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 the student-seminar assignment problem 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, weighted graph yields the best results for most metrics and instances. Additionally, an interactive web system is presented that allows for managing and solving such matching problems by employing the presented algorithms.

Contents

List of Figures

List of Tables

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 [1]. There are different goals for these matching problems which can range from efficiency to fairness and finding a mechanism that fulfills these goals is an important task to efficiently design such matching 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. Additionally, the distribution of preferences needs to be considered as well. For instance, in the hospital-residents problem, both the hospitals and residents supply preferences over the other set. In that case it is important to consider if incomplete preference lists or ties should be allowed in the instances.

We will 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 to many universities, which often require their students to participate in a seminar. 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 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, because there are many 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 are of high popularity among the student body.

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 a set of metrics that makes sense for 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 one-sided preferences. Lastly, an interactive web system is presented, which allows a school's administration to find a student-seminar matching using one of the presented algorithms.

1.2. Formal Definition: Capacitated House Allocation Problem

Many economists and game theorists [2] study 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 $H := \{h_1, h_2, ..., h_n\}$ is the set of houses and $A := \{a_1, a_2, ..., a_m\}$ 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 $M : A \to H$, so that for every applicant a_i, the house $M(a_i)$ is on the applicants preference list [3].

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, one-to-many, variant of the problem is also referred to as the "Capacitated House Allocation Problem", denoted by CHA [4]. The following chapters of this thesis explore performance indicators and algorithms for finding matchings in the context of students and seminars, which is semantically equivalent to the CHA problem.

1.3. Formal Definition: Student-to-Seminar Matching

Similarly to CHA, we can describe the problem of assigning students to seminars as a many-to-one matching problem, with a set of students $S := \{s_1, s_2, ..., s_n\}$ and a set of seminars $T := \{t_1, t_2, ..., t_m\}$. Every student $s_i \in S$ provides a strict preference order over a subset of T, and every seminar $t_j \in T$ has a capacity of $c_j \in \mathbb{N}$ students. The goal is to find a matching $M : S \to T$, 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 discuss different optimality criteria which can be used for defining an objective function for the 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 for CHA 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 can be created and managed as part of the system. Lastly, we will briefly explore extensions to the problem, such as the many-to-many case and finally summarize the results and make recommendations for the implementation and usage of such a matching 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 1.2)
      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, A.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 [5]. 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*, maximum-cardinality 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 picks 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 [5].

It is important to note that the algorithm produces different results if the women are the proposers: It has been shown that all executions of the algorithm with all possible permutations of the input 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 [5]. 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 [5].

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 [5]. 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 also 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 [7]. 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 1.2) 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 does not exist is NP-complete [8]. 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 possible before [9].

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, maximum-cardinality, 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 $W : S \times T \to \mathbb{R}$, to find a map $M : S \to T$, 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 maximum-cardinality, minimum-weight matching in polynomial time [10]. 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' preferences 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 [4].

2. Optimality criteria

Looking at the previously presented problems, it becomes clear that there are different objective functions or optimality criteria for matching problems. 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 as many students as possible 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 [11]. 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 that students' preferences are not considered when computing a 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 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 $\succ_{pareto}$

on $\mathcal{M}$, where given two matchings $M, M' \in \mathcal{M}$ the following holds true: $M' \succ_{pareto} M$ 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 $M' \succ_{pareto} M$ [4]. 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) [12, 13]. However, the greedy algorithm does not always produce a Pareto-optimal matching of maximum cardinality, which would be desirable for student-seminar matchings [14].

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 [15]. 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 $M' \succ_{pop} M$, 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. $M' \succ_{pop} M$ [16, 17]. Because of that definition, this criteria is often also referred to as the majority assignment [18].

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 [16].

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 [17].

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 ith component indicates the number of students obtaining their ith-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 $\langle p_1, ..., p_{r^*} \rangle$, with $r^* = r(M)$ and for each $k \in [1, r^*]$, the kth component is defined as: $p_k = |\{(s_i, t_j) \in M : rank(s_i, t_j) = k\}|$ [4].

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. 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 [5]. In the literature, the term for a mechanism, wherein no agent can benefit from misrepresenting their preferences, is called strategy-proof [16]. We would desire that a matching mechanism encourages students to truthfully disclose their preferences by giving the students no opportunity for strategically manipulating the outcome of the used matching algorithm.

Such strategy-proof 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 [19] 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 negatively impacting other students.

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 preferred seminar.

As we have already seen, there does not always exist a popular matching or a pareto-optimal matching that is also *agent complete*, i.e. matches all students. 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. 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 also 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 [20]. 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 RSD-Matching(S, T)
    M := ∅
    for each s ∈ 𝒮 in random order do
        t := highest ranked, available seminar on preference list of s
        if t ≠ null then
            M = M ∪ {(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 [21]. □

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 [16].

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:

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

In this example, each seminar only has a capacity of 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_2 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 [14] have proposed a 3-phase algorithm for computing a maximum cardinality matching for the house allocation problem, which was extended by Sng [3] for the many-to-one case (CHA). 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 [14].

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 [14].

3. **Cyclic coalition**: M contains a cyclic coalition, if there exists a sequence of distinct assigned students $C = \langle s_0, s_1, \ldots, s_{r-1} \rangle$ 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 [14].

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 [14].*

Using this lemma, the authors [3, 14] 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-cardinality 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 [22]. 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 [14].

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 $\mathcal{O}(E\sqrt{V})$ [14] when using the Hopcroft-Karp algorithm. Phase 1 and 2 both take $\mathcal{O}(|E|)$ of time [3], 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 that the assignment problem tries to find a perfect matching, meaning that all agents are assigned, and that a one-to-one (HA), contrary to one-to-many (CHA), matching is found.

For this problem, there exist algorithms [10, 23] 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 performing one-to-many matchings.

However, we can see that the assignment problem is equivalent to finding a perfect, minimum-weight matching in bipartite, weighted graph. Therefore, 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: $E := \{(s, t) \mid s \in S \land t \in T \land \text{t is on the preference list of s}\}$. Additionally, a weight function $W : E \to \mathbb{N}$ 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 [3] and has the minimum weight property, and therefore is rank-maximal;[3] however, a large drawback is that this mechanism is not strategy-proof. Students are again 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 [17] 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 [15] 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 [15] define and proof 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 applicant $a_i \in A$ 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 [15]. The authors prove the following lemma [15]:

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 defintion of $f(a_i)$ and $s(a_i)$. Manlove and Sng, again prove the following two lemmas [15]:

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 $s(a_j)$. 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 $l(a_i)$ [15]. Manlove and Sng prove the following theorem to fully characterize popular matchings [15]:

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 at least 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 [15]. 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 [24].
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 [24], which computes the maximum cardinality matching in G' in $\mathcal{O}(\sqrt{C}n_1)$ [15]. Alternatively, a modified version of the Hopcroft-Karp algorithm could be used for computing the maximum cardinality matching in $\mathcal{O}(E\sqrt{V})$[25] 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.

3.5. Comparison of Mechanisms

In the previous section, we studied several different algorithmic approaches that guarantee different optimality criteria. 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 practical results that were obtained by performing experiments of matching mechanisms with one- and two-sided preferences by Diebold and Bichler [26].

3.5.1. Theoretical Results

Referring 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 non trivial. 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	$\mathcal{O}(n)$	$\mathcal{O}(\sqrt{n} * m)$	$\approx \mathcal{O}(n^3)$	$\mathcal{O}(\sqrt{C} * n_1 + m)$

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.

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 always produces maximum-cardinality matchings.

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 of 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.

3.5.2. Practical Results in the Literature

Diebold et al. have published results for extensive experiments on matching mechanisms with both one- and two-sided preferences [26]. They used real course registration data from TUM for investigating properties, including size, rank and popularity, of matchings produced by several mechanisms. The mechanisms are the same as described in Section 3, with one exception being that the *ProB-CHAT* algorithm [26] is used in place of the Hungarian algorithm for finding rank-maximal matchings.

The authors found that all algorithms' matchings achieve an average cardinality of atleast 97.48%. For one of the 9 instances, Popular-CHA failed to find a matching, but when it found one, the matchings' average ranks of 1.33 got close to the ones produced by ProB-CHAT of 1.26. Unsurprisingly, ProB-CHAT performed best on the rank metrics and also produced more popular matchings than all other algorithms, but Popular-CHA. However, its' max runtime was the worst with 33.852s, compared to the second highest 2.458s of Popular-CHA on a dataset with 915 students and 51 courses.

Another surprising finding is that RSD performed better on rank metrics with an average rank of 1.41 than Max-PaCHA with an average rank of 1.51. Generally, the average ranks of all algorithms are somewhat close by being in a range of 1.26 (ProB-CHAT) to 1.51 (Max-PaCHA). Unfortunately, the authors do not disclose more detailled information about their datasets and structure of matchings.

While these results confirm some of the theoretical observations, it will be interesting to get more insights with differently structured datasets being used. Additionally, most of the data contains ties and we do not get any meaningful insights on the distribution of preferences.

4. Implementation

As part of this thesis, we implement the algorithms from Section 3, as well as tools for a benchmark and an interactive web system consisting of a web server and front-end. This section will give a brief overview of the implementation of those components.

4.1. Overview

To allow for further experimentation with our findings, we will describe a prototype for an interactive web-system, built for computing student-seminar matchings. 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, editing and deleting student and seminar data.
2. Allow for computing a matching using the previously entered data.
3. Output the matching for a chosen algorithm and its properties.

This system produces matchings by using efficient implementations of the algorithms mentioned in Section 3. While the web server itself is build in the Kotlin programming language and therefore runs on the JVM, the algorithms are implemented in C++ for performance reasons. 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.

4.2. Algorithm Implementation

All of the algorithms mentioned in Section 3 are implemented in C++. Additionally, a modified version of the Max-Popular-Algorithm is implemented, which always computes a matching by excluding students matched to their last-resort seminar, as opposed to making the whole computation fail.

4.2.1. Input and Execution

All five algorithms are compiled into a single binary seminar_assignment using CMake. No external 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 looks 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 in Listing 1.

Each line begins with a character indicating what type of information the line contains, where d represents metadata about the instance, t represents seminar data and s represents

```
d seminar_count student_count
t seminar_id seminar_capacity
...
s student_id pref_list_length pref_0 pref_1 .. pref_n
...
```

Listing 1: Program input format

student data. An example for this is the following instance in Listing 2 with 4 students and seminars:

```
d 4 4
t 0 1
t 1 3
t 2 4
t 3 1
s 0 3 0 1 2
s 1 1 0
s 2 4 0 3 2 1
s 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. 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 resulting matching is then collected by parsing the process' stdout and creating a JVM-internal representation of it.

4.2.2. RSD Implementation

This mechanism is 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.

4.2.3. Max-PaCHA Implementation

This algorithm makes use of the Hopcroft Karp algorithm for finding an initial maximum-cardinality 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 matching 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, we implement 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 (maximum $n_1/2$ trades, with n_1 being the number of students). 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.

4.2.4. Assignment Problem Implementation

For the assignment problem, we 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]. Our implementation transforms the vectors of students and seminars into a cost matrix, which is then used by the Hungarian algorithm to produce a matching. As expected, this algorithm has the worst theoretical 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.

4.2.5. Popular-CHA 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 4.2.3; however, the runtime bottleneck is improved significantly due to the fact that there are 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 leaves students unmatched, 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 most preferred seminar. In total, the runtime is again dominated by executing the Hopcroft-Karp algorithm, however, because it is operating on a reduced input, the actual runtime is expected to be better than in 4.2.3.

4.2.6. Modified Popular-CHA Implementation

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

4.3. Web Server Implementation

For the implementation of the web server we choose 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 built 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, updating and deleting students and seminars, downloading all data in JSON-format, uploading a dataset, computing a matching 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 sent through the WebSocket in a JSON-representation. Any updates to the app's data from any connection are 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 endpoint 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.

4.3.1. 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 evaluating key metrics of the produced matchings. 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. Section 5 explains this in more detail.

4.4. 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 establishes a WebSocket connection with the web server to fetch the initial data and listen for updates. The website consists of a navigation drawer that provides access to the following 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 and updating student and seminar entities by filling out a simple form with the required data, as well as deleting any of those entities. Create, update 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 is also always up to date on changes that other users could be making. Additionally, the website offers the capability to download the currently used data as a JSON-file, as well as changing the dataset using one of the data generators used in Section 5 or by uploading a JSON-file.

4.5. Accessing the System and Source Code

A live version of the web system is hosted at `https://aaronoe.github.io`. Additionally, the source code for all components can be found in the following GitHub repository: `https://github.com/aaronoe/matching_thesis`. Detailed instructions on building from source are included in the repositories' README file, as well as shell scripts for running the components.

5. Experimental Evaluation

In Section 3.5.2, we looked at other results in the literature that gave a decent comparison between some of the algorithms, however, we did not learn much about the datasets, especially the structure of the preference list and distribution of those preferences. Therefore, we are conducting experiments by benchmarking the implemented algorithms with both real and artificial datasets of varying sizes and preference structures.

5.1. 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 formalize some open questions as research questions that these experiments try 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 Popular-CHA to not produce a matching using different datasets?
3. How does the Modified-Popular-CHA algorithm from Section 4.2.6 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?

5.2. Experimental Setup

The goal of these experiments 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 the previous subsection. 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 are 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.

5.2.1. Benchmark Setup

The experiments are 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, which are saved into a file. When using synthetic data, the program generates between 10-50 different instances to account for the randomness, but only one instance runs at a time to allow for ideal CPU utilization. All experiments are run on a workstation with an Intel Core i5 8250U processor and 16GB of memory. The following types of instances are being used:

1. Real-world datasets
2. Small, synthetic datasets (∼200 students) with real-world similar preferences
3. Large, synthetic datasets (∼2500 students) with real-world similar preferences
4. Large, synthetic dataset with uniform and incomplete preferences (5000 Students)
5. Large, synthetic dataset with uniform and complete preferences (5000 Students)

5.2.2. Sample Data

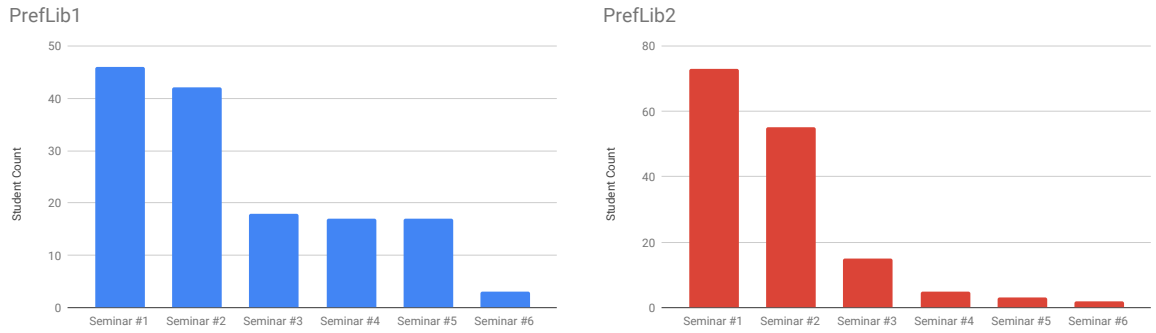
Due to the fact that real data for seminar registrations is mostly kept private, a majority of the benchmark will rely on synthetic data. However, there are a few publicly available datasets available that will be used for evaluation and also for better understanding real-world preference distributions.

Using Graph Generators Using graph generators to synthetically create instances to the seminar-student matching problem turns out to be a big challenge. Essentially, we need a graph generator for bipartite, weighted graphs, that also considers seminar capacities. One option is generating as many nodes as the total capacity of all seminars plus students, but that could result in having duplicate edges, i.e. preferences in the generated instance. Therefore we will use a custom, domain specific mechanism for generating instances.

Random Uniform Preference Lists Probably the simplest way to synthetically generate data is 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 to measure the effect of incomplete-preference lists.

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 orders for each dataset with 9 courses and 146 students, or 7 courses and 153 students respectively. Unfortunately,

there are no capacities given; however, those can easily be computed synthetically. An interesting characteristic of those datasets is that in each dataset, all students rank the same seminar with their first preference. We remove these seminars from the dataset, as to not assume such a preference distribution in the real world.



(a) PrefLib1 Dataset (b) PrefLib2 Dataset

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

Figure 1 shows the preference distribution for the student's first choice seminars in both datasets after removing the always first-ranked seminar. We can see that in both datasets, students clearly strongly prefer two seminars. In the first dataset, however, 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.

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. One type of Power Law distributions are Zipfian Distributions [30], which can be used for creating synthetic seminar distributions. 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:

To create the dataset, for each student a seminar is randomly drawn using the Zipfian distribution and added to 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.

5.2.3. Metrics

We already know that all of the algorithms under consideration produce pareto-optimal matchings; however, it will also be necessary to measure popularity. If the Popular-CHA algorithm finds a matching, we know that it is the popular matching, but 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,

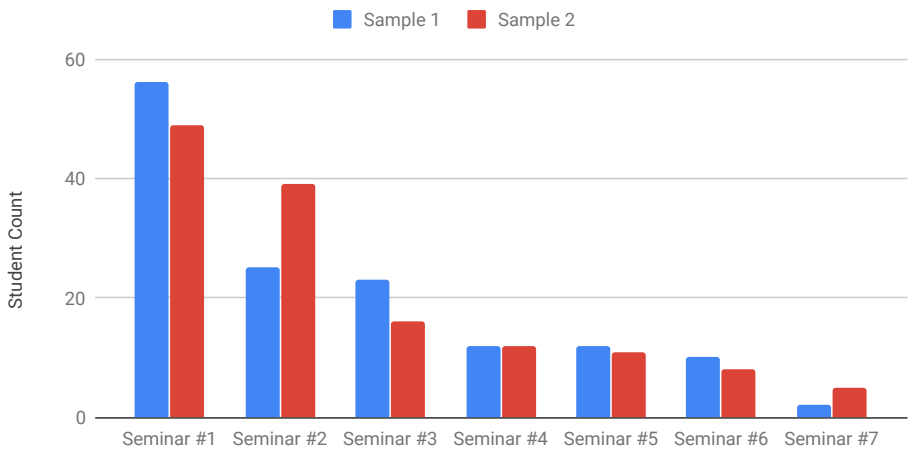


Figure 2: Two sample Zipfian preference distribution for first-ranked seminars

it is not possible to check if a given matching is popular without comparing it to all other possible 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 5.1.

5.3. Experimental Results

The following subsections provide a detailed summary of the results, grouped by the type of dataset being used. Afterwards, the research questions from Section 5.1 are answered and a summary of results is drawn.

5.3.1. PrefLib Datasets

The two PrefLib datasets contain strict, complete preferences for a small number of students and seminars.

PrefLib1 Dataset Figure 3 shows the rank distribution for the algorithms that find a matching.

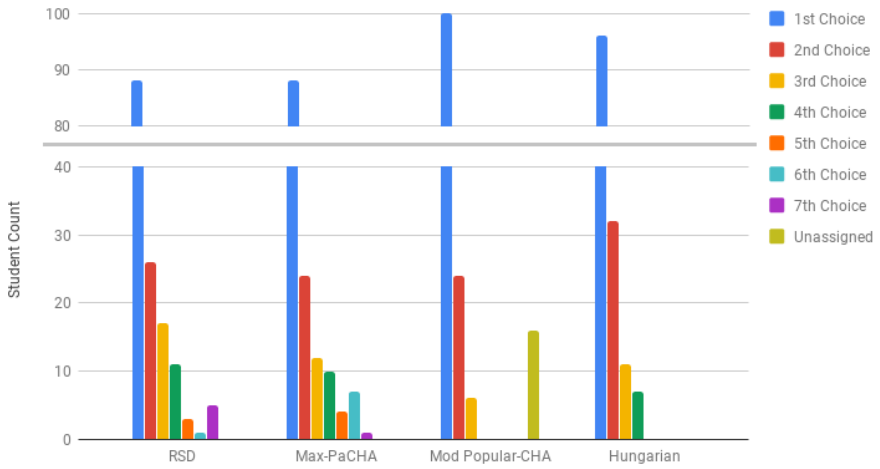


Figure 3: Rank distribution for PrefLib1 Dataset. Note: Popular-CHA failed.

The result makes clear that Mod-Popular-CHA 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 metrics, even though both their matchings are of 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.

Table 3 shows the results in detail. Unfortunately, the Popular-CHA algorithm does not find a matching, but we can also see that the Mod-Popular-CHA algorithm finds 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.When not taking the unassigned students into account, the Mod-Popular-CHA algorithm performs even better on the rank metrics; however, when taking the unassigned students into account, it performs worst among all algorithms.

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 3: Summary of the results for PrefLib1

PrefLib2 Dataset Figure 4 shows the preference distribution of all five algorithms using the PrefLib2 dataset. 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; however the differences in profile among the algorithms are not that significant compared to the PrefLib1 dataset.

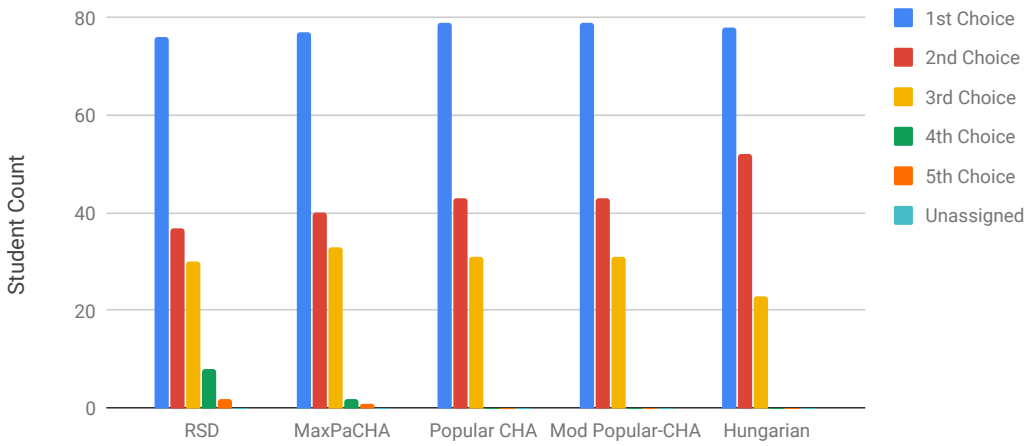


Figure 4: Rank distribution for PrefLib2 Dataset.

Table 4 shows the results for the PrefLib2 dataset in more detail. 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 also 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 Max-PaCHA 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 should be noted that the matching produced by RSD gets close in terms of rank to the

Metric	RSD	Max-PaCHA	Hungarian	Popular-CHA	Mod-Popular-CHA
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 4: Summary of the results for PrefLib2

matching produced by the Hungarian algorithm, even though it is about 50 times faster.

5.3.2. 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.

Small Zipfian Datasets We begin by looking at a similarly sized dataset as the Preflib datasets, and then begin deviating the student & seminar count. We begin by looking at a similarly sized dataset as the Preflib datasets, and then begin deviating the student & seminar count. Figure 5 shows the average ranks for 50 test runs using the Zipfian Dataset with 10 seminars and about 200 students.

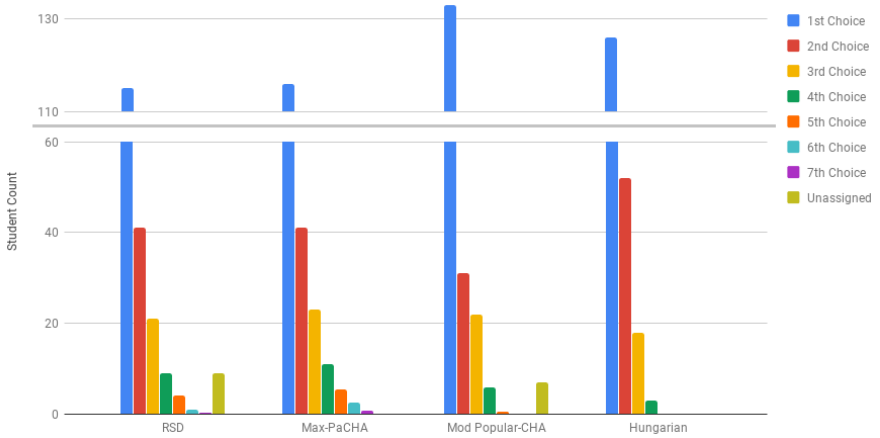


Figure 5: Rank distribution for small Zipfian dataset.

We can clearly see that the Hungarian algorithm performs best in terms of worst-rank and profile in general; however, Mod-Popular-CHA assigns more students to their first choice at the cost of leaving on average 7 students unassigned.

Table 5 shows the average results in more detail and shows that, out of the 50 instances, Popular-CHA only finds a matching for four instances. When it finds a matching, it does

Metric	RSD	Max-PaCHA	Hungarian	Popular-CHA	Mod-Popular-CHA
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 5: Average results for small Zipfian dataset with 50 runs

so quite fast and the matching is, on average, better rank-wise than all matchings produced by the other algorithms, except for the ones produced by the Hungarian algorithm. In terms of popularity, Mod-Popular-CHA actually outperforms 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 produces more popular matchings than most other algorithms, and, for some instances, even more popular ones than the Mod-Popular-CHA algorithm.

Large Zipfian Datasets Using a large dataset with a Zipfian distribution yields similar results. Figure 6 shows the rank distributions for 10 test runs with datasets that contain about 2500 students and 35 seminars. We can see that Mod Popular-CHA is able to outperform the other algorithms in terms of popularity by maximizing the number of students being assigned to their first choice. However, the Hungarian algorithm produces more balanced matchings, where most students are assigned to their top three choices, while also leaving no student unassigned.

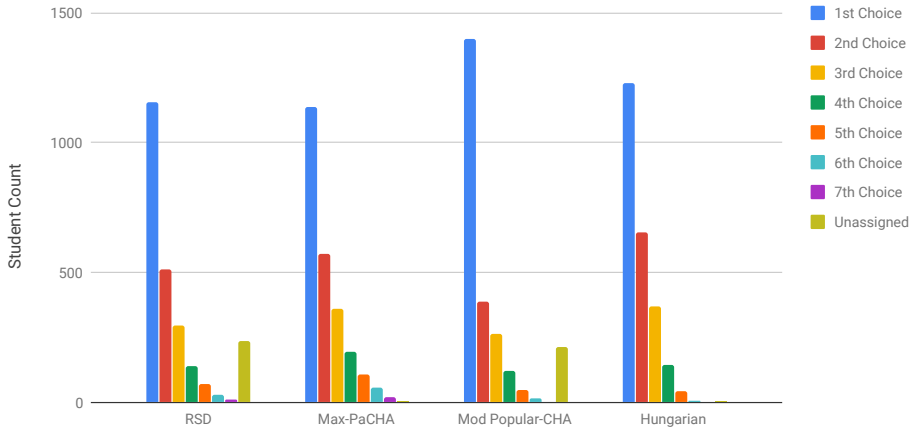


Figure 6: Rank distribution for large Zipfian dataset.

Table 6 summarizes the results in detail and shows that Popular-CHA is not very successful and finds no matching in any of the test runs. However, Mod-Popular-CHA always produces more popular matchings than all of the other algorithms, even though

Metric	RSD	Max-PaCHA	Hungarian	Popular-CHA	Mod-Popular-CHA
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 6: Average results for large Zipfian dataset (2500 Students) with 10 runs

8.6% of the students were left unassigned on average. When taking the unassigned students out of the rank metrics, Mod-Popular-CHA produces matchings with the best average rank and standard deviation, while having the second lowest runtime. In contrast to that, the Hungarian algorithm always assigns all students, even though it takes, on average, 65 seconds to do so. This confirms the theoretical observations about runtime and shows that using the Hungarian algorithm is potentially less desirable for large instances.

5.3.3. Large Uniform Datasets

Datasets with Incomplete Preferences This large uniform dataset contains 50 seminars and 5000 students, who each randomly pick 10 seminars with equal probability.

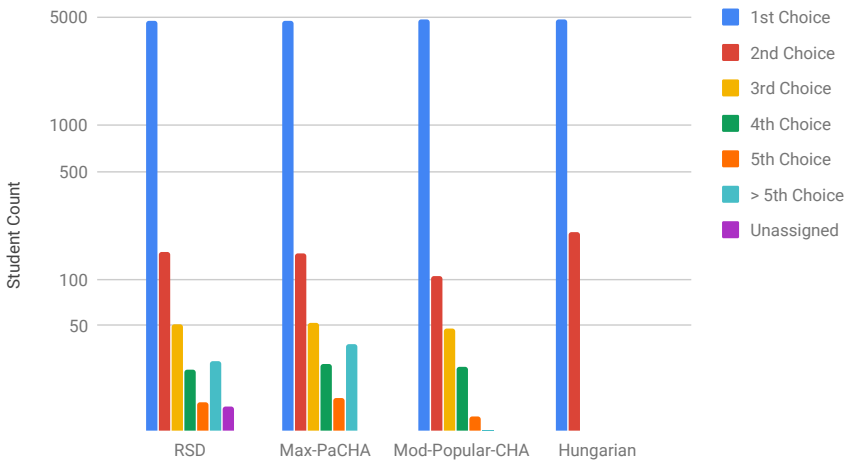


Figure 7: Rank distribution on log-scale for uniform dataset with incomplete preferences, with ranks greater than 5 being grouped.

Figure 7 shows the average ranks when executing each algorithm 10 times using incomplete preference lists. We can see that all algorithms assign a vast majority (94%) of students to their first choice; however, all algorithms, but the Hungarian algorithm, also assign students to their lower preferences and therefore have a worse profile.

Metric	RSD	Max-PaCHA	Hungarian	Popular-CHA	Mod-Popular-CHA
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 7: Average results for large uniform dataset with incomplete preferences.

In Table 7 we can see that Popular-CHA only finds a matching for one out of the 10 instances, but when it does, it performs quite well in terms of rank. However, Mod-Popular-CHA is on average able to find better matchings in less time than Popular-CHA. What also stands out is that Mod-Popular-CHA and the Hungarian algorithm are tied for popularity for every instance, but runtime-wise, Mod-Popular-CHA performs far better than the Hungarian algorithm.

Datasets with Complete Preferences When using the same size dataset as before, but with complete preference lists, the results show some interesting insights.

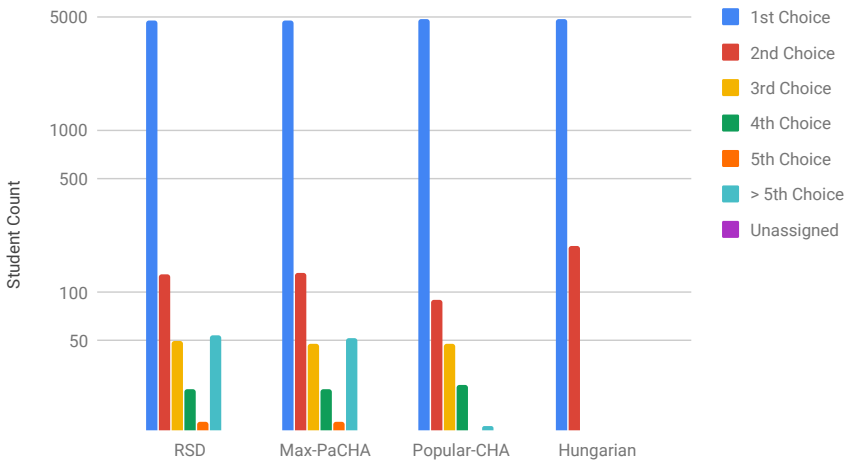


Figure 8: Rank distribution on log-scale for uniform dataset with complete preferences, with ranks greater than 5 being grouped.

Figure 8 shows the average ranks when executing each algorithm 10 times using complete preference lists. We see the same pattern as with incomplete preferences, namely that the Hungarian algorithm's matchings have the best profile and a low worst rank of two. Compared to that RSD and Max-PaCHA assign around 50 students to their 5th choice or worse.

Besides that, Table 8 shows that Popular-CHA finds 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-CHA
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 8: Average results for large uniform dataset with complete preferences.

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

5.4. Conclusions of Experiments

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 3 and 4), the runtime differences for the different mechanisms are insignificant. For large instances, the modified Popular-CHA algorithm performs well and it could still be optimized even further for matching more students. The Mod-Popular-CHA algorithm as it was in described in Section 4.2.6, does not try to assign students that were left unmatched when finding a maximum-cardinality matching. This could easily be improved by either using a simple mechanism like RSD or even using the Hungarian algorithm on the set of unassigned students to increase the cardinality of the matching. Neither modification guarantees that the final matching would be popular or of maximum cardinality, but it could improve the result. 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.

In terms of average rank, Figure 9 summarizes the average ranks for each dataset and algorithm, except for Popular-CHA. We can see that for some instances the differences in

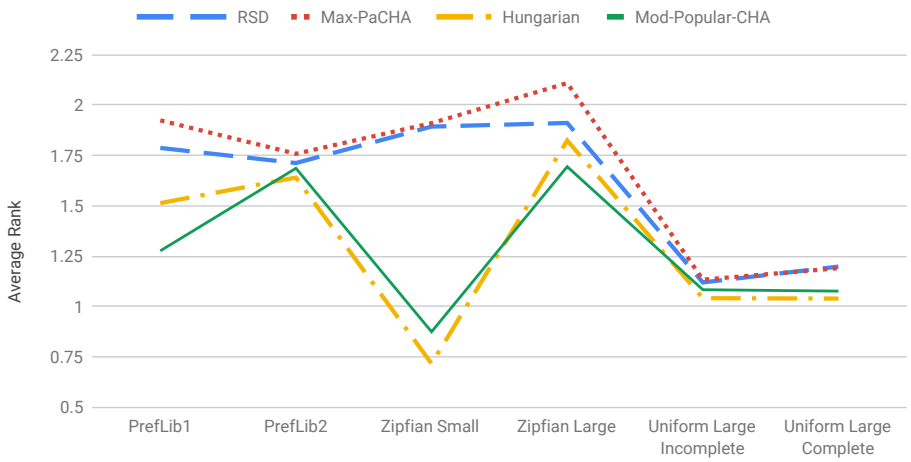


Figure 9: Average Rank comparison between algorithms and datasets.

rank are not very significant, but for other instances there is a bigger difference. Generally, the Hungarian algorithm unsurprisngly performs the best; however, for two types of datasets Mod-Popular-CHA beats the Hungarian algorithm in the average rank metric at the cost of leaving students unassigned. At the same time, the Hungarian algorithm also produces matchings with the lowest rank standard deviation, i.e. lowest fluctuation of rank, which makes this mechanism fair towards all students.

5.4.1. Answers to the Research Questions

The experiments reveal some interesting insights that should help answering the research questions from Section 5.1.

Effect of Different Preference Distributions: Unsurprisingly, the algorithms perform better on the rank metrics, when the preferences are distributed uniformly - in fact, the Hungarian algorithm is able to assign on average more than 94% (Table 8) of the students to their first choice when the preference distribution was uniform.

Failure Rate of Popular-CHA: The experiments show 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 6, 5), Popular-CHA only finds a matching for 4/50 instances. Even for uniform distributions and incomplete preference lists (Table 7), the algorithm only finds a matching for 1/10 instances. Making the preference lists complete makes it easier for the algorithm to find matchings as shown in Table 8.

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 typically also

producing more or equally popular matchings as the other algorithms. For many instances, Mod-Popular-CHA ties the Hungarian algorithm for popularity and was able to produce good matchings in terms of the rank metrics in a fraction of the time.

Cost of Giving up on Strategy-proofness: Surprisingly, RSD produces, on average, better matchings on most metrics than Max-PaCHA, while also being by far the fastest algorithm. However, RSD usually produces the highest percentage of unassigned students. Additionally, the modified Popular-CHA algorithm typically outperforms RSD on most metrics, while also having a relatively low runtime. In the real-world datasets (PrefLib), RSD leaves no students unassigned and got close to the Hungarian algorithm in terms of rank, but also produced 7 as the worst rank, compared to 4 for the Hungarian algorithm.

Popularity as a Meaningful Metric: For many instances, we cannot find the one popular matching, as Popular-CHA simply fails - but when it finds a matching, the worst rank, average rank and rank standard deviation is higher than what the Hungarian algorithm produes. Another interesting insight is that matchings produced by Mod-Popular-CHA usually tie in popularity with those produced by the Hungarian algorithm, even though Mod-Popular-CHA often left a few students unassigned. For instance, in Figure 6, Mod-Popular-CHA leaves, on average, 8.6% of the students unassigned, but the matchings are still classified as more popular than, 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 no seminar at all. This makes it questionable as to whether or not popularity is a meaningful optimality criteria for the student and seminar use case, 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 satisfied students.

The Effect of Short Preference Lists: The experiments show 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.

6. 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.

6.1. 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. In particular, 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 (Section 3.5.2) and our experiment show 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 as to whether or not Popularity should be considered when designing such a matching system. The results from Table 4 and 8 show 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, we 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 is that Max-PaCHA usually performs worse than all other algorithms on the rank metrics, including RSD. This can easily be 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.

6.2. Potential Algorithmic Improvements

In Section 4.2.6, we 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 presented the approach of reducing the matching problem to the assignment problem or by using flow-networks to find a rank-maximal matching. 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 7 and 8), 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. [3]. 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 3), the runtime of the Hungarian algorithm was still below 20ms.

6.3. 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 all other 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 [3] could be used for finding an exact result or, for very large instances, the Mod-Popular-CHA algorithm can be used. To improve the 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.

7. Conclusion

The goal of this thesis has been to present and analyze different matching mechanisms and their properties for one-to-many matching scenarios with one-sided preferences. A variety of optimality criteria exist, but in the end, it is evident that, none of the algorithms presented produce matchings that fulfill all of the optimality criteria.

The experiments conducted 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; but, 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. Still, 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 non-existent match. Therefore, we can recommend the Hungarian or Rank-Max algorithm for the use case of student-seminar matching. To improve the resulting matchings and make strategic manipulation harder, 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.

Appendices

A. 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.

A.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. A similar problem exists in the United States, where graduating medical students need to be matched to residency positions in hospitals.

Since we are considering 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 do not rely on the graph representation of the problem.

A.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.

A.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 [5]. In the context of student seminar matchings, it would make sense to use the resident/student optimal algorithm, which goes as follows [5]:

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 [5].

A.2. Many-to-Many Matchings

A natural extension of the many-to-one matching setting is the many-to-many matching problem. 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:

A.2.1. One-sided Preferences

Many-to-many matching scenarios with one-sided preferences are also commonly referred to as the *Course allocation* problem, which is a combinatorial assignment problem with the goal of assigning students to courses, based on the students' preferences [31]. Compared

to the student and seminar scenario we now allow students to get matched to more than one course.

A common mechanism for solving course allocation problems is using an *Auction model* or *Bidding points mechanism* in which students are given a set amount of artificial currency, which they then use to bid on seats in classes. After all bids are submitted, the system assigns the seats to the highest bidders. The bids can be used to infer students' preferences; however, the true preferences may differ significantly, which in turn could cause conflicts [32].

Another commonly used mechanism is the *Draft*, in which students are asked to pick a course with remaining seats based on a draft order. Intuitively, this mechanism is an extension of RSD (Section 3.1), where the whole process is repeated until no student makes a pick anymore. Such draft mechanisms are commonly used in course allocation settings; one example being Harvard Business School [31]. However, it has been shown that students have successfully manipulated the draft due to the fact that this mechanism is not strategy-proof.

A mechanism that is more robust towards strategic manipulation is a *proxy bidding mechanism* presented by Kominers et al. [31]. The mechanism uses proxies that act on behalf of the students by using their true preferences in the draft.

A.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[5] 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 [5].

A.3. Online Variants

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 [33].

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.

A.3.1. Online Maximum 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 [33]. 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: $C.R. = \frac{ALG(I)}{OPT(I)}$ [33].

A simple algorithm for this online problem is a greedy algorithm, which 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 $\frac{1}{2}$ [33]. An optimal yet simple algorithm was introduced by Karp et al. [34], which achieves a competitive ratio of $1 - \frac{1}{e} \simeq 0.63$. The algorithm, called Ranking, begins by permuting 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 ∈ V do
    if v has no available neighbors then
        continue
    end if
    Match v to the neighbor u ∈ U with the smallest value π(u)
end for
```

References

[1] 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.

[2] 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.

[3] 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.

[4] D. F. Manlove, *Algorithmics of Matching Under Preferences*, vol. 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013.

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

[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] A. Roth, "The theory and practice of market design," Nobel Prize in Economics documents 2012-5, Nobel Prize Committee, 2012.

[8] E. Ronn, "Np-complete stable matching problems," *Journal of Algorithms*, vol. 11, no. 2, pp. 285 – 304, 1990.

[9] 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.

[10] 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.

[11] D. B. West, *Introduction to Graph Theory (2nd Edition)*. Pearson, 2000.

[12] 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.

[13] 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.

[14] 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.

[15] D. Manlove and C. Sng, "Popular matchings in the capacitated house allocation problem," September 2006.

[16] 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.

[17] 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.

[18] P. Gärdenfors, "Match making: Assignments based on bilateral preferences," *Behavioral Science*, vol. 20, no. 3, pp. 166–173, 1975.

[19] A. E. Roth, "Incentive compatibility in a market with indivisible goods," *Economics Letters*, vol. 9, no. 2, pp. 127 – 132, 1982.

[20] M. Manea, "Serial dictatorship and pareto optimality," *Games and Economic Behavior*, vol. 61, no. 2, pp. 316 – 330, 2007.

[21] J. Kun, "Serial dictatorships and house allocation." `https://jeremykun.com/2015/10/26/serial-dictatorships-and-house-allocation/`, Nov 2015.

[22] L. Shapley and H. Scarf, "On cores and indivisibility," *Journal of Mathematical Economics*, vol. 1, no. 1, pp. 23 – 37, 1974.

[23] 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.

[24] 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.

[25] J. E. Hopcroft and R. M. Karp, "A n5/2 algorithm for maximum matchings in bipartite graphs," in *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pp. 122–125, Oct 1971.

[26] 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.

[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," 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] S. D. Kominers, M. Ruberry, and J. Ullman, "Course allocation by proxy auction," in *Internet and Network Economics* (A. Saberi, ed.), (Berlin, Heidelberg), pp. 551–558, Springer Berlin Heidelberg, 2010.

[32] T. Sönmez and M. U. Ünver, "Course bidding at business schools," *International Economic Review*, vol. 51, no. 1, pp. 99–123, 2010.

[33] A. Mehta, "Online matching and ad allocation," *Found. Trends Theor. Comput. Sci.*, vol. 8, pp. 265–368, Oct. 2013.

[34] 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 August 11, 2019 .....