Student Course Allocation with Constraints

Akshay Utture[2] *, Vedant Somani[1], Prem Krishnaa[1], and Meghana Nasre[1]

[1] Indian Institute of Technology Madras
{vedant, jpk, meghana}@cse.iitm.ac.in
[2] University of California, Los Angeles
akshayutture@ucla.edu

Abstract. Real-world matching scenarios like the matching of students to courses in a university setting involves complex constraints like credit limits, time-slot constraints for courses, basket constraints (say, at most one humanities elective for a student), in addition to the preferences of students over courses and vice versa, and class capacities. We model this problem as a many-to-many bipartite matching problem where both students and courses specify preferences over each other and students have a set of downward feasible constraints. We propose an Iterative Algorithm Framework which uses a many-to-one matching algorithm and outputs a many-to-many matching that satisfies all the constraints. We prove that the output of such an algorithm is Pareto-optimal from the student-side if the many-to-one algorithm used is Pareto-optimal from the student side. For a given matching, we propose a new metric called the Mean Effective Average Rank (MEAR) for a student, which quantifies the goodness of allotted courses for that student with respect to the matching. We empirically evaluate two many-to-one matching algorithms with synthetic data modeled on real-world instances and present the evaluation of these two algorithms on different metrics including MEAR scores, matching size and number of unstable pairs.

1 Introduction

Consider an academic institution where each semester students choose elective courses to credit in order to meet the credit requirements. Each student has a fixed number of credits that need to be satisfied by crediting electives. Each course has a credit associated with it and a capacity denoting the maximum number of students it can accommodate. Both students and courses have a strict preference ordering over a subset of elements from the other set. In addition, it is common to have constraints like a student wants to be allotted at most one course from a basket of courses, a student does not want to bet allotted to time-conflicting courses. The goal is to compute an *optimal* assignment of elective courses to students satisfying the curricular restrictions while respecting the course capacities and satisfying the credit requirements.

* Part of this work was done when the author was a Dual Degree student at the Indian Institute of Technology Madras.

We model the course allocation problem as a many-to-many bipartite matching problem with two sided preferences where one side of the bipartition allows *downward feasible constraints* (as described by [8]) and the other side of the bipartition has capacity constraints. Formally, we have a bipartite graph $G = (S \cup C, E)$ where S represents the set of students, C represents the set of courses and an edge $(s, c) \in E$ denotes that the course c can be assigned to the student s. Each course $c \in C$ has an integer valued capacity represented by $q(c)$, which represents the maximum number of students that can be assigned to c. Each vertex $x \in S \cup C$ ranks its neighbours in G in a strict order and this ordering is called the preference list of the vertex x. Each student s defines a set of *downward feasible constraints* $X(s)$ over the neighbours in G. A set of downward feasible constraints is one in which if $C \subset C'$, and C' is feasible, then C must be feasible. In other words, if a set of courses is feasible for a student, any subset of those courses should also be feasible. There are many types of constraints expressible as downward feasible constraints.

1. **Student Capacity:** A student may have an upper limit on the number of courses to take in a semester.
2. **Student Credits:** A student may specify the maximum number of credits allowed for him or her. This assumes that each course has a variable number of credits.
3. **Time slots:** Each course runs in a particular time slot, and courses with overlapping time slots cannot be assigned to a student.
4. **Curricular constraints:** A student may specify an upper limit on a subset of courses that are of interest to him. This upper limit denotes the maximum number of courses that can be assigned to him from the subset. For instance, in a semester, a student may want to be assigned at most two Humanities electives amongst the multiple Humanities electives in his preference list.

There exist natural examples of constraints like minimum number of students required in a course (for the course to be operational) or pre-requisites on courses which *cannot* be expressed as downward feasible constraints.

Fig. 1 shows an example instance of the student course allocation problem with downward feasible constraints. Here there are three students and three courses and the preference lists of the participants can be read from the figure. We assume all courses have uniform credits and therefore students specify a maximum capacity on the number of desired courses. Each student has a capacity of two. Each course can accommodate at most two students. Finally, for each student, there is a constraint that the student can be allotted at most one of the two courses $\{c_2, c_3\}$. Such a constraint could be imposed if the courses c_2 and c_3 run in overlapping time-slots. The student s_3 has a constraint that he can be allotted at most one of $\{c_1, c_3\}$. All these constraints (tabulated in Fig. 1), including the capacity constraints, are downward feasible constraints.

A matching M in a student course allocation setting is a subset of the edges of the underlying bipartite graph; matching M is said to be feasible if M respects all downward feasible constraints specified. As seen in Fig. 1 an instance may admit

multiple feasible matchings. This motivates the need for a notion of optimality in order to select a matching from the set of all feasible matchings.

Student	Capacity	Student Preference List	Course	Capacity	Course Preference List
s_1	2	c_1, c_2, c_3	c_1	2	s_1, s_2, s_3
s_2	2	c_1, c_2, c_3	c_2	2	s_1, s_2, s_3
s_3	2	c_1, c_2, c_3	c_3	2	s_1, s_2, s_3
Constraints: Each student can take only 1 of $\{c_2, c_3\}$. s_3 can take only 1 of $\{c_1, c_3\}$					

Fig. 1: Each student s_i, for $i = 1, 2, 3$ prefers c_1 followed by c_2 followed by c_3. Each course has the same preference order. Matchings $M_1 = \{(s_1, c_1), (s_2, c_1), (s_1, c_2), (s_2, c_2), (s_3, c_3)\}$ and $M_2 = \{(s_1, c_1), (s_2, c_1), (s_3, c_2), (s_1, c_2), (s_2, c_3)\}$ are both feasible in this instance.

To the best of our knowledge, this problem setting has not be considered in the literature; settings with a subset of these constraints have been studied before. Cechlarova et al. [8] study a similar problem for one-sided preferences where they consider computing pareto-optimal matchings (defined below). In the two sided preference list model, *laminar classifications* [19,15] have been studied which can be defined as follows. Each vertex $x \in S \cup C$ specifies a family of sets called classes over the neighbours of x in G. A class J_x^i is allowed to have an upper quota $q^+(J_x^i)$ and a lower quota $q^-(J_x^i)$ which specifies the maximum and minimum number of vertices from J_x^i that need to be matched to x in any feasible matching. The family of classes for a vertex x is laminar if for any two classes of a vertex either one is contained inside the other or they are disjoint. Note that if there are no lower quotas associated with classes, then a laminar classification is a special case of downward feasibility. In fact, downward feasibility allows for non-laminar classes and course credits but as noted earlier, cannot capture lower quotas. Huang [19] and later Fleiner and Kamiyama [15] extended the notion of *stability* for laminar classifications. Their results show that in a many-to-many bipartite matching problem with two sided preferences and two sided laminar classifications with upper and lower quotas, it is possible to decide in polynomial time whether there exists a stable matching. In contrast, if classifications are non-laminar, it is NP-Complete to decide whether a stable matching exists [19]. Below we discuss two well-studied notions of optimality.

Stability: In the presence of two sided preferences, stability is a de-facto notion of optimality. We recall the definition of stability as in [15] which is applicable to our setting as well. Let M be a feasible matching in an instance G of the student course allocation problem with downward feasible constraints. For a vertex $u \in S \cup C$ let $M(u)$ denote the set of partners assigned to u in M. An edge $(s, c) \in E \setminus M$ is blocking w.r.t. M if both the conditions below hold:

- Either $M(s) \cup \{c\}$ is feasible for s or there exists a $c' \in M(s)$ such that s prefers c over c' and $(M(s) \setminus \{c'\}) \cup \{c\}$ is feasible for s, and
- Either $M(c) \cup \{s\}$ is feasible for c or there exists an $s' \in M(c)$ such that c prefers s over s' and $(M(c) \setminus \{s'\}) \cup \{s\}$ is feasible for c.

A matching M is stable if no pair $(s, c) \in E \setminus M$ blocks M. It is easy to verify that the matching M_1 in Example 1 is stable whereas the matching M_2 is unstable since the pair (s_2, c_2) blocks M_2.

Student Pareto-optimality: We now recall the definition of pareto-optimality from [8] which we consider for the student side. A student s prefers a set $M(s)$ over another set $M'(s)$ if $M(s)$ is lexicographically better than $M'(s)$; we denote this as $M(s) >_s M'(s)$. A matching M dominates another matching M' if there exists at least one student s such that $M(s) >_s M'(s)$ and for all students $s' \in S \setminus s$, we have $M(s') \geq'_s M'(s')$. A matching M is pareto-optimal if there is no matching M' that dominates M. Since we consider the domination only from the student side, we call this student pareto-optimality. However, since there is no ambiguity, we will simply denote it as pareto-optimal. For the example in Fig. 1, both M_1 and M_2 are pareto-optimal.

We now contrast these two notions via a simple example as shown in Fig. 2. There is only one student and three course and the credits of the student and courses as well as preferences of the student can be read from the figure. There are two feasible matchings M_s and M_p both of which are stable yet only M_p is pareto-optimal. Note that M_p matches s to its top choice and satisfies the credit requirements. The example illustrates that in the presence of credits, stability may not be the most appealing notion of optimality. However, the only Pareto-optimal matching is M_p is more suitable in this scenario.

Student Preference List (s)	c_1, c_2, c_3
Max Credit Limit (s)	2
Credits(c_1) = 2; Credits(c_2) = Credits(c_3) = 1	

Fig. 2: Stability versus pareto-optimality. $M_s = \{(s, c_2), (s, c_3)\}$, $M_p = \{(s, c_1)\}$.

Our Contributions: In this work, we propose an Iterative Algorithm Framework for the student course allocation problem under downward feasible constraints. Our framework uses a many-to-one bipartite matching algorithm as a subroutine to output a many-to-many matching. We show the following results using the Iterative Algorithm framework.

- We provide an efficient algorithm for the many-to-many matching problem with two sided preferences, downward feasible constraints on one side, and capacity constraints on the other. Our framework can be easily extended to solve the many-to-many matching problem with downward feasible constraints on both sides. (Section 2)
- We show that if the many-to-one matching algorithm used in the framework is student-pareto-optimal then our output for the many-to-many matching problem is also student pareto-optimal. (Section 3)
- We introduce a new evaluation metric Mean Effective Average Rank (MEAR) score, a variation of the average rank metric, to quantify the quality of the matchings produced in a such a problem setting. (Section 4)
- We empirically evaluate the well-known Gale-Shapely [16] stable matching algorithm and First Preference Allotment algorithm in the Iterative Algorithm Framework, using synthetic data-sets modeled on real-world instances, on

different metrics including MEAR scores, matching size and number of unstable pairs. (Section 5)

Related Work: The closest to our work is the work by Cechlarova et al. [8] where they consider the many-to-many matchings under downward feasible constraints. They give a characterization and an algorithm for Pareto-optimal many-to-many matchings under one sided preferences Variants of the student course allocation problem deal with constraints like course prerequisites [7], lower-quotas for courses [9], or ties in the preference list [6]. Our problem setting deals with a similar setup but allows two-sided preferences.

The Gale and Shapley algorithm used by us is a classical algorithm to compute stable matching in the well-studied Hospital Residents (HR) problem [16] The HR problem is a special case of our problem with no downward feasible constraints, and a unit capacity for every student. Variants of the HR problem with constraints include allowing lower quotas [18], class constraints [19,15] and exchange-stability [21,10]. Other variations allow multiple partners [4,27], couples [25], colleague preferences [12] and ties [22,20,23,24]. The algorithms for the models considered above have strong mathematical guarantees, unlike our solution. However, the constraints studied do capture all the complexities of the course allocation problem considered in our setting.

An empirical analysis of matching algorithms has been done in the context of the National Residency Matching Program in the U.S. [2,14,33,13,31], and its counter-part in the U.K. [3,32,5]. Manlove et al. [29] introduce a constraint programming model to solve the Hospital Residents problem with couples and justify its quality by its empirically obtained low execution time and number of blocking pairs. Krishnapriya et al. [26] empirically study the quality of the matchings produced by a popular matching on metrics of practical importance, like size, number of blocking pairs and number of blocking residents. Can we say a little about some of these citations?[30,17,28,11] also conduct experiments to empirically evaluate the quality of their matching algorithms. In this paper, we use a similar experimental approach to demonstrate the practical importance of our proposed algorithm.

2 Algorithm Description

In this section, we present a framework called the Iterative Algorithm Framework (IAF for short) into which one can insert any many-to-one matching algorithm with two-sided preferences, and get a concrete algorithm which solves the many-to-many student-course matching problem with two-sided preferences and downward feasible constraints. As an example, we show two such many-to-one matching algorithms, namely the Gale-Shapley algorithm [16] and the First Preference Allotment. Finally, we suggest a simple extension to deal with downward feasible constraints on both sides of the bipartition.

2.1 Iterative Algorithm Framework

The Iterative Algorithm Framework (IAF) computes a many-to-many matching by a series of iterations of many-to-one matchings. We show in Section 3 that this framework gives a pareto-optimal matching from the side of the students if the many-to-one algorithm used in each iteration is pareto-optimal among the students allotted in that iteration.

Algorithm. 1 gives the pseudo-code for the IAF. As input, we require the set of students (S), the set of courses (C), the set of constraints $X(s)$ which determines what subset of courses from C is feasible for student s, the preference list of each student s ($P(s)$), the preference list of each course c ($P(c)$), and the capacity of each course c (q(c)). The framework outputs for each student s the set of allotted courses (M(s)).

The residual capacity of a course (denoted by $r(c)$) is the remaining capacity of a course at some point in the algorithm. It is initialized to its total capacity $q(c)$. Initialize the set of allotted courses $M(s)$ of each student s to be the empty set. We also maintain for every student s a reduced preference list ($R(s)$) which is intialized to the original preference list ($P(s)$). During the course of the algorithm the reduced preference list contains the set of courses in the preference list which have not yet been removed by the algorithm.

The framework is iterative and as long as some student has a non-empty reduced preference list, it invokes the `solve` function. The `solve` function is invoked with the capacities of the courses being set to the residual capacities. The `solve` function can be substituted with any many-to-one matching algorithm with two-sided preferences like the Gale-Shapley algorithm. An important characteristic of the IAF is that the allotments made at the end of the `solve` function are frozen, and matched student-course pair cannot get unmatched in a future iteration. This characteristic is essential to ensure the termination of the algorithm. After the execution of the `solve` function, the residual capacities of the courses are recalculated based on the allotment in `solve`.

In the remaining part of the loop the algorithm removes preferences which can no longer be matched given the current set of allotted courses. If the course c was allotted to student s in this iteration, we remove c from the preference list of s. Additionally, we remove courses with no residual capacity, because all allotments up to this point are frozen and none of the residual capacity is going to free up in a future iteration. In order to maintain feasibility in the future iterations, we also need to remove all courses on a student's preference list which are infeasible with the current partial allotment of courses. In this part, it is implicit that if a course c is removed from the preference list of student s, even s is removed from the preference list of c.

2.2 Gale-Shapley Algorithm in the Iterative Algorithm Framework

Here we use the well-known Gale and Shapley algorithm [16] in the IAF model. The Gale-Shapley algorithm in the context of students and courses with the students proposing works as follows. The algorithm works in a series of rounds until

Data: S = set of students, C = set of courses, $X(s)$ = constraints for student s, $P(s)$ = preference list for student s, $P(c)$ = preference list for each course c, $q(c)$ = capacity for each course c
Result: For each student s, $M(s)$ = set of allotted courses

Let $r(c) = q(c), \forall c \in C$;
Let $M(s) = \emptyset, \forall s \in S$;
Let $R(s) = P(s), \forall s \in S$;
Let $R(c) = P(c), \forall c \in C$;
while $\exists s \in S, R(s) \neq \emptyset$ **do**
    Invoke *solve*() using the residual capacities ;
    Freeze the every allotment (s, c) made in *solve*() ;
    `/* (s,c) cannot be unmatched in a future iteration */`
    Calculate the new $r(c), \forall c \in C$;
    foreach *student* $s \in S, R(s) \neq \emptyset$ **do**
        c = course allotted to s in *solve*() of current iteration ;
        Remove c from $R(s)$ and s from $R(c)$;
        From $R(s)$ remove every c' such that $r(c') = 0$;
        `/* Also remove corresponding` s `from` $R(c')$ `*/`
        From $R(s)$ remove every c' such that $M(s) \cup \{c'\}$ is not feasible according to $X(s)$;
        `/* Also remove corresponding` s `from` $R(c')$ `*/`
    end
end

Algorithm 1: Iterative Algorithm Framework

Iteration	Allotment in current Iteration	Partial allotment so far
1	$\{(s_1, c_1), (s_2, c_1), (s_3, c_2)\}$	$\{(s_1, c_1), (s_2, c_1), (s_3, c_2)\}$
2	$\{(s_1, c_2), (s_2, c_3)\}$	$\{(s_1, c_1), (s_2, c_1), (s_3, c_2), (s_1, c_2), (s_2, c_3)\}$

Table 1: Gale-Shapley+IAF algorithm for the example in Fig. 1

each student has exhausted his preference list. In each round, every unallotted student applies to his most preferred course that he has not applied to before, and if the course is either not full or prefers this student to its least preferred allotted student, the course will provisionally accept this student (and reject its least preferred allotted student in case it was full). Consider applying the Gale-Shapley algorithm [16] with the IAF to the example shown in Fig. 1. Table 1 gives the partial allotments made after each iteration. We note that the allotments made in the first iteration are frozen, that is they cannot be modified.

Note that in this example, the student-optimal stable marriage returned by the Gale Shapley algorithm is pareto-optimal among the students it allots. However, it is known that this is not true always. Hence Gale and Shapley in the IAF framework cannot guarantee a pareto-optimal matching. Fig. 3 shows an example where the student-optimal stable matching is $\{(s_1, c_2), (s_3, c_1), (s_4, c_1)\}$. This is not student pareto-optimal because (s_1, s_3) can exchange their partners and both be better off. We discuss why we might still use the Gale-Shapley algorithm with the IAF in Section 5.

Student	Student Preference List	Course	Course Capacity	Course Preference List
s_1	c_1, c_2	c_1	2	s_4, s_3, s_1
s_2	c_2	c_2	1	s_1, s_2, s_3
s_3	c_2, c_1			
s_4	c_1			

Fig. 3: Student-optimal stable matching is not pareto-optimal for students

2.3 First Preference Allotment in the Iterative Algorithm Framework

The First Preference Allotment is a simple many-to-one allotment, where each student is temporarily allotted to the first course on his or her preference list. If a course c, with capacity $q(c)$ is oversubscribed by k (i.e. $q(c) + k$ students are temporarily allotted to it), c rejects its k least preferred students. The algorithm terminates here, and the students who get rejected from courses which are oversubscribed do not apply again to other courses on their preference list. The First Preference Allotment algorithm is pareto-optimal among the set of students it matches (even though not pareto-optimal among the entire student set), and hence results in a pareto-optimal matching when inserted into the IAF.

Consider applying this algorithm to the example in Fig. 1. Table. 2 gives the partial allotments after each iteration of the First Preference Allotment algorithm when used with the `solve` method of the IAF. In the first iteration, all students apply to their top choice course (c_1 for all), and since c_1 only has a capacity of 2, it accepts $\{s_1, s_2\}$ and rejects its worst preferred student (s_3). The allotment so far is frozen, and in the next iteration, all students again apply to the next top choice course in their preference list (c_2 for all). Again, since c_2 only has a capacity of 2, it accepts s_1, s_2 and rejects its worst preferred student (s_3). In the final iteration, s_3 is the only student left with any capacity, and it applies to its next top choice course (c_3) and gets accepted.

Iteration	Allotment in current Iteration	Partial allotment so far
1	$\{(s_1, c_1), (s_2, c_1)\}$	$\{(s_1, c_1), (s_2, c_1)\}$
2	$\{(s_1, c_2), (s_2, c_2)\}$	$\{(s_1, c_1), (s_2, c_1), (s_1, c_2), (s_2, c_2)\}$
3	$\{(s_3, c_3)\}$	$\{(s_1, c_1), (s_2, c_1), (s_1, c_2), (s_2, c_2), (s_3, c_3)\}$

Table 2: First Preference Allotment+IAF algorithm for the example in Fig. 1

2.4 Extending the Iterative Algorithm Framework to additionally allow downward feasible constraints for courses

We can trivially extend the IAF to work in situations where courses express downward feasible constraints over students, instead of simple course capacities. The `solve` method now needs a one-to-one matching algorithm, and line 13 in Algorithm 1 needs to additionally remove preferences which are infeasible for the course constraints. The proof of student pareto-optimality is almost identical to the one shown in Section 3.2 for Algorithm 1, and is hence skipped for brevity. Also, since the problem is now symmetric from the student and course sides, we can similarly show course pareto-optimality if the `solve` method is pareto-optimal among the courses it allots.

3 Theoretical Guarantees

In this section, we first characterize a pareto-optimal matching, and then prove that if the many-to-one matching used in the `solve` method of the Iterative Algorithm Framework is pareto-optimal among the students it allots, then the final many-to-many matching given by the Iterative Algorithm Framework is also pareto-optimal. Note that the condition of requiring the `solve` method to be pareto-optimal among the students allotted is weaker than requiring it to be pareto-optimal among the entire set of students, and hence gives us more flexibility in picking a many-to-one matching.

3.1 Characterization of Pareto Optimality in the Many-to-Many setting

Cechlarova et. al [8] prove that a matching is pareto-optimal if and only if it is Maximal, Trade-In Free, and Coalition Free. These terms are defined as follows.

1. *Maximal:* M is maximal if no student-course pair (s, c) exists such that $r(c) > 0$, and the allotment of c to s is feasible with $M(s)$.
2. *Trade-In Free:* M is trade-in free if no student-course pair (s, c) exists such that $r(c) > 0$, and s can drop the courses $C' \in M(s)$ for c, because s prefers c over each of the courses in C'
3. *Coalition:* M is coalition free if there exists no set of students $S' \subset S$ who can exchange courses with one another, (and drop some lower preferred courses if needed to maintain feasibility) to get a new matching M', in which lexicographically $M'(s) > M(s), \forall s \in S'$

3.2 Proof of Pareto Optimality from the Student Side

To prove pareto-optimality it is sufficient to prove that the allotment is Maximal, Trade-In Free and Coalition Free. Theorems in this subsection use the assumption that the `solve` method is pareto-optimal among the subset of students it allotted a course.

Lemma 1. *At any point of the algorithm, for an unallotted student-course preference (s, c), if $c \notin R(s)$, then either $r(c) = 0$ or c is not feasible with $M(s)$.*

Proof: In Algorithm 1 the only place an unallotted student-course preference (s, c) gets removed from $R(s)$ is on Lines 12 and 13. The reasons for removal of a course c in these lines is either $r(c) = 0$ or c is not feasible with $M(s)$

Lemma 2. *The matching given by Algorithm 1 is Maximal*

Proof: Since Algorithm 1 terminates when $R(s) = \emptyset, \forall s \in S$, every unallotted preference (s, c) is not in $R(s)$. Then, according to Lemma 1, every unallotted preference (s, c) satisfied at least one of these - i) $r(c) = 0$ or ii) c is not feasible with $M(s)$. Hence the final output matching must be Maximal.

Lemma 3. *The matching given by Algorithm 1 is Trade-in Free.*

Proof: Let us assume that the allotment is not Trade-in free. Let s be a student who wants to trade-in the set of courses C for the course c, which s prefers over every course in C. By the definition of trading-in stated above, $r(c) > 0$ at the end of the algorithm. This implies that $r(c) > 0$ throughout the algorithm because $r(c)$ never increases after an iteration.

Consider the start of the first iteration where some $c_1 \in C'$ was allotted to s. By the definition of trading-in, c is feasible with $M(s) \setminus C'$, and hence should have been feasible at this point. The allotment of (s, c_1) instead of (s, c) contradicts the fact that the `solve` method is pareto-optimal. Hence the assumption that the allotment is not Trade-in free must be false.

Lemma 4. *The matching given by Algorithm 1 is Coalition Free.*

Proof: Let all arithmetic here be modulo n. Assume that $\mathrm{K} = \{(s_1, c_1), ..., (s_n, c_n)\}$ is a coalition in M. Let $\{(s_i, c_i), ..., (s_k, c_k)\}$ be the largest consecutive set of applicants from K who were allotted their coalition courses in the earliest iteration. $r(c_{k+1}) > 0$ throughout this iteration because $r(c)$ does not increase for all c, and (s_{k+1}, c_{k+1}) got matched in a later iteration. During the current iteration, s_k selected c_k instead of the more preferred c_{k+1}, even though $r(c_{k+1}) > 0$ throughout this iteration. This contradicts the condition that `solve` is pareto-optimal among the students it allots in an iteration. Hence the assumption that a coalition exists must be false.

3.3 Time Complexity

Let n be the total number of students and courses, and m be the sum of the sizes of the preference lists. The outer while loop runs for $\mathcal{O}(m)$ iterations in the worst case because at least 1 allotment happens in each iteration. The inner foreach loop runs through each student's preference list a constant number of times, and hence has complexity $\mathcal{O}(m)$ (assuming that the feasibility checking of the constraints for a preference list of length l is $\mathcal{O}(l)$). The complexity of the `solve` method depends on the algorithm inserted. Hence, the overall complexity of the Iterative Algorithm Framework is $\mathcal{O}(m) * (\mathcal{O}(m) + Complexity(\texttt{solve}))$. However, for some many-to-one algorithms like the Gale-Shapley, we can obtain a tighter bound on the number of iterations of the outer while loop. Each iteration of the Gale-Shapley algorithm runs in $\mathcal{O}(m)$ time and allots each student at least one course on his or her preference list (unless all the courses on the preference list are full), and hence the number of while loop iterations is $\mathcal{O}(n)$, bringing the total complexity to $\mathcal{O}(mn)$. The complexity of the `solve` method with the First Preference Allotment algorithm is $\mathcal{O}(m)$, since each student only applies to his top choice course and each course checks its preference list once, resulting in an overall complexity of $\mathcal{O}(m^2)$

4 Evaluation Metrics

In this section we look at some evaluation metrics used to quantify the quality of a matching produced in the presence of downward feasible constraints.

4.1 Mean Effective Average Rank

We define a new metric called Mean Effective Average Rank (MEAR for short) for quantifying the goodness of a matching in this problem setting. We discuss the MEAR in the context of students (shortened to MEAR-S), but it is applicable for courses as well (shortened to MEAR-C). MEAR is a mean of the Effective Average Rank (EAR) over all the students. EAR is a variation of the average rank metric and is defined in Eq. 2. The definition uses the Reduced Rank (see Eq. 1) which is the actual rank in the preference list, minus the number of courses above this preference which were removed due to some infeasibility with a higher ranked course. The intuition behind using the Reduced Ranks in the definition of EAR is that if a student has expressed the constraint that he be allotted only one of the first ten courses on his preference list, and he gets allotted his first and eleventh choice, then the sum of ranks is (1+10=11) 11, but the sum of Reduced Ranks ($1 + 2 = 3$) is 3, which is a more accurate representation of the allotment.

$$ReducedRank_{M(s)}(c) = Rank(c) - |\{c' : c' \in T(s), c' >_s c\}| \tag{1}$$

$$EAR(s) = \frac{\sum_{c \in M(s)} ReducedRank_s(c)}{|M(s)|^2} \tag{2}$$

Here $M(s)$ is the set of courses allotted to s, $T(s)$ is the set of courses removed due to infeasibility with higher ranked courses in $M(s)$, and the notation $c' >_s c$ means that student s prefers course c' over c.

Average Rank uses $|M(s)|$ in the denominator, but Eq. 2 uses its square instead, for normalization. For a student is allotted her first 5 choices, the sum of allotted Reduced Ranks is $(1 + 2 + 3 + 4 + 5 = 15)$. Dividing by the number of courses, gives us $(15/5 = 3)$, whereas a student allotted only his second preference gets an average of 2. An Average Rank of 3 sounds like a poor outcome for the student, but the student got all his top 5 choices. Hence dividing by $|M(s)|^2$ gives us $(15/25 = 0.6)$ which represents the quality of the allotment more accurately.

Consider the example in Table. 3, to understand the EAR calculation for student s. The table lists the preferences, and the constraint is that only one of $\{c_1, c_2\}$ can be allotted to s. $M(s) = \{c_1, c_4\}$. The Reduced Rank of c_1 is 1, and since c_2 got removed because of the allotment to c_1, the reduced ranks of c_3, c_4 and c_5 are 2, 3 and 4 respectively. Finally, the EAR is the sum of Reduced Ranks $(1 + 3)$ divided by the square of the number of courses allotted (2^2).

Some properties of the MEAR-score are as follows. A lower EAR score implies a better allotment. The lowest EAR value (and MEAR value) possible is 0.5. If the top k courses are allotted, $EAR = (k * (k + 1))/2k^2$, which approaches 0.5 as k approaches infinity. EAR favours larger matches because of the square term

Student Preference List	c_1, c_2, c_3, c_4, c_5
Constraint	At most 1 out of $\{c_1, c_2\}$
Allotment for student	$\{c_1, c_4\}$
Reduced Ranks (in brackets)	$c_1(1), c_2(removed), c_3(2), c_4(3), c_5(4)$
Effective Average Rank (EAR)	$(1+3)/(2^2) = 1$

Table 3: Example for Effective Average Rank Calculation

in the denominator. MEAR favours fairer allotments because an average is taken over all students irrespective of the number of courses allotted to each student. This is exemplified in Example 1 (in Table. 1), where M_1 and M_2 have a similar total sum of ranks across all students, but M_2 gets a lower MEAR score (1.25 instead of 1.5).

4.2 Other Metrics

Even though MEAR gives a single number to assess the quality of a matching, there are other metrics of interest like the allotment size and number of unstable pairs. Another possible metric is the exchange blocking pairs (studied in a similar context by [21] and [10]), defined as the number of student pairs who can exchange one of their allotted courses and both be better off. By definition, a pareto-optimal matching will not have exchange blocking pairs, but if a non pareto-optimal many-to-one matching is used with the Iterative Algorithm Framework, this becomes an interesting metric. Exchange-blocking pairs and unstable-pairs are important metrics because they quantify the dissatisfaction among students. For example, two students forming an exchange-blocking pair will want to swap their courses because they can both be better off.

5 Experimental Results

In this section we empirically evaluate the quality of the matchings produced by the Gale-Shapley algorithm and First Preference Allotment algorithms in the `solve` method of the Iterative Algorithm Framework (shortened as GS+IAF and FP+IAF respectively) and compare these results against a Maximum Cardinality Matching (shortened as MCM), which serves as a baseline. Other baselines are not available since this problem setting has never been studied before. The Integer Linear Program used to compute the MCM is described in Appendix B. Using an input generator (a modified version of the one used by [26]), we first generate synthetic-data which models common downward feasible constraints in the student-course matching scenario, and then study the effect of varying parameters like the instance size, competition for courses and preference list lengths. The matching sizes are reported as a fraction of the MCM size and the unstable pairs are reported as a fraction of the number of unallotted pairs. All numbers reported in this section are averaged over 10 instances. Instructions on reproducing the experiments are detailed in Appendix C.

5.1 Input Data Generator

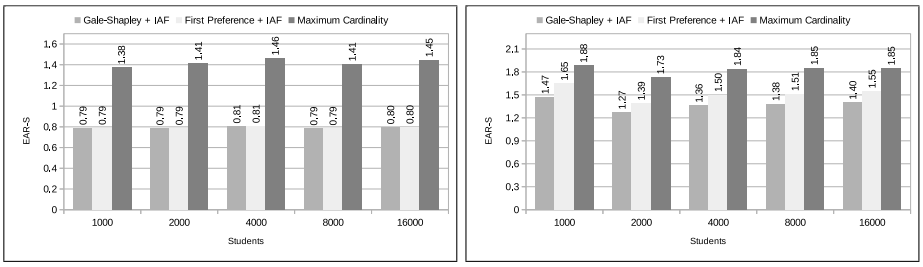
We generate two kinds of data sets: the *Shuffle Dataset* and *Master Dataset*. The generation of these datasets is identical, except that the Shuffle dataset represents one extreme of the possible skew in preference ordering where preference orders are fully random, and the Master dataset represents the other extreme where preference orders are identical. In the Master dataset, there exists a universal ordering among courses, and all student preference lists respect this relative ordering among the courses in their preference list. All course preference lists use a similar universal ordering of students.

The Shuffle Dataset is generated as follows. The number of students, courses and the range of preference list sizes (default is [3,12]) are taken as input. For each student, the set of acceptable courses is chosen according to a geometric distribution (with parameter 0.1) among the courses. The preference ordering is random. The credits of a course and the student credit limit follow geometric distributions (with parameter 0.1) among the values (40, 50, 60, 70) and (5, 10, 15, 20) respectively. The average capacity of a course is adjusted so that the number of seats offered by courses is 1.5× the total demand from students. A set of universal class constraints across all students mimics the type of constraint where no student can take time-overlapping courses. There are 20 such constraints, and in each of them, a student can pick at most k_1 (random integer in [1,3]) courses from a set of $(n_{courses}/20)$ courses. Individual class constraints (unique to each student) mimic the constraint of a student wanting at most p_1 courses from a set of p_2 courses (eg. a student wants at most 2 out of the 7 humanities courses on his/her preference list), where p_2 is a random integer from [2, preference list size] and p_1 is a random integer from [1,$p_2 - 1$]. The number of such constraints per student is a random integer from [0, preference list size]. The values used for the generator are chosen to mimic the typical values in a university setting.

5.2 Effect of varying instance size

Figure 4 shows the effect of varying the number of students and courses on MEAR-S for GS+IAF, FP+IAF and MCM. On the Shuffle Dataset, GS+IAF (mean=0.79) and FP+IAF (mean=0.79) perform similarly, and both clearly outperform the MCM (mean=1.42). On the Master Dataset, the GS+IAF (mean=1.38) performs slightly better FP+IAF (mean=1.52), which in turn performs better than the MCM (mean=1.83). Increasing the size of the matching does not affect this trend on either data set.

Tables 4 and 5 show the effect of varying instance size on other parameters for the two datasets. GS+IAF and FP+IAF give similar scores on all metrics. The GS+IAF, however, gives a few exchange-blocking pairs because the Gale-Shapley algorithm is not pareto-optimal among the students it allots. Both IAF algorithms outperform MCM in the Unstable Pairs (almost 0 for both IAF algorithms) and Exchange Blocking Pairs by a huge margin, but do slightly worse on MEAR-C and, as expected, the matching size. The MEAR-C is slightly higher because the GS+IAF and FP+IAF algorithms are inherently biased towards the student side.



(a) Shuffle Dataset (b) Master Dataset

Fig. 4: MEAR-S values for varying instance sizes (student-course ratio is 1:6)

Students	MEAR-C			Matching Size			Unstable-Pair Ratio			Ex. Blocking Pairs		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
1000	1.42	1.43	1.23	0.87	0.87	1.0	0.11	0.11	0.38	0.3	0	69.9
2000	1.39	1.40	1.23	0.87	0.87	1.0	0.10	0.10	0.39	0.6	0	70.9
4000	1.49	1.49	1.31	0.86	0.86	1.0	0.10	0.10	0.40	0.4	0	68.9
8000	1.39	1.40	1.21	0.87	0.87	1.0	0.10	0.10	0.38	0.1	0	50.8
16000	1.44	1.45	1.26	0.87	0.87	1.0	0.10	0.10	0.38	0	0	66.3

Table 4: Effect of varying instance size on other metrics for the Shuffle Dataset

It is impossible to get an Exchange-blocking pair for any algorithm used on the Master Dataset, and hence it is not shown. Any 2 courses common to 2 student preference lists will be listed in the same relative order, and exchanging them will leave exactly 1 student worse off.

5.3 Effect of Increasing Competition

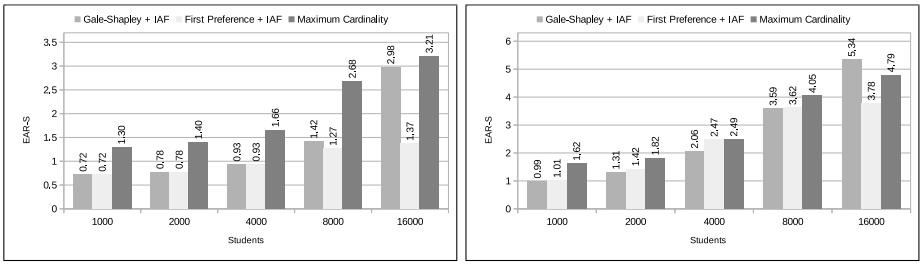
Figure 5 shows the effect of varying competition on the MEAR-S metric. The number of students is varied, while the number of courses is kept constant at 250. On the Shuffle Dataset, FP+IAF (mean=0.98) outperforms the GS+IAF (mean=1.17) on the larger input size, and both clearly outperform the Maximum Cardinality Matching (mean=1.92). On the Master Dataset, the FP+IAF (mean=2.17) performs slightly better than GS+IAF (mean=2.19), which in turn performs better than the MCM (mean=2.69). Tables 6 and 7 show the effect of varying student competition on other parameters for the two datasets. With increasing competition, MCM deteriorates significantly on EAR-C and Exchange-blocking pairs, and loses most of its advantage in matching size. FP+IAF on the other hand scales well on all measures (including EAR-S) and hence is the appropriate choice for a high-competition scenario.

5.4 Effect of Varying Preference List sizes

Figure 6 shows the effect of varying preference list sizes on the MEAR-S metric, on an input of 8000 students and 1333 courses. On the Shuffle Dataset, GS+IAF (mean=0.81) and FP+IAF (mean=0.81) clearly outperform the MCM

Students	MEAR-C			Matching Size			Unstable-Pair Ratio		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
1000	2.10	2.05	1.27	0.82	0.82	1.0	0.17	0.24	0.57
2000	1.99	1.97	1.17	0.84	0.84	1.0	0.14	0.20	0.56
4000	2.07	2.05	1.23	0.84	0.84	1.0	0.14	0.20	0.57
8000	2.04	2.04	1.24	0.84	0.84	1.0	0.15	0.21	0.57
16000	2.17	2.17	1.23	0.79	0.79	1.0	0.15	0.23	0.57

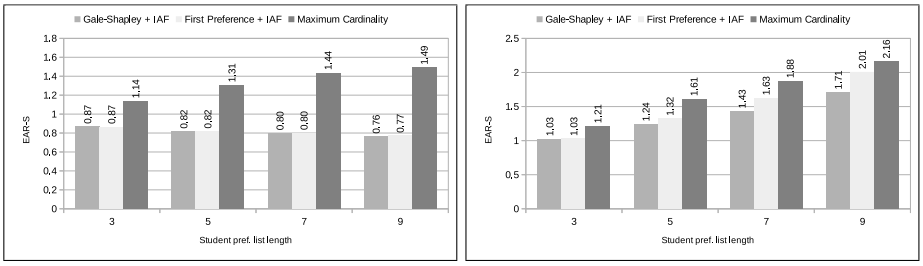
Table 5: Effect of varying instance size on other metrics for the Master Dataset



(a) Shuffle Dataset (b) Master Dataset

Fig. 5: MEAR-S values for varying competition for courses (by varying the number of students while keeping the number of courses constant at 250)

(mean=1.34). On the Master Dataset, the GS+IAF (mean=1.33) performs slightly better FP+IAF (mean=1.45), which in turn outperforms the MCM (mean=1.68). The observations here are qualitatively identical to those obtained by varying the instance size (Section 5.2).



(a) Shuffle Dataset (b) Master Dataset

Fig. 6: MEAR-S values for varying student preference list sizes

5.5 Discussion

From the preceding observations, we can conclude that GS+IAF and FP+IAF give similar results on the Shuffle Dataset, but GS+IAF does better on the Master Dataset. Even though it does not have a theoretical pareto-optimal guarantee, GS+IAF gives a negligible number of exchange-blocking pairs. It also consistently fares better on the Unstable Pairs metric because it avoids Unstable Pairs within

Students	MEAR-C			Matching Size			Unstable-Pair Ratio			Ex. Blocking Pairs		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
1000	1.40	1.40	1.27	0.90	0.90	1.0	0.02	0.02	0.34	0	0	66.5
2000	1.37	1.37	1.20	0.87	0.87	1.0	0.09	0.09	0.39	0	0	131.6
4000	1.43	1.46	1.41	0.85	0.84	1.0	0.24	0.24	0.53	6.7	0	321
8000	1.58	1.87	2.57	0.94	0.93	1.0	0.32	0.38	0.75	181.1	0	3598.2
16000	1.05	2.59	5.11	0.99	0.98	1.0	0.10	0.40	0.89	6287.4	0	5554.9

Table 6: Effect of varying competition on other metrics for the Shuffle Dataset

Students	MEAR-C			Matching Size			Unstable-Pair Ratio		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
1000	2.88	2.87	1.50	0.88	0.88	1.0	0.06	0.08	0.44
2000	2.39	2.44	1.28	0.84	0.85	1.0	0.13	0.19	0.56
4000	1.67	1.64	1.38	0.80	0.80	1.0	0.24	0.39	0.79
8000	1.90	2.00	2.56	0.91	0.91	1.0	0.25	0.54	0.90
16000	2.39	3.31	5.34	0.98	0.98	1.0	0.16	0.59	0.95

Table 7: Effect of varying competition on other metrics for the Master Dataset

an iteration. Hence, these observations empirically justify the use of GS+IAF, even though it may not have the same guarantees as FP+IAF. MCM on the other hand clearly outputs inferior matchings for the Shuffle Dataset on all parameters except size. Since all preference lists are similar in the Master Dataset, ignoring preferences does not hurt much, and MCM fares decently. Note that the MCM does not scale to larger instances because it is obtained by an ILP.

6 Discussion

In this paper, we presented the Iterative Algorithm Framework to solve the many-to-many matching problem with two-sided preferences and downward feasible constraints. We proved that if the many-to-one `solve` sub-routine is pareto-optimal among the students it allots, the framework outputs a pareto-optimal matching satisfying all the constraints. To quantify the quality of a matching, we introduce a new measure called the Mean Effective Average Rank (MEAR). We show that the Gale-Shapely and First Preference Allotment algorithms used with the Iterative Algorithm Framework, get significantly higher student MEAR-scores on two different synthetic datasets, and these results hold even when the instance size, competition or preference list lengths are changed. The Iterative Algorithm Framework algorithms also perform significantly better on other metrics like the number of unstable pairs and exchange-blocking pairs. Future work can include studying extensions which, in addition to the existing downward feasible constraints, allow ties or non-downward feasible constraints like lower quotas or course-prerequisites on one or both sides of the matching bipartition.

References

1. IBM CPLEX Optimizer. `https://www.ibm.com/analytics/cplex-optimizer`

Length	MEAR-C			Matching Size			Unstable-Pair Ratio			Ex. Blocking Pairs		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
3	0.97	0.98	0.89	0.93	0.93	1.00	0.08	0.08	0.34	0	0	2.5
5	1.14	1.15	1.01	0.90	0.90	1.00	0.11	0.11	0.43	0.2	0	20
7	1.34	1.34	1.17	0.87	0.87	1.00	0.12	0.12	0.43	0.5	0	53.6
9	1.44	1.46	1.23	0.86	0.84	1.00	0.11	0.12	0.41	0.1	0	81.3

Table 8: Effect of varying student preference list sizes on other metrics for the Shuffle Dataset

Length	MEAR-C			Matching Size			Unstable-Pair Ratio		
	GS+I	FP+I	MCM	GS+I	FP+I	MCM	GS+I	FP+I	MCM
3	1.23	1.23	0.94	0.92	0.92	1.00	0.04	0.06	0.42
5	1.55	1.58	1.02	0.86	0.86	1.00	0.12	0.16	0.63
7	1.87	1.92	1.16	0.83	0.82	1.00	0.16	0.22	0.63
9	2.40	2.42	1.39	0.81	0.80	1.00	0.18	0.26	0.60

Table 9: Effect of varying student preference list sizes on other metrics for the Master Dataset

2. National Residency Matching Program. `https://www.nrmp.org`
3. Scottish Foundation Association Scheme. `https://www.matching-in-practice.eu/the-scottish-foundation-allocation-scheme-sfas`
4. Bansal, V., Agrawal, A., Malhotra, V.S.: Polynomial time algorithm for an optimal stable assignment with multiple partners. Theor. Comput. Sci. **379**(3), 317–328 (Jul 2007)
5. Biró, P., Irving, R.W., Schlotter, I.: Stable matching with couples: An empirical study. J. Exp. Algorithmics **16**, 1.2:1.1–1.2:1.27 (May 2011)
6. Cechlárová, K., Eirinakis, P., Fleiner, T., Magos, D., Manlove, D., Mourtos, I., Ocelźáková, E., Rastegari, B.: Pareto optimal matchings in many-to-many markets with ties. Theor. Comp. Sys. **59**(4), 700–721 (Nov 2016)
7. Cechlárová, K., Klaus, B., Manlove, D.F.: Pareto optimal matchings of students to courses in the presence of prerequisites. Discrete Optimization **29**, 174–195 (2018)
8. Cechlárová, K., Eirinakis, P., Fleiner, T., Magos, D., Mourtos, I., Potpinkov, E.: Pareto optimality in many-to-many matching problems. Discrete Optimization **14**, 160 – 169 (2014)
9. Cechlárová, K., Fleiner, T.: Pareto optimal matchings with lower quotas. Mathematical Social Sciences **88**, 3 – 10 (2017)
10. Cechlárová, K., Manlove, D.F.: The exchange-stable marriage problem. Discrete Applied Mathematics **152**(1), 109 – 122 (2005)
11. Diebold, F., Aziz, H., Bichler, M., Matthes, F., Schneider, A.: Course allocation via stable matching. Business & Information Systems Engineering **6**(2), 97–110 (Apr 2014)
12. Dutta, B., Mass, J.: Stability of matchings when individuals have preferences over colleagues. Journal of Economic Theory **75**(2), 464 – 475 (1997)
13. E Roth, A., Peranson, E.: The effects of the change in the nrmp matching algorithm. national resident matching program. JAMA : the journal of the American Medical Association **278**, 729–32 (09 1997)
14. E. Roth, A., Peranson, E.: The redesign of the matching market for american physicians: Some engineering aspects of economic design. American Economic Review **89**, 748–780 (02 1999)

15. Fleiner, T., Kamiyama, N.: A matroid approach to stable matchings with lower quotas. In: Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 135–142. SODA '12 (2012)
16. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. The American Mathematical Monthly **69**(1), 9–15 (1962)
17. Giannakopoulos, I., Karras, P., Tsoumakos, D., Doka, K., Koziris, N.: An equitable solution to the stable marriage problem. In: Proceedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence ICTAI. pp. 989–996. ICTAI '15 (2015)
18. Hamada, K., Iwama, K., Miyazaki, S.: The hospitals/residents problem with lower quotas. Algorithmica **74**(1), 440–465 (Jan 2016)
19. Huang, C.C.: Classified stable matching. In: Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1235–1253. SODA '10 (2010)
20. Irving, R.W.: Stable marriage and indifference. Discrete Appl. Math. **48**(3), 261–272 (Feb 1994)
21. Irving, R.W.: Stable matching problems with exchange restrictions. Journal of Combinatorial Optimization **16**(4), 344–360 (Nov 2008)
22. Irving, R.W., Manlove, D.F., O'Malley, G.: Stable marriage with ties and bounded length preference lists. Journal of Discrete Algorithms **7**(2), 213 – 219 (2009)
23. Irving, R.W., Manlove, D.F., Scott, S.: The hospitals/residents problem with ties. In: Algorithm Theory - SWAT 2000. pp. 259–271 (2000)
24. Iwama, K., Manlove, D., Miyazaki, S., Morita, Y.: Stable marriage with incomplete lists and ties. In: Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings. pp. 443–452 (1999)
25. Klaus, B., Klijn, F.: Stable matchings and preferences of couples. Journal of Economic Theory **121**(1), 75 – 106 (2005)
26. M, K.A., Nasre, M., Nimbhorkar, P., Rawat, A.: How good are popular matchings? In: Proceedings of the 17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy. pp. 9:1–9:14 (2018)
27. Malhotra, V.S.: On the stability of multiple partner stable marriages with ties. In: Albers, S., Radzik, T. (eds.) Algorithms – ESA 2004. pp. 508–519 (2004)
28. Manlove, D., Milne, D., Olaosebikan, S.: An integer programming approach tothe student-project allocation problem with preferences over projects. In: Lee, J., Rinaldi, G., Mahjoub, A.R. (eds.) Combinatorial Optimization. pp. 313–325 (2018)
29. Manlove, D.F., McBride, I., Trimble, J.: "almost-stable" matchings in the hospitals / residents problem with couples. Constraints **22**(1), 50–72 (Jan 2017)
30. Olaosebikan, S., Manlove, D.: Super-stability in the student-project allocation problem with ties. In: Kim, D., Uma, R.N., Zelikovsky, A. (eds.) Combinatorial Optimization and Applications. pp. 357–371 (2018)
31. Peranson, E., R Randlett, R.: The nrmp matching algorithm revisited. Academic Medicine **70**, 477–84 (06 1995)
32. Roth, A.E.: A natural experiment in the organization of entry-level labor markets: regional markets for new physicians and surgeons in the united kingdom. The American economic review **81 3**, 415–40 (1991)
33. Williams, K.J.: A reexamination of the nrmp matching algorithm. national resident matching program. Academic medicine : journal of the Association of American Medical Colleges **70**, 470–6; discussion 490 (07 1995)

A Example to show that stable matching need not exist

In this appendix, we show an example where a stable matching does not exist because of the presence of general downward feasible constraints. Table 10 lists the preferences and capacities of the courses, and the preferences, credits and constraints of the students. Table 11 gives the 5 feasible maximal matchings and lists out a blocking pair in each. A non-maximal matching will have a student-course pair which can feasibly be allotted, and hence has a blocking pair.

Student	Credits	Student Preference List	Course	Capacity	Course Preference List
s_1	2	c_1, c_2, c_3	c_1	1	s_2, s_1
s_2	2	c_3, c_1, c_2	c_2	1	s_1, s_2
			c_3	1	s_1, s_2
Each student can take only 1 of $\{c_1, c_2\}$ - master class constraint					
s_1 can take only 1 of $\{c_2, c_3\}$ - student class constraint					
s_2 can take only 1 of $\{c_1, c_3\}$ - student class constraint					

Table 10: Example instance which does not exhibit a stable matching with downward feasible constraints.

Matching	Blocking pair
$\{(s_1, c_1), (s_1, c_3), (s_2, c_2)\}$	(s_2, c_1)
$\{(s_1, c_1), (s_2, c_2), (s_2, c_3)\}$	(s_1, c_3)
$\{(s_1, c_2), (s_2, c_3)\}$	(s_1, c_1)
$\{(s_1, c_2), (s_2, c_1)\}$	(s_1, c_2)
$\{(s_1, c_3), (s_2, c_1)\}$	(s_2, c_3)

Table 11: Possible maximal matchings and their blocking pairs

B Integer Linear Program to solve the Maximum Cardinality Matching

In this appendix, we describe the Integer Linear Program used to compute the Maximum Cardinality Matching, which serves as a baseline for our empirical evaluation. The Linear Program, given below, is solved using the IBM CPLEX Optimizer [1]. Here, x_{sc} denotes the inclusion edge (s, c) in the matching, J_s^i denotes the i^{th} class constraint for student s, and $q(J_s^i)$ denotes it's capacity. For course c, $z(c)$ denotes its credits and $q(c)$ it's maximum capacity. Similarly, for each student s, $q(s)$ denotes the student's credit limit. $P(a)$ denotes the preference list of a.

Objective:

$$\textbf{Maximize} \sum_{(s,c)\in E} x_{sc}$$

Constraints:

$$\sum_{c\in P(s)} z(c)\, x_{sc} \leq q(s),\ \forall s \in S \qquad \text{(Credit Constraints)}$$

$$\sum_{s \in P(c)} x_{sc} \leq q(c),\ \forall c \in C \qquad \text{(Course capacity constraints)}$$

$$\sum_{c \in J_s^i} x_{sc} \leq q(J_s^i),\ \forall s \in S, 1 \leq i \leq n_s \qquad \text{(Class constraints)}$$

$$x_{sc} \in \{0,1\},\ \forall (s,c) \in E \qquad \text{(Integrality constraints)}$$

C Experimental Workflow

In this appendix we describe the experimental workflow used to report our tabulated results. The code used can be found in the following Github repository: https://github.com/ved5288/student-course-allocation-with-constraints.

C.1 Software Dependencies and Code Compilation

Following are the software dependencies for running the experiments: `python 3.5, javac, java`. Additionally `numpy` and `IBM ILOG CPLEX Optimization Studio v12.8 - python 3.5 API` are required for the maximum cardinality matching. To compile the java code, run `make` from the project folder. This will create a `CourseAllocationTool.jar` in the same directory.

C.2 Input Data Generator

Run the following terminal command to generate an input instance (Directory: /code/input_generator):

`python3 generate_instance.py <`n_1`> <`n_2`> <`k_{low}`> <`k_{high}`> <flag> <output_folder>`

where n_1 is the number of residents/students, n_2 is the number of hospitals/courses, k_{low}, k_{up} specify the range for the length of student preference lists and *output_folder* is the folder to which the instance is generated. Note that when $flag = 0$, the instance generated corresponds to the master dataset, where the preference lists of students and courses are ordered based on a master list, and when $flag = 1$, the instance generated corresponds to the shuffle dataset, where the preference lists are ordered randomly. These are the parameters used for the three experiments.

- *Varying instance size:* We fix $n_1 : n_2$ as $6 : 1$, while scaling up the size. We also fix $k_{low} = 3, k_{up} = 12$.
- *Increasing competition:* We fix $k_{low} = 3, k_{up} = 12$ and $n_2 = 250$, while we increase the value of n_1.
- *Varying preference list size:* We fix $n_1 = 4000, n_2 = 666$ and $k_{low} = k_{up}$, while scaling k_{low}.

C.3 Maximum Cardinality Matching using CPLEX

Run (`python3 run_cplex.py <input_dir>`) to generate the Maximum Cardinality Matching (MCM) by solving a Linear Program using the CPLEX solver (Directory: /code/input_generator). *input_dir* corresponds to the folder containing the instance. On execution, it automatically generates the LP constraints into `input_dir/max_card_lp.txt` file and invokes CPLEX to solve the LP. The output is written to `input_dir/output.csv`.

C.4 Running the Iterative Algorithm Framework with Gale-Shapley or First Preference algorithms

Run the (`java -jar CourseAllocationTool.jar`), and input the location of the various source files like student list, course list, student preference list, etc., when prompted. Finally, specify the algorithm to use (Gale-Shapley + Iterative Allotment Framework or First Preference + Iterative Allotment Framework). Alternatively, you can load the Maximum Cardinality Matching output to get its statistics. The output is a matching, along with several statistics The `output_dir/aggregateStatistics.csv` file lists the MEAR-S score, MEAR-C score, total number of preferences (TP), size of matching (SM), total number of credits allotted (TC), and the average number of class constraints per student (AC)