_i) = h_j\}$ and the size of that set as $f_j = |f(h_j)|$. For a matching M in I , we now say that a house $h_j \in H$ is full if $|M^{-1}(h_j)| = c_j$, i.e. the maximum number of applicants is matched to that house, and undersubscribed if $|M(h_j)| < c_j$. Additionally, for every applicant $a_i \in A$, we append a last-resort house $l(a_i)$ with capacity 1 to a_i 's preference list [16]. The authors prove the following lemma [16]:

Lemma 3. *Let M be a popular matching in I . Then for every f-house h_j , $|M^{-1}(h_j) \cap f(h_j)| = \min\{c_j, f_j\}$.*

In other words, for a popular matching M in I , every f-house h_j is matched to at least the number of applicants who have h_j as their first-preference but at most h_j 's capacity c_j . Next, for every applicant a_i , we define $s(a_i)$ to be the most-preferred house h_j on his preference list, such that either (i) h_j is not an f-house, meaning that it is not the first choice of any applicant, or (ii) h_j is an f-house, but $h_j \neq f(a_i)$ and $f_j < c_j$. In simple terms, $s(a_i)$ is the first undersubscribed house on a_i 's preference list after $f(a_i)$. We will refer to such houses as s-houses. It is important to note that such a house $s(a_i)$ always exists, due to the introduction of $l(a_i)$. In a popular matching, an agent a_i may only be matched to either $f(a_i)$ or $s(a_i)$, as every house in between those two is full according to the definition of $f(a_i)$ and $s(a_i)$. Manlove and Sng, again prove the following two lemmas [16]:

Lemma 4. *Let M be a popular matching in I . Then no agent $a_i \in A$ can be matched in M to a house between $f(a_i)$ and $s(a_i)$ on a_i 's preference list.*

Lemma 5. *Let M be a popular matching in I . Then no agent $a_i \in A$ can be matched in M to a house worse than $s(a_i)$ on a_i 's preference list.*

To summarize, so far for every applicant $a_i \in A$ we have defined the applicant's most preferred house $f(a_i)$ and his second most-preferred, but available house $s(a_i)$. We have seen that, in a popular matching, applicants can only be matched to either of those houses $f(a_i)$ or $s(a_i)$. Using this information and a description of an instance as a weighted-bipartite-graph $G = (V = (A, H), E)$, we can now construct a subgraph G' of G , by removing all edges in G from every applicant a_i , except the ones to $f(a_i)$ and $s(a_i)$. We now say that a matching M is agent-complete in G' if it matches all agents in A and no agent a_i is matched to their last-resort house $l(a_i)$ [16]. Manlove and Sng prove the following theorem to fully characterize popular matchings [16]:

Theorem 6. *A matching M is popular in I iff:*

1. *for every f -house h_j*
 - a) *if $f_j \leq c_j$, then $f(h_j) \subseteq M(h_j)$*
 - b) *if $f_j > c_j$, then $|M(h_j)| = c_j$ and $M(h_j) \subseteq f(h_j)$*
2. *M is an agent complete matching in the reduced graph G'*

Intuitively, this means that in every popular matching, each house that would be undersubscribed or just full, based on the applicants who name that house as their first preference, will be matched to atleast all of those applicants. Additionally, every house that would be oversubscribed is matched to only a subset of the applicants naming the house as their first choice. For the remaining unmatched applicants it must now be possible to find an applicant-complete matching so that none of the applicants is matched to their last-resort house.

3.4.2 Algorithm

Using Theorem 6, the authors develop the algorithm Popular-CHA for finding a maximum popular matching or reporting that none exists [16]. The algorithm works as follows:

1. Reduce G to G' .
2. Match all agents to their first-choice house h_j , if $f_j \leq c_j$, i.e. the house would be undersubscribed or just full afterwards. This will satisfy condition 1a of Theorem 6.
3. Remove all applicants and their incident edges, that were matched in the previous step, from G' . Additionally, update the capacities for each previously matched house h_j as $c'_j = c_j - f_j$. All full and isolated houses and their incident edges are also removed from G' .

4. Compute a maximum cardinality matching M' on G' using the updated capacities. For this step Manlove and Sng use Gabow's algorithm [25].
5. If M' is not agent complete, then no popular matchings exists. Otherwise merge the matchings M and M' .
6. As a last step, to fulfill condition 1b of Theorem 6, promote any agent $a_i \in M$ who is matched to their s-house to their f-house, if it is undersubscribed.

3.4.3 Properties

Due to the fact that step 2, 4 and 6 make the computed matching fulfill the criteria outlined in Theorem 6, the algorithm produces a popular matching of maximum cardinality, if it exists. If just one applicant can't be matched to his f-house or s-house or is matched to his last-resort house, the algorithm fails and no matching is produced. In that case the popular matching for that instance is not of maximum cardinality and therefore is not computed by the algorithm.

Furthermore, the algorithm's runtime complexity is $\mathcal{O}(\sqrt{C}n_1 + |E|)$, where C is the sum of the capacities of the houses and n_1 the number of applicants. $|E|$ is equivalent to the sum of the agents' preference list lengths. The runtime is dominated by Gabow's algorithm, which computes the maximum cardinality matching in G' in $\mathcal{O}(\sqrt{C}n_1)$ [16]. Alternatively, a modified version of the Hopcroft-Karp algorithm could be used for computing the maximum cardinality matching in $\mathcal{O}(E\sqrt{V})$ [26] time, which is what Abraham et al. use for the one-to-one case.

If the applicants do not have knowledge over the overall preference distribution, this mechanism is strategy-proof due to the existence of the l-houses. The mechanism does not prioritize students with short preference lists, because it only considers at most two of the students preferences. However, if students provide short preference lists, it is more likely for this algorithm to not find a matching, as the chances of a student being matched to his last-resort house are increased.

4 Comparison of mechanisms

In the previous section we have studied several different algorithmic approaches, which guarantee different optimality criteria and rely on different mechanisms. To better evaluate the algorithms against the optimality criteria we defined in Section 2 we will now summarize and compare the properties of the algorithms. Afterwards we will look at some practical results, that were obtained by a benchmark of matching mechanisms with one- and two-sided preferences by Diebold and Bichler [1]. These results will provide a guideline in picking algorithms for implementation and will be described in the next chapter.

4.1 Theoretical results

Drawing back to the list of desirable properties defined in Section 2.6, let us now recap and compare the aforementioned algorithms to evaluate which one could be applicable for the problem of matching students to seminars. Unfortunately none of the algorithms guarantee all of the optimality criteria at the same time, which makes the choice of an algorithm unclear. Table 2 gives an overview of the presented algorithms and their properties. Each of the algorithms is listed in the same order that they were presented, and for each optimality criteria, a yes/no encoding is used to make a statement about which properties an algorithm guarantees. It is important to note here that a "no" in a column does not strictly mean that the given optimality criteria cannot be fulfilled by the algorithm, but rather that the algorithm does not guarantee it. For instance, a matching computed with the greedy algorithm can be of maximum cardinality or be popular. Only the results for strategy-proofness are a strict yes or no, since fulfilling strategy-proofness does not depend on the instance of the problem, but only of the mechanism being used.

	RSD	Max-PaCHA	Assignment	Popular-CHA
Maximum Cardinality	no	yes	yes	yes
Pareto-Optimal	yes	yes	yes	yes
Popular	no	no	no	yes
Rank Maximal	no	no	yes	no
Always Exists	yes	yes	yes	no
Strategy Proof	yes	no	no	yes
Time Complexity	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt{nm})$	$\approx \mathcal{O}(n^3)$	$\mathcal{O}(\sqrt{C}n_1 + m)$

Table 2: Comparison of different algorithmic approaches

To summarize the results, we can see that all of the algorithms guarantee pareto-optimality, however only the Popular-CHA algorithm guarantees popularity. At the same time, only the greedy approach and Popular-CHA also guarantee strategy-proofness, which makes Popular-CHA particularly interesting for the student-seminar problem.

4.1.1 Strategy-proofness and maximum cardinality

One interesting observation is that fulfilling maximum cardinality comes at the cost of either not being strategy-proof, or not guaranteeing that a matching exists at all. Indeed, only the greedy and Popular-CHA algorithm guarantee strategy-proofness. However, ensuring strategy-proofness and maximum cardinality at the same time comes at the cost of not always finding a matching. If we look back at the algorithm Popular CHA, we remember that a maximum cardinality matching M' is computed on the reduced graph G' . We saw that a maximum popular matching does not exist, iff the matching is not agent-complete, meaning that one of the agents is matched to their last-resort house. While this mechanism ensures strategy-proofness, it is also not always possible to find such a maximum cardinality matching using the Popular CHA algorithm. Therefore, it remains an open question whether or not a mechanism exists that both is strategy-proof and produces maximum-cardinality matchings.

4.1.2 Max-PaCHA and the assignment problem

Another important thing to notice is the similarity of the properties between the Max-PaCHA and assignment problem algorithm. Except for the fact that the assignment algorithm guarantees rank maximality, the two algorithms produce matchings with very similar characteristics, which then begs the question why one should use the Max-PaCHA algorithm. But looking at the runtime complexity of the algorithms, we see that, while both algorithms run in polynomial time, the assignment problem takes longer to be solved.

4.2 Practical results in the literature

Diebold et al. have published results for an extensive experiment on matching mechanisms with both one- and two-sided preferences [1]. They used real course registration data from TUM, grouped into 28 datasets for investigating properties, including size, rank and popularity, of matchings produced by several mechanisms. Before conducting our own experiment we will study their results.

4.2.1 Dataset

The data with one-sided preferences is comprised of 9 datasets from the official tutorial registration at TUM in the period between October 2012 and October 2015. All of those datasets contain incomplete preference lists and all but two of them contain ties. Each dataset contains between 136 and 1035 students and between 5 and 51 courses with total capacities ranging from 130 to 1282. For the dataset with 51 courses, the authors provide a histogram of the length of student's preference lists, showing that a majority of students gave preferences for between 10 and 15 tutorials [1]. Unfortunately the rest of the data is kept private and therefore it is not possible to get information on the distribution of the individual preferences lists, or in other words the popularity of each seminar from the students' perspective.

4.2.2 Methodology

Diebold et al. used most of the mechanism described in Section 3, with some small differences. Particularly, instead of solving the assignment problem they used the ProB CHAT algorithm that produces the same matchings. They implemented all matching mechanisms in Python 3.4.2 and used randomization to break ties. Additionally, every algorithm was run 100 times and the average of the metrics were obtained for each mechanism. Besides size, average rank and popularity, the authors also used the *area under the profile curve ration (AUPCR)* to compare the rank of the produced matchings. They define it as follows: "The AUPCR up to a specific rank describes the probability that a matching mechanism will rank a randomly chosen student higher than his n-th preference." [1]

4.2.3 Results

A summary of the benchmark is provided in Table 3. Unsurprisingly the Popular CHA mechanism produces more popular matchings than all other mechanisms, but it is also interesting to note that ProB CHAT produces more popular matchings than all mechanisms but Pop CHAT. Besides that we can see that all mechanisms achieve an average size of atleast 97.4%, while Max-PaCHA and ProB CHAT unsurprisingly find max-cardinality matchings.

Metric	RSD	Max PaCHA	ProB CHAT	Popular CHA
Average AUPCR	94.95%	96.77%	97.83%	97.31%
More Popular	20%	0%	80%	100%
Average rank	1.41	1.51	1.26	1.33
Average size	97.48%	100%	100%	99.78%
Max Runtime	0.014s	0.522s	33.852s	2.458s

Table 3: Summary of one-sided matching mechanisms from Diebold et al. [1]

Looking at the profile and rank, we can see that ProB CHAT performs best, which is not a surprise, but at the same time Popular CHA got very close in terms of average AUPCR and average rank. However it is important to note that Popular CHA did not find a matching for one of the 9 datasets. At first glance it would seem that ProB CHAT always produces really good results in terms of the metrics used, however we also need to note that its maximum runtime in the benchmark was 33.852 seconds for the largest dataset with 915 students and 51 courses with a total capacity of 1080. In contrast, the RSD mechanism still produces acceptable results at runtimes of a fraction of a second, while being strategy-proof.

4.2.4 Learnings

In this benchmark, we have seen that the ProB CHAT mechanism unsurprisingly produces some of the best results, while also being limited by its runtime which is worse

among all mechanisms. Additionally, we have seen that Popular CHA also produces good matchings at a more acceptable runtime. One interesting observation is that Popular CHA produces better results than MPO CHA, if it can find a matching. While these results confirm some of the theoretical observations, it will be interesting to get more insights with different data being used. In this benchmark, most of the data contained ties and we didn't get any insights on the distribution of preferences.

5 Implementation

5.1 Overview

To allow for further experimentation with my findings, I will describe a prototype for an interactive web-system, built for computing student-seminar matchings. The requirements for this system were quite simple to keep it in scope of this thesis. The system consists of a website, which communicates with a server that manages the underlying data and computes the matchings. The requirements can be listed as follows:

1. Allow for adding and deleting student and seminar data.
2. Allow for computing a match using the previously entered data.
3. Output the matching and its properties.

This system produces matchings by using an efficient implementation of the algorithms mentioned in Section 3. While the webserver itself is build in the Kotlin programm language and therefore runs on the JVM, I decided to implement the algorithms in C++ for the performance benefit I noticed when moving from a Kotlin to a C++ implementation. However, the Kotlin server is still used for generating mock testing data, gathering statistics about the results, as well as communicating with the front-end of the system.

5.2 Algorithm Implementation

While prototyping with a Kotlin implementation of a subset of the algorithms from Section 3 I noticed that performance could be significantly improved by moving to a C++ implementation of the algorithms. All of the algorithms mentioned in Section 3 were implemented. Additionally a modified version of the Max-Popular-Algorithm was implemented, which always computes a matching by exluding students matched to their last-resort seminar instead of making the whole computation fail.

5.2.1 Input and Execution

All five algorithms are compiled into a single binary `seminar_assignment` using CMake. No external libraries dynamic or static libraries need to be linked. The program accepts an optional argument which specifies the algorithm to be used and then reads the problem instance on stdin. In detail, a call to the program could look like this: `./seminar_assignment mode < instance.in`

In this call, `mode` should be replaced by one of the following arguments to select an algorithm for execution: "hungarian", "popular", "popular-modified", "rsd" (default) or "max-pareto". The input format is proprietary and domain-specific and is defined as follows:

```

seminar_count student_count
seminar_id seminar_capacity
...
student_id pref_list_length pref_0 pref_1 .. pref_n
...
```

Listing 1: Program input format

An example for this is the following instance with 4 students and seminars:

```

4 4
0 1
1 3
2 4
3 1
0 3 0 1 2
1 1 0
2 4 0 3 2 1
3 1 1
```

Listing 2: Example instance input

The program expects that the IDs of both students and seminars are enumerated consecutively beginning at zero, because those entities are collected into vectors, where the entities' IDs are being used as an index of the respective vector. The output of the program is written to stdout and consists of the algorithms runtime in milliseconds, followed by a newline, then the number of matched pairs, followed by a new-line and then a line for each matched pair consisting of the student-ID and space separated seminar-ID. An example of this is the following Listing 3, which was produced by executing the Hungarian algorithm on the instance from Listing 2:

```

20
4
0 1
1 0
2 3
3 1
```

Listing 3: Output for input from Listing 2 with the Hungarian algorithm

The C++ program does not produce any statistics or any other side-effects. It is the responsibility of the web-server to process the matching and gather statistics. The server uses Java's Runtime API internally to create a process that executes the C++ program, by writing the problem's instance in the previously described format to the new process' stdin. The server launches each new process in a new coroutine that suspends execution until the process terminates. The resulting matching is then collected by parsing the process' stdout and creating an internal representation of it.

5.2.2 RSD Implementation

This mechanism was the easiest to implement: The vector of students is first shuffled and then iterated over. Every student is directly assigned to their most-preferred, available seminar. The exact runtime lies somewhere between $\mathcal{O}(n)$ and $\mathcal{O}(n * m)$ depending on the instance, with n being the number of students and m the number of seminars.

5.2.3 Max-Pareto Implementation

This algorithm makes use of the Hopcroft Karp algorithm for finding a matching. In order to use the algorithm, the vector of seminars is first expanded to take the capacities into account, by creating one new seminar for every each item of capacity. Using the vector of students and this expanded vector of seminars, a maximum cardinality is then computed. This matching is then processed, by trying to promote any student to a more-preferred seminar, if it is available, and then implementing any possible trades. For the last part, I implemented a simple graph data structure, which is used for creating a preference-graph for detecting cycles which correspond to trades. The algorithm loops until that graph is empty, i.e. all trades have been implemented. At that point the matching is pareto-optimal as described in Section 3.2 and is then returned. The runtime of the implementation is dominated by executing the Hopcroft-Karp algorithm and therefore yields a total of $\mathcal{O}(|E|\sqrt{n + C})$ time complexity, where C is the sum of capacities of all seminars.

5.2.4 Assignment Problem Implementation

For the assignment problem, I used an existing open-source implementation of the *Kuhn-Munkres/Hungarian* algorithm that computes a min-weight matching in $\mathcal{O}(n^3)$ time [27]. The algorithm transforms the vectors of students and seminars into a cost matrix, which is then used by the algorithm to produce a matching. As expected, this algorithm performs the poorest in terms of runtime complexity among the other algorithms. It has to be noted that, due to the input transformation the actual runtime of the algorithm is $\mathcal{O}((n + C)^3)$, where C is the sum of capacities of the seminars. This makes this algorithm less desirable for larger instances.

5.2.5 Max-Popular Implementation

This algorithm first assigns all students whose first preference would be under-subscribed if all students are assigned to their first preference. Afterwards the Hopcroft-Karp algorithm is used again to match the unmatched students to seminars. The input is transformed the same way as in Section 5.2.3, however the runtime bottleneck is improved significantly due to the fact that there is only two edges per student, one for the f-house and one for the s-house. Therefore the runtime of this step is $\mathcal{O}(2n_1\sqrt{n_1 + C_1})$, where n_1 is the number of unassigned students and C_1 is the total capacity left at this point. If the matching produced by the Hopcroft-Karp algorithm is

not agent-complete, the algorithm returns an empty matching. Otherwise as a last step, the algorithm tries to promote students to better seminars, because the Hopcroft-Karp algorithm does not guarantee that students are matched to their more preferred seminar. In total, the runtime is again dominated by executing the Hopcroft-Karp algorithm, however due to the fact that it is operating on a reduced input, the complexity is better than in 5.2.3.

5.2.6 Modified Max-Popular Implementation

Due to the fact, that the Max-Popular algorithm does not always produce a matching, I implemented a modified version, which simply removes students assigned to their last-resort house, instead of making the algorithm fail. During the evaluation it will be interesting to see, if the matchings produced by this algorithm are still more popular than the ones produced by the other mechanisms.

5.3 Web-Server Implementation

For the implementation of the Web-Server I chose the Kotlin programming language in conjunction with the Ktor framework. Just like Java, Kotlin compiles to Java Bytecode, but provides meaningful extensions to the language such as nullable types and coroutines. Ktor is a web-server-framework that is build on top of Kotlin's coroutines to provide a declarative API for defining web-endpoints, while allowing non-blocking execution due to the coroutines' suspending nature. To represent the student and seminar data internally the server utilizes classes to represent students and seminars and persists them to a file by serializing the data model to JSON, whenever a value changes. Ideally, for something other than a research prototype, an alternative persistence solution, such as a database, should be used. The server provides endpoints for creating and deleting students and seminars, downloading all data in JSON-format, computing a match and finally a *WebSocket* endpoint that provides real-time data-updates to the clients. When establishing a connection with the WebSocket, the app's data consisting of a list of the currently saved students and seminars is send through the WebSocket in a JSON-representation. Any updates to the app's data from any connection is then also propagated through any open WebSocket connections. This allows the clients to implement real time updates, while keeping the load of the server low by suspending the execution of each connection until data is updated. In order to compute a matching clients make a GET-request to the server's matching input, by providing the name of the algorithm in the request's path. The server then, executes the C++ program using that algorithm and the currently stored student and seminar data. The response of the endpoint is a JSON payload consisting of a map of seminars to students, the profile of the matching and the count of unassigned students.

5.3.1 Benchmark Tools

In addition to handling HTTP-requests, the web-server also includes infrastructure for running and evaluating experiments with the algorithms. A variety of test-data with different preference distributions can be generated and used for evaluation key metrics of the algorithms. Section 6 will explain this in more detail. Implementation wise, the benchmark launches a coroutine for each execution of the program to allow concurrent execution. The matching data is then processed, analysed and a directory with results is being created. The result directory includes the inputs, statistics and charts visualizing the results.

5.4 Web-Interface Implementation

The Front-End of the system is implemented with the Flutter framework, which is a declarative UI-framework based on Google's Dart programming language that can target the Web, Android, iOS and many desktop platforms. For the Front-End the Dart-code is transpiled into Javascript-code that draws the UI specified in the source code on a canvas in the body of the HTML page. When loading the page, the website established a WebSocket connection with the web-server to fetch the initial data and also to listen for updates. The website consists of three tabs: The first tab lists the students, the second tab lists the seminars and the last tab allows for computing a matching and viewing the results. Additionally the first two tabs allow for creating new student and seminar entities by filling out a simple form with the required data, as well as deleting any of those entities. Create and delete operations don't directly manipulate any data in the browser, but instead make HTTP-requests to change the data on the server, which then propagates the changes through the WebSocket connection to the website again. That way, the website treats the server as the single-source of truth for all app-data and implements real-time capabilities.

6 Evaluation

We have already looked at other results in the literature in Section 4.2 which gave a decent comparison between some of the algorithms. However, we did not get a lot of knowledge about the datasets, especially the structure of the preference list and distribution of those preferences. Therefore we will conduct an experiment, by benchmarking the implemented algorithms with both real and artificial datasets of varying sizes and preference structures.

6.1 Research Questions

Table 2 already compared and summarized the algorithms from a theoretical perspective, however some of the results would benefit from further quantification and clarity. For instance, we know that the Popular-CHA algorithm, as described in Section 3.4, does not always produce a matching. It would be beneficial to empirically investigate if certain inputs cause this problem. Therefore we will formalize some open questions as research questions, that this experiment hopes to answer:

1. What is the effect of different preference distributions on the matchings produced by the different algorithms.
2. How likely is it for the Popular-CHA algorithm to not produce a matching using different datasets?
3. How does the Modified-Popular algorithm from Section 5.2.6 compare in terms of popularity to the other algorithms.
4. What is the cost of giving up on strategy proofness in terms of rank, i.e. how well does RSD perform compared to other mechanisms?
5. Is Popularity a meaningful metric for the student-seminar use case?
6. What is the impact of short-preference lists on the quality and existence of matchings?

6.2 Sample Data

Due to the fact that real data for seminar registrations is mostly kept private, a majority of the benchmark will rely on synthetic data. However, there are a few publicly available datasets available that will be used for evaluation and also for better understanding real-world preference distributions.

6.2.1 Using graph generators

Using graph generators to synthetically create instances to the seminar-student matching problem turned out to be a big challenge. Essentially, we would need a graph generator

for bipartite, weighted graphs, that also considers the seminar capacity. One option would be generating as many nodes as the total capacity of all seminars, but that could result in duplicate edges or i.e. preferences. Therefore we will use a custom, domain specific mechanism for generating instances.

6.2.2 Random Uniform Preference Lists

Probably the simplest way to synthetically generate data, was picking a size for the instance and then generating seminars and students based on a uniform random generator. The preference lists are created by shuffling the list of all seminars for each student. Additionally, we can simply shorten each preference list by a random factor as well for measuring the effect of incomplete-preference lists.

6.2.3 PrefLib Datasets

Preflib.org is an open collection of more than 3000 community-contributed datasets of preference data for different domains [28]. Fortunately, there exist two datasets of students' course preferences at the polish AGU University [29]. The two datasets contain strict and complete preference lists for each dataset with 9 courses and 146 students or 7 courses and 153 students respectively. Unfortunately there are no capacities given, however those will be computed synthetically. An interesting characteristic of those datasets is that in each dataset, all students rank the same seminar with their first preference. I decided to remove those seminars from the dataset, because I wouldn't assume such a preference distribution in the real world.

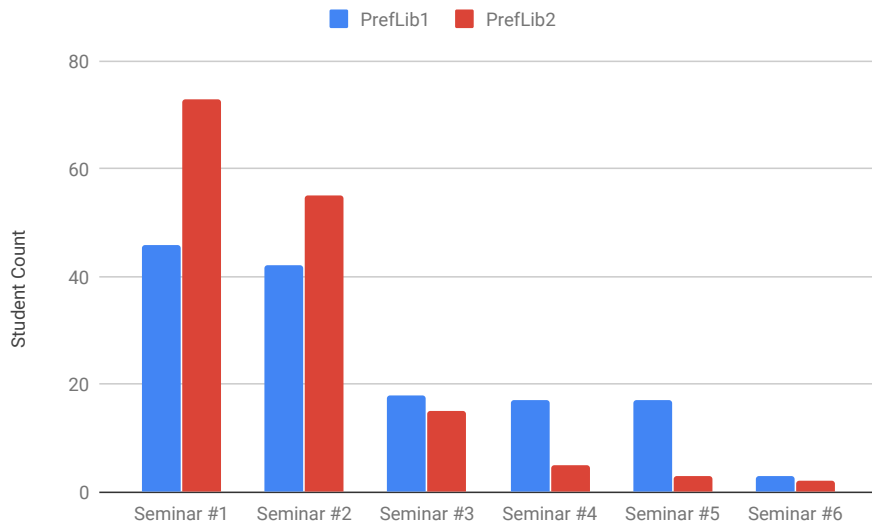


Figure 1: Preference distributions for first-ranked seminars for both PrefLib datasets

Figure 1 shows the preference distributions for the student's first choice in both datasets after removing the always first-ranked seminar. We can see that in both

datasets, students clearly strongly prefer two seminars. However in the first dataset, there are 3 more seminars that are also decently popular, while in the second dataset the majority of students really prefers the two dominant seminars. Generally, those datasets indicate that the preference structure of real-world datasets is not likely to be uniformly distributed.

6.2.4 Random Zipf-Distributed Preference Lists

To better simulate real-world preference structures than by using a uniform distribution, a power-law distribution can be applied. According to Zipf’s law the frequency of a word in a large sample, is proportional to it’s position in a frequency table [30]. This law can also be applied for creating synthetic seminar distribution. Using this type of distribution with some additional randomization yields the following preference list structure when seminar and student counts are similar to the first PrefLib dataset:

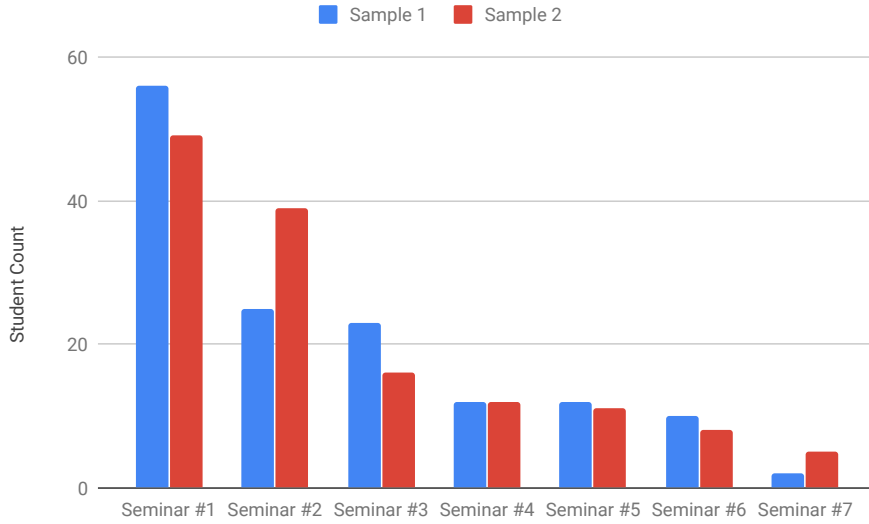


Figure 2: Two sample Zipfian preference distribution for first-ranked seminars

To create the dataset, for each student a seminar is randomly drawn using the Zipfian distribution and added to the his preference list until the desired preference-list length has been reached. Comparing Figure 2 to Figure 1 shows somewhat similar results, which should make this generator relevant for the benchmark.

6.3 Methodology

The goal of this experiment is quantifying some of the optimality criteria defined in Section 2 to compare the selection of algorithms empirically and find answers to the research questions proposed in Section 6.1. Due to the fact that real-world data for student enrollments is not widely available, it is important to note that the results of this experiment will be biased by the selection of data available. However, by using the

available real datasets and a few different synthetic datasets it should be possible to find answers to the proposed research questions and to better understand the differences of different approaches.

6.3.1 Benchmark Setup

The benchmark is performed by a Kotlin program that generates test data and then executes each of the algorithms with the same input. The program then collects the results and computes statistics on each matching, which are saved into a file. When using synthetic data, the program will generate between 10-50 different instances to account for the randomness. All experiments are run on a workstation with a 4-core (8 threads) Intel Core i5 8250U processor and 16GB of memory. The following types of instances will be used:

1. PrefLib datasets
2. Small Zipfian dataset (~ 200 students)
3. Large Zipfian dataset (~ 2500 students)
4. Large Uniform with incomplete preferences (5000 Students)
5. Large Uniform dataset with complete preferences (5000 Students)

6.3.2 Metrics

We already know that all of the algorithms under consideration produce pareto-optimal matchings, however, it will be necessary to measure popularity. If the Popular-CHA algorithm finds a matching, we know that it's the popular matching, however if the algorithm cannot find a matching it could be possible that one of the other algorithms produces such a matching. In general, given the definitions for Popularity that we have used before, it is not possible to check if a given matching is popular without comparing it to all other matchings. However this is infeasible due to the exponential runtime complexity of that comparison, which is why we will just compare the matchings produced by the five algorithms for Popularity. Given two matchings $m, m' \in \mathcal{M}$, we say that m is more popular than m' if the number of students preferring m is greater than the number of students preferring m' . In addition to comparing Popularity, the following other metrics will be used:

- **Profile:** The profile of the matching as defined in Section 2.4 given as an array.
- **Average Rank & Standard Deviation:** The average and standard deviation of the matched students' ranks. A matched student's rank corresponds to the position of his match on his preference list. In case there are unassigned students, two numbers will be given for this metric. First the metric excluding unassigned students is given and then in parentheses the metrics including unassigned students, weighted with the maximum rank is given.

- **Worst Rank:** In conjunction with the previous metric, we will also look at the worst rank that exists in a matching.
- **Unassigned-Count:** The number of unassigned students in a matching.
- **Runtime:** The runtime of the algorithm in milliseconds. Only the runtime of the actual algorithm in C++ is measured without parsing the input data or printing the result.
- **Existence:** This metric is only interesting for the Popular-CHA algorithm and will indicate if the algorithm was able to compute a matching for a given instance.

These metrics should be a sufficient selection to quantify a matching’s optimality and therefore should make it possible to answer the research questions from section 6.1.

6.4 Results

6.4.1 PrefLib1 dataset

The PrefLib datasets contain strict, complete preferences for a small number of students and seminars. Table 4 shows results for executing the algorithms on the first dataset. Unfortunately, the Popular CHA algorithm did not compute a matching, however we can also see that the Mod-Popular algorithm computed a matching that ties in popularity with the matching produced by the Hungarian algorithm. Unsurprisingly, in terms of rank the Hungarian algorithm produces the best matching with an average rank of 1.513 and a standard deviation of 0.829.

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.787	1.924	1.513	n/a	1.276 (2.342)
Rank SD	1.136	1.426	0.829	n/a	0.540 (3.079)
Unassigned-Count	0	0	0	146	16
Runtime	<1ms	19ms	13ms	1ms	1ms
More Popular	1.5	1.5	3.5	0	3.5
Worst Rank	7	7	4	-	3 / Unassigned
Exists	yes	yes	yes	no	yes

Table 4: Summary of the results for PrefLib1

When not taking the unassigned students into account, the Mod-Popular CHA algorithm performs even better on the rank metrics, however if taking the unassigned students into account it performs worst among all algorithms. Figure 3 shows the rank distribution for the algorithms that produced a matching and makes clear that, the Mod-Popular CHA algorithm assigns more students to their first rank at the cost of leaving a lot of students unassigned.

What else is surprising with this instance is that the greedy RSD algorithm performs better than Max PaCHA in regards to the rank metric, even though both are of

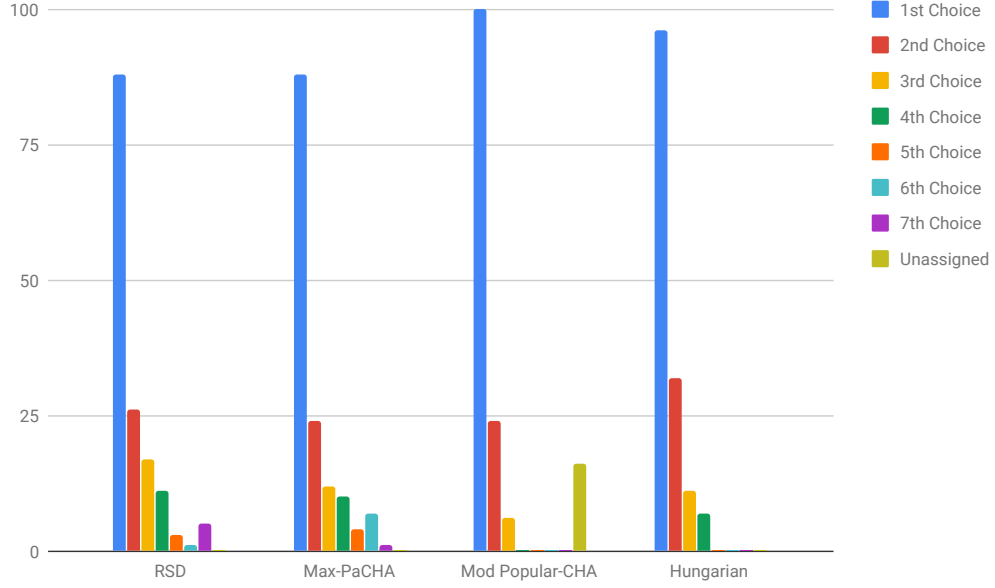


Figure 3: Rank distribution for PrefLib1 Dataset. Note: Popular CHA failed.

maximum cardinality. Surprisingly the algorithms also tie for popularity for this instance, even though RSD performs better from a rank perspective. Runtime wise, there are no surprises other than the fact that the Hungarian algorithm is faster than Max PaCHA for this instance. However, with this size of input the runtime results are not that significant.

6.4.2 PrefLib2 dataset

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.712	1.758	1.640	1.686	1.686
Rank SD	0.797	0.878	0.728	0.787	0.787
Unassigned-Count	0	0	0	0	0
Runtime	<1ms	41ms	45ms	9ms	10ms
More Popular	1	0	2 (+2 Ties)	2 (+2 Ties)	2 (+2 Ties)
Worst Rank	5	5	3	3	3
Exists	yes	yes	yes	yes	yes

Table 5: Summary of the results for PrefLib2

Table 5 shows the results for the PrefLib2 dataset. We can see that the Popular CHA algorithm finds a matching, which is why the last two columns are identical. What's interesting about the results is that the matchings produced by the Hungarian and Popular CHA algorithm tie in popularity, even though they are different in regards to the rank metrics.

What else stands out is that RSD again produces a better matching than Max PaCHA in regards to the rank metrics. In fact, the matching produced by the Max PaCHA algorithm performs worst in terms of rank, even though its runtime is the second highest. Overall the runtimes are still low, with all of them being lower than 50ms, which should make all algorithms usable for real world applications for similarly sized instances. However it has to be noted that the matching produced by RSD gets close in terms of rank to the matching produced by the Hungarian algorithm, even though it is about 50 times faster.

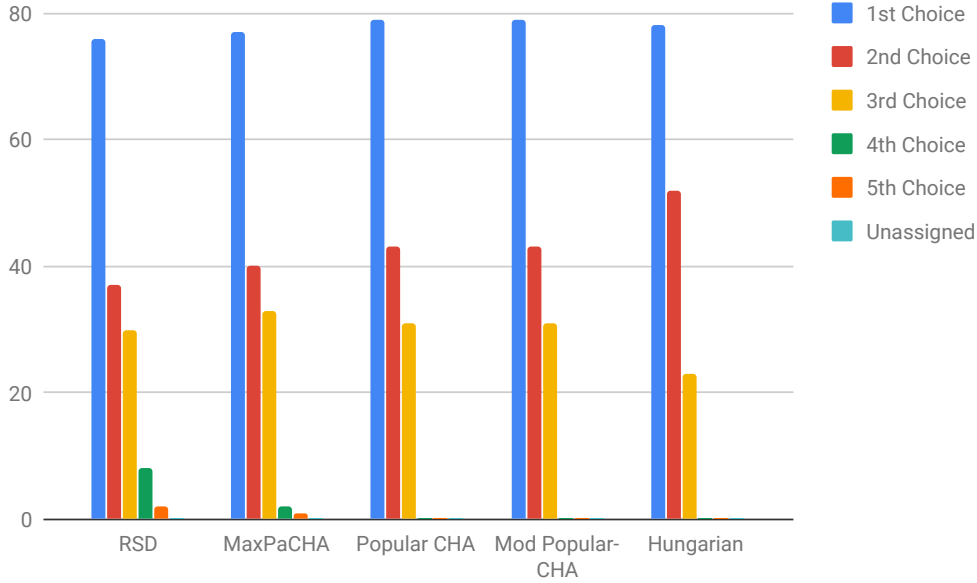


Figure 4: Rank distribution for PrefLib2 Dataset.

Figure 4 shows the preference distribution of all five algorithms. We can see that RSD and Max-PaCHA assign some students to their 4th or 5th choice, which explains why they perform worse in terms of rank. Unsurprisingly the matching produced by the Hungarian algorithm has the best rank-profile.

6.4.3 Small Zipfian datasets

Since the Zipfian datasets have a somewhat similar preference distribution as the PrefLib datasets we should see similar results, however parameters can be tuned such as preference list length and student count. We will begin by looking at a similarly sized dataset as the Preflib datasets and then begin deviating the student & seminar count. Table 6 shows the average results for 50 test runs using the Zipfian Dataset with 10 seminars and about 200 students. Out of the 50 instances, the Popular CHA algorithm only found a matching for four instances. When it found a matching, it did so quite fast and the matching was on average better rank-wise than all other algorithms except for the Hungarian algorithm. In terms of popularity, the Mod-Popular CHA

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.893 (2.014)	1.910	1.481	1.646	1.523 (1.913)
Rank SD	1.451 (1.782)	1.457	0.716	1.034	0.874 (2.121)
Unassigned-Count	2	0	0	0	7
Runtime	<1ms	54ms	60ms	7ms	10ms
More Popular	1	1.65	3.05	0.7	3.6
Worst Rank	8	8	5	5	6
Exists	yes	yes	yes	4/50 exist	yes

Table 6: Average results for small Zipfian dataset with 50 runs

algorithm actually outperformed all algorithms (equal to Popular-CHA if the matching exists) for most instances, even though rank-wise it does not perform as well as the Hungarian algorithm. However, the Hungarian algorithm produced more popular matchings than most other algorithms, and for some instances even more popular than the Mod-Popular CHA algorithm.

6.4.4 Large Zipfian datasets

Using a large dataset with a Zipfian distribution yields similar results. Table 7 summarizes the average results for 10 test runs with datasets that contain about 2500 students and 35 seminars.

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.911 (5.287)	2.109	1.824	-	1.694 (4.746)
Rank SD	1.225 (10.41)	1.383	1.029	-	1.087 (9.967)
Unassigned	10%	0%	0%	100%	8.6%
Runtime	3ms	2190ms	65730ms	-	168ms
More Popular	1.1	1.65	3	0	4
Worst Rank	7	7	6	-	7
Exists	yes	yes	yes	0/10	yes

Table 7: Average results for large Zipfian dataset (2500 Students) with 10 runs

Again, the Popular CHA algorithm was not very successful and found no matching in any of the test runs. However, the Mod-Popular CHA algorithm always produced more popular matchings than all of the other algorithms, even though 8.6% of the students were left unassigned on average. When taking the unassigned students out of the rank metrics, the Mod-Popular CHA algorithm produces matchings with the best average rank and standard deviation, while having the second lowest runtime. In contrast to that, the Hungarian algorithm always assigned all students, however needed on average 65 seconds to do so. This confirms the theoretical observations about runtime and shows that using the Hungarian algorithm is potentially not feasible for large instances.

Figure 5 shows the rank distributions for all algorithms but Popular CHA. We can see that Mod Popular-CHA is able to outperform the other algorithms in terms of

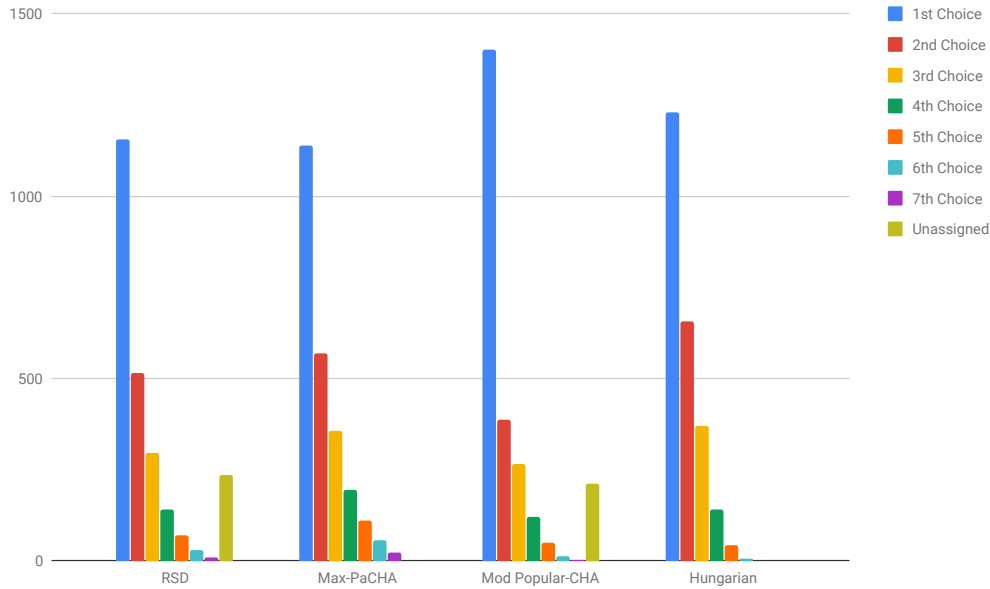


Figure 5: Rank distribution for large Zipfian dataset.

popularity by maximizing the number of students being assigned to their first choice. However, the Hungarian algorithm produced more balanced matchings, where the most students were assigned to their top three choices, while also leaving no student unassigned.

6.4.5 Large uniform dataset with incomplete preferences

This large uniform dataset contains 50 seminars and 5000 students, who each randomly picked 10 seminars with equal probability.

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.119 (1.265)	1.133	1.041	1.093	1.083
Rank SD	0.647 (2.778)	0.713	0.2	0.508	0.486
Unassigned	0.28%	0%	0%	0% (90%)	0%
Runtime	13ms	6175ms	55815ms	257ms	221ms
More Popular	1.65	1.15	3.45	0.3	3.45
Worst Rank	10	10	2	8	10
Exists	yes	yes	yes	1/10	yes

Table 8: Average results for large uniform dataset with incomplete preferences.

Table 8 shows the average results when executing each algorithm 10 times. Unfortunately the Popular CHA algorithm only found a matching for one out of the 10 instances, but when it did, it performed quite well in terms of rank. However, the Mod-Popular CHA algorithm was on average able to find better matchings in less time

than the Popular CHA mechanism. What also stands out is that Mod-Popular CHA and the Hungarian algorithm are tied for popularity in every instance, but runtime wise Mod-Popular CHA performs far better than the Hungarian algorithm. What needs to be noted though, is that the Hungarian algorithm outperforms all other algorithms in terms of profile by having the worst rank in all matchings be 2. All algorithms assign more than 94% of the students to their first choice, but the Hungarian algorithm only assigns students to their first or second choice.

6.4.6 Large uniform dataset with complete preferences

When using the same size dataset as before, but with complete preference lists, the results show some interesting insights. Table 9 shows that Popular-CHA found a matching for every instance, which helps answer the question if preference list length has an effect on Popular CHA. For this dataset, Popular CHA and the Hungarian algorithm always produce matchings of the same popularity, which means that there exists more than one popular matching for every instance.

Metric	RSD	Max PaCHA	Hungarian	Popular CHA	Mod-Popular
Average Rank	1.198	1.190	1.039	1.076	1.076
Rank SD	1.543	1.469	0.193	0.453	0.453
Unassigned	0%	0%	0%	0%	0%
Runtime	11ms	7922ms	57141ms	228ms	219ms
More Popular	0.5	0.5	3	3	3
Worst Rank	30	27	2	9	9
Exists	yes	yes	yes	10/10	yes

Table 9: Average results for large uniform dataset with complete preferences.

Another things that stands out is the fact that RSD and Max PaCHA produce matchings with high worst-ranks of 30 and 27 respectively. Overall these two algorithms perform worse or equal on all metrics than Popular CHA and the Hungarian algorithm, which makes them less appealing for this kind of dataset. Given the low average runtime of 219ms the Mod-Popular CHA algorithm seems to be a good option when the size of the instance is too large for the Hungarian, while still performing well on the rank-related metrics. Even though Popular-CHA and the Hungarian algorithm tie in popularity, it should be assumed that a matching with a worst rank of 2 is more popular among the students than a matching of worst rank 9.

6.5 Conclusion

The experiments have revealed some interesting insights that should help answering the research questions from section 6.1.

1. **Effect of different preference distributions:** Unsurprisingly, the algorithms performed better on the rank metrics, when the preferences were distributed

uniformly. In fact, the Hungarian algorithm was able to assign on average more than 94% (Table 9) of the students to their first choice when the preference distribution was uniform.

2. **Failure rate of Popular-CHA:** The experiments have shown that Popular-CHA fails to find a matching quite often, especially if the preference lists are short and not uniformly distributed. When using the Zipfian datasets (Tables 7, 6), Popular-CHA only found a matching for 4/60 instances. Even for uniform distributions and incomplete preference lists (Table 8) the algorithm only found a matching for 1/10 instances. Making the preference lists complete makes it easier for the algorithm to find matchings as shown in Table 9.
3. **Performance of Mod-Popular-CHA:** The modified version of Popular-CHA was able to find comparatively good matchings in a short amount of time, while also usually producing more or equally popular matchings as the other algorithms. For many instances Mod-Popular-CHA tied the Hungarian algorithm for popularity and was able to produce good matchings in terms of the rank metrics in a fraction of the time.
4. **Cost of giving up on strategy-proofness:** Surprisingly RSD produced on average better matchings on most metrics than Max-PaCHA, while also being by far the fastest algorithm. However, RSD usually produced the highest percentage of unassigned students, but the modified Popular-CHA algorithm usually outperformed RSD on most metrics, while also having a relatively low runtime. In the real-world datasets (PrefLib), RSD left no students unassigned and got close to the Hungarian algorithm in terms of rank, but also had 7 as the worst rank, compared to 4 for the Hungarian algorithm.
5. **Popularity as a meaningful metric:** For many instances we could not find the one popular matching, as Popular-CHA simply failed, but when it didn't, the worst rank, average rank and rank standard deviation was higher than what the Hungarian algorithm produced. Another interesting insight was that matchings produced by Mod-Popular-CHA usually tied in popularity with those produced by the Hungarian algorithm, even though Mod-Popular-CHA often left a few students unassigned. For instance in Figure 5, Mod-Popular-CHA left on average 8.6% of the students unassigned but the matchings were still classified as more popular or just as popular as the matchings produced by the Hungarian algorithm. This shows that a popular matching can maximize the number of students ranked to their first choice at the cost of some students being matched to a low-ranking seminar or even none. This makes it questionable, whether popularity is a meaningful optimality criteria for the student and seminar usecase, as it should be a goal to assign as many students as possible and to make all students equally satisfied with their match instead of finding the biggest possible subset of students, where everyone is satisfied.

6. **The effect of short preference lists:** The experiments have shown that short preference lists make it more likely for Popular-CHA to not find a matching, as well as making it more likely for RSD to produce matchings with more unassigned students.

In summary, the matchings produced by the Hungarian algorithm usually perform best in all of the metrics, except for runtime where it performs the worst by far. However, for the student-seminar use case that should not be a problem, because most instances will be smaller than what we have tested with. In fact, when using the real world datasets (see Table 4 and 5), the runtime differences for the different mechanisms were insignificant. For large instances, the modified Popular-CHA algorithm performed well and it could still be optimized even further for matching more students. The algorithm as it was in described in section 5.2.6, does not try to assign students that were left unmatched by the Hopcroft-Karp algorithm, which could easily be done by either using a simple mechanism like RSD or even using the Hungarian algorithm on the set of unassigned students. Neither modification guarantees that the final matching would be popular or of maximum cardinality, but it could improve the matching. Therefore it would make the Mod-Popular-CHA algorithm a fast heuristic that should, according to the experiments, perform better than RSD and Max-PaCHA for most instances, while still keeping the runtime low.

7 Extensions to the problem

For the bulk of this thesis, we have looked at many-to-one matching problems with one-sided preferences, as this settings makes the most sense for the student-seminar application. However, there is a wide range of similar problems and extensions that are also worth mentioning. This section will present some of those problems and key results from the literature.

7.1 Two-Sided preferences

In many practical one-to-many matching schemes, both parties will express preferences over the other. For instance, many countries use such schemes for matching students to university programs, which can be viewed as an instance of a one-to-many matching problem with two-sided preferences. The same problem exists in the US, where graduating medical students need to be matched to residency positions in hospitals.

Since we are dealing with two-sided preferences now, it is not possible to simply transform this problem into a weighted bipartite graph anymore, since edges now have two weights. It could be possible to use the product of those two weights as an edge weight, however, in practice different mechanisms are being used that don't rely on the graph representation of the problem.

7.1.1 Stable matchings

In the context of two-sided preferences Stability is a commonly used optimality criteria that matchings should fulfill. It can be seen as the two-sided preference extension to Pareto-optimality and is defined as follows. Given a set H of hospitals and a set R of residents, a matching M of an instance I is stable iff there is no pair $(r_i, h_j) \in M, r_i \in R, h_j \in H$ with:

- r_i and h_j are on each other's preference lists
- either r_i is assigned in M , or r_i prefers h_j to $M(r_i)$
- either h_j is undersubscribed in M , or h_j prefers r_i to it is least preferred assigned resident in $M(h_j)$

Simply put, in a stable matching M , no pair of agents (r_i, h_j) should have an incentive to give up their current match to get matched to each other.

7.1.2 Algorithm

Gale and Shapley [6] proved that for every instance of the problem, there exists a stable matching that can be computed using a linear-time algorithm. There are two variants of the algorithm, one that computes a resident-optimal, hospital pessimal matching and one that does the opposite. That means, that in the first case, each resident is matched to the best hospital among all stable matchings, while each hospital is matched to the

worst set of residents among all stable matchings [7]. In the context of student-seminar matchings it would make sense to use the resident/student optimal algorithm, which goes as follows [7]:

Algorithm 2 Resident-oriented deferred acceptance algorithm

Input: set of Residents with preferences R , set of Seminars T

Output: Stable Matching M

```

while Some resident  $r$  is free and has a non-empty preference list do
   $h :=$  first hospital on  $r$ 's preference list
  if  $h$  is fully subscribed then
     $r' :=$  worst resident provisionally assigned to  $h$ 
    unassign  $r'$  from  $h$ 
  end if
  provisionally assign  $r$  to  $h$ 
  if  $h$  is fully subscribed then  $s :=$  worst resident provisionally assigned to  $h$ 
    for each successor  $s'$  of  $s$  on  $h$ 's list do
      remove  $s'$  and  $h'$  from each other's list
    end for
  end if
end while

```

An advantage of using this algorithm for matching students to seminars is that the order in which students are processed does not matter. Gusfield shows that for any permutation of the input, each hospital is assigned the same number of residents, the same set of residents is unassigned and each undersubscribed hospital is matched to the same set of residents [7].

7.2 Many-to-Many matchings

A natural extension of the many-to-one matching setting is many-to-many matching. It could very well be a requirement that students need to be matched to more than one seminar. We assume that students now also provide a capacity, i.e. the number of courses they want to be matched to. In that case we need to differentiate between one- and two-sided preferences again:

7.2.1 One-sided preferences

WIP

7.2.2 Two-sided preferences

For the case of two-sided preferences we have to slightly modify the definition of Stability again to accomodate for the fact that each entity can be assigned more than once: Given the set of hospitals H and residents R , let $h \in H$ and $r \in R$, so that h and r are acceptable to each other, unmatched and the following holds true:

- either h has unfilled places or prefers r to one of his matched partners **and**
- either r has unfilled places or prefers h to one of his matched partners

Using this definition of stability, Gusfield[7] proposes that an algorithm can be constructed using ideas from the hospital-oriented and resident-oriented (see Algorithm 2) algorithm, that finds a stable matching that's optimal for either of the sets. Again, such a stable matching exists for any instance of the problem [7].

7.3 Online variant

When solving the online-variant of the problem, the whole input is not available from the start. That means that the input needs to be processed piece by piece, or more formally: Given a bipartite weighted graph (U, V, E) , where U is known to the algorithm, vertices in V are unknown, but arrive one at a time, while also revealing their incident edges, find a matching that maximizes some objective function. These algorithms could be of interest in the case of a first-come first-serve course allocation system, or in other areas such as DVD-rental or online-advertisement allocation systems [31].

In the case of student-seminar assignments, we would assume that the set of courses and their capacities is known beforehand, and the students arrive later. One of the algorithms we have seen in Section 3 can be used for this problem, namely the RSD-algorithm. As a matter of fact, the algorithm will produce the same results for the offline and online case, given that the order in which students are processed is identical.

7.3.1 Online max-cardinality matching

There has been lots of research in particular on finding maximum-cardinality matchings with online inputs. The online-inputs are classified by how much information the algorithm possesses about the input order. For now, we will only consider the **Adversarial order**, where we assume no knowledge of the query sequence, which means that only U is known at the beginning of the algorithm, while we have no knowledge of V and E or the order they appear in [31]. To measure performance we will use the **competitive ratio** of an algorithm which is defined as follows. Given an instance of the problem I , the value of the objective function for the online algorithm is given as $ALG(I)$, and the value of the objective function for the best offline algorithm is given as $OPT(I)$. The competitive ratio is now computed as follows: $C.R. = \frac{ALG(I)}{OPT(I)}$ [31].

Simple algorithms for this online problem, are a greedy algorithm, that matches arriving vertices to any available neighbor, or a random approach that matches arriving vertices to a random neighbor. These mechanisms achieve a competitive ratio of $\frac{1}{2}$ [31]. An optimal, but yet simple algorithm was introduced by Karp et al. [32], which achieves a competitive ratio of $1 - \frac{1}{e} \simeq 0.63$. The algorithm, called Ranking, begins by permutating the known vertices of U in a random permutation π , i.e. we assign a random priority number to each $u \in U$. Each incoming vertex $v \in V$ is then assigned

to an available neighbor, with the smallest value of $\pi(u)$. In detail the algorithm looks like this:

Algorithm 3 Ranking

Offline: Pick a random, uniform permutation π of U

for each arriving vertex $v \in V$ **do**

if v has no available neighbors **then**

 continue

end if

 Match v to the neighbor $u \in U$ with the smallest value $\pi(u)$

end for

8 Discussion

The goal of this chapter is to provide an overview of both the theoretical and practical results and to give detailed comparisons and a trade-off analysis of the algorithms.

8.1 Discussion of theoretical and practical results

Drawing back to the theoretical results from Table 2, we have already seen that none of the algorithms produce matchings will all of the desired criteria. Especially strategy-proofness cannot be achieved if trying to maximize a matching's cardinality. Popular-CHA, which theoretically is strategy-proof, unfortunately does not always find a matching which makes it less suitable for real-world applications. Both the results from Diebold's (Table 3) as well as our experiment has shown that Popular CHA fails for about 90% of the tested instances. There is also no known algorithm that efficiently computes the popular matching if it is not of maximum cardinality, which makes it questionable whether Popularity should be considered when designing such a matching system. The results from Table 5 and 9 have shown that the matchings produced by Popular CHA and the Hungarian algorithm often tie in terms of popularity, even though the Hungarian algorithm clearly performs better on other metrics. Looking at the rank distributions of some matchings in Figure 4 and 2, we have seen that (Mod-) Popular-CHA attempts to maximize the number of students assigned to their first choice in order to find a popular matching. It is highly questionable if this property is desirable over having a matching with a lower rank average and especially a lower rank standard deviation, which is what the Hungarian algorithm typically produces. For this reason, I would argue that *Popularity* as an optimality criteria should not be of high importance when selecting a matching mechanism. The results have shown that popular matchings, compared to profile-based optimal matchings, do not explicitly attempt to provide a good match for every student, but instead give some students a bad or no match in order to give the majority of students a better match.

Another surprising finding from our experiment was that Max-PaCHA usually performed worse than all other algorithms, including RSD, on the rank metrics. This can easily explained by the fact that Max-PaCHA essentially performs a local search from an initial (maximum cardinality) matching and terminates when finding one and only one local optima. Contrary to that, the greedy approach (RSD) also performs a local search, however achieves better average ranks due to the fact that it does not attempt to maximize the matching's cardinality. Comparing RSD and Max-PaCHA to the other algorithms, the Hungarian algorithm and Mod-Popular-CHA should be preferred due to the fact that they simply performed better on almost all other metrics in the experiments. However, if using a strategy-proof mechanism is a requirement, RSD should obviously be used.

8.2 Potential algorithmic improvements

In Section 5.2.6, we have briefly discussed a modified version of the Popular-CHA algorithm that was used in the experiment to gain insights on failing instances of Popular-CHA. However, the modified algorithm leaves some room for optimization, as it does not attempt to match students that were either unmatched or assigned to their last-resort preference after performing the maximum cardinality matching. A simple way of improving this algorithm is to use one of the other algorithms (e.g. RSD) on that unmatched subset of students. Doing this would not worsen the matching's popularity, because being matched to any seminar is better than not being matched at all from a popularity perspective. Using this modified mechanism could be a good alternative to using the Hungarian algorithm when using large instances where runtime could be a problem.

Besides that, different algorithmic approaches could be used for computing a profile-based optimal matching. In Section 3.3 we have presented the approach of reducing the matching problem to the assignment problem and by using flow-networks to solve the problem. For the use case of student and seminar matchings, it can be assumed that instances will be relatively small, however the experiments have also shown that the Hungarian algorithm's runtime is quite poor compared to the other algorithms when using larger instances. For instance, when using about 5000 students (Table 8 and 9) the Hungarian algorithm took on average about 60 seconds to find a matching, compared to about 230ms for the Popular-CHA mechanism. A faster algorithm for finding a profile-based optimal matching, Rank-Max is presented by Sng et al. in [4]. They show that for a given instance I of the CHAT (with ties) problem, the algorithm finds a rank-maximal matching in $\mathcal{O}(\min(z^*\sqrt{C}, C + z^*)m)$, where z^* is the maximal rank of an edge in an optimal solution of I , C is the total capacity of the houses in I and m is the sum of the lengths of all preference lists in I . Compared to that, the version of the Hungarian algorithm used for the experiments has a runtime of $\mathcal{O}((C * n_1)^3)$ with n_1 being the number of students and C again being the total capacity of all seminars. While the theoretical runtime of Rank-Max is much better than of the Hungarian algorithm, we have seen that for small, real-world instances like the PrefLib datasets (Table 4) the runtime of the Hungarian algorithm was still below 20ms.

8.3 System design recommendations

As alluded to in the previous two subsections, the Hungarian algorithm provides the best results and from a distribution perspective probably the fairest distribution out of all algorithms. Important properties of the algorithm are that it finds matchings of maximal cardinality as well as matchings with the lowest average rank combined with a low rank standard deviation. While the runtime of that algorithm was the highest among the algorithms presented, performance should not be a problem for real-world student-seminar matching scenarios, where the instances are somewhat small. To improve performance the Max-Rank algorithm [4] could be used for finding an exact

result or, for really large instances, the Mod-Popular-CHA algorithm can be used. To improve performance of all algorithms it would also help to require students to supply at least k preferences, where k could be a fixed fraction of the total seminar count. The experiments have indicated that longer preference lists, especially when power-law-like preference distributions are used, make the algorithms perform better.

The online variant of the problem can be solved using a first-come first-serve mechanism like RSD or an algorithm like Ranking (Algorithm 3) that maximizes cardinality in the online scenario.

9 Conclusion

The goal of this thesis has been presenting and analyzing different matching mechanisms and their properties for one-to-many matchings with one-sided preferences. A variety of optimality criteria exist and it turns out that, none of the algorithms presented produce matchings that fulfill all of the optimality criteria.

The experiments have shown that the resulting matchings vary quite a lot by algorithm and structure of the instance. In summary, the Hungarian algorithm usually outperformed all other algorithms on most metrics, however it also had by far the highest runtime. To improve the runtime, different algorithms can be used, such as Rank-Max, that also compute profile-based matchings in almost linear time. Another insight from the experiments was that the Popular-CHA algorithm failed to find matchings for a majority of the instances, however we discussed a simple modified version of the algorithm that usually produced better results than RSD and Max-PaCHA, while having a much lower runtime than the Hungarian algorithm. However, according to the results from the experiment, *Popularity* seemed to be a less desirable criteria, due to the fact that popular matchings often only optimized the match for a subset of the students, while leaving the other students with a bad or no match. Therefore we can recommend to use the Hungarian or Rank-Max algorithm for the use case of student-seminar matching. To improve on performance and lower the bar for strategic manipulation, a matching system could require students to supply at least k preferences. This would also partially eliminate the Hungarian's algorithm biggest disadvantage, which was that students are encouraged to provide short preference lists.

As part of this thesis, we have also presented an interactive web-system that allows university administrations to manage and solve such matching problems. The system relies on the C++ algorithms, that were used for the benchmark and should offer good performance for typical instances.

References

- [1] F. Diebold and M. Bichler, “Matching with indifferences: A comparison of algorithms in the context of course allocation,” *European Journal of Operational Research*, vol. 260, no. 1, pp. 268 – 282, 2017.
- [2] A. E. Roth, “The evolution of the labor market for medical interns and residents: A case study in game theory,” *Journal of Political Economy*, vol. 92, no. 6, pp. 991–1016, 1984.
- [3] S. P. Fekete, M. Skutella, and G. J. Woeginger, “The complexity of economic equilibria for house allocation markets,” *Information Processing Letters*, vol. 88, no. 5, pp. 219 – 223, 2003.
- [4] C. Thiam Soon Sng, *Efficient Algorithms for Bipartite Matching Problems with Preferences*. PhD thesis, University of Glasgow, 7 2008. A thesis submitted to the Faculty of Information and Mathematical Sciences at the University of Glasgow for the degree of Doctor of Philosophy.
- [5] D. F. Manlove, *Algorithmics of Matching Under Preferences*, vol. 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013.
- [6] D. Gale and L. S. Shapley, “College admissions and the stability of marriage,” *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.
- [7] D. Gusfield and R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*. Cambridge, MA, USA: MIT Press, 1989.
- [8] A. Roth, “The theory and practice of market design,” Nobel Prize in Economics documents 2012-5, Nobel Prize Committee, 2012.
- [9] E. Ronn, “Np-complete stable matching problems,” *Journal of Algorithms*, vol. 11, no. 2, pp. 285 – 304, 1990.
- [10] A. E. Roth and E. Peranson, “The redesign of the matching market for american physicians: Some engineering aspects of economic design,” Working Paper 6963, National Bureau of Economic Research, February 1999.
- [11] J. Munkres, “Algorithms for the assignment and transportation problems,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.
- [12] D. B. West, *Introduction to Graph Theory (2nd Edition)*. Pearson, 2000.
- [13] A. E. Roth and M. A. O. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Econometric Society Monographs, Cambridge University Press, 1990.

- [14] A. Abdulkadiroglu and T. Sonmez, “Random Serial Dictatorship and the Core from Random Endowments in House Allocation Problems,” *Econometrica*, vol. 66, pp. 689–702, May 1998.
- [15] D. J. Abraham, K. Cechlárová, D. F. Manlove, and K. Mehlhorn, “Pareto optimality in house allocation problems,” in *Proceedings of the 16th International Conference on Algorithms and Computation, ISAAC’05*, (Berlin, Heidelberg), pp. 1163–1175, Springer-Verlag, 2005.
- [16] D. Manlove and C. Sng, “Popular matchings in the capacitated house allocation problem,” September 2006.
- [17] B. Klaus, D. F. Manlove, and F. Rossi, “Matching under preferences,” in *Handbook of Computational Social Choice* (F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, eds.), pp. 333–355, Cambridge ; New York: Cambridge University Press, April 2016.
- [18] D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn, “Popular matchings,” in *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’05*, (Philadelphia, PA, USA), pp. 424–432, Society for Industrial and Applied Mathematics, 2005.
- [19] P. Gärdenfors, “Match making: Assignments based on bilateral preferences,” *Behavioral Science*, vol. 20, no. 3, pp. 166–173, 1975.
- [20] A. E. Roth, “Incentive compatibility in a market with indivisible goods,” *Economics Letters*, vol. 9, no. 2, pp. 127 – 132, 1982.
- [21] M. Manea, “Serial dictatorship and pareto optimality,” *Games and Economic Behavior*, vol. 61, no. 2, pp. 316 – 330, 2007.
- [22] J. Kun, “Serial dictatorships and house allocation.” <https://jeremykun.com/2015/10/26/serial-dictatorships-and-house-allocation/>, Nov 2015.
- [23] L. Shapley and H. Scarf, “On cores and indivisibility,” *Journal of Mathematical Economics*, vol. 1, no. 1, pp. 23 – 37, 1974.
- [24] R. Jonker and A. Volgenant, “A shortest augmenting path algorithm for dense and sparse linear assignment problems,” *Computing*, vol. 38, pp. 325–340, Dec 1987.
- [25] H. N. Gabow, “An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems,” in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, STOC ’83*, (New York, NY, USA), pp. 448–456, ACM, 1983.

- [26] J. E. Hopcroft and R. M. Karp, “A $n^5/2$ algorithm for maximum matchings in bipartite graphs,” in *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pp. 122–125, Oct 1971.
- [27] C. Ma, “hungarian-algorithm-cpp.” <https://github.com/mcximing/hungarian-algorithm-cpp>, 2016.
- [28] N. Mattei and T. Walsh, “Preflib: A library of preference data [HTTP://PREFLIB.ORG](http://preflib.org),” in *Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT 2013)*, Lecture Notes in Artificial Intelligence, Springer, 2013.
- [29] “PrefLib: A library for preferences - agh course preferences.” <http://www.preflib.org/data/election/agh/>.
- [30] D. M. W. Powers, “Applications and explanations of zipf’s law,” in *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL ’98, (Stroudsburg, PA, USA), pp. 151–160, Association for Computational Linguistics, 1998.
- [31] A. Mehta, “Online matching and ad allocation,” *Found. Trends Theor. Comput. Sci.*, vol. 8, pp. 265–368, Oct. 2013.
- [32] R. M. Karp, U. V. Vazirani, and V. V. Vazirani, “An optimal algorithm for on-line bipartite matching,” in *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC ’90, (New York, NY, USA), pp. 352–358, ACM, 1990.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den July 22, 2019

.....