Super-stability in the Student-Project Allocation Problem with Ties

Sofiat Olaosebikan* and David Manlove**

School of Computing Science, University of Glasgow,
e-mail: s.olaosebikan.1@research.gla.ac.uk, David.Manlove@glasgow.ac.uk

Abstract. The *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S) involves assigning students to projects based on student preferences over projects, lecturer preferences over students, and the maximum number of students that each project and lecturer can accommodate. This classical model assumes that preference lists are strictly ordered. Here, we study a generalisation of SPA-S where ties are allowed in the preference lists of students and lecturers, which we refer to as the *Student-Project Allocation problem with lecturer preferences over Students with Ties* (SPA-ST). We investigate stable matchings under the most robust definition of stability in this context, namely *super-stability*. We describe the first polynomial-time algorithm to find a super-stable matching or to report that no such matching exists, given an instance of SPA-ST. Our algorithm runs in $O(L)$ time, where L is the total length of all the preference lists. Finally, we present results obtained from an empirical evaluation of the linear-time algorithm based on randomly-generated SPA-ST instances. Our main finding is that, whilst super-stable matchings can be elusive, the probability of such a matching existing is significantly higher if ties are restricted to the lecturers' preference lists.

1 Introduction

The *Student-Project Allocation problem* (SPA) [5,16] involves sets of students, projects and lecturers, where students are to be assigned to projects offered by lecturers. Applications of SPA can be found in many university departments, for example, the School of Computing Science, University of Glasgow [15], the Faculty of Science, University of Southern Denmark [6], the Department of Computing Science, University of York [14], and elsewhere [3,5,8]. In this setting, lecturers provide a list of projects, and students are required to rank a subset of these projects that they find acceptable, in order of preference. Typically there may be upper bounds on the number of students that each project and lecturer can accommodate. Considering the preferences and the capacities of projects and lecturers, the problem then is to find a *matching* (i.e., an assignment of students

* Supported by a College of Science and Engineering Scholarship, University of Glasgow. Orcid ID: 0000-0002-8003-7887.

** Supported by grant EP/P028306/1 from the Engineering and Physical Sciences Research Council. Orcid ID: 0000-0001-6754-7308.

to projects such that each student is assigned at most one project, and the capacity constraints on projects and lecturers are not violated), which is optimal in some sense according to the stated preferences.

In this work, we will concern ourselves with a variant of SPA that involves lecturer preferences over students, which is known as the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S). In this context, it has been argued [22] that a natural property for a matching to satisfy is that of *stability*. Informally, a *stable matching* ensures that no student and lecturer who are not matched together would rather be assigned to each other than remain with their current assignees. Such a pair would have an incentive to form a private arrangement outside of the matching, undermining its integrity. Other variants of SPA in the literature involve lecturer preferences over their proposed projects [13,18,19], lecturer preferences over (student, project) pairs [2], and no lecturer preferences at all [15]. See [6] for a recent survey.

The classical SPA-S model assumes that preferences are strictly ordered. However, this might not be achievable in practice. For instance, a lecturer may be unable or unwilling to provide a strict ordering of all the students who find her projects acceptable. Such a lecturer may be happier to rank two or more students equally in a tie, which indicates that the lecturer is indifferent between the students concerned. This leads to a generalisation of SPA-S which we refer to as the *Student-Project Allocation problem with lecture preferences over Students with Ties* (SPA-ST). If we allow ties in the preference lists of students and lecturers, different stability definitions naturally arise. Suppose M is a matching in an instance of SPA-ST. Informally, we say M is *weakly stable, strongly stable* or *super-stable* if there is no student and lecturer such that if they decide to form an arrangement outside the matching, respectively,

(i) both of them would be better off,
(ii) one of them would be better off and the other no worse off,
(iii) neither of them would be worse off.

With respect to this informal definition, clearly a super-stable matching is strongly stable, and a strongly stable matching is weakly stable. These concepts were first defined and studied by Irving [9] in the context of the Stable Marriage problem with Ties, and subsequently extended to the Hospitals/Residents problem with Ties (HRT) [10,11] (where HRT is the special case of SPA-ST in which each lecturer offers only one project, and the capacity of each project is the same as the capacity of the lecturer offering the project).

Considering the weakest of the three stability concepts mentioned above, every instance of SPA-ST admits a weakly stable matching (this follows by breaking the ties in an arbitrary fashion and applying the stable matching algorithm described in [1] to the resulting SPA-S instance). However, such matchings could be of different sizes [17]. Thus opting for weak stability leads to the problem of finding a weakly stable matching that matches as many students to projects as possible – a problem that is known to be NP-hard [12,17], even for the so-called *Stable Marriage problem with Ties and Incomplete lists*, which is the special case of HRT

in which each project (hospital) has capacity 1. Further, a $\frac{3}{2}$-approximation algorithm was described in [7] for the problem of finding a maximum weakly stable matching in an instance of SPA-ST.

Choosing super-stability avoids the problem of finding a weakly stable matching with optimal cardinality, because (i) analogous to the HRT case, all super-stable matchings have the same size [10], (ii) finding one or reporting that none exists can be accomplished in linear-time (as we will see in this paper), and (iii) if a super-stable matching M exists then all weakly stable matchings are of the same size (equal to the size of M), and match exactly the same set of students (see [20] for proof). Furthermore, Irving *et al.* argued in [10] that super-stability is a very natural solution concept in cases where agents have incomplete information. Central to their argument is the following proposition, stated for HRT in [10, Proposition 2], which extends naturally to SPA-ST as follows (see [20] for proof).

Proposition 1. *Let* I *be an instance of SPA-ST, and let M be a matching in* I*. Then M is super-stable in* I *if and only if M is stable in every instance of SPA-S obtained from* I *by breaking the ties in some way.*

In a practical setting, suppose that a student s_i has incomplete information about two or more projects and decides to rank them equally in a tie T, and a super-stable matching M exists in the corresponding SPA-ST instance I, where s_i is assigned to a project in T. Then M is stable in every instance of SPA-S (obtained from I by breaking the ties) that represents the true preferences of s_i. Consequently, we will focus on the concept of super-stability in the SPA-ST context.

Unfortunately not every instance of SPA-ST admits a super-stable matching. This is true, for example, in the case where there are two students, two projects and one lecturer, where the capacity of each project is 1, capacity of the lecturer is 2, and every preference list is a single tie of length 2. Nonetheless, it should be clear from the discussion above that a super-stable matching should be preferred in practical applications when one does exist.

Irving *et al.* [10] described an algorithm to find a super-stable matching given an instance of HRT, or to report that no such matching exists. However, merely reducing an instance of SPA-ST to an instance of HRT and applying the algorithm described in [10] to the resulting HRT instance does not work in general (see [20] for a further explanation).

Our Contribution. In this paper, we describe the first polynomial-time algorithm to find a super-stable matching or to report that no such matching exists, given an instance of SPA-ST – thus solving an open problem given in [1,16]. Our algorithm runs in time linear in the size of the problem instance. The remaining sections of this paper are structured as follows. We give a formal definition of the SPA-S problem, the SPA-ST variant, and the super-stability concept in Sect. 2. We describe our algorithm for SPA-ST under super-stability in Sect. 3. Further, Sect. 3 also presents our algorithm's correctness results and some structural properties satisfied by the set of super-stable matchings in an instance of SPA-ST. In Sect. 4,

we present results arising from an empirical evaluation that investigates how the nature of the preference lists would affect the likelihood of a super-stable matching existing, with respect to randomly-generated SPA-ST instances. Our main finding is that the probability of a super-stable matching existing is significantly higher if ties are restricted to the lecturers' preference lists. Finally, Sect. 5 presents some concluding remarks and potential direction for future work.

2 Preliminary definitions

2.1 Formal definition of SPA-S

An instance I of SPA-S involves a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_1}\}$ of *students*, a set $\mathcal{P} = \{p_1, p_2, \ldots, p_{n_2}\}$ of *projects* and a set $\mathcal{L} = \{l_1, l_2, \ldots, l_{n_3}\}$ of *lecturers*. Each student s_i ranks a subset of $\mathcal{P}$ in strict order. We denote by A_i the ranked set of projects that s_i finds acceptable. We say that s_i finds p_j *acceptable* if $p_j \in A_i$.

Each lecturer $l_k \in \mathcal{L}$ offers a non-empty set of projects P_k, where $P_1, P_2, \ldots, P_{n_3}$ partitions $\mathcal{P}$, and l_k provides a preference list, denoted by $\mathcal{L}_k$, ranking in strict order of preference those students who find at least one project in P_k acceptable. Also l_k has a capacity $d_k \in \mathbb{Z}^+$, indicating the maximum number of students she is willing to supervise. Similarly each project $p_j \in \mathcal{P}$ has a capacity $c_j \in \mathbb{Z}^+$ indicating the maximum number of students that it can accommodate.

We assume that for any lecturer l_k, $\max\{c_j : p_j \in P_k\} \leq d_k \leq \sum\{c_j : p_j \in P_k\}$ (i.e., the capacity of l_k is (i) at least the highest capacity of the projects offered by l_k, and (ii) at most the sum of the capacities of all the projects l_k is offering). We denote by $\mathcal{L}_k^j$, the *projected preference list* of lecturer l_k for p_j, which can be obtained from $\mathcal{L}_k$ by removing those students that do not find p_j acceptable (thereby retaining the order of the remaining students from $\mathcal{L}_k$).

An *assignment* M is a subset of $\mathcal{S} \times \mathcal{P}$ such that $(s_i, p_j) \in M$ implies that s_i finds p_j acceptable. If $(s_i, p_j) \in M$, we say that s_i *is assigned to* p_j, and p_j *is assigned* s_i. For convenience, if s_i is assigned in M to p_j, where p_j is offered by l_k, we may also say that s_i is assigned to l_k, and l_k is assigned s_i.

For any student $s_i \in \mathcal{S}$, we let $M(s_i)$ denote the set of projects assigned to s_i in M. For any project $p_j \in \mathcal{P}$, we denote by $M(p_j)$ the set of students assigned to p_j in M. Project p_j is *undersubscribed*, *full* or *oversubscribed* according as $|M(p_j)|$ is less than, equal to, or greater than c_j, respectively. Similarly, for any lecturer $l_k \in \mathcal{L}$, we denote by $M(l_k)$ the set of students assigned to l_k in M. Lecturer l_k is *undersubscribed*, *full* or *oversubscribed* according as $|M(l_k)|$ is less than, equal to, or greater than d_k, respectively.

A *matching* M is an assignment such that each student is assigned to at most one project in M, each project is assigned at most c_j students in M, and each lecturer is assigned at most d_k students in M. If s_i is assigned to some project in M, for convenience we let $M(s_i)$ denote that project.

In what follows, l_k is the lecturer who offers project p_j.

Definition 1 (stability). Let $\mathtt{I}'$ be an instance of SPA-S, and let M be a matching in $\mathtt{I}'$. We say M is *stable* if it admits no blocking pair, where a *blocking pair* is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that (a) s_i is either unassigned in M or prefers p_j to $M(s_i)$, and (b) either

(i) p_j is undersubscribed and l_k is undersubscribed, or
(ii) 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 $M(l_k)$, or
(iii) p_j is full and l_k prefers s_i to the worst student in $M(p_j)$.

For a full description of an algorithm to find a stable matching in this setting, we refer the interested reader to [1,16].

2.2 Ties in the preference lists

We now define formally the generalisation of SPA-S in which preference lists can include ties. In the preference list of lecturer $l_k \in \mathcal{L}$, a set T of r students forms a *tie of length* r if, for any $s_i, s_{i'} \in T$, l_k does not prefer s_i to $s_{i'}$ (i.e., l_k is *indifferent* between s_i and $s_{i'}$). A tie in a student's preference list is defined similarly. For convenience, in what follows we consider a non-tied entry in a preference list as a tie of length one. We denote by SPA-ST the generalisation of SPA-S in which the preference list of each student (respectively lecturer) comprises a strict ranking of ties, each comprising one or more projects (respectively students).

An example SPA-ST instance $\mathtt{I}_1$ is given in Fig. 1, which involves the set of students $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5\}$, the set of projects $\mathcal{P} = \{p_1, p_2, p_3\}$ and the set of lecturers $\mathcal{L} = \{l_1, l_2\}$. Ties in the preference lists are indicated by round brackets.

Student preferences	Lecturer preferences	
s_1: p_1	l_1: s_5 $(s_1$ $s_2)$ s_3 s_4	l_1 offers p_1, p_2
s_2: $(p_1$ $p_3)$	l_2: s_4 s_5 s_2	l_2 offers p_3
s_3: p_2		
s_4: p_2 p_3	Project capacities: $c_1 = c_3 = 1,\ c_2 = 2$	
s_5: p_3 p_1	Lecturer capacities: $d_1 = 2,\ d_2 = 1$	

Fig. 1. An example instance $\mathtt{I}_1$ of SPA-ST.

In the context of SPA-ST, we assume that all notation and terminology carries over from Sect. 2.1 as defined for SPA-S with the exception of stability, which we now define. When ties appear in the preference lists, three levels of stability arise (as in the HRT context [10,11]), namely *weak stability, strong stability and super-stability*. The formal definition for weak stability in SPA-ST follows from the definition for stability in SPA-S (see Definition 1). Moreover, the existence of a weakly stable matching in an instance $\mathtt{I}$ of SPA-ST is guaranteed by breaking the ties in $\mathtt{I}$ arbitrarily, thus giving rise to an instance $\mathtt{I}'$ of SPA-S. Clearly, a stable matching in $\mathtt{I}'$ is weakly stable in $\mathtt{I}$. Indeed a converse of sorts holds, which gives rise to the following proposition (see [20] for proof).

Proposition 2. *Let* I *be an instance of* SPA-ST*, and let* M *be a matching in* I*. Then* M *is weakly stable in* I *if and only if* M *is stable in some instance* I′ *of* SPA-S *obtained from* I *by breaking the ties in some way.*

As mentioned earlier, super-stability is the most robust concept to seek in a practical setting. Only if no super-stable matching exists in the underlying problem instance should other forms of stability be sought.

Definition 2 (super-stability). Let I be an instance of SPA-ST, and let M be a matching in I. We say M is *super-stable* if it admits no blocking pair, where a *blocking pair* is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that (a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them; and (b) either

(i) p_j is undersubscribed and l_k is undersubscribed, or
(ii) 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 $M(l_k)$ or is indifferent between them, or
(iii) p_j is full and l_k prefers s_i to the worst student in $M(p_j)$ or is indifferent between them.

It may be verified that the matching $M = \{(s_3, p_2), (s_4, p_3), (s_5, p_1)\}$ is super-stable in the SPA-ST instance shown in Fig. 1. Clearly, a super-stable matching is weakly stable.

3 An algorithm for SPA-ST under super-stability

In this section we present our algorithm for SPA-ST under super-stability, which we will refer to as `Algorithm SPA-ST-super`. First, we note that our algorithm is a non-trivial extension of Algorithm SPA-student for SPA-S from [1] and Algorithm HRT-Super-Res for HRT from [10]. Due to the more general setting of SPA-ST, `Algorithm SPA-ST-super` requires some new ideas, and the proofs of the correctness results are more complex than for the aforementioned algorithms for SPA-S and HRT. In Sect. 3.1, we give a description of our algorithm, before presenting it in pseudocode form. We present the algorithm's correctness results in Sect. 3.2.

3.1 Description of the algorithm

First, we present some definitions relating to the algorithm. In what follows, I is an instance of SPA-ST, (s_i, p_j) is an acceptable pair in I and l_k is the lecturer who offers p_j. Further, if (s_i, p_j) belongs to some super-stable matching in I, we call (s_i, p_j) a *super-stable pair* and s_i a *super-stable partner* of p_j (and vice-versa).

During the execution of the algorithm, students become *provisionally assigned* to projects. It is possible for a project to be provisionally assigned a number of students that exceed its capacity. This holds analogously for a lecturer. The algorithm proceeds by deleting from the preference lists certain (s_i, p_j) pairs that cannot be super-stable. By the term *delete* (s_i, p_j), we mean the removal of

p_j from s_i's preference list and the removal of s_i from $\mathcal{L}_k^j$ (the projected preference list of lecturer l_k for p_j). In addition, if s_i is provisionally assigned to p_j at this point, we break the assignment. By the *head* of a student's preference list at a given point, we mean the set of one or more projects, tied in her preference list after any deletions might have occurred, that she prefers to all other projects in her list.

For project p_j, we define the *tail* of $\mathcal{L}_k^j$ as the least-preferred tie in $\mathcal{L}_k^j$ after any deletions might have occurred (recalling that a tie can be of length one). In the same fashion, we define the *tail* of $\mathcal{L}_k$ (preference list of lecturer l_k) as the least-preferred tie in $\mathcal{L}_k$ after any deletions might have occurred. If s_i is provisionally assigned to p_j, we define the *successors* of s_i in $\mathcal{L}_k^j$ as those students that are worse than s_i in $\mathcal{L}_k^j$. An analogous definition holds for the successors of s_i in $\mathcal{L}_k$.

We now describe our algorithm. `Algorithm` SPA-ST-`super` begins by initialising an empty set M which will contain the provisional assignments of students to projects (and intuitively to lecturers). We remark that such assignments can subsequently be broken during the algorithm's execution. Also, each project is initially assigned to be empty (i.e., not assigned to any student).

The `while` loop of the algorithm involves each student s_i who is not provisionally assigned to any project in M and who has a non-empty list applying in turn to each project p_j at the head of her list. Immediately, s_i becomes provisionally assigned to p_j in M (and to l_k). If, by gaining a new student, p_j becomes oversubscribed, it turns out that none of the students s_t at the tail of $\mathcal{L}_k^j$ can be assigned to p_j in any super-stable matching – such pairs (s_t, p_j) are deleted. Similarly, if by gaining a new student, l_k becomes oversubscribed, none of the students s_t at the tail of $\mathcal{L}_k$ can be assigned to any project offered by l_k in any super-stable matching – such pairs (s_t, p_u), for each project $p_u \in P_k$ that s_t finds acceptable, are deleted.

Regardless of whether any deletions occurred as a result of the two conditionals described in the previous paragraph, we have two further (possibly non-disjoint) cases in which deletions may occur. If p_j becomes full, we let s_r be any worst student provisionally assigned to p_j (according to $\mathcal{L}_k^j$), and we delete (s_t, p_j) for each successor s_t of s_r in $\mathcal{L}_k^j$. Similarly if l_k becomes full, we let s_r be any worst student provisionally assigned to l_k, and we delete (s_t, p_u), for each successor s_t of s_r in $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable. As we will prove later, none of the (student, project) pairs deleted when a project or a lecturer becomes full can be a super-stable pair.

At the point where the `while` loop terminates (i.e., when every student is provisionally assigned to one or more projects or has an empty list), if some project p_j that was previously full ends up undersubscribed, we let s_r be any one of the most preferred students (according to $\mathcal{L}_k^j$) who was provisionally assigned to p_j during some iteration of the algorithm but is not assigned to p_j at this point (for convenience, we henceforth refer to such s_r as the most preferred student rejected from p_j according to $\mathcal{L}_k^j$). If the students at the tail of $\mathcal{L}_k$ (recalling that the tail of $\mathcal{L}_k$ is the least-preferred tie in $\mathcal{L}_k$ after any deletions might have occurred) are no better than s_r, it turns out that none of these students s_t can

be assigned to any project offered by l_k in any super-stable matching – such pairs (s_t, p_u), for each project $p_u \in P_k$ that s_t finds acceptable, are deleted. The `while` loop is then potentially reactivated, and the entire process continues until every student is provisionally assigned to a project or has an empty list.

At the termination of the `repeat-until` loop, if the set M, containing the provisional assignments of students to projects, is super-stable relative to the given instance `I` then M is output as a super-stable matching in `I`. Otherwise, the algorithm reports that no super-stable matching exists in `I`. We present `Algorithm` SPA-ST-`super` in pseudocode form in Algorithm 1.

3.2 Correctness of `Algorithm` SPA-ST-`super`

We now present the following results regarding the correctness of `Algorithm` SPA-ST-`super`. For several lemmas in this section, we either omit the proof or provide a sketch proof; see [20] for the full proofs. The first of these results deals with the fact that no super-stable pair is ever deleted during the execution of the algorithm. In what follows, `I` is an instance of SPA-ST, (s_i, p_j) is an acceptable pair in `I` and l_k is the lecturer who offers p_j.

Lemma 1. *If a pair (s_i, p_j) is deleted during the execution of `Algorithm` SPA-ST-`super`, then (s_i, p_j) does not belong to any super-stable matching in `I`.*

Proof (Sketch). Suppose (s_i, p_j) is the first super-stable pair to be deleted during some execution of the algorithm, which belongs to some super-stable matching, say M^*. Let l_k be the lecturer who offers p_j. We consider five points in the algorithm at which (s_i, p_j) could be deleted. If (s_i, p_j) is deleted because s_i is in the tail of $\mathcal{L}_k^j$ when p_j became oversubscribed, we show that one of the students provisionally assigned to p_j at this point must form a blocking pair for M^* with p_j, a contradiction. Similarly, if (s_i, p_j) is deleted because s_i is in the tail of $\mathcal{L}_k$ when l_k became oversubscribed, we show that there is some project $p_{j'} \in P_k$ and some student s_r, with s_r provisionally assigned to $p_{j'}$ in M^* at this point, such that $(s_r, p_{j'})$ must form a blocking pair for M^*, a contradiction. Further, if (s_i, p_j) is deleted because s_i is a successor of a worst student provisionally assigned to p_j when p_j became full, we show that one of the students provisionally assigned to p_j at this point must form a blocking pair for M^* with p_j, a contradiction. Similarly, if (s_i, p_j) is deleted because s_i is a successor of a worst student provisionally assigned to l_k when l_k became full, we show that there is some project $p_{j'} \in P_k$ and some student s_r, with s_r provisionally assigned to $p_{j'}$ in M^* at this point, such that $(s_r, p_{j'})$ must form a blocking pair for M^*, a contradiction. Finally, suppose (s_i, p_j) is deleted at line 33 of the algorithm, suppose $p_{j'}$ is a project offered by l_k which was previously full but ended up undersubcribed in line 27 of the algorithm, and suppose s_r is the most preferred student rejected from $p_{j'}$ according to $\mathcal{L}_k^{j'}$. We show that, in order to avoid the pair $(s_{i'}, p_{j'})$ from blocking M^*, we can construct an infinite sequence of distinct students, a contradiction to the finite size of the instance. □

Algorithm 1 `Algorithm` SPA-ST-super

Input: SPA-ST instance `I`
Output: a super-stable matching M in `I` or "no super-stable matching exists in `I`"

$M \leftarrow \emptyset$
for each $p_j \in \mathcal{P}$ **do**
    `full`(p_j) = `false`
repeat
    while some student s_i is unassigned and has a non-empty list **do**
        for each project p_j at the head of s_i's list **do**
            $l_k \leftarrow$ lecturer who offers p_j
            /* s_i applies to p_j */
            $M \leftarrow M \cup \{(s_i, p_j)\}$ / *provisionally assign s_i to p_j (and to l_k) */
            if p_j is oversubscribed **then**
                for each student s_t at the tail of $\mathcal{L}_k^j$ **do**
                    delete (s_t, p_j)
            else if l_k is oversubscribed **then**
                for each student s_t at the tail of $\mathcal{L}_k$ **do**
                    for each project $p_u \in P_k \cap A_t$ **do**
                        delete (s_t, p_u)
            if p_j is full **then**
                `full`(p_j) = `true`
                $s_r \leftarrow$ worst student assigned to p_j according to $\mathcal{L}_k^j$ {any if > 1}
                for each successor s_t of s_r on $\mathcal{L}_k^j$ **do**
                    delete (s_t, p_j)
            if l_k is full **then**
                $s_r \leftarrow$ worst student assigned to l_k according to $\mathcal{L}_k$ {any if > 1}
                for each successor s_t of s_r on $\mathcal{L}_k$ **do**
                    for each project $p_u \in P_k \cap A_t$ **do**
                        delete (s_t, p_u)
    if some project p_j is undersubscribed and `full`(p_j) is `true` **then**
        $l_k \leftarrow$ lecturer who offers p_j
        $s_r \leftarrow$ most preferred student rejected from p_j according to $\mathcal{L}_k^j$ {any if > 1}
        if the students at the tail of $\mathcal{L}_k$ are no better than s_r **then**
            for each student s_t at the tail of $\mathcal{L}_k$ **do**
                for each project $p_u \in P_k \cap A_t$ **do**
                    delete (s_t, p_u)
until every unassigned student has an empty list
if M is a super-stable matching in `I` **then**
    return M
else
    return "no super-stable matching exists in `I`"

The next three lemmas deal with the case that Algorithm SPA-ST-super reports the non-existence of a super-stable matching in I.

Lemma 2. *If any student is multiply assigned at the termination of Algorithm SPA-ST-super, then I admits no super-stable matching.*

Lemma 3. *If some lecturer l_k becomes full during some execution of Algorithm SPA-ST-super and l_k subsequently ends up undersubscribed at the termination of the algorithm, then I admits no super-stable matching.*

Lemma 4. *If the pair (s_i, p_j) was deleted during some execution of Algorithm SPA-ST-super, and at the termination of the algorithm s_i is not assigned to a project better than p_j, and each of p_j and l_k is undersubscribed, then I admits no super-stable matching.*

The next lemma shows that the final assignment may be used to determine the existence, or otherwise, of a super-stable matching in I.

Lemma 5. *If at the termination of Algorithm SPA-ST-super, the assignment M is not super-stable in I, then no super-stable matching exists in I.*

Proof. Suppose M is not super-stable in I. If some student s_i is multiply assigned in M, then I admits no super-stable matching, by Lemma 2. Hence suppose no student is multiply assigned in M. Then M is a matching, since no project or lecturer is oversubscribed in M. Let (s_i, p_j) be a blocking pair of M in I, then s_i is either unassigned in M or prefers p_j to $M(s_i)$ or is indifferent between them. Whichever of these is the case, (s_i, p_j) has been deleted and therefore does not belong to any super-stable matching, by Lemma 1. Let l_k be the lecturer who offers p_j. If (s_i, p_j) was deleted as a result of l_k being full or oversubscribed, (s_i, p_j) could only block M if l_k ends up undersubscribed in M. If this is the case, then I admits no super-stable matching, by Lemma 3.

Now suppose (s_i, p_j) was deleted as a result of p_j being full or oversubscribed. Suppose firstly that p_j ends up full in M. Then (s_i, p_j) cannot block M irrespective of whether l_k is undersubscribed or full in M, since l_k prefers the worst assigned student(s) in $M(p_j)$ to s_i. Hence suppose p_j ended up undersubscribed in M. As p_j was previously full, each pair (s_t, p_u), for each s_t that is no better than s_i at the tail of $\mathcal{L}_k$ and each $p_u \in P_k \cap A_t$, would have been deleted in line 33 of the algorithm. Thus if l_k is full in M, then (s_i, p_j) does not block M. Suppose l_k is undersubscribed in M. If l_k was full at some point during the execution of the algorithm, then I admits no super-stable matching, by Lemma 3. Suppose l_k was never full during the algorithm's execution. As (s_i, p_j) is a blocking pair of M, s_i cannot be assigned in M a project that she prefers to p_j. Hence I admits no super-stable matching, by Lemma 4.

Finally suppose (s_i, p_j) was deleted (at line 33) because some other project $p_{j'}$ offered by l_k was previously full and ended up undersubscribed at line 27. Then l_k must have identified her most preferred student, say s_r, rejected from

$p_{j'}$ such that s_i is at the tail of $\mathcal{L}_k$ and s_i is no better than s_r in $\mathcal{L}_k$. Moreover, every project offered by l_k that s_i finds acceptable would have been deleted from s_i's preference list at the for loop iteration in line 33. If p_j ends up full in M, then (s_i, p_j) does not block M. Suppose p_j ends up undersubscribed in M. If l_k is full in M, then (s_i, p_j) does not block M, since $s_i \notin M(l_k)$ and l_k prefers the worst student in $M(l_k)$ to s_i. Suppose l_k is undersubscribed in M, again by Lemma 4, I admits no super-stable matching. □

The next lemma shows that `Algorithm SPA-ST-super` may be implemented to run in linear time.

Lemma 6. *`Algorithm SPA-ST-super` may be implemented to run in $O(L)$ time and $O(n_1 n_2)$ space, where n_1, n_2, and L are the number of students, number of projects, and the total length of the preference lists, respectively, in `I`.*

Lemma 1 shows that there is an optimality property for each assigned student in any super-stable matching found by the algorithm, whilst Lemma 5 establishes the correctness of `Algorithm SPA-ST-super`. The following theorem collects together Lemmas 1, 5 and 6.

Theorem 1. *For a given instance `I` of SPA-ST, `Algorithm SPA-ST-super` determines, in $O(L)$ time and $O(n_1 n_2)$ space, whether or not a super-stable matching exists in `I`. If such a matching does exist, all possible executions of the algorithm find one in which each assigned student is assigned to the best project that she could obtain in any super-stable matching, and each unassigned student is unassigned in all super-stable matchings.*

3.3 Properties of super-stable matchings in SPA-ST

The following theorem, which we will refer to as the *Unpopular Projects Theorem*, gives some properties of super-stable matchings in an instance of SPA-ST that are analogous to those satisfied by stable matchings in SPA-S [1, Theorem 4.1].

Theorem 2. *For a given instance `I` of SPA-ST, the following holds.*

1. *Each lecturer is assigned the same number of students in all super-stable matchings.*
2. *Exactly the same students are unassigned in all super-stable matchings.*
3. *A project offered by an undersubscribed lecturer has the same number of students in all super-stable matchings.*

4 Empirical Evaluation

In this section, we evaluate an implementation of `Algorithm SPA-ST-super`. We implemented our algorithm in Python[1], and performed our experiments on a system with dual Intel Xeon CPU E5-2640 processors with 64GB of RAM, running Ubuntu 14.04. For our experiment, we were primarily concerned with the following question: how does the nature of the preference lists in a given SPA-ST instance affect the existence of a super-stable matching?

[1] https://github.com/sofiatolaosebikan/spa-st-super-stability

4.1 Datasets

For datasets, there are clearly several parameters that can be varied, such as the number of students, projects and lecturers; the lengths of the students' preference list as well as a measure of the density of ties present in the preference lists. We denote by t_d, the measure of the density of ties present in the preference lists. In each student's preference list, the tie density t_{d_s} ($0 \leq t_{d_s} \leq 1$) is the probability that some project is tied to its successor. The tie density t_{d_l} in each lecturer's preference list is defined similarly. At $t_{d_s} = t_{d_l} = 1$, each preference lists would be contained in a single tie while at $t_{d_s} = t_{d_l} = 0$, no tie would exist in the preference lists, thus reducing the problem to an instance of SPA.

4.2 Experimental Setup

For each range of values for the aforementioned parameters, we randomly-generated a set of SPA-ST instances, involving n_1 students (which we will henceforth refer to as the size of the instance), $0.5n_1$ projects, $0.2n_1$ lecturers and $1.5n_1$ total project capacity which was randomly distributed amongst the projects. The capacity for each lecturer l_k was chosen randomly to lie between the highest capacity of the projects offered by l_k and the sum of the capacities of the projects that l_k offers. In each set, we measured the proportion of instances that admit a super-stable matching. It is worth mentioning that when we varied the tie density on both the students' and lecturers' preference lists between 0.1 and 0.5, super-stable matchings were very elusive, even with an instance size of 100 students. Thus, for the purpose of our experiment, we decided to choose a low tie density.

Experiment 1 In the first experiment, we increased the number of students n_1 while maintaining a constant ratio of projects, lecturers, project capacities and lecturer capacities as described above. For various values of n_1 ($100 \leq n_1 \leq 1000$) in increments of 100, we created 1000 randomly-generated instances. The length of each student's preference list was fixed at 50. For all the preference lists, we set $t_{d_s} = t_{d_l} = 0.005$ (on average, 1 out of 5 students has a single tie of length 2 in their preference list, and this holds similarly for the lecturers). The result displayed in Fig. 2 shows that the proportion of instances that admit a super-stable matching decreases as the number of students increases.

Experiment 2 In our second experiment, we varied the length of each student's preference list while maintaining the number of students, projects, lecturers, project capacities and lecturer capacities, and tie density in the students' and lecturers' preference lists, as in Experiment 1. For various values of n_1 ($100 \leq n_1 \leq 1000$) in increments of 100, we created 1000 randomly-generated instances. In the first part of this experiment, the length of each student's preference list was set to 10, and in the second part, the length of each student's preference list was chosen randomly to lie between $0.25n_1$ and $0.5n_1$. The result is displayed in Fig. 3. Although this result shows that more instances admitted a super-stable matching when the preference list was longer compared to when the preference list was fixed at 10, but this difference is not very significant.

Experiment 3 In our last experiment, we investigated how the variation in tie density in both the students' and lecturers' preference lists affects the existence of a super-stable matching. To achieve this, we varied the tie density in the students' preference lists t_{d_s} ($0 \leq t_d \leq 0.05$) and the tie density in the lecturers' preference lists t_{d_l} ($0 \leq t_d \leq 0.05$), both in increments of 0.005. For each pair of tie densities in $t_{d_s} \times t_{d_l}$ we randomly-generated 1000 SPA-ST instances for various values of n_1 ($100 \leq n_1 \leq 1000$) in increments of 100. For each of these instances, we maintained the same ratio of projects, lecturers, project capacities and lecturer capacities as in Experiment 1. We also fixed the length of each student's preference list at 50. The result displayed in Fig. 4 shows that increasing the tie density in both the students' and lecturers' preference lists reduces the proportion of instances that admit a super-stable matching. In fact, this proportion reduces further as the size of the instance increases. However, it was interesting to see that when we fixed the tie density in the students' preference lists at 0 and varied the tie density in the lecturers' preference lists, about 75% of the randomly-generated SPA-ST instances involving 1000 students admitted a super-stable matching.

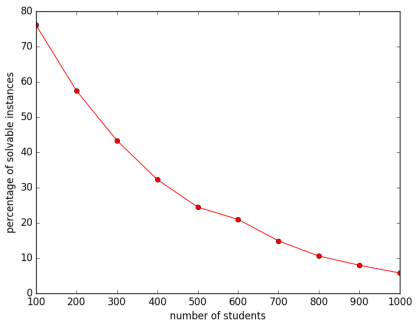

Fig. 2. Result for Experiment 1.

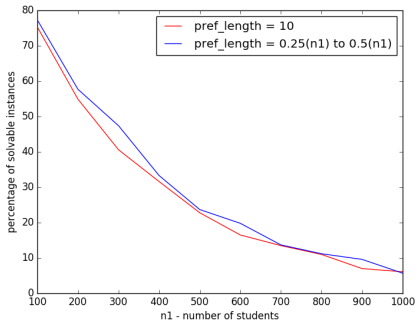

Fig. 3. Result for Experiment 2.

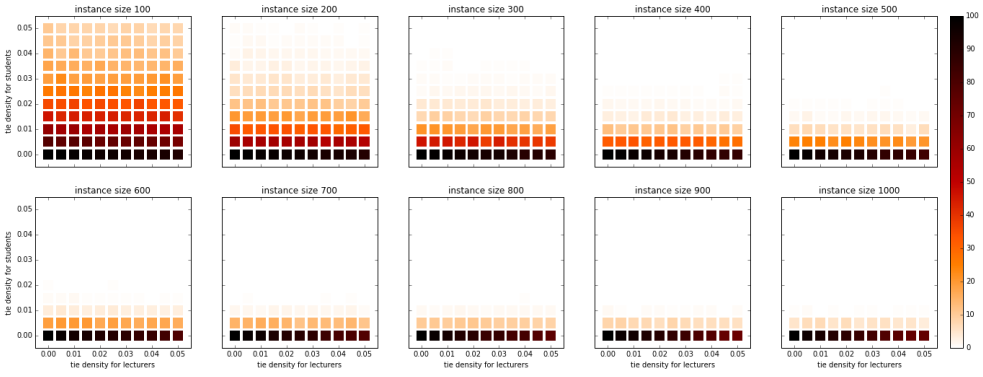

Fig. 4. Each of the coloured square boxes represent the proportion of the 1000 randomly-generated SPA-ST instances that admits a super-stable matching, with respect to the tie density in the students' and lecturers' preference lists. See the colour bar transition, as this proportion ranges from dark (100%) to light (0%).

5 Discussions and Concluding Remarks

Based on the instances we generated randomly, the experimental results suggest that as we increase the size of the instance and the density of ties in the preference lists, the likelihood of a super-stable matching existing decreases. There was no significant uplift in this likelihood even as we increased the lengths of the students' preference lists. Moreover, when the ties occur only in the lecturers' preference lists, we found that a significantly higher proportion of instances admit a super-stable matching. Given that there are typically more students than lecturers in practical applications, it could be that only lecturers are permitted to have some form of indifference over the students that they find acceptable, whilst each student might be able to provide a strict ordering over what may be a small number of projects that she finds acceptable. On the other hand we did not find the same uplift for the case where ties belong to the students' preference lists only.

Further evaluation of our algorithm could investigate how other parameters (e.g., the popularity of some projects, or the position of the ties in the preference lists) affect the existence of a super-stable matching. It would also be interesting to examine the existence of super-stable matchings in real SPA-ST datasets. From a theoretical perspective, an interesting question would be: what is the probability of a super-stable matching existing, given an arbitrary SPA-ST instance? This question has been explored for the Stable Roommates problem (a non-bipartite generalisation of the Stable Marriage problem) [21].

To cope with the possible non-existence of a super-stable matching, a natural strategy would be to seek a strongly stable matching if one exists, and if not, settle for a weakly stable matching. As noted in Section 1, every instance of SPA-ST admits a weakly stable matching. We leave open the problem of constructing an algorithm for SPA-ST under the strong stability criterion.

Acknowledgements

The authors would like to thank Frances Cooper and Kitty Meeks for valuable comments that helped to improve the presentation of this paper.

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(1):79–91, 2007.
2. A.H. Abu El-Atta and M.I. Moussa. Student project allocation with preference lists over (student,project) pairs. In *Proceedings of ICCEE 09: the 2nd International Conference on Computer and Electrical Engineering*, pages 375–379. IEEE, 2009.
3. A.A. Anwar and A.S. Bahaj. Student project allocation using integer programming. *IEEE Transactions on Education*, 46(3):359–367, 2003.
4. G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.

5. 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.
6. M. Chiarandini, R. Fagerberg, and S. Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research*, 2017.
7. F. Cooper and D.F. Manlove. A 3/2-approximation algorithm for the Student-Project Allocation problem. In *Proceedings of SEA 2018*, vol. 103 of *LIPIcs*, pg 8:1–8:13.
8. P.R. Harper, V. de Senna, I.T. Vieira, and A.K. Shahani. A genetic algorithm for the project assignment problem. *Computers and Operations Research*, 32:1255–1265, 2005.
9. R.W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.
10. R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT '00*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2000.
11. R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS '03*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer, 2003.
12. K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Proceedings of ICALP '99*, volume 1644 of *Lecture Notes in Computer Science*, pages 443–452. Springer, 1999.
13. K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation bounds for the student-project allocation problem with preferences over projects. *Journal of Discrete Algorithms*, 13:59–66, 2012.
14. D. Kazakov. Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science. Available from `http://www-users.cs.york.ac.uk/kazakov/papers/proj.pdf` (last accessed 8 March 2018), 2001.
15. 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*, volume 8986 of *LNCS*, pages 213–225. Springer, 2015.
16. D.F. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.
17. 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.
18. D.F. Manlove, D. Milne, and S. Olaosebikan. An Integer Programming Approach to the Student-Project Allocation Problem with Preferences over Projects. Lecture Notes in Computer Science, vol 10856, pages 313-325, Springer, 2018.
19. D.F. Manlove and G. O'Malley. Student project allocation with preferences over projects. *Journal of Discrete Algorithms*, 6:553–560, 2008.
20. S. Olaosebikan and D.F. Manlove. Super-stability in the Student-Project Allocation problem with Ties. Available from `http://arxiv.org/abs/1805.09887`.
21. B.G. Pittel and R.W. Irving. An upper bound for the solvability probability of a random stable roommates instance. *Random Structures and Algorithms*, 5:465–486, 1994.
22. 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.