A $\frac{3}{2}$-Approximation Algorithm for the Student-Project Allocation Problem

Frances Cooper[1]
School of Computing Science, University of Glasgow
Glasgow, Scotland, UK
f.cooper.1@research.gla.ac.uk
https://orcid.org/0000-0001-6363-9002

David Manlove[2]
School of Computing Science, University of Glasgow
Glasgow, Scotland, UK
david.manlove@glasgow.ac.uk
https://orcid.org/0000-0001-6754-7308

Abstract

The *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S) comprises three sets of agents, namely students, projects and lecturers, where students have preferences over projects and lecturers have preferences over students. In this scenario we seek a *stable matching*, that is, an assignment of students to projects such that there is no student and lecturer who have an incentive to deviate from their assignee/s. We study SPA-ST, the extension of SPA-S in which the preference lists of students and lecturers need not be strictly ordered, and may contain ties. In this scenario, stable matchings may be of different sizes, and it is known that MAX SPA-ST, the problem of finding a maximum stable matching in SPA-ST, is NP-hard. We present a linear-time $\frac{3}{2}$-approximation algorithm for MAX SPA-ST and an Integer Programming (IP) model to solve MAX SPA-ST optimally. We compare the approximation algorithm with the IP model experimentally using randomly-generated data. We find that the performance of the approximation algorithm easily surpassed the $\frac{3}{2}$ bound, constructing a stable matching within 92% of optimal in all cases, with the percentage being far higher for many instances.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Matching problems, Approximation, Algorithms, Stability

Digital Object Identifier 10.4230/LIPIcs.SEA.2018.8

1 Introduction

Background and motivation. In universities all over the world, students need to be assigned to projects as part of their degree programmes. Lecturers typically offer a range of projects, and students may rank a subset of the available projects in preference order. Lecturers may have preferences over students, or over the projects they offer, or they may not have explicit preferences at all. There may also be capacity constraints on the maximum numbers of students that can be allocated to each project and lecturer. The problem of allocating students to projects subject to these preference and capacity constraints is called the *Student-Project Allocation problem* (SPA) [7, Section 5.5][2, 3]. Variants of this problem can be defined for the cases that lecturers have preferences over the students that rank their projects [1],

[1] Supported by an Engineering and Physical Sciences Research Council Doctoral Training Account
[2] Supported by Engineering and Physical Sciences Research Council grant EP/P028306/01

17th International Symposium on Experimental Algorithms (SEA 2018).
Editor: Gianlorenzo D'Angelo; Article No. 8; pp. 8:1–8:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or over the projects they offer [9], or not at all [6]. In this paper we focus on the first of these cases, where lecturers have preferences over students – the so-called *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S).

Finding an optimal allocation of students to projects manually is time-consuming and error-prone. Consequently many universities automate the allocation process using a centralised algorithm. Given the typical sizes of problem instances (e.g., 130 students at the University of Glasgow, School of Computing Science), the efficiency of the matching algorithm is of paramount importance. In the case of SPA-S, the desired matching must be *stable* with respect to the given preference lists, meaning that no student and lecturer have an incentive to deviate from the given allocation and form an assignment with one another [10].

Abraham et al. [1] described a linear-time algorithm to find a stable matching in an instance I of SPA-S when all preference lists in I are strictly ordered. They also showed that, under this condition, all stable matchings in I are of the same size. In this paper we focus on the variant of SPA-S in which preference lists of students and lecturers can contain ties, which we refer to as the *Student-Project Allocation problem with lecturer preferences over Students including Ties* (SPA-ST). Ties allow both students and lecturers to express indifference in their preference lists (in practice, for example, lecturers may be unable to distinguish between certain groups of students). A stable matching in an instance of SPA-ST can be found in linear time by breaking the ties arbitrarily and using the algorithm of Abraham et al. [1].

The *Stable Marriage problem with Ties and Incomplete lists* (SMTI) is a special case of SPA-ST in which each project and lecturer has capacity 1, and each lecturer offers one project. Given an instance of SMTI, it is known that stable matchings can have different sizes [8], and thus the same is true for SPA-ST. Yet in practical applications it is desirable to match as many students to projects as possible. This motivates MAX SPA-ST, the problem of finding a maximum (cardinality) stable matching in an instance of SPA-ST. This problem is NP-hard, since the corresponding optimisation problem restricted to SMTI, which we refer to as MAX SMTI, is NP-hard [8]. Király [5] described a $\frac{3}{2}$-approximation algorithm for MAX SMTI. He also showed how to extend this algorithm to the case of the *Hospitals-Residents problem with Ties* (HRT), where HRT is the special case of SPA-ST in which each lecturer l offers one project p, and the capacities of l and p are equal. Yanagisawa [11] showed that MAX SMTI is not approximable within a factor of $\frac{33}{29}$ unless P=NP; the same bound applies to MAX SPA-ST.

Our contribution. In this paper we describe a linear-time $\frac{3}{2}$-approximation algorithm for MAX SPA-ST. This algorithm is a non-trivial extension of Király's approximation algorithm for HRT as mentioned above. We also describe an Integer Programming (IP) model to solve MAX SPA-ST optimally. Through a series of experiments on randomly-generated data, we then compare the sizes of stable matchings output by our approximation algorithm with the sizes of optimal solutions obtained from our IP model. Our main finding is that the performance of the approximation algorithm easily surpassed the $\frac{3}{2}$ bound on the generated instances, constructing a stable matching within 92% of optimal in all cases, with the percentage being far higher for many instances.

Note that a natural "cloning" technique, involving transforming an instance I of SPA-ST into an instance I' of SMTI, and then using Király's $\frac{3}{2}$-approximation algorithm for SMTI [5] in order to obtain a similar approximation in SPA-ST, does not work in general, as shown in [4, Appendix A]. This motivates the need for a bespoke algorithm for the SPA-ST case.

Structure of this paper. Section 2 gives a formal definition of SPA-ST. Section 3 describes the $\frac{3}{2}$-approximation algorithm, and the IP model for MAX SPA-ST is given in Section 4. The experimental evaluation is described in Section 5, and Section 6 discusses future work.

2 Formal definition of SPA-ST

An instance I of SPA-ST comprises a set $S = \{s_1, s_2, ..., s_{n_1}\}$ of *students*, a set $P = \{p_1, p_2, ..., p_{n_2}\}$ of *projects*, and a set $L = \{l_1, l_2, ..., l_{n_3}\}$ of *lecturers*. Each project is *offered* by one lecturer, and each lecturer l_k *offers* a set of projects $P_k \subseteq P$, where $P_1, \ldots, P_k$ partitions P. Each project $p_j \in P$ has a *capacity* $c_j \in \mathbb{Z}_0^+$, and similarly each lecturer $l_k \in L$ has a *capacity* $d_k \in \mathbb{Z}_0^+$. Each student $s_i \in S$ has a set $A_i \subseteq P$ of *acceptable* projects that they rank in order of preference. *Ties* are allowed in preference lists, where a tie t in a student s_i's list indicates that s_i is indifferent between all projects in t. Each lecturer $l_k \in L$ has a preference list over the students s_i for which $A_i \cap P_k \neq \emptyset$. Ties may also exist in lecturer preference lists. The *rank* of project p_j on student s_i's list, denoted $\text{rank}(s_i, p_j)$, is defined as 1 plus the number of projects that s_i strictly prefers to p_j. An analogous definition exists for the rank of a student on a lecturer's list, denoted $\text{rank}(l_k, s_i)$.

An *assignment* M in I is a subset of $S \times P$ such that, for each pair $(s_i, p_j) \in M$, $p_j \in A_i$, that is, s_i finds p_j acceptable. Let $M(s_i)$ denote the set of projects assigned to a student $s_i \in S$, let $M(p_j)$ denote the set of students assigned to a project $p_j \in P$, and let $M(l_k)$ denote the set of students assigned to projects in P_k for a given lecturer $l_k \in L$. A *matching* M is an assignment such that $|M(s_i)| \leq 1$ for all $s_i \in S$, $|M(p_j)| \leq c_j$ for all $p_j \in P$ and $|M(l_k)| \leq d_k$ for all $l_k \in L$. If $s_i \in S$ is assigned in a matching M, we let $M(s_i)$ denote s_i's assigned project, otherwise $M(s_i)$ is empty.

Given a matching M in I, let $(s_i, p_j) \in (S \times P) \backslash M$ be a student-project pair, where p_j is offered by lecturer l_k. Then (s_i, p_j) is a *blocking pair* of M [1] if 1, 2 and 3 hold as follows:

1. s_i finds p_j acceptable;
2. s_i either prefers p_j to $M(s_i)$ or is unassigned in M;
3. Either a, b or c holds as follows:
   a. p_j is *undersubscribed* (i.e., $|M(p_j)| < c_j$) and l_k is *undersubscribed* (i.e., $|M(l_k)| < d_k$);
   b. p_j is undersubscribed, l_k is full and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$;
   c. p_j is full and l_k prefers s_i to the worst student in $M(p_j)$.

Let (s_i, p_j) be a blocking pair of M. Then we say that (s_i, p_j) is of *type* $(3x)$ if 1, 2 and $3x$ are true in the above definition, where $x \in \{a, b, c\}$. In order to more easily describe certain stages of the approximation algorithm, blocking pairs of type $(3b)$ are split into two subtypes as follows. $(3bi)$ defines a blocking pair of type $(3b)$ where s_i is already assigned to another project of l_k's. $(3bii)$ defines a blocking pair of type $(3b)$ where this is not the case.

A matching M in an instance I of SPA-ST is *stable* if it admits no blocking pair. Define MAX SPA-ST to be the problem of finding a maximum stable matching in SPA-ST and let M_{opt} denote a maximum stable matching for a given instance. Similarly, let MIN SPA-ST be the problem of finding a minimum stable matching in SPA-ST.

3 Approximation algorithm

3.1 Introduction and preliminary definitions

We begin by defining key terminology before describing the approximation algorithm itself in Section 3.2, which is a non-trivial extension of Király's HRT algorithm [5].

A student $s_i \in S$ is either in *phase* 1, 2 or 3. In *phase* 1 there are still projects on s_i's list that they have not applied to. In *phase* 2, s_i has iterated once through their list and are doing so again whilst a priority is given to s_i on each lecturer's preference list, compared to other students who tie with s_i. In *phase* 3, s_i is considered unassigned and carries out no more applications. A project p_j is *fully available* if p_j and l_k are both undersubscribed, where lecturer l_k offers p_j. A student s_i *meta-prefers* project p_{j_1} to p_{j_2} if either (i) rank$(s_i, p_{j_1}) <$ rank(s_i, p_{j_2}), or (ii) rank$(s_i, p_{j_1}) =$ rank(s_i, p_{j_2}) and p_{j_1} is fully available, whereas p_{j_2} is not. In phase 1 or 2, s_i may be either *available*, *provisionally assigned* or *confirmed*. Student s_i is *available* if they are not assigned to a project. Student s_i is *provisionally assigned* if s_i has been assigned in phase 1 and there is a project still on s_i's list that meta-prefers to p_j. Otherwise, s_i is *confirmed*.

If a student s_i is a provisionally assigned to project p_j, then (s_i, p_j) is said to be *precarious*. A project p_j is *precarious* if it is assigned a student s_i such that (s_i, p_j) is precarious. A lecturer is *precarious* if they offer a project p_j that is precarious. Lecturer l_k *meta-prefers* s_{i_1} to s_{i_2} if either (i) rank$(l_k, s_{i_1}) <$ rank(l_k, s_{i_2}), or (ii) rank$(l_k, s_{i_1}) =$ rank(l_k, s_{i_2}) and s_{i_1} is in phase 2, whereas s_{i_2} is not. The *favourite* projects F_i of a student s_i are defined as the set of projects on s_i's preference list for which there is no other project on s_i's list meta-preferred to any project in F_i. A *worst assignee* of lecturer l_k is defined to be a student in $M(l_k)$ of worst rank, with priority given to phase 1 students over phase 2 students. Similarly, a *worst assignee of lecturer l_k in $M(p_j)$* is defined to be a student in $M(p_j)$ of worst rank, prioritising phase 1 over phase 2 students, where l_k offers p_j.

We remark that some of the above terms such as *favourite* and *precarious* have been defined for the SPA-ST setting by extending the definitions of the corresponding terms as given by Király in the HRT context [5].

3.2 Description of the algorithm

Algorithm 1 begins with an empty matching M which will be built up over the course of the algorithm's execution. All students are initially set to be available and in phase 1. The algorithm proceeds as follows. While there are still available students in phase 1 or 2, choose some such student s_i. Student s_i applies to a favourite project p_j at the head of their list, that is, there is no project on s_i's list that s_i meta-prefers to p_j. Let l_k be the lecturer who offers p_j. We consider the following cases.

- If p_j and l_k are both undersubscribed then (s_i, p_j) is added to M. Clearly if (s_i, p_j) were not added to M, it would potentially be a blocking pair of type $(3a)$.
- If p_j is undersubscribed, l_k is full and l_k is precarious where precarious pair $(s_{i'}, p_{j'}) \in M$ for some project p'_j offered by l_k, then we remove $(s_{i'}, p_{j'})$ from M and add pair (s_i, p_j). This notion of precariousness allows us to find a stable matching of sufficient size even when there are ties in student preference lists (there may also be ties in lecturer preference lists). Allowing a pair $(s_{i'}, p_{j'}) \in M$ to be precarious means that we are noting that $s_{i'}$ has other fully available project options in their preference list at equal rank to $p_{j'}$. Hence, if another student applies to $p_{j'}$ when $p_{j'}$ is full, or to a project offered by l_k where l_k is full, we allow this assignment to happen removing $(s_{i'}, p_{j'})$ from M, since there is a chance that the size of the resultant matching could be increased.
- If on the other hand p_j is undersubscribed, l_k is full and l_k meta-prefers s_i to a worst assignee $s_{i'}$, where $(s_{i'}, p_{j'}) \in M$ for some project $p_{j'}$ offered by l_k, then we remove $(s_{i'}, p_{j'})$ from M and add pair (s_i, p_j). It makes intuitive sense that if l_k is full and gets an offer to an undersubscribed project from a student s_i that they prefer to a worst assigned student $s_{i'}$, then l_k would want to remove $s_{i'}$ from $p_{j'}$ and take on s_i for $p_{j'}$. Student $s_{i'}$ will subsequently remove $p_{j'}$ from their preference list as l_k will not want to assign to them on re-application. This is done via the Remove-pref method (Algorithm 2).

Algorithm 1 3/2-approximation algorithm for SPA-ST.

Require: An instance I of SPA-ST
Ensure: Return a stable matching M where $|M| \geq \frac{2}{3}|M_{opt}|$
$M \leftarrow \emptyset$
all students are initially set to be available and in phase 1
while there exists an available student $s_i \in S$ who is in phase 1 or 2 **do**
  let l_k be the lecturer who offers p_j
  s_i applies to a favourite project $p_j \in A(s_i)$
  if p_j is fully available **then**
    $M \leftarrow M \cup \{(s_i, p_j)\}$
  else if p_j is undersubscribed, l_k is full **and** (l_k is precarious **or** l_k meta-prefers s_i to a worst assignee) **then** ▷ according to the *worst assignee* definition in Section 3.1
    if l_k is precarious **then**
      let $p_{j'}$ be a project in P_k such that there exists $(s_{i'}, p_{j'}) \in M$ that is precarious
    else ▷ l_k is not precarious
      let $s_{i'}$ be a worst assignee of l_k such that l_k meta-prefers s_i to $s_{i'}$ and let $p_{j'} = M(s_{i'})$
      Remove-Pref($s_{i'}, p_{j'}$)
    end if
    $M \leftarrow M \backslash \{(s_{i'}, p_{j'})\}$
    $M \leftarrow M \cup \{(s_i, p_j)\}$
  else if p_j is full **and** (p_j is precarious **or** l_k meta-prefers s_i to a worst assignee in $M(p_j)$) **then**
    if p_j is precarious **then**
      identify a student $s_{i'} \in M(p_j)$ such that $(s_{i'}, p_j)$ is precarious
    else ▷ p_j is not precarious
      let $s_{i'}$ be a worst assignee of l_k in $M(p_j)$ such that l_k meta-prefers s_i to $s_{i'}$
      Remove-Pref($s_{i'}, p_j$)
    end if
    $M \leftarrow M \backslash \{(s_{i'}, p_j)\}$
    $M \leftarrow M \cup \{(s_i, p_j)\}$
  else
    Remove-Pref(s_i, p_j)
  end if
end while
Promote-students(M)
return M;

- If p_j is full and precarious then pair (s_i, p_j) is added to M while precarious pair $(s_{i'}, p_j)$ is removed. As before, this allows $s_{i'}$ to potentially assign to other fully available projects at the same rank as p_j on their list. Since $s_{i'}$ does not remove p_j from their preference list, $s_{i'}$ will get another chance to assign to p_j if these other applications to fully available projects at the same rank are not successful.
- If p_j is full and l_k meta-prefers s_i to a worst assignee $s_{i'}$ in $M(p_j)$, then pair (s_i, p_j) is added to M while $(s_{i'}, p_j)$ is removed. As this lecturer's project is full (and not precarious) the only time they will want to add a student s_i to this project (meaning the removal of another student) is if s_i is preferred to a worst student $s_{i'}$ assigned to that project. Similar to before, $s_{i'}$ will not subsequently be able to assign to this project and so removes it from their preference list via the Remove-pref method (Algorithm 2).

Algorithm 2 Remove-Pref(s_i, p_j) – remove a project from a student's preference list.

Require: An instance I of SPA-ST and a student s_i and project p_j
Ensure: Return an instance I where p_j is removed from s_i's preference list
remove p_j from s_i's preference list
if s_i's preference list is empty **then**
    reinstate s_i's preference list
    if s_i is in phase 1 **then**
        move s_i to phase 2
    else if s_i is in phase 2 **then**
        move s_i to phase 3
    end if
end if
return instance I

Algorithm 3 Promote-students(M) – remove all blocking pairs of type ($3bi$).

Require: SPA-ST instance I and matching M that does not contain blocking pairs of type ($3a$), ($3bii$) or ($3c$).
Ensure: Return a stable matching M.
while there are still blocking pairs of type ($3bi$) **do**
    Let $(s_i, p_{j'})$ be a blocking pair of type ($3bi$)
    $M \leftarrow M \backslash \{(s_i, M(s_i))\}$
    $M \leftarrow M \cup \{(s_i, p_{j'})\}$
end while
return M

When removing a project from a student s_i's preference list (the Remove-pref operation of Algorithm 2), if s_i has removed all projects from their preference list and is in phase 1 then their preference list is reinstated and they are set to be in phase 2. If on the other hand they were already in phase 2, then they are set to be in phase 3 and are hence inactive. The proof that Algorithm 1 produces a stable matching (see [4, Appendix B]) relies only on the fact that a student iterates once through their preference list. Allowing students to iterate through their preference lists a second time when in phase 2 allows us to find a stable matching of sufficient size when there are ties in lecturer preference lists (there may also be ties in student preference lists). This is due to the meta-prefers definition where a lecturer favours one student s_i over another $s_{i'}$ if they are the same rank and s_i is in phase 2 whereas $s_{i'}$ is not. Similar to above, this then allows s_i to steal a position from $s_{i'}$ with the chance that $s_{i'}$ may find another assignment and increase the size of the resultant matching.

After the main while loop has terminated, the final part of the algorithm begins where all blocking pairs of type ($3bi$) are removed using the Promote-students method (Algorithm 3).

3.3 Proof of correctness

▶ **Theorem 1.** *Let M be a matching found by Algorithm 1 for an instance I of* SPA-ST*. Then M is stable and $|M| \geq \frac{2}{3}|M_{opt}|$, where M_{opt} is a maximum stable matching in I.*

Proof. Theorems 18, 22 and Theorem 30, proved in [4, Appendix B], show that M is stable, and that Algorithm 1 runs in polynomial time and has performance guarantee $\frac{3}{2}$. The proofs required for this algorithm are naturally longer and more complex than given by Király [5] for SMTI, as SPA-ST generalises SMTI to the case that lecturers can offer multiple projects, and projects and lecturers may have capacities greater than 1. These extensions add extra components to the definition of a blocking pair (given in Section 2) which in turn adds complexity to the algorithm and its proof of correctness. ◀

Appendix B.5 in [4] gives a simple example instance where a matching found by Algorithm 1 is *exactly* $\frac{2}{3}$ times the optimal size, hence the analysis of the performance guarantee is tight.

4 IP model

In this section we present an IP model for MAX SPA-ST. For the stability constraints in the model, it is advantageous to use an equivalent condition for stability, as given by the following lemma, whose proof can be found in [4, Appendix C].

▶ **Lemma 2.** *Let I be an instance of SPA-ST and let M be a matching in I. Then M is stable if and only if the following condition, referred to as condition (*) holds: For each student $s_i \in S$ and project $p_j \in P$, if s_i is unassigned in M and finds p_j acceptable, or s_i prefers p_j to $M(s_i)$, then either:*

- *l_k is full, $s_i \notin M(l_k)$ and l_k prefers the worst student in $M(l_k)$ to s_i or is indifferent between them, or;*
- *p_j is full and l_k prefers the worst student in $M(p_j)$ to s_i or is indifferent between them, where l_k is the lecturer offering p_j.*

The key variables in the model are binary-valued variables x_{ij}, defined for each $s_i \in S$ and $p_j \in P$, where $x_{ij} = 1$ if and only if student s_i is assigned to project p_j. Additionally, we have binary-valued variables α_{ij} and β_{ij} for each $s_i \in S$ and $p_j \in P$. These variables allow us to more easily describe the stability constraints below. For each $s_i \in S$ and $l_k \in L$, let

$$T_{ik} = \{s_u \in S : \text{rank}(l_k, s_u) \leq rank(l_k, s_i) \wedge s_u \neq s_i\}.$$

That is, T_{ik} is the set of students ranked at least as highly as student s_i in lecturer l_k's preference list not including s_i. Also, for each $p_j \in P$, let

$$T_{ijk} = \{s_u \in S : \text{rank}(l_k, s_u) \leq rank(l_k, s_i) \wedge s_u \neq s_i \wedge p_j \in A(s_u)\}.$$

That is, T_{ijk} is the set of students s_u ranked at least as highly as student s_i in lecturer l_k's preference list, such that project p_j is acceptable to s_u, not including s_i. Finally, let $S_{ij} = \{p_r \in P : \text{rank}(s_i, p_r) \leq rank(s_i, p_j)\}$, that is, S_{ij} is the set of projects ranked at least as highly as project p_j in student s_i's preference list, including p_j. Figure 1 shows the IP model for MAX SPA-ST.

Equation (1) enforces $x_{ij} = 0$ if s_i finds p_j unacceptable. Inequality (2) ensures that a student may be assigned to a maximum of one project. Inequalities (3) and (4) ensure that project and lecturer capacities are enforced. In the left hand side of Inequality (5), if $1 - \sum_{p_r \in S_{ij}} x_{ir} = 1$, then either s_i is unmatched or s_i prefers p_j to $M(s_i)$. This also ensures that either $\alpha_{ij} = 1$ or $\beta_{ij} = 1$, described in Inequalities (6) and (7). Inequality (6) ensures that, if $\alpha_{ij} = 1$, the number of students ranked at least as highly as student s_i by l_k (not including s_i) and assigned to l_k must be at least l_k's capacity d_k. Inequality (7) ensures that, if $\beta_{ij} = 1$, the number of students ranked at least as highly as student s_i in lecturer l_k's preference list (not including s_i) and assigned to p_j must be at least p_j's capacity c_j.

Finally, for our optimisation we maximise the sum of all x_{ij} variables in order to maximise the number of students assigned. The following result, proved in [4, Appendix C], establishes the correctness of the IP model.

$$
\begin{array}{lll}
\text{maximise:} & \displaystyle\sum_{s_i \in S} \sum_{p_j \in P} x_{ij} & \\
\text{subject to:} & & \\
1. & x_{ij} = 0 & \forall s_i \in S \;\; \forall p_j \in P,\; p_j \notin A(s_i) \\
2. & \displaystyle\sum_{p_j \in P} x_{ij} \le 1 & \forall s_i \in S \\
3. & \displaystyle\sum_{s_i \in S} x_{ij} \le c_j & \forall p_j \in P \\
4. & \displaystyle\sum_{s_i \in S} \sum_{p_j \in P_k} x_{ij} \le d_k & \forall l_k \in L \\
5. & 1 - \displaystyle\sum_{p_r \in S_{ij}} x_{ir} \le \alpha_{ij} + \beta_{ij} & \forall s_i \in S \;\; \forall p_j \in P \\
6. & \displaystyle\sum_{s_u \in T_{ik}} \sum_{p_r \in P_k} x_{ur} \ge d_k \alpha_{ij} & \forall s_i \in S \;\; \forall p_j \in P \\
7. & \displaystyle\sum_{s_u \in T_{ijk}} x_{uj} \ge c_j \beta_{ij} & \forall s_i \in S \;\; \forall p_j \in P \\
 & x_{ij} \in \{0,1\}, \quad \alpha_{ij} \in \{0,1\}, \quad \beta_{ij} \in \{0,1\} & \forall s_i \in S \;\; \forall p_j \in P
\end{array}
$$

Figure 1 IP model for MAX SPA-ST.

▶ **Theorem 3.** *Given an instance I of* SPA-ST*, let J be the IP model as defined in Figure 1. A maximum stable matching in I corresponds to an optimal solution in J and vice versa.*

5 Experimental evaluation

5.1 Methodology

Experiments were conducted on the approximation algorithm and the IP model using randomly-generated data in order to measure the effects on matching statistics when changing parameter values relating to (1) instance size, (2) probability of ties in preference lists, and (3) preference list lengths. Two further experiments (referred to as (4) and (5) below) explored scalability properties for both techniques. Instances were generated using both existing and new software. The existing software is known as the *Matching Algorithm Toolkit* and is a collaborative project developed by students and staff at the University of Glasgow.

For a given SPA-ST instance, let the total project and lecturer capacities be denoted by c_P and d_L, respectively. Note that these capacities were distributed randomly, subject to there being a maximum difference of 1 between the capacities of any two projects or any two lecturers (to ensure uniformity). The minimum and maximum size of student preference lists is given by l_{min} and l_{max}, and t_s represents the probability that a project on a student's preference list is tied with the next project. Lecturer preference lists were generated initially from the student preference lists, where a lecturer l_k must rank a student if a student ranks a project offered by l_k. These lists were randomly shuffled and t_l denotes the ties probability for lecturer preference lists. A linear distribution was used to make some projects more popular than others and in all experiments the most popular project is around 5 times more popular than the least. This distribution influenced the likelihood of a student finding a given project acceptable. Parameter details for each experiment are given below.

(1) Increasing instance size: 10 sets of $10,000$ instances were created (labelled SIZE1, ..., SIZE10). The number of students n_1 increased from 100 to 1000 in steps of 100, with $n_2 = 0.6n_1$, $n_3 = 0.4n_1$, $c_P = 1.4n_1$, $d_L = 1.2n_1$. The probabilities of ties in preference lists were $t_s = t_l = 0.2$ throughout all instance sets. Lengths of preference lists $l_{min} = 3$ and $l_{max} = 5$ also remained the same and were kept low to ensure a wide variability in stable matching size per instance.

(2) Increasing probability of ties: 11 sets of $10,000$ instances were created (labelled TIES1, ..., TIES11). Throughout all instance sets $n_1 = 300$, $n_2 = 250$, $n_3 = 120$, $c_P = 420$, $d_L = 360$, $l_{min} = 3$ and $l_{max} = 5$. The probabilities of ties in student and lecturer preference lists increased from $t_s = t_l = 0.0$ to $t_s = t_l = 0.5$ in steps of 0.05.

(3) Increasing preference list lengths: 10 sets of $10,000$ instances were generated (labelled PREF1, ..., PREF10). Similar to the TIES cases, throughout all instance sets $n_1 = 300$, $n_2 = 250$, $n_3 = 120$, $c_P = 420$ and $d_L = 360$. Additionally, $t_s = t_l = 0.2$. Preference list lengths increased from $l_{min} = l_{max} = 1$ to $l_{min} = l_{max} = 10$ in steps of 1.

(4) Instance size scalability: 5 sets of 10 instances were generated (labelled SCALS1, ..., SCALS5). All instance sets in this experiment used the same parameter values as the SIZE experiment, except the number of students n_1 increased from $10,000$ to $50,000$ in steps of $10,000$.

(5) Preference list scalability: Finally, 6 sets of 10 instances were created (labelled SCALP1, ..., SCALP6). Throughout all instance sets $n_1 = 500$ with the same values for other parameters as the SIZE experiment. However in this case ties were fixed at $t_s = t_l = 0.4$, and $l_{min} = l_{max}$ increasing from 25 to 150 in steps of 25.

For each generated instance, we ran the $\frac{3}{2}$-approximation algorithm and then used the IP model to find a maximum stable matching. We also computed a minimum stable matching using a simple adaptation of our IP model for MAX SPA-ST, in order to measure the spread in the sizes of stable matchings. A timeout of 1800 seconds (30 minutes) was imposed on all instance runs. All experiments were conducted using a machine with 32 cores, 8×64GB RAM and Dual Intel® Xeon® CPU E5-2697A v4 processors. The operating system was Ubuntu version 17.04 with all code compiled in Java version 1.8, where the IP models were solved using Gurobi version 7.5.2. Each approximation algorithm instance was run on a single thread while each IP instance was run on two threads. No attempt was made to parallelise Java garbage collection. Repositories for the code and data can be found at `https://doi.org/10.5281/zenodo.1183221` and `https://doi.org/10.5281/zenodo.1186823` respectively.

Correctness testing was conducted over all generated instances. This consisted of (1) ensuring that each matching produced by the approximation algorithm was at least $\frac{2}{3}$ the size of maximum stable matching, as found by the IP, and, (2) testing that a given allocation was stable and adhered to all project and lecturer capacities. This was run over all output from both the approximation algorithm and the IP-based algorithm.

5.2 Experimental results

Experimental results can be seen in Tables 1, 2, 3 and 4. Tables 1, 2 and 3 show the results from Experiments 1, 2 and 3 respectively (in which the instance size, probability of ties and preference list lengths were increased, respectively). From this point onwards an *optimal* matching refers to a maximum stable matching. In these tables, column 'minimum A/Max' gives the minimum ratio of approximation algorithm matching size to optimal matching size that occurred, '% A=Max' displays the percentage of times the approximation algorithm achieved an optimal result, and '% A$\geq$ 0.98Max' shows the percentage of times the approximation algorithm achieved a result at least 98% of optimal. The 'average size' columns are somewhat self explanatory, with sub-columns 'A/Max' and 'Min/Max' showing the average approximation algorithm matching size and minimum stable matching size as a fraction of optimal. Finally, 'average total time' indicates the time taken for model creation, solving and outputting results *per instance*. The main findings are summarised below.

- *The approximation algorithm consistently far exceeds its $\frac{3}{2}$ bound.* Considering the column labelled 'minimum A/Max' in Tables 1, 2 and 3, we see that the smallest value was within the SIZE1 instance set with a ratio of 0.9286. This is well above the required bound of $\frac{2}{3}$.
- *On average the approximation algorithm provides results that are closer in size to the average maximum stable matching than the minimum stable matching.* The columns 'A/Max' and 'Min/Max' show that, on average, for each instance set, the approximation algorithm produces a solution that is within 98% of maximum and far closer to the maximum size than to the minimum size.

Table 4 shows the scalability results for increasing instance sizes (Experiment 4) and increasing preference list lengths (Experiment 5). The 'instances completed' column indicates the number of instances completed before timeout occurred. In addition to showing the average total time taken (where 'total' includes model creation time and solution time), the column 'average solve time' displays the time taken to either execute the approximation algorithm, or solve the IP model (in both cases, model creation time is excluded).

For Experiment 4, the number of instances solved within the 30-minute timeout reduced from 10 to 0 for the IP-based algorithm finding the maximum stable matching. However, even for the largest instance set sizes the approximation algorithm was able to solve all instances on average within a total of 21 seconds (0.8 seconds of which was used to actually execute the algorithm).

For Experiment 5, with a higher probability of ties and increasing preference list lengths, the IP-based algorithm was only able to solve all the instances of one instance set (SCALP2) within 30 minutes each, however the approximation algorithm took less than 0.3 seconds on average to return a solution for each instance. This shows that the approximation algorithm is useful for either larger or more complex instances than the IP-based algorithm can handle, motivating its use for real world scenarios.

6 Future work

This paper has described a $\frac{3}{2}$-approximation algorithm for MAX SPA-ST. It remains open to describe an approximation algorithm that has a better performance guarantee, and/or to prove a stronger lower bound on the inapproximability of the problem than the current best bound of $\frac{33}{29}$ [11]. Further experiments could also measure the extent to which the order that students apply to projects in Algorithm 1 affects the size of the stable matching generated.

The work in this paper has mainly focused on the size of stable matchings. However, it is possible for a stable matching to admit a *blocking coalition*, where a permutation of student assignments could improve the allocations of the students and lecturers involved without harming anyone else. Since permutations of this kind cannot change the size of the matching they are not studied further here, but would be of interest for future work.

Table 1 Increasing instance size experimental results.

Case	minimum A/Max	% A=Max	% A≥ 0.98Max	average size A	Min	Max	A/Max	Min/Max	average total time (ms) A	Min	Max
SIZE1	0.9286	17.8	62.7	96.4	92.0	97.8	0.986	0.941	43.3	147.6	137.8
SIZE2	0.9585	1.6	62.6	192.6	183.4	195.7	0.984	0.937	51.2	230.6	210.6
SIZE3	0.9556	0.1	63.7	288.7	274.9	293.7	0.983	0.936	56.6	346.4	313.4
SIZE4	0.9644	0.0	65.6	384.9	366.4	391.7	0.983	0.935	59.7	488.7	429.3
SIZE5	0.9654	0.0	66.5	481.0	457.7	489.6	0.982	0.935	62.8	660.3	555.6
SIZE6	0.9641	0.0	66.8	577.2	549.3	587.7	0.982	0.935	66.4	862.3	713.0
SIZE7	0.9679	0.0	65.4	673.3	640.5	685.7	0.982	0.934	69.8	1127.8	900.6
SIZE8	0.9684	0.0	67.4	769.5	732.0	783.8	0.982	0.934	73.0	1437.3	1098.2
SIZE9	0.9653	0.0	68.6	865.6	823.4	881.7	0.982	0.934	76.5	1784.3	1343.9
SIZE10	0.9701	0.0	68.0	961.7	914.7	979.7	0.982	0.934	86.6	2281.2	1651.0

Table 2 Increasing probability of ties experimental results.

Case	minimum A/Max	% A=Max	% A≥ 0.98Max	average size A	Min	Max	A/Max	Min/Max	average total time (ms) A	Min	Max
TIES1	1.0000	100.0	100.0	284.0	284.0	284.0	1.000	1.000	59.2	184.0	186.9
TIES2	0.9792	38.0	100.0	284.9	282.0	285.8	0.997	0.987	61.2	192.4	194.7
TIES3	0.9722	12.1	99.3	285.9	279.9	287.9	0.993	0.972	61.7	201.0	203.1
TIES4	0.9655	3.4	95.2	287.0	277.6	289.9	0.990	0.958	62.3	213.3	214.5
TIES5	0.9626	1.0	82.5	288.0	275.1	291.9	0.986	0.942	62.9	234.3	231.0
TIES6	0.9558	0.4	66.7	289.2	272.4	294.0	0.984	0.927	64.2	274.2	260.6
TIES7	0.9486	0.2	52.9	290.3	269.4	295.7	0.982	0.911	64.3	358.3	311.3
TIES8	0.9527	0.2	46.4	291.4	266.2	297.2	0.980	0.896	64.2	577.3	380.7
TIES9	0.9467	0.2	50.4	292.5	262.7	298.3	0.980	0.880	65.2	1234.1	427.5
TIES10	0.9529	0.5	61.9	293.7	258.9	299.1	0.982	0.866	59.6	2903.4	409.1
TIES11	0.9467	1.0	74.2	294.8	254.8	299.5	0.984	0.851	60.4	5756.9	377.4

Table 3 Increasing preference list length experimental results.

Case	minimum A/Max	% A=Max	% A≥ 0.98Max	average size A	average size Min	average size Max	average size A/Max	average size Min/Max	average total time (ms) A	average total time (ms) Min	average total time (ms) Max
PREF1	1.0000	100.0	100.0	215.0	215.0	215.0	1.000	1.000	74.3	107.5	105.1
PREF2	0.9699	12.3	99.0	262.1	249.1	264.1	0.993	0.943	67.5	133.8	128.7
PREF3	0.9617	1.2	84.0	280.9	266.4	284.7	0.987	0.936	68.1	181.4	174.0
PREF4	0.9623	1.0	82.8	290.0	277.0	293.9	0.987	0.943	69.1	249.7	242.6
PREF5	0.9661	4.2	95.1	294.8	283.9	297.7	0.990	0.954	68.3	346.7	340.3
PREF6	0.9732	15.7	99.5	297.3	288.7	299.1	0.994	0.965	66.1	472.4	440.6
PREF7	0.9767	36.2	100.0	298.7	292.1	299.7	0.997	0.975	64.5	638.3	550.9
PREF8	0.9833	58.2	100.0	299.3	294.4	299.9	0.998	0.982	64.1	811.9	660.3
PREF9	0.9866	75.5	100.0	299.7	296.1	299.9	0.999	0.987	63.4	1032.2	789.1
PREF10	0.9900	87.3	100.0	299.8	297.4	300.0	1.000	0.991	104.3	1239.4	931.0

Table 4 Scalability experimental results.

Case	instances completed A	instances completed Min	instances completed Max	average solve time (ms) A	average solve time (ms) Min	average solve time (ms) Max	average total time (ms) A	average total time (ms) Min	average total time (ms) Max
SCALS1	10	10	10	136.5	126162.8	225917.9	1393.8	127980.3	227764.3
SCALS2	10	10	9	242.4	348849.4	1091424.2	5356.7	353272.3	1096045.6
SCALS3	10	10	0	491.7	777267.7	N/A	13095.3	785421.2	N/A
SCALS4	10	7	0	718.8	1049122.0	N/A	18883.5	1062076.4	N/A
SCALS5	10	7	0	803.5	1288961.1	N/A	20993.0	1307728.7	N/A
SCALP1	10	0	9	25.1	N/A	93086.0	193.3	N/A	94242.9
SCALP2	10	1	10	23.3	1425177.0	626774.9	189.4	1428844.0	631225.2
SCALP3	10	0	3	31.7	N/A	867107.7	196.6	N/A	882251.0
SCALP4	10	0	1	37.8	N/A	1551376.0	248.5	N/A	1594201.0
SCALP5	10	0	0	59.0	N/A	N/A	283.7	N/A	N/A
SCALP6	10	0	0	45.7	N/A	N/A	288.4	N/A	N/A

References

1 D.J. Abraham, R.W. Irving, and D.F. Manlove. Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5:73–90, 2007.

2 R. Calvo-Serrano, G. Guillén-Gosálbez, S. Kohn, and A. Masters. Mathematical programming approach for optimally allocating students' projects to academics in large cohorts. *Education for Chemical Engineers*, 20:11–21, 2017.

3 M. Chiarandini, R. Fagerberg, and S. Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research,* to appear, 2018.

4 F. Cooper and D. Manlove. A $\frac{3}{2}$-approximation algorithm for the Student-Project Allocation problem. Technical Report 1804.02731, Computing Research Repository, Cornell University Library, 2018. Available from `http://arxiv.org/abs/1804.02731`.

5 Z. Király. Linear time local approximation for maximum stable marriage. *Algorithms*, 6:471–484, 2013.

6 A. Kwanashie, R.W. Irving, D.F. Manlove, and C.T.S. Sng. Profile-based optimal matchings in the Student–Project Allocation problem. In *Proceedings of IWOCA '14: the 25th International Workshop on Combinatorial Algorithms*, volume 8986 of *Lecture Notes in Computer Science*, pages 213–225. Springer, 2015.

7 D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.

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

9 D.F. Manlove and G. O'Malley. Student-project allocation with preferences over projects. *Journal of Discrete Algorithms*, 6:553–560, 2008.

10 A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

11 H. Yanagisawa. *Approximation Algorithms for Stable Marriage Problems*. PhD thesis, Kyoto University, School of Informatics, 2007.