Journal of Combinatorial Optimization (2022) 43:1203–1239
https://doi.org/10.1007/s10878-020-00632-x

Super-stability in the student-project allocation problem with ties

Sofiat Olaosebikan[1] · **David Manlove**[1]

Published online: 12 August 2020

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 each project is offered by one lecturer and 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 when ties are present in the students' and lecturers' preference lists, the probability of such a matching existing is significantly higher if ties are restricted to the lecturers' preference lists.

Keywords Student-project allocation · Stable matching · Super-stability · Polynomial-time algorithm · Empirical evaluation

A preliminary version of a part of this paper appeared in Olaosebikan and Manlove (2018).
The first author was supported by a College of Science and Engineering Scholarship from the University of Glasgow, and the second author was supported by Grant EP/P028306/1 from the Engineering and Physical Sciences Research Council.

✉ Sofiat Olaosebikan
Sofiat.Olaosebikan@glasgow.ac.uk

David Manlove
David.Manlove@glasgow.ac.uk

[1] School of Computing Science, University of Glasgow, Glasgow, Scotland, UK

1 Introduction

The *Student-Project Allocation problem* (SPA) (Abraham et al. 2007; Chiarandini et al. 2019; Manlove 2013) is a many-one matching problem which involves three sets of entities: students, projects and lecturers. Each project is proposed by one lecturer and each student is required to rank a subset of these projects that she finds acceptable, in order of preference. Further, each lecturer may have preferences over the students that find her projects acceptable and/or the projects that she offers. Typically there may be capacity constraint on the number of students that each project and lecturer can accommodate. The goal is to find a *matching*, i.e., an assignment of students to projects based on the stated preferences such that each student is assigned to at most one project, and the capacity constraints on projects and lecturers are not violated.

Applications of SPA can be found in many university departments, for example, the School of Computing Science, University of Glasgow (Kwanashie et al. 2015), the Faculty of Science, University of Southern Denmark (Chiarandini et al. 2019), the Department of Computing Science, University of York (Kazakov 2001), and elsewhere (Anwar and Bahaj 2003; Calvo-Serrano et al. 2017; Harper et al. 2005). 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) (Abraham et al. 2007; Manlove 2013). This variant falls under the category of bipartite matching problem with two-sided preferences.[1] In this context, it has been argued that a natural property for a matching to satisfy is that of *stability* (Roth 1984, 1990, 1991). Informally, a *stable matching* ensures that no student and lecturer would have an incentive to deviate from the matching by forming a private arrangement involving some project.

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 lecturer preferences over Students with Ties* (SPA- ST).

If we allow ties in the preference lists of students and lecturers, three different stability definitions naturally arise. Suppose M is a matching in an instance of SPA-ST. Informally, we say that 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 would be no worse off,
(iii) neither of them would be worse off.

With respect to this informal definition, a super-stable matching is also strongly stable, and a strongly stable matching is also weakly stable. These concepts were first defined and studied by Irving (1994) in the context of the *Stable Marriage problem*

[1] For further reading on the classification of matching problems, we refer the interested reader to Manlove (2013).

with Ties (SMT), and subsequently extended to the *Hospitals/Residents problem with Ties* (HRT) (Irving et al. 2000, 2003) (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; and SMT is a restriction of HRT where the capacity of each hospital is 1).

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 Abraham et al. (2007) to the resulting SPA- S instance). However, such matchings could be of different sizes (Manlove et al. 2002). 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 (Iwama et al. 1999; Manlove et al. 2002), even for the so-called *Stable Marriage problem with Ties and Incomplete lists* (SMTI), which is an extension of SMT in which the preference lists need not be complete. However, we note that a $\frac{3}{2}$-approximation algorithm was described in Cooper and Manlove (2018a) for the problem of finding a maximum size weakly stable matching, given an instance of SPA- ST.[2]

Although a super-stable matching can be elusive, it avoids the problem of finding a maximum size weakly stable matching, because, as we will show in this paper, analogous to the HRT case (Irving et al. 2000): (i) all super-stable matchings have the same size; (ii) finding one or reporting that none exists can be accomplished in linear-time; 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. Furthermore, Irving et al. (2000) argued 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 (Irving et al. 2000, Proposition 2), which extends naturally to SPA- ST as follows (see Sect. 2.2 for a 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. 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, the capacity of each project is 1, the capacity of the lecturer is 2, and every preference list is a single tie of length 2; any matching will be undermined by some student s_i and the lecturer involving a project that s_i is not assigned to. Nonetheless, it should be clear from the discussions above that a super-stable matching should be preferred in practical applications when one does exist.

[2] This approximation algorithm finds a weakly stable matching that is at least two-thirds the size of a maximum weakly stable matching.

Related work Irving et al. (2000) 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 Irving et al. (2000) to the resulting HRT instance does not work in general (we explain this further in Sect. 2.3). Other variants of SPA in the literature involve lecturer preferences over their proposed projects (Iwama et al. 2012; Manlove et al. 2018; Manlove and O'Malley 2008), lecturer preferences over (student, project) pairs (Abu El-Atta and Moussa 2009), and no lecturer preferences at all (Kwanashie et al. 2015) (see Chiarandini et al. 2019 for a more detailed survey in this latter case). A similar model known as the *Student-Project-Resource Matching-Allocation problem* (SPR) was recently considered in Ismaili et al. (2019). This model is different from SPA- S in the following ways: (i) in SPA- S, the capacity of each project is fixed by the lecturer offering it, while in SPR, the capacity of each project is determined by the resources allocated to it; (ii) in SPA- S, each lecturer has a fixed capacity on the total number of students that can be assigned to her projects, while in SPR, there is no notion of lecturer capacity.

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 Abraham et al. (2007), Manlove (2013). Our algorithm is student-oriented because it involves the students applying to projects. Moreover, the algorithm returns the student-optimal super-stable matching, in the sense that if the given instance admits a super-stable matching then our algorithm will output a solution in which each assigned student has the best project that she could obtain in any super-stable matching that the instance admits. We also present the results of an empirical evaluation based on an implementation of our algorithm 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.[3] Our main finding from the empirical evaluation is that super-stable matchings are very elusive with ties in the students' and lecturers' preference lists. However, if the preference lists of the students are strictly ordered and only the lecturers express ties in their preference lists, the probability of a super-stable matching existing is significantly higher.

The remainder of this paper is 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 the experimental results obtained from our algorithm's empirical evaluation. Finally, Sect. 5 presents some concluding remarks and potential direction for future work.

[3] From a theoretical perspective, the likelihood of a stable matching existing has been explored for the Stable Roommates problem—a non-bipartite generalisation of the Stable Marriage problem (Pittel and Irving 1994).

2 Preliminary definitions and results

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, which forms s_i's preference list. We say that s_i finds p_j *acceptable* if p_j is in s_i's preference list, and we denote by A_i the set of projects that s_i finds acceptable. 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 $\mathcal{P}$. Also, l_k ranks in strict order of preference those students who find at least one project in P_k acceptable, which forms l_k's preference list. We say that l_k finds s_i *acceptable* if s_i is in l_k's preference list, and we denote by $\mathcal{L}_k$ the set of students that l_k finds acceptable.

For any pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$, where p_j is offered by l_k, we refer to (s_i, p_j) as an *acceptable pair* if s_i and l_k both find each other acceptable, i.e., if $p_j \in A_i$ and $s_i \in \mathcal{L}_k$. Each project $p_j \in \mathcal{P}$ has a capacity $c_j \in \mathbb{Z}^+$ indicating the maximum number of students that can be assigned to p_j. Similarly, each lecturer $l_k \in \mathcal{L}$ has a capacity $d_k \in \mathbb{Z}^+$ indicating the maximum number of students that l_k is willing to supervise. 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 that are assigned to s_i in M. For any project $p_j \in \mathcal{P}$, we denote by $M(p_j)$ the set of students that are assigned to p_j in M. Project p_j is *undersubscribed*, *full* or *oversubscribed* in M 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 that are assigned to l_k in M. Lecturer l_k is *undersubscribed*, *full* or *oversubscribed* in M 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 (i.e., $|M(s_i)| \leq 1$ for each $s_i \in \mathcal{S}$, $|M(p_j)| \leq c_j$ for each $p_j \in \mathcal{P}$, and $|M(l_k)| \leq d_k$ for each $l_k \in \mathcal{L}$). 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 I be an instance of SPA- ST, and let M be a matching in I. We say that M is *stable* if it admits no blocking pair, where a *blocking pair* is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$ such that (a) and (b) holds as follows:

(a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$;
(b) either (i), (ii) or (iii) holds as follows:

(i) each of p_j and l_k is undersubscribed in M;
(ii) p_j is undersubscribed in M, l_k is full in M and either
  (1) $s_i \in M(l_k)$, or
  (2) l_k prefers s_i to the worst student in $M(l_k)$;
(iii) p_j is full in M and l_k prefers s_i to the worst student in $M(p_j)$.

To find a stable matching in an instance of SPA- S, two linear-time algorithms were described in Abraham et al. (2007). The stable matching produced by the first algorithm is *student-optimal* (i.e., each assigned student has the best-possible project that she could obtain in any stable matching) while the one produced by the second algorithm is *lecturer-optimal* (i.e., each lecturer has the best set of students that she could obtain in any stable matching). The set of stable matchings in a given instance of SPA- S satisfy several interesting properties that together form what we will call the *Unpopular Projects Theorem* [analogous to the Rural Hospitals Theorem for HR Irving et al. 2000], which we state as follows.

Theorem 1 (Abraham et al. 2007) *For a given instance of* SPA- S*, the following holds:*

1. *each lecturer is assigned the same number of students in all stable matchings;*
2. *exactly the same students are unassigned in all stable matchings;*
3. *a project offered by an undersubscribed lecturer is assigned the same number of students in all stable matchings.*

As we will see later in this paper, when ties are present in the preference lists of students and lecturers, the set of super-stable matchings also satisfy each of the properties in Theorem 1.

2.2 Ties in the preference lists

We now define formally the generalisation of SPA- S in which the 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 l_k does not prefer s_i to $s_{i'}$ for any $s_i, s_{i'} \in T$ (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, henceforth, 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 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 $\mathcal{L} = \{l_1, l_2\}$, with $P_1 = \{p_1, p_2\}$ and $P_2 = \{p_3\}$. Ties in the preference lists are indicated by round brackets.

Students' preferences	Lecturers' 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 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 Irving et al. 2000, 2003), 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 I of SPA- ST is guaranteed by breaking the ties in I arbitrarily, thus giving rise to an instance I' of SPA- S. Clearly, a stable matching in I' is weakly stable in I. Indeed a converse of sorts holds, which gives rise to the following proposition.

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.*

Proof Let I be an instance of SPA- ST and let M be a matching in I. Suppose that M is weakly stable in I. Let I' be an instance of SPA- S obtained from I by breaking the ties in the following way. For each student s_i in I such that the preference list of s_i includes a tie T containing two or more projects, we order the preference list of s_i in I' as follows: if s_i is assigned in M to a project p_j in T then s_i prefers p_j to every other project in T; otherwise, we order the projects in T arbitrarily. For each lecturer l_k in I such that l_k's preference list includes a tie X, if X contains students that are assigned to l_k in M and students that are not assigned to l_k in M then l_k's preference list in I' is ordered in such a way that each $s_i \in X \cap M(l_k)$ is preferred to each $s_{i'} \in X \backslash M(l_k)$; otherwise, we order the students in X arbitrarily. Now, suppose (s_i, p_j) forms a blocking pair for M in I'. Given how the ties in I were removed to obtain I', this implies that (s_i, p_j) forms a blocking pair for M in I, a contradiction to our assumption that M is weakly stable in I. Thus M is stable in I'.

Conversely, suppose M is stable in some instance I' of SPA- S obtained from I by breaking the ties in some way. Now suppose that M is not weakly stable in I. Then some pair (s_i, p_j) forms a blocking pair for M in I. It is then clear from the definition of weak stability and from the construction of I' that (s_i, p_j) is a blocking pair for M in I', a contradiction. □

As mentioned earlier, super-stability is the most robust concept to seek. Only if no super-stable matching exists in the underlying problem instance should other forms of stability be sought in a practical setting. Thus, for the remainder of this paper, we focus on super-stability in the SPA- ST context.

Definition 2 (Super-stability) Let I be an instance of SPA- ST, and let M be a matching in I. We say that M is *super-stable* if it admits no blocking pair, where a *blocking pair* is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$ such that (a) and (b) holds as follows:

(a) either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them;
(b) either (i), (ii), or (iii) holds as follows:
   (i) each of p_j and l_k is undersubscribed in M;
   (ii) p_j is undersubscribed in M, l_k is full in M and either
      (1) $s_i \in M(l_k)$, or
      (2) l_k prefers s_i to the worst student/s in $M(l_k)$ or is indifferent between them;
   (iii) p_j is full in M and l_k prefers s_i to the worst student/s 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 Fig. 1. Clearly, a super-stable matching is also weakly stable. Moreover, the super-stability definition gives rise to Proposition 1, which can be regarded as an analogue of Proposition 2 for super-stability, restated as follows.

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.*

Proof Let I be an instance of SPA- ST and let M be a matching in I. Suppose that M is super-stable in I. We want to show that M is stable in every instance of SPA- S obtained from I by breaking the ties in some way. Now, let I' be an arbitrary instance of SPA- S obtained from I by breaking the ties in some way, and suppose M is not stable in I'. This implies that M admits a blocking pair (s_i, p_j) in I'. Since I' is an arbitrary SPA- S instance obtained from I by breaking the ties in some way, it follows that in I: (i) if s_i is assigned in M then s_i either prefers p_j to $M(s_i)$ or is indifferent between them, (ii) if p_j is full in M then l_k either prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, and (iii) if l_k is full in M then either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them. This implies that (s_i, p_j) forms a blocking pair for M in I, a contradiction to the super-stability of M.

Conversely, suppose M is stable in every instance of SPA- S obtained from I by breaking the ties in some way. Now suppose M is not super-stable in I. This implies that M admits a blocking pair (s_i, p_j) in I. We construct an instance I' of SPA- S from I by breaking the ties in the following way: (i) if s_i is assigned in M and s_i is indifferent between p_j and $M(s_i)$ in I then s_i prefers p_j to $M(s_i)$ in I'; otherwise we break the ties in s_i's preference list arbitrarily, and (ii) if some student, say $s_{i'}$, different from s_i is assigned to l_k in M such that l_k is indifferent between s_i and $s_{i'}$ in I then l_k prefers s_i to $s_{i'}$ in I'; otherwise we break the ties in l_k's preference list arbitrarily. Thus (s_i, p_j) forms a blocking pair for M in I', i.e., M is not stable in I', a contradiction to the fact that M is stable in every instance of SPA- S obtained from I by breaking the ties in some way. □

The following proposition, which is a consequence of Propositions 1 and 2 , and Theorem 1, tells us that if a super-stable matching M exists in I then all weakly stable

matchings in I are of the same size (equal to the size of M) and match exactly the same set of students.

Proposition 3 *Let I be an instance of* SPA- ST*, and suppose that I admits a super-stable matching M. Then the Unpopular Projects Theorem holds for the set of weakly stable matchings in I.*

Proof Let I be an instance of SPA- ST. Let M be a super-stable matching in I and let M' be a weakly stable matching in I. Then by Proposition 2, M' is stable in some instance I' of SPA- S obtained from I by breaking the ties in some way. Also M is stable in I' by Proposition 1. By Theorem 1, each lecturer is assigned the same number of students in M and M', exactly the same students are unassigned in M and M', and a project offered by an undersubscribed lecturer is assigned the same number of students in M and M'. Hence, the Unpopular Projects Theorem holds for the set of weakly stable matchings in I. □

2.3 Cloning from SPA- ST to HRT does not work in general

As mentioned earlier, Irving et al. (2000) described a polynomial-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of HRT. The authors referred to their algorithm as Algorithm HRT-Super-Res. One might assume that reducing a given instance of SPA- ST to an instance of HRT (using a "cloning" technique) and subsequently applying Algorithm HRT-Super-Res to the resulting instance would solve our problem. However, this is not always true. In what follows, we describe an obvious method to clone an instance of SPA- ST to an instance of HRT, and we show that applying the super-stable matching algorithm described in Irving et al. (2000) to the resulting HRT instance does not work in general.

A method to derive an instance I' of HRT from an instance I of SPA- ST was described by Cooper and Manlove (2018b). We explain this method as follows. The students and projects involved in I are converted into residents and hospitals respectively in I', i.e., each $s_i \in \mathcal{S}$ becomes r_i in the cloned instance, and each $p_j \in \mathcal{P}$ becomes h_j. Residents inherit their preference lists naturally from students, i.e., if r_i corresponds to s_i then the preference list of r_i in I' is A_i, with each project in A_i being replaced by the associated hospital. Hospitals inherit their preference lists from the projected preference list of the associated project according to the lecturer offering the project, i.e., if p_j corresponds to h_j (where p_j is offered by l_k) then the preference list of h_j in I' is $\mathcal{L}_k^j$, with each student in $\mathcal{L}_k^j$ being replaced by the associated resident. Each hospital also inherits its capacity from the project, i.e., for each h_j associated with p_j, the capacity of h_j is c_j.

Let l_k be an arbitrary lecturer in I. In order to translate l_k's capacity into the HRT instance, we create n *dummy residents*[4] for each hospital h_j corresponding to a project $p_j \in P_k$, where n is the difference between the sum of the capacities of all the projects in P_k and the capacity of l_k (recall that $\sum_{p_j \in P_k} c_j \geq d_k$). The preference list for each of these dummy residents will be a single tie consisting of all the hospitals corresponding

[4] The dummy residents created for each hospital will offset the difference between the corresponding lecturer capacity and the total capacity of her proposed projects.

Students' preferences	Lecturers' preferences	
s_1: p_1	l_1: s_1 (s_2 s_3)	l_1 offers p_1, p_2
s_2: (p_1 p_2)	l_2: s_3	l_2 offers p_3
s_3: p_2 p_3		
	Project capacities: $c_1 = c_2 = c_3 = 1$	
	Lecturer capacities: $d_1 = d_2 = 1$	

Fig. 2 An instance I of SPA- ST

Residents' preferences	Hospitals' preferences
r_1: h_1	h_1: r_{d_1} r_1 r_2
r_2: (h_1 h_2)	h_2: r_{d_1} (r_2 r_3)
r_3: h_2 h_3	h_3: r_3
r_{d_1}: (h_1 h_2)	
	Hospital capacities: $c_1 = c_2 = c_3 = 1$

Fig. 3 An instance I' of HRT cloned from the SPA- ST instance illustrated in Fig. 2

to a project in P_k. Further, the preference list for each hospital corresponding to a project in P_k will include a tie in its first position consisting of all the dummy residents associated with l_k.

Next, we describe how to map between matchings in I and in I'. Let M and M' be a matching in I and I' respectively. Let r_i be the resident associated with s_i and let h_j be the hospital associated with p_j. If s_i is assigned in M to project p_j, then r_i is assigned in M' to hospital h_j. To illustrate the cloning technique described above, we give an example instance I of SPA- ST in Fig. 2 as well as the corresponding cloned HRT instance I' in Fig. 3. Also, we give an intuition as to why this technique will not work in general.

With respect to Figs. 2 and 3 , each resident r_1, r_2 and r_3 in I' corresponds to student s_1, s_2 and s_3 in I, respectively; and the preference list of each resident is adapted from the preference list of the associated student. Also, each hospital h_1, h_2 and h_3 in I' corresponds to project p_1, p_2 and p_3 in I, respectively. The preference list of hospitals h_1 and h_2 is $\mathcal{L}_1^1$ and $\mathcal{L}_1^2$ respectively, since l_1 is the lecturer that offers both p_1 and p_2. Similarly, the preference list of hospital h_3 is $\mathcal{L}_2^3$, since l_2 is the lecturer that offers p_3. Further, for lecturer l_1 who offers both p_1 and p_2, since $c_1 + c_2 = 2 > 1 = d_1$, we add one dummy resident r_{d_1} to the cloned instance. The preference list of r_{d_1} is a single tie consisting of h_1 and h_2; and the preference list of both h_1 and h_2 includes r_{d_1} in first position.

The reader can easily verify that matching $M = \{(s_1, p_1), (s_3, p_3)\}$ is super-stable in the SPA- ST instance I illustrated in Fig. 2. Now, following our description of how to map between matchings in I and in I', a matching in I' is $M' = \{(r_{d_1}, h_2), (r_1, h_1), (r_3, h_3)\}$, with $(s_1, p_1) \in M$ corresponding to $(r_1, h_1) \in M'$ and $(s_3, p_3) \in M$ corresponding to $(r_3, h_3) \in M'$. Clearly, M' is not super-stable in I' as (r_{d_1}, h_1) forms a blocking pair. In fact, the HRT instance I' admits no super-stable matching. The justification for this is as follows: irrespective of the hospital that the dummy resident r_{d_1} is assigned to in any matching obtained from I', r_{d_1} will block this matching via the other hospital tied in her preference list (since the hospital would be better off taking on r_{d_1}, and r_{d_1} would be no worse off).

One way to avoid this problem would be to strictly order the hospitals in r_{d_1}'s preference list; however, the order in which the hospitals appear will lead to different possibilities. For instance: if r_{d_1} prefers h_1 to h_2, the reader can verify that the corresponding HRT instance admits no super-stable matching; however, if r_{d_1} prefers h_2 to h_1, again the reader can verify that the corresponding HRT instance admits the super-stable matching $\{(r_{d_1}, h_2), (r_1, h_1), (r_3, h_3)\}$. The downside of this strategy is that there is no obvious reason as to why r_{d_1} should prefer h_2 to h_1 in the cloned HRT instance in Fig. 3 by merely looking at the original SPA- ST instance in Fig. 2. Hence, in order to make this technique work in general, we will need to generate every HRT instance obtained by ordering the dummy residents' preference lists in some way. This is exponential in the problem instance.

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. Before we proceed, we briefly describe Algorithm HRT-Super-Res (Irving et al. 2000). The algorithm involves a sequence of proposals from the residents to the hospitals. Each resident proposes in turn to all of the hospitals tied together at the head of her preference list, and all proposals are provisionally accepted. If a hospital h becomes oversubscribed then none of h's worst assignees nor any resident tied with these assignees in h's preference list can be assigned to h in any super-stable matching—such pairs (r, h) are deleted from each other's preference lists. If a hospital h is full then no resident strictly worse than h's worst assignees can be assigned to h in any super-stable matching—again such (r, h) pairs are deleted from each other's preference lists. The proposal sequence terminates once every resident is either assigned to a hospital or has an empty preference list. At this point, if the constructed assignment of residents to hospitals is super-stable in the original HRT instance then the assignment is returned as a super-stable matching. Otherwise, the algorithm reports that no super-stable matching exists.

We note that our algorithm is a non-trivial extension of Algorithm HRT-Super-Res for HRT (Irving et al. 2000). Due to the more general setting of SPA- ST, Algorithm SPA-ST-super requires some new ideas (precisely lines 27–34 of the algorithm on page 14), and the proofs of the correctness results are more complex than for the aforementioned algorithm for HRT. We give definitions relating to the algorithm in Sect. 3.1. We give a description of our algorithm in Sect. 3.2, before presenting it in pseudocode form. In Sect. 3.3, we illustrate an execution of our algorithm with respect to an example SPA- ST instance. We present the algorithm's correctness results in Sect. 3.4. Finally, in Sect. 3.5, we show that the set of super-stable matchings in an instance of SPA- ST satisfy analogous properties to those given in Theorem 1.

3.1 Definitions relating to 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*.

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. If s_i has been deleted from every projected preference list of l_k that she originally belonged to, we will implicitly assume that s_i has been deleted from l_k's preference list. 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 $\mathcal{L}_k$ (the 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 $\mathcal{L}_k$.

3.2 Description of the algorithm

We now describe our algorithm, shown in pseudocode form in Algorithm 1. Algorithm SPA-ST-super begins by initialising an empty set M which will contain the provisional assignments of students to projects (and implicitly 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 preference 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—the 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 that we delete is 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 preference 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—the 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 preference list, at which point the `repeat-until` loop terminates.

Upon termination of the `repeat-until` loop, if the set M, containing the assignment 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.

3.3 Example algorithm execution

We illustrate an execution of Algorithm SPA-ST-super with respect to the SPA- ST instance shown in Fig. 1 (page 7). We initialise $M = \{\}$, which will contain the provisional assignment of students to projects. For each project $p_j \in \mathcal{P}$, we set `full`(p_j) = `false` (`full`(p_j) will be set to `true` when p_j becomes full, so that we can easily identify any project that was full during an iteration of the algorithm and ended up undersubscribed). We assume that the students become provisionally assigned to each project at the head of their list in subscript order. Table 1 illustrates how this execution of Algorithm SPA-ST-super proceeds with respect to I_1.

3.4 Correctness of algorithm SPA-ST-super

We now present a series of results concerning the correctness of Algorithm SPA-ST-super. The first of these results deals with the fact that no super-stable pair is deleted during an 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 an execution of Algorithm* SPA-ST-super*, then (s_i, p_j) does not belong to any super-stable matching in I.*

In order to prove Lemma 1, we present Lemmas 2 and 3 .

Lemma 2 *If a pair (s_i, p_j) is deleted within the* `while` *loop during an execution of Algorithm* SPA-ST-super *then (s_i, p_j) does not belong to any super-stable matching in I.*

Table 1 An execution of Algorithm SPA-ST-super with respect to Fig. 1

While loop iterations	Student applies to project	Consequence
1	s_1 applies to p_1	$M = \{(s_1, p_1)\}$. `full`(p_1) = `true`.
2	s_2 applies to p_1	$M = \{(s_1, p_1), (s_2, p_1)\}$. p_1 becomes oversubscribed. The tail of $\mathcal{L}_1^1$ contains s_1 and s_2—thus we delete the pairs (s_1, p_1) and (s_2, p_1) (and we break the provisional assignments).
	s_2 applies to p_3	$M = \{(s_2, p_3)\}$. `full`(p_3) = `true`.
3	s_3 applies to p_2	$M = \{(s_2, p_3), (s_3, p_2)\}$.
4	s_4 applies to p_2	$M = \{(s_2, p_3), (s_3, p_2), (s_4, p_2)\}$. `full`$(p_2)$ = `true`.
5	s_5 applies to p_3	$M = \{(s_2, p_3), (s_3, p_2), (s_4, p_2), (s_5, p_3)\}$. p_3 becomes oversubscribed. The tail of $\mathcal{L}_2^3$ contains only s_2—thus we delete the pair (s_2, p_3) (and we break the provisional assignment).
The first iteration of the `while` loop terminates since every unassigned student (i.e., s_1 and s_2) has an empty preference list. At this point, `full`(p_1) is `true` and p_1 is undersubscribed. Moreover, the student at the tail of $\mathcal{L}_1$ (i.e., s_4) is no better than s_1, where s_1 was previously assigned to p_1 and s_1 is also the most-preferred student rejected from p_1 according to $\mathcal{L}_1^1$; thus we delete the pair (s_4, p_2). The `while` loop is then reactivated.		
6	s_4 applies to p_3	$M = \{(s_3, p_2), (s_5, p_3), (s_4, p_3)\}$. p_3 becomes oversubscribed. The tail of $\mathcal{L}_2^3$ contains only s_5—thus we delete the pair (s_5, p_3).
7	s_5 applies to p_1	$M = \{(s_3, p_2), (s_4, p_3), (s_5, p_1)\}$.
Again, every unassigned students has an empty preference list. We also have that `full`(p_2) is `true` and p_2 is undersubscribed; however no further deletion is carried out in line 34 of the algorithm, since the student at the tail of $\mathcal{L}_1$ (i.e., s_3) is better than s_4, where s_4 was previously assigned to p_2 and s_4 is also the most-preferred student rejected from p_2 according to $\mathcal{L}_1^2$. Hence, the `repeat-until` loop terminates and the algorithm outputs $M = \{(s_3, p_2), (s_4, p_3), (s_5, p_1)\}$ as a super-stable matching. It is clear that M is super-stable in the original instance I_1.		

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 preference list **do**
    for each project p_j at the head of s_i's preference 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)
  for each $p_j \in \mathcal{P}$ **do**
    if 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 preference list
if M is super-stable in I **then**
  return M
else
  return "no super-stable matching exists in I"

Proof Without loss of generality, suppose that the first super-stable pair to be deleted within the `while` loop during an arbitrary execution E of the algorithm is (s_i, p_j), which belongs to some super-stable matching, say M^*. Suppose that M is the assignment immediately after the deletion. Let us denote this point in the algorithm where the deletion is made by ‡. During E, there are four cases that would lead to the deletion of any (student, project) pair within the `while` loop.

(1) p_j *is oversubscribed.* Suppose that (s_i, p_j) is deleted because some student (possibly s_i) became provisionally assigned to p_j during E, causing p_j to become oversubscribed. If p_j is full or undersubscribed at point ‡, since $s_i \in M^*(p_j) \setminus M(p_j)$ and no project can be oversubscribed in M^*, then there is some student $s_r \in M(p_j) \setminus M^*(p_j)$ such that l_k prefers s_r to s_i or is indifferent between them. We note that s_r cannot be assigned to a project that she prefers to p_j in any super-stable matching. Otherwise, since p_j must have been in the head of s_r's preference list when she applied, this would mean that a super-stable pair was deleted before (s_i, p_j). Thus either s_r is unassigned in M^* or s_r prefers p_j to $M^*(s_r)$ or s_r is indifferent between them. Clearly, for any combination of l_k and p_j being full or undersubscribed in M^*, it follows that (s_r, p_j) blocks M^*, a contradiction.

(2) l_k *is oversubscribed.* Suppose that (s_i, p_j) is deleted because some student (possibly s_i) became provisionally assigned to a project offered by lecturer l_k during E, causing l_k to become oversubscribed. At point ‡, none of the projects offered by l_k is oversubscribed in M, otherwise we will be in case (1). Similar to case (1), if l_k is full or undersubscribed at point ‡, since $s_i \in M^*(p_j) \setminus M(p_j)$ and no lecturer can be oversubscribed in M^*, it follows that there is some project $p_{j'} \in P_k$ and some student $s_r \in M(p_{j'}) \setminus M^*(p_{j'})$ such that l_k prefers s_r to s_i or is indifferent between them. We consider two subcases.

   (i) If $p_{j'} = p_j$ then $s_r \neq s_i$. Moreover, as in case (1), either s_r is unassigned in M^* or s_r prefers $p_{j'}$ to $M^*(s_r)$ or s_r is indifferent between them. For any combination of l_k and $p_{j'}$ being full or undersubscribed in M^*, we have that $(s_r, p_{j'})$ blocks M^*, a contradiction.

   (ii) If $p_{j'} \neq p_j$. Assume firstly that $s_r \neq s_i$. Then as $p_{j'}$ has fewer assignees in M^* than it has provisional assignees in M, and as in (i) above, $(s_r, p_{j'})$ blocks M^*, a contradiction. Finally assume $s_r = s_i$. Then s_i must have applied to $p_{j'}$ at some point during E before ‡. Clearly, either s_i prefers $p_{j'}$ to p_j or s_i is indifferent between them, since $p_{j'}$ must have been in the head of s_i's preference list when s_i applied. Since $s_i \in M^*(l_k)$ and $p_{j'}$ is undersubscribed in M^*, it follows that $(s_i, p_{j'})$ blocks M^*, a contradiction.

(3) p_j *is full.* Suppose that (s_i, p_j) is deleted because p_j became full during E. At point ‡, p_j is full in M. Thus at least one of the students in $M(p_j)$, say s_r, will not be assigned to p_j in M^*, for otherwise p_j will be oversubscribed in M^*. This implies that either s_r is unassigned in M^* or s_r prefers p_j to $M^*(s_r)$ or s_r is indifferent between them. For otherwise, we obtain a contradiction to (s_i, p_j) being the first super-stable pair to be deleted. Since l_k prefers s_r to s_i, it follows that (s_r, p_j) blocks M^*, a contradiction.

(4) l_k *is full.* Suppose that (s_i, p_j) is deleted because l_k became full during E. We consider two subcases.

   (i) All the students assigned to p_j in M at point ‡ (if any) are also assigned to p_j in M^*. This implies that p_j has one more assignee in M^* than it has provisional assignees in M, namely s_i. Thus, some other project $p_{j'} \in P_k$ has fewer assignees in M^* than it has provisional assignees in M, for otherwise l_k would be oversubscribed in M^*. Hence there exists some student $s_r \in$

$M(p_{j'})\backslash M^*(p_{j'})$. It is clear that $s_r \neq s_i$, since s_i plays the role of s_t at some for loop iteration in line 24 of the algorithm. Also, s_r cannot be assigned to a project that she prefers to $p_{j'}$ in M^*, as explained in case (1). Moreover, since $p_{j'}$ is undersubscribed in M^* and l_k prefers s_r to s_i, it follows that $(s_r, p_{j'})$ blocks M^*, a contradiction.

(ii) Some student, say s_r, who is assigned to p_j in M is not assigned to p_j in M^*, i.e., $s_r \in M(p_j)\backslash M^*(p_j)$. Since s_r cannot be assigned in M^* to a project that she prefers to p_j and since l_k prefers s_r to s_i, it follows that (s_r, p_j) blocks M^*, a contradiction.

□

Lemma 3 *If a pair (s_i, p_j) is deleted in line 34 of Algorithm* SPA-ST-super *then (s_i, p_j) does not belong to any super-stable matching in I.*

Proof Without loss of generality, suppose that the first super-stable pair to be deleted during an arbitrary execution E of the algorithm is (s_i, p_j), which belongs to some super-stable matching, say M^*. Then by Lemma 2, (s_i, p_j) was deleted in line 34 during E. Let l_k be the lecturer who offers p_j. Suppose that M is the assignment during the iteration of the `repeat-until` loop where (s_i, p_j) was deleted.

Let $p_{j'}$ be some other project offered by l_k which was full during a previous `repeat-until` loop iteration and subsequently ends up undersubscribed in the current `repeat-until` loop iteration, i.e., $p_{j'}$ plays the role of p_j in line 28. Suppose that $s_{i'}$ plays the role of s_r in line 30, i.e., $s_{i'}$ is the most-preferred student rejected from $p_{j'}$ according to $\mathcal{L}_k^{j'}$ (possibly $s_{i'} = s_i$). Moreover $s_{i'}$ was provisionally assigned to $p_{j'}$ during a previous `repeat-until` loop iteration but $(s_{i'}, p_{j'}) \notin M$ in the current `repeat-until` loop iteration. Thus $(s_{i'}, p_{j'})$ has been deleted before the deletion of (s_i, p_j) occurred; and thus, $(s_{i'}, p_{j'}) \notin M^*$, since (s_i, p_j) is the first super-stable pair to be deleted. Further, l_k either prefers $s_{i'}$ to s_i or is indifferent between them, since s_i plays the role of s_t at some for loop iteration in line 32.

We remark that no student who is provisionally assigned to some project in M can be assigned to a project better than her current assignment in any super-stable matching. For otherwise, this would mean a super-stable pair must have been deleted before (s_i, p_j), since each student who is assigned in M applies to projects in the head of her preference list. So, either $s_{i'}$ is unassigned in M^* or $s_{i'}$ prefers $p_{j'}$ to $M^*(s_{i'})$ or s_i is indifferent between them. By the super-stability of M^*, $p_{j'}$ is full in M^* and l_k prefers every student in $M^*(p_{j'})$ to $s_{i'}$; for otherwise, $(s_{i'}, p_{j'})$ blocks M^*, a contradiction.

Let $l_{z_0} = l_k$, $p_{t_0} = p_{j'}$ and $s_{q_0} = s_{i'}$. Just before the deletion of (s_i, p_j) occurred, p_{t_0} is undersubscribed in M. Since p_{t_0} is full in M^*, there exists some student $s_{q_1} \in M^*(p_{t_0})\backslash M(p_{t_0})$. We note that l_{z_0} prefers s_{q_1} to s_{q_0}; for otherwise, $(s_{i'}, p_{j'})$ blocks M^*, a contradiction. Let $p_{t_1} = p_{t_0}$. Since (s_i, p_j) is the first super-stable pair to be deleted, s_{q_1} is assigned in M to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. For otherwise, as each student applies to projects at the head of her preference list, that would mean (s_{q_1}, p_{t_1}) must have been deleted before (s_i, p_j), a contradiction. We note that $p_{t_2} \neq p_{t_1}$, since $(s_{q_1}, p_{t_2}) \in M$ and $(s_{q_1}, p_{t_1}) \notin M$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the super-stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefer the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2})\backslash M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1})\backslash M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i}\rangle_{i\geq 1}$ of students, a sequence $\langle p_{t_i}\rangle_{i\geq 1}$ of projects, and a sequence $\langle l_{z_i}\rangle_{i\geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

First we claim that for each new project that we identify, $p_{t_{2i}} \neq p_{t_{2i-1}}$ for $i \geq 1$. Suppose $p_{t_{2i}} = p_{t_{2i-1}}$ for some $i \geq 1$. From above s_{q_i} was identified by $l_{z_{i-1}}$ such that $(s_{q_i}, p_{t_{2i-1}}) \in M^*\backslash M$. Moreover $(s_{q_i}, p_{t_{2i}}) \in M$. Hence we reach a contradiction. Clearly, for each student s_{q_i} that we identify, for $i \geq 1$, s_{q_i} must be assigned to distinct projects in M and in M^*.

Next we claim that for each new student s_{q_i} that we identify, $s_{q_i} \neq s_{q_t}$ for $1 \leq t < i$. We prove this by induction on i. For the base case, clearly $s_{q_2} \neq s_{q_1}$. We assume that the claim holds for some $i \geq 1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}$ consists of distinct students. We show that the claim holds for $i+1$, i.e., the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_i}, s_{q_{i+1}}$ also consists of distinct students. Clearly $s_{q_{i+1}} \neq s_{q_i}$ since l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}. Thus, it suffices to show that $s_{q_{i+1}} \neq s_{q_j}$ for $1 \leq j \leq i-1$. Now, suppose $s_{q_{i+1}} = s_{q_j}$ for $1 \leq j \leq i-1$. This implies that s_{q_j} was identified by l_{z_i} and clearly l_{z_i} prefers s_{q_j} to $s_{q_{j-1}}$. Now since $s_{q_{i+1}}$ was also identified by l_{z_i} to avoid the blocking pair $(s_{q_i}, p_{t_{2_i}})$ in M^*, it follows that either (i) $p_{t_{2i}}$ is full in M^*, or (ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. We consider each cases further as follows.

(i) If $p_{t_{2i}}$ is full in M^*, we know that $(s_{q_i}, p_{t_{2i}}) \in M\backslash M^*$. Moreover s_{q_j} was identified by $l_{z_{i+1}}$ because of case (i). Furthermore $(s_{q_{j-1}}, p_{t_{2i}}) \in M\backslash M^*$. In this case, $p_{t_{2i+1}} = p_{t_{2i}}$ and we have that

$$(s_{q_i}, p_{t_{2i+1}}) \in M\backslash M^* \text{ and } (s_{q_{i+1}}, p_{t_{2i+1}}) \in M^*\backslash M,$$
$$(s_{q_{j-1}}, p_{t_{2i+1}}) \in M\backslash M^* \text{ and } (s_{q_j}, p_{t_{2i+1}}) \in M^*\backslash M.$$

By the inductive hypothesis, the sequence $s_{q_1}, s_{q_2}, \ldots, s_{q_{j-1}}, s_{q_j}, \ldots, s_{q_i}$ consists of distinct students. This implies that $s_{q_i} \neq s_{q_{j-1}}$. Thus since $p_{t_{2i+1}}$ is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ to avoid the blocking pairs $(s_{q_{j-1}}, p_{t_{2i+1}})$ and $(s_{q_i}, p_{t_{2i+1}})$ respectively in M^*, a contradiction.

(ii) $p_{t_{2i}}$ is undersubscribed in M^* and l_{z_i} is full in M^*. Similarly as in case (i) above, we have that

$$s_{q_i} \in M(l_{z_i})\backslash M^*(l_{z_i}) \text{ and } s_{q_{i+1}} \in M^*(l_{z_i})\backslash M(l_{z_i}),$$
$$s_{q_{j-1}} \in M(l_{z_i})\backslash M^*(l_{z_i}) \text{ and } s_{q_j} \in M^*(l_{z_i})\backslash M(l_{z_i}).$$

Since $s_{q_i} \neq s_{q_{j-1}}$ and l_{z_i} is full in M^*, l_{z_i} should have been able to identify distinct students s_{q_j} and $s_{q_{i+1}}$ corresponding to students $s_{q_{j-1}}$ and s_{q_i} respectively, a contradiction.

This completes the induction step. As the sequence of distinct students and projects we are identifying is infinite, we reach an immediate contradiction. □

Lemmas 2 and 3 immediately give rise to Lemma 1. The next lemma will be used as a tool in the proof of the remaining lemmas.

Lemma 4 *Let M be the assignment at the termination of Algorithm* SPA-ST-super *and let M^* be any super-stable matching in I. Let l_k be an arbitrary lecturer: (i) if l_k is undersubscribed in M^* then every student who is assigned to l_k in M is also assigned to l_k in M^*; and (ii) if l_k is undersubscribed in M then l_k has the same number of assignees in M^* as in M.*

Proof Let l_k be an arbitrary lecturer. First, we show that (i) holds. Suppose otherwise, then there exists a student, say s_i, such that $s_i \in M(l_k) \backslash M^*(l_k)$. Moreover, there exists some project $p_j \in P_k$ such that $s_i \in M(p_j) \backslash M^*(p_j)$. By Lemma 1, s_i cannot be assigned to a project that she prefers to p_j in M^*. Also, by the super-stability of M^*, p_j is full in M^* and l_k prefers the worst student/s in $M^*(p_j)$ to s_i.

Let $l_{z_0} = l_k$, $p_{t_0} = p_j$, and $s_{q_0} = s_i$. As p_{t_0} is full in M^* and no project is oversubscribed in M, there exists some student $s_{q_1} \in M^*(p_{t_0}) \backslash M(p_{t_0})$ such that l_{z_0} prefers s_{q_1} to s_{q_0}. Let $p_{t_1} = p_{t_0}$. By Lemma 1, s_{q_1} is assigned in M to a project p_{t_2} such that s_{q_1} prefers p_{t_2} to p_{t_1}. We note that s_{q_1} cannot be indifferent between p_{t_2} and p_{t_1}; for otherwise, as each student applies to projects at the head of her preference list, since $(s_{q_1}, p_{t_1}) \notin M$, that would mean (s_{q_1}, p_{t_1}) must have been deleted during the algorithm's execution, contradicting Lemma 1. It follows that $s_{q_1} \in M(p_{t_2}) \backslash M^*(p_{t_2})$. Let l_{z_1} be the lecturer who offers p_{t_2}. By the super-stability of M^*, either (i) or (ii) holds as follows:

(i) p_{t_2} is full in M^* and l_{z_1} prefers the worst student/s in $M^*(p_{t_2})$ to s_{q_1};
(ii) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^*, $s_{q_1} \notin M^*(l_{z_1})$ and l_{z_1} prefers the worst student/s in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \backslash M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \backslash M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}. Now, suppose $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). It is clear that $s_{q_2} \neq s_{q_1}$. Applying similar reasoning as for s_{q_1}, student s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. Let l_{z_2} be the lecturer who offers p_{t_4}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 3, we can identify an infinite sequence of distinct students and projects, a contradiction. Hence, if l_k is

undersubscribed in M^* then every student who is assigned to l_k in M is also assigned to l_k in M^*.

Next, we show that (ii) holds. By the first claim, any lecturer who is full in M is also full in M^*, and any lecturer who is undersubscribed in M has as many assignees in M^* as she has in M. Hence

$$\sum_{l_k \in \mathcal{L}} |M(l_k)| \leq \sum_{l_k \in \mathcal{L}} |M^*(l_k)| \ . \tag{1}$$

We note that if a student s_i is unassigned in M, by Lemma 1, s_i is unassigned in M^*. Equivalently, if s_i is assigned in M^* then s_i is assigned in M. Let S_1 denote the set of students who are assigned to at least one project in M, and let S_2 denote the set of students who are assigned to a project in M^*; it follows that $|S_2| \leq |S_1|$. Further, we have that

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| = |S_2| \leq |S_1| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)|, \tag{2}$$

From Inequalities (1) and (2), it follows that $|M(l_k)| = |M^*(l_k)|$ for each $l_k \in \mathcal{L}$. □

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 5 *If a student is assigned to two or more projects at the termination of Algorithm* SPA-ST-super *then I admits no super-stable matching.*

Proof Let M be the assignment at the termination of the algorithm. Suppose for a contradiction that there exists a super-stable matching M^* in I. Suppose that a student is assigned to two or more projects in M. Then either (a) any two of these projects are offered by different lecturers or (b) all of these projects are offered by the same lecturer.

Firstly, suppose (a) holds. Then some lecturer has fewer assignees in M^* than in M. Suppose not, then

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| \geq \sum_{l_k \in \mathcal{L}} |M(l_k)| \ . \tag{3}$$

Let S_1 and S_2 be as defined in the proof of Lemma 4, it follows that $|S_2| \leq |S_1|$. Hence,

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| = |S_2| \leq |S_1| < \sum_{l_k \in \mathcal{L}} |M(l_k)|, \tag{4}$$

since some student in S_1 is assigned in M to two or more projects offered by different lecturers. Inequality (4) contradicts Inequality (3). Hence, our claim is established. As some lecturer l_k has fewer assignees in M^* than in M, it follows that l_k is undersubscribed in M^*, since no lecturer is oversubscribed in M. In particular, there exists

some project $p_j \in P_k$ and some student, say s_i, such that p_j is undersubscribed in M^* and $(s_i, p_j) \in M \backslash M^*$. Since $(s_i, p_j) \in M$, then p_j must have been in the head of s_i's preference list when s_i applied to p_j during the algorithm's execution. By Lemma 1, either s_i is unassigned in M^* or s_i prefers p_j to $M^*(s_i)$ or s_i is indifferent between them. Hence (s_i, p_j) blocks M^*, a contradiction.

Next, suppose (b) holds. Then $|S_1| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)|$. As in case (a), since $|S_2| \leq |S_1|$, it follows that

$$\sum_{l_k \in \mathcal{L}} |M^*(l_k)| \leq \sum_{l_k \in \mathcal{L}} |M(l_k)| \ .$$

Suppose first that $|M^*(l_k)| < |M(l_k)|$ for some $l_k \in \mathcal{L}$. Then l_k has fewer assignees in M^* than in M, and following a similar argument as in case (a) above, we reach an immediate contradiction. Hence, $|M^*(l_k)| = |M(l_k)|$ for all $l_k \in \mathcal{L}$. For each $l_k \in \mathcal{L}$, we claim that every student who is assigned to l_k in M is also assigned to l_k in M^*. Suppose otherwise. Let l_{z_1} be an arbitrary lecturer in $\mathcal{L}$. Then there exists some student $s_{q_1} \in M(l_{z_1}) \backslash M^*(l_{z_1})$. Let $M(s_{q_1}) = p_{t_2}$. By Lemma 1, s_{q_1} is assigned in M^* to a project p_{t_1} such that s_{q_1} prefers p_{t_2} to p_{t_1}. Clearly, p_{t_1} is not offered by l_{z_1}, since $s_{q_1} \in M(l_{z_1}) \backslash M^*(l_{z_1})$. We also note that s_{q_1} cannot be indifferent between p_{t_2} and p_{t_1}. Otherwise, the argument follows from (a), since s_{q_1} is assigned in M to two projects offered by different lecturers, and we reach an immediate contradiction. By the super-stability of M^*, either (i) or (ii) holds as follows:

(a) p_{t_2} is full in M^* and l_{z_1} prefers every student in $M^*(p_{t_2})$ to s_{q_1};
(b) p_{t_2} is undersubscribed in M^*, l_{z_1} is full in M^* and l_{z_1} prefers every student in $M^*(l_{z_1})$ to s_{q_1}.

Otherwise, (s_{q_1}, p_{t_2}) blocks M^*. In case (i), there exists some student $s_{q_2} \in M^*(p_{t_2}) \backslash M(p_{t_2})$. Let $p_{t_3} = p_{t_2}$. In case (ii), there exists some student $s_{q_2} \in M^*(l_{z_1}) \backslash M(l_{z_1})$. We note that l_{z_1} prefers s_{q_2} to s_{q_1}, and clearly $s_{q_2} \neq s_{q_1}$. Let $M^*(s_{q_2}) = p_{t_3}$ (possibly $p_{t_3} = p_{t_2}$). Applying similar reasoning as for s_{q_1}, student s_{q_2} is assigned in M to a project p_{t_4} such that s_{q_2} prefers p_{t_4} to p_{t_3}. We are identifying a sequence $\langle s_{q_i} \rangle_{i \geq 1}$ of students, a sequence $\langle p_{t_i} \rangle_{i \geq 1}$ of projects, and a sequence $\langle l_{z_i} \rangle_{i \geq 1}$ of lecturers, such that, for each $i \geq 1$

1. s_{q_i} prefers $p_{t_{2i}}$ to $p_{t_{2i-1}}$,
2. $(s_{q_i}, p_{t_{2i}}) \in M$ and $(s_{q_i}, p_{t_{2i-1}}) \in M^*$,
3. l_{z_i} prefers $s_{q_{i+1}}$ to s_{q_i}; also, l_{z_i} offers both $p_{t_{2i}}$ and $p_{t_{2i+1}}$ (possibly $p_{t_{2i}} = p_{t_{2i+1}}$).

Following a similar argument as in the proof of Lemma 3, we can identify an infinite sequence of distinct students and projects, a contradiction.

Now, let s_i be an arbitrary student such that s_i is assigned in M to two or more projects offered by a lecturer, say l_k. Then $s_i \in M^*(l_k)$. Moreover, there exists some project $p_j \in P_k$ such that $(s_i, p_j) \in M \backslash M^*$. We claim that p_j is undersubscribed in M^*. Suppose otherwise. Let $l_{z_0} = l_k$, $p_{t_0} = p_j$ and $s_{q_0} = s_i$. Then there exists some student $s_{q_1} \in M^*(p_{t_0}) \backslash M(p_{t_0})$, since p_{t_0} is not oversubscribed in M and $s_{q_0} \in M(p_{t_0}) \backslash M^*(p_{t_0})$. Again, by Lemma 1, s_{q_1} is assigned in M to a project p_{t_1} such that s_{q_1} prefers p_{t_1} to p_{t_0}. Let l_{z_1} be the lecturer who offers p_{t_1}. Following a similar

argument as in the proof of Lemma 3, we can identify a sequence of distinct students and projects, and as this sequence is infinite, we reach a contradiction. Hence our claim holds, i.e., p_j is undersubscribed in M^*. Finally, since s_i cannot be assigned to any project that she prefers to p_j in M^* and since $(s_i, p_j) \in M^*(l_k)$, we have that (s_i, p_j) blocks M^*, a contradiction. □

Lemma 6 *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.*

Proof Let M be the assignment at the termination of the algorithm. Suppose for a contradiction that there exists a super-stable matching M^* in I. Let l_k be the lecturer who became full during some execution of the algorithm and subsequently ends up undersubscribed in M. By Lemma 4, $|M(l_k)| = |M^*(l_k)|$ and thus l_k is undersubscribed in M^*. At the point in the algorithm where l_k became full (line 22), we note that none of the projects offered by l_k is oversubscribed. Since l_k ended up undersubscribed in M, it follows that there is some project $p_j \in P_k$ that has fewer assignees in M at the termination of the algorithm than it had at some point during the algorithm's execution, thus p_j is undersubscribed in M.

We claim that each project offered by l_k has the same number of assignees in M^* as in M. Suppose otherwise, then there is some project $p_t \in P_k$ such that $|M^*(p_t)| < |M(p_t)|$; thus p_t is undersubscribed in M^*, since no project is oversubscribed in M. It follows that there exists some student $s_r \in M(p_t) \backslash M^*(p_t)$. By Lemma 1, s_r is either unassigned in M^* or prefers p_t to $M^*(s_r)$. Since l_k is undersubscribed in M^*, (s_r, p_t) blocks M^*, a contradiction. Hence $|M^*(p_t)| \geq |M(p_t)|$. Moreover, since $|M(l_k)| = |M^*(l_k)|$, we have that $|M(p_t)| = |M^*(p_t)|$ for all $p_t \in P_k$.

Hence p_j undersubscribed in M implies that p_j is undersubscribed in M^*. Moreover, there is some student s_i who was provisionally assigned to p_j at some point during the execution of the algorithm but s_i is not assigned to p_j in M. Thus, the pair (s_i, p_j) was deleted during the algorithm's execution, so that $(s_i, p_j) \notin M^*$ by Lemma 1. It follows that either s_i is unassigned in M^* or s_i prefers p_j to $M^*(s_i)$ or s_i is indifferent between them. Hence, (s_i, p_j) blocks M^*, a contradiction. □

Lemma 7 *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.*

Proof Suppose for a contradiction that there exists a super-stable matching M^* in I. Let (s_i, p_j) be a pair that was deleted during an arbitrary execution E of the algorithm. This implies that $(s_i, p_j) \notin M^*$ by Lemma 1. Let M be the assignment at the termination of E. By the hypothesis of the lemma, l_k is undersubscribed in M. This implies that l_k is undersubscribed in M^*, by Lemma 4. Since p_j is offered by l_k, and p_j is undersubscribed in M, it follows from the proof of Lemma 6 that p_j is undersubscribed in M^*. Further, by the hypothesis of the lemma, either s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$ or is indifferent between them. By Lemma 1, this is true for s_i in M^*. Hence (s_i, p_j) blocks M^*, a contradiction. □

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 8 *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 assigned to two or more projects in M then I admits no super-stable matching, by Lemma 5. Hence every student is assigned to at most one project in M. Moreover, since no project or lecturer is oversubscribed in M, it follows that M is a matching. Let (s_i, p_j) be a blocking pair for M, then s_i is either unassigned in M or prefers p_j to $M(s_i)$ or is indifferent between them. Whichever is the case, (s_i, p_j) has been deleted. Let l_k be the lecturer who offers p_j. In what follows, we will identify the point in the algorithm at which (s_i, p_j) was deleted, and consequently, we will arrive at a conclusion that no super-stable matching exists.

Firstly, suppose (s_i, p_j) was deleted as a result of p_j being full or oversubscribed (on lines 12 or 21). Suppose p_j is 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 p_j is 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 on line 34 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 6. Hence l_k was never full during the algorithm's execution. Recall that each of p_j and l_k is undersubscribed in M. As (s_i, p_j) is a blocking pair of M, s_i cannot be assigned in M to a project that she prefers to p_j. Hence I admits no super-stable matching, by Lemma 7.

Next, suppose (s_i, p_j) was deleted as a result of l_k being full or oversubscribed (on lines 16 or 26), (s_i, p_j) could only block M if l_k is undersubscribed in M. If this is the case then I admits no super-stable matching, by Lemma 6.

Finally, suppose (s_i, p_j) was deleted (on line 34) because some other project $p_{j'}$ offered by l_k was previously full and ended up undersubscribed on line 28. Then l_k must have identified the most-preferred student, say s_r, who was previously assigned to $p_{j'}$ but subsequently got rejected from $p_{j'}$. At this point, 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 34. If p_j is full in M then (s_i, p_j) does not block M. Hence p_j is 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/s in $M(l_k)$ to s_i. Hence l_k is undersubscribed in M. Again by Lemma 7, I admits no super-stable matching.

Since (s_i, p_j) is an arbitrary pair, this implies that I admits no super-stable matching. □

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

Lemma 9 *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*.*

Proof The algorithm's time complexity depends on how efficiently we can execute the operation of a student applying to a project and the operation of deleting a (student, project) pair, each of which occur once for any (student, project) pair. It turns out that both operations can be implemented to run in constant time, giving Algorithm SPA-ST-super an overall complexity of $\Theta(L)$, where L is the total length of all the preference lists. In what follows, we describe the non-trivial aspects of such an implementation. We remark that the data structures discussed here are inspired by, and extend, those detailed in Abraham et al. (2007, Section 3.3), for Algorithm SPA-student.

For each student s_i, build an array $position_{s_i}$, where $position_{s_i}(p_j)$ is the position of project p_j in s_i's preference list. For example, if s_i's preference list is $(p_2\ p_5\ p_3)\ p_7\ (p_6\ p_1)$ then $position_{s_i}(p_5) = 2$ and $position_{s_i}(p_1) = 6$. In general, position captures the order in which the projects appear in the preference list when read from left to right, ignoring any ties. Represent s_i's preference list by embedding doubly linked lists in an array $preference_{s_i}$. For each project $p_j \in A_i$, $preference_{s_i}(position_{s_i}(p_j))$ stores the list node containing p_j. This node contains two next pointers (and two previous pointers)—one to the next project in s_i's preference list (after deletions, this project may not be located at the next array position), and another pointer to the next project $p_{j'}$ in s_i's preference list, where $p_{j'}$ and p_j are both offered by the same lecturer. Construct the latter list by traversing through s_i's preference list, using a temporary array to record the last project in the list offered by each lecturer. Use virtual initialisation (described in Brassard and Bratley 1996, p. 149) for these arrays, since the overall $O(n_1 n_3)$ initialisation may be too expensive.

To represent the ties in s_i's preference list, build an array $successor_{s_i}$. For each project p_j in s_i's preference list, $successor_{s_i}(position_{s_i}(p_j))$ stores the `true` boolean if p_j is tied with its successor in A_i and `false` otherwise. After the deletion of any (student, project) pair, update the successor booleans. As an illustration, with respect to s_i's preference list given in the previous paragraph, $successor_{s_i}$ is the array [`true, true, false, false, true, false`]. Now, suppose p_3 was deleted from s_i's preference list, since $successor_{s_i}(position_{s_i}(p_3))$ is `false` and $successor_{s_i}(position_{s_i}(p_5))$ is `true`, set $successor_{s_i}(position_{s_i}(p_5))$ to `false` (since p_5 is the predecessor of p_3). Clearly using these data structures, we can find the next project at the head of each student's preference list, find the next project offered by a given lecturer on each student's preference list, as well as delete a project from a given student's preference list in constant time.

For each lecturer l_k, build two arrays $preference_{l_k}$ and $successor_{l_k}$, where $preference_{l_k}(s_i)$ is the position of student s_i in l_k's preference list, and $successor_{l_k}(preference_{l_k}(s_i))$ stores the position of the first strict successor (with respect to position) of s_i in $\mathcal{L}_k$ or a null value if s_i has no strict successor[5]. Represent l_k's preference list (i.e., $\mathcal{L}_k$) by the array $preference_{l_k}$, with an additional pointer, $last_{l_k}$. Initially, $last_{l_k}$ stores the index of the last position in $preference_{l_k}$. To represent the ties in l_k's preference list, build an array $predecessor_{l_k}$. For each $s_i \in \mathcal{L}_k$,

[5] For example, if l_k's preference list is $s_5\ (s_3\ s_1\ s_6)\ s_7\ (s_2\ s_8)$ then $successor_{l_k}$ is the array [2 5 5 5 6 0 0].

$predecessor_{l_k}(preference_{l_k}(s_i))$ stores the `true` boolean if s_i is tied with its predecessor in $\mathcal{L}_k$ and `false` otherwise.

When l_k becomes full, make $last_{l_k}$ equivalent to l_k's worst assigned student through the following method. Perform a backward traversal through the array $preference_{l_k}$, starting at $last_{l_k}$, and continuing until l_k's worst assigned student, say $s_{i'}$, is encountered (each student stores a pointer to their assigned project, or a special null value if unassigned). Deletions must be carried out in the preference list of each student who is worse than $s_{i'}$ on l_k's preference list (precisely those students whose position in $preference_{l_k}$ is greater than or equal to that stored in $successor_{l_k}(preference_{l_k}(s_{i'}))$)[6].

When l_k becomes oversubscribed, we can find and delete the students at the tail of l_k by performing a backward traversal through the array $preference_{l_k}$, starting at $last_{l_k}$, and continuing until we encounter a student, say $s_{i'}$, such that $predecessor_{l_k}(preference_{l_k}(s_{i'}))$ stores the `false` boolean. If l_k becomes undersubscribed after we break the assignment of students encountered on this traversal (including $s_{i'}$) to l_k, rather than update $last_{l_k}$ immediately, which could be expensive, we wait until l_k becomes full again. The cost of these traversals taken over the algorithm's execution is thus linear in the length of l_k's preference list.

For each project p_j offered by l_k, build the arrays $preference_{p_j}$, $successor_{p_j}$ and $predecessor_{p_j}$ corresponding to $\mathcal{L}_k^j$, as described in the previous paragraph for $\mathcal{L}_k$. Represent the projected preference list of l_k for p_j (i.e., $\mathcal{L}_k^j$) by the array $preference_{p_j}$, with an additional pointer, $last_{p_j}$. These project preference arrays are used in much the same way as the lecturer preference arrays

Since we only visit a student at most twice during these backward traversals, once for the lecturer and once for the project, the asymptotic running time remains linear. □

Lemma 1 shows that there is an optimality property for each assigned student in any super-stable matching found by the algorithm, whilst Lemma 8 establishes the correctness of Algorithm SPA-ST-super. The following theorem collects together Lemmas 1, 8 and 9 .

Theorem 2 *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.*

Given the optimality property established by Theorem 2, we define the super-stable matching found by Algorithm SPA-ST-super to be *student-optimal*.

[6] For efficiency, we remark that it is not necessary to make deletions from the preference lists of lecturers or projected preference lists of lecturers for each project the lecturer offers, since the while loop of Algorithm SPA-ST-super involves students applying to projects in the head of their preference list.

Students' preferences	Lecturers' preferences	
s_1: p_1	l_1: s_5 s_6 s_4 (s_1 s_2) s_3	l_1 offers p_1, p_2
s_2: (p_1 p_3)	l_2: s_3 s_4 s_5 s_6 s_2	l_2 offers p_3, p_4
s_3: p_2 p_3		
s_4: p_2 p_3		
s_5: p_3 p_2	Project capacities: $c_1 = c_4 = 1$, $c_2 = c_3 = 2$	
s_6: p_2 p_4	Lecturer capacities: $d_1 = 2$, $d_2 = 3$	

Fig. 4 Instance I_2 of SPA- ST

3.5 Properties of super-stable matchings in SPA- ST

In this section, we consider properties of the set of super-stable matchings in an instance of SPA- ST. We show that the Unpopular Projects Theorem for SPA- S (see Theorem 1) holds for SPA- ST under super-stability.

Theorem 3 *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.*

Proof Let M and M^* be two arbitrary super-stable matchings in I. Let I' be an instance of SPA- S obtained from I by breaking the ties in I in some way. Then by Proposition 1, each of M and M^* is stable in I'. Thus by Theorem 1, each lecturer is assigned the same number of students in M and M^*, exactly the same students are unassigned in M and M^*, and a project offered by an undersubscribed lecturer has the same number of students in M and M^*. □

To illustrate this, consider the SPA- ST instance I_2 given in Fig. 4, which admits the super-stable matchings $M_1 = \{(s_3, p_3), (s_4, p_2), (s_5, p_3), (s_6, p_2)\}$ and $M_2 = \{(s_3, p_3), (s_4, p_3), (s_5, p_2), (s_6, p_2)\}$. Each of l_1 and l_2 is assigned the same number of students in both M_1 and M_2, illustrating part (1) of Theorem 3. Also, each of s_1 and s_2 is unassigned in both M_1 and M_2, illustrating part (2) of Theorem 3. Finally, l_2 is undersubscribed in both M_1 and M_2, and each of p_3 and p_4 has the same number of students in both M_1 and M_2, illustrating part (3) of Theorem 3.

4 Empirical evaluation

In this section, we evaluate an implementation of Algorithm SPA-ST-super. We implemented our algorithm in Python,[7] and performed our experiments on a system with dual Intel Xeon CPU E5-2640 processors with 64GB of RAM, running Ubuntu 17.10. 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?

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

4.1 Datasets

When generating random 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 lists 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 list comprises 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- S.

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 uniformly at random 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.[8] 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.

4.2.1 Correctness testing

To test the correctness of our algorithm's implementation, we implemented an Integer Programming (IP) model for super-stability in SPA- ST (see Appendix 1) using the Gurobi optimisation solver in Python (see Footnote 7). We randomly generated 10, 000 SPA- ST instances, each consisting of 100 students and a constant ratio of projects, lecturers, project capacities and lecturer capacities as described above. Also, each student's preference list was fixed at 10, with a tie density of 0.1. With this setup, we verified consistency between the outcomes of our implementation of Algorithm SPA-ST-super and our implementation of the IP-based algorithm in terms of the existence or otherwise of a super-stable matching.

4.2.2 Experiment 1

In our first experiment, we examined how the length of the students' preference lists affects the existence of a super-stable matching. We increased the number of students

[8] We remark that the parameter space was chosen to ensure that projects could typically accommodate more than one student, that the total capacity of the projects exceeded the number of students, and that each lecturer typically offered multiple projects, without reflecting any specific real-world application.

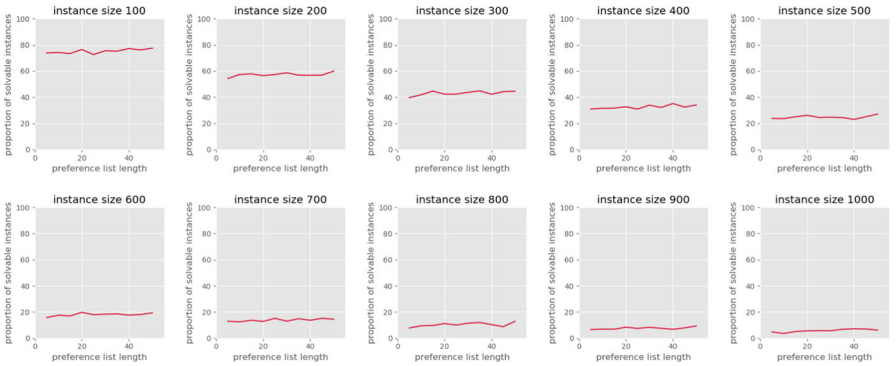

Fig. 5 Proportion of instances that admit a super-stable matching as the size of the instance increases while varying the length of the preference lists with tie density fixed at 0.005 in both the students' and lecturers' preference lists

Table 2 Time (in seconds) for our algorithm's implementation to terminate averaged over 1000 for each instance size, with the students' preference lists length fixed at 50

n_1	100	200	300	400	500	600	700	800	900	1000
Time	0.017	0.046	0.082	0.120	0.160	0.203	0.248	0.298	0.349	0.399

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 varied the length of each student's preference list for various values of x ($5 \leq x \leq 50$) in increments of 5; and with each of these parameters, we randomly generated 1000 instances. 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, which is displayed in Fig. 5, shows that as we varied the length of the preference list, there was no significant uplift in the number of instances that admitted a super-stable matching. In most cases, we observed that the proportion of instances that admit a super-stable matching is slightly higher when the preference list length is 50 compared to when the preference list length is 5. The result also shows that the proportion of instances that admit a super-stable matching decreases as the number of students increases. Further, we recorded the time taken for our algorithm's implementation to terminate, and as can be seen in Table 2, for an instance size of 1000 and preference list length 50, the algorithm terminates in approximately 0.4 second.

4.2.3 Experiment 2

In our second 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} ($0 \leq t_{d_s} \leq 0.05$) and the tie density in the lecturers' preference lists t_{d_l} ($0 \leq t_{d_l} \leq 0.05$), both in increments of 0.005. For each pair of tie densities in $t_{d_s} \times t_{d_l}$,

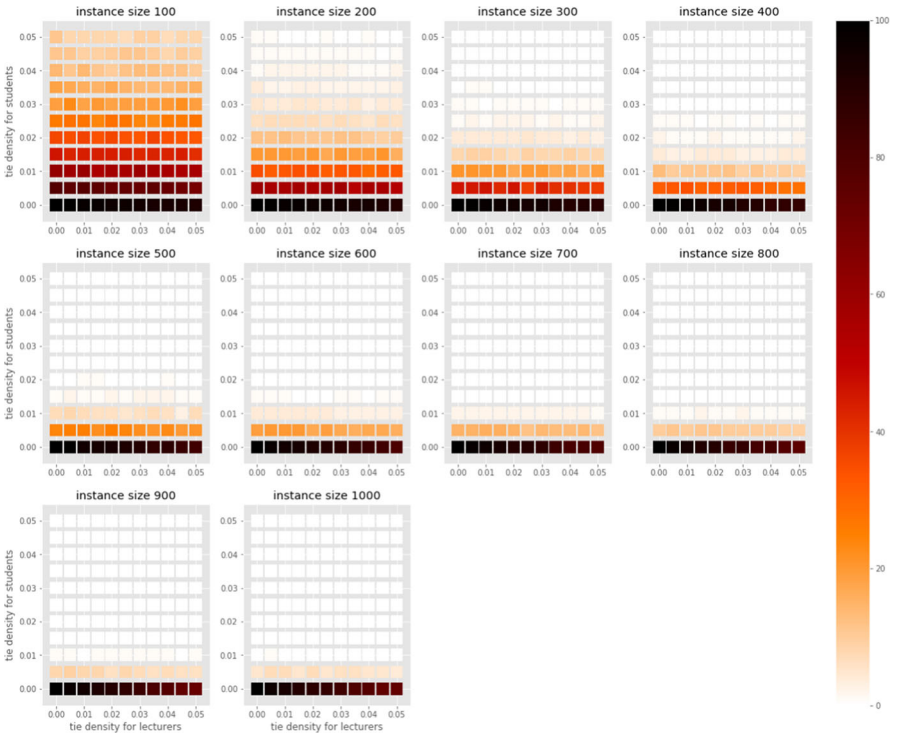

Fig. 6 Result for Experiment 2. Each of the coloured square boxes represents the proportion of the 1000 randomly-generated SPA- ST instances that admit a super-stable matching, with respect to the tie density in the students' and lecturers' preference lists. See the colour bar transition, as this proportion ranges from dark (100%) to light (0%)

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. Considering our discussion from Experiment 1, we fixed the length of each student's preference list at 50.

The result displayed in Fig. 6 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. When ties occur only in the lecturers' preference lists, we found that a significantly higher proportion of instances admit a super-stable matching—about 74% of the randomly-generated SPA- ST instances involving 1000 students admitted a super-stable matching. The confidence interval for this value is (0.71, 0.77). However, the reverse is the case when ties occur only in the students' preference lists. We have no explanation for this outcome.

5 Discussion and concluding remarks

In this paper, we have described a linear-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of SPA- ST. We established that for instances that do admit a super-stable matching, our algorithm produces the student-optimal super-stable matching, in the sense that each assigned student has the best project that she could obtain in any super-stable matching. We leave open the formulation of a lecturer-oriented counterpart to our algorithm.

Further, we carried out an empirical evaluation of our algorithm's implementation. The purpose of our experiments was to investigate how the nature of the preference lists affects the existence (or otherwise) of super-stable matchings in an arbitrary instance of SPA- ST. 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. When the ties occur only in the lecturers' preference lists, we found that a significantly higher proportion of instances admit a super-stable matching. However, the reverse is the case when the ties occur only in the students' preference lists.

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. 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, the following are other directions for future work. Let I be an arbitrary instance of SPA- ST.

1. Can we formalise the results on the probability of a super-stable matching existing in I? As mentioned in Section 1, this question has been partially explored for the Stable Roommates problem (Pittel and Irving 1994).
2. Is there a characterisation of the set of super-stable matchings in I in terms of a lattice structure? It is known that the set of super-stable matchings in an instance of SMT forms a distributive lattice under the dominance relation (Manlove 2002; Spieker 1995). To generalise this structural result for SPA- ST, ideas from Manlove (2002) and Spieker (1995) would certainly be useful.

Acknowledgements The authors would like to thank Frances Cooper, Kitty Meeks, Patrick Prosser, and also the anonymous reviewers, for valuable comments that helped to improve the presentation of this paper.

Appendix A: An IP model for super-stability in SPA-ST

A.1 Introduction

In this section, we describe an IP model for super-stability in SPA- ST. Although a super-stable matching in an instance of SPA- ST can be found in polynomial-time (as illustrated by Theorem 2), our reason for this is purely experimental. Let I be an instance of SPA- ST involving 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. We construct an IP model J of I as follows. Firstly, we create binary variables $x_{i,j} \in \{0, 1\}$ $(1 \leq i \leq n_1, 1 \leq j \leq n_2)$ for each acceptable pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ such that $x_{i,j}$ indicates whether s_i is assigned to p_j in a solution or not. Henceforth, we denote by S a solution in the IP model J, and we denote by M the matching derived from S in the following natural way: if $x_{i,j} = 1$ under S then s_i is assigned to p_j in M, otherwise s_i is not assigned to p_j in M.

A.2 Constraints

In this section, we give the set of constraints to ensure that the assignment obtained from a feasible solution in J is a matching, and that the matching admits no blocking pair.

Matching constraints

The feasibility of a matching can be ensured with the following three set of constraints.

$$\sum_{p_j \in A_i} x_{i,j} \leq 1 \qquad (1 \leq i \leq n_1), \tag{5}$$

$$\sum_{i=1}^{n_1} x_{i,j} \leq c_j \qquad (1 \leq j \leq n_2), \tag{6}$$

$$\sum_{i=1}^{n_1} \sum_{p_j \in P_k} x_{i,j} \leq d_k \qquad (1 \leq k \leq n_3)\ . \tag{7}$$

Note that Inequality (5) ensures that each student $s_i \in \mathcal{S}$ is not assigned to more than one project, while Inequalities (6) and (7) ensure that the capacity of each project $p_j \in \mathcal{P}$ and each lecturer $l_k \in \mathcal{L}$ is not exceeded.

Given an acceptable pair (s_i, p_j), we define $rank(s_i, p_j)$, the *rank* of p_j on s_i's preference list, to be $r + 1$, where r is the number of projects that s_i prefers to p_j. Clearly, projects that are tied together on s_i's preference list have the same rank. Given a lecturer $l_k \in \mathcal{L}$ and a student $s_i \in \mathcal{L}_k$, we define $rank(l_k, s_i)$, the *rank* of s_i on l_k's preference list, to be $r + 1$, where r is the number of students that l_k prefers to s_i.

Similarly, students that are tied together on l_k's preference list have the same rank. With respect to an acceptable pair (s_i, p_j), we define $S_{i,j} = \{p_{j'} \in A_i : rank(s_i, p_{j'}) < rank(s_i, p_j)\}$, the set of projects that s_i prefers to p_j. Let l_k be the lecturer who offers p_j. We also define $T_{i,j,k} = \{s_{i'} \in \mathcal{L}_k^j : rank(l_k, s_{i'}) < rank(l_k, s_i)\}$, the set of students that are better than s_i on the projected preference list of l_k for p_j. Finally, we define $D_{i,k} = \{s_{i'} \in \mathcal{L}_k : rank(l_k, s_{i'}) < rank(l_k, s_i)\}$, the set of students that are better than s_i on l_k's preference list.

In what follows, we fix an arbitrary acceptable pair (s_i, p_j) and we enforce constraints to ensure that (s_i, p_j) does not form a blocking pair for the matching M. Henceforth, l_k is the lecturer who offers p_j.

Blocking pair constraints First, we define $\theta_{i,j} = 1 - x_{i,j} - \sum_{p_{j'} \in S_{i,j}} x_{i,j'}$. Intuitively, $\theta_{i,j} = 1$ if and only if s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$ or is indifferent between them. Henceforth, if (s_i, p_j) forms a blocking for M then we refer to (s_i, p_j) as a blocking pair of type (i), type (ii) or type (iii), according as (s_i, p_j) satisfies condition (i), (ii), or (iii) of Definition 2, respectively. We describe the constraints to avoid these types of blocking pair as follows.

Type (i). First, we create a binary variable α_j in J such that if p_j is undersubscribed in M then $\alpha_j = 1$. We enforce this condition by imposing the following constraint.

$$c_j \alpha_j \geq c_j - \sum_{i'=1}^{n_1} x_{i',j}, \tag{8}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$. If p_j is undersubscribed in M then the RHS of Inequality (8) is at least 1 and this implies that $\alpha_j = 1$, otherwise α_j is not constrained. Next, we create a binary variable β_k in J such that if l_k is undersubscribed in M then $\beta_k = 1$. We enforce this condition by imposing the following constraint:

$$d_k \beta_k \geq d_k - \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'}, \tag{9}$$

where $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} = |M(l_k)|$. If l_k is undersubscribed in M then the RHS of Inequality (9) is at least 1 and this implies that $\beta_k = 1$, otherwise β_k is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (i) blocking pair for M.

$$\boxed{\theta_{i,j} + \alpha_j + \beta_k \leq 2 \ .} \tag{10}$$

Type (ii). We create a binary variable η_k in J such that if l_k is full in M then $\eta_k = 1$. We enforce this condition by imposing the following constraint.

$$d_k \eta_k \geq \left(1 + \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'}\right) - d_k \ . \tag{11}$$

If l_k is full in M then the RHS of Constraint (11) is at least 1 and this implies that $\eta_k = 1$, otherwise η_k is not constrained. Next, we create a binary variable $\delta_{i,k}$ in J such that if $s_i \in M(l_k)$, or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them, then $\delta_{i,k} = 1$. We enforce this condition by imposing the following constraint.

$$d_k \delta_{i,k} \geq \sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} - \sum_{s_{i'} \in D_{i,k}} \sum_{p_{j'} \in P_k} x_{i',j'} \ . \tag{12}$$

Note that if $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or l_k is indifferent between them, then the RHS of Constraint (12) is at least 1 and this implies that $\delta_{i,k} = 1$, otherwise $\delta_{i,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (ii) blocking pair for M.

$$\boxed{\theta_{i,j} + \alpha_j + \eta_k + \delta_{i,k} \leq 3 \ .} \tag{13}$$

Type (iii). Next we create a binary variable γ_j in J such that if p_j is full in M then $\gamma_j = 1$. We enforce this condition by imposing the following constraint.

$$c_j \gamma_j \geq \left(1 + \sum_{i'=1}^{n_1} x_{i',j}\right) - c_j \ . \tag{14}$$

where $\sum_{i'=1}^{n_1} x_{i',j} = |M(p_j)|$. If p_j is full in M then the RHS of Inequality (14) is at least 1 and this implies that $\gamma_j = 1$, otherwise γ_j is not constrained. Next, we create a binary variable $\lambda_{i,j,k}$ in J such that if l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, then $\lambda_{i,j,k} = 1$. We enforce this condition by imposing the following constraint.

$$c_j \lambda_{i,j,k} \geq \sum_{i'=1}^{n_1} x_{i',j} - \sum_{s_{i'} \in T_{i,j,k}} x_{i',j} \ . \tag{15}$$

Note that if l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them, then the RHS of Inequality (15) is at least 1 and this implies that $\lambda_{i,j,k} = 1$, otherwise $\lambda_{i,j,k}$ is not constrained. The following constraint ensures that (s_i, p_j) does not form a type (iii) blocking pair for M.

$$\boxed{\theta_{i,j} + \gamma_j + \lambda_{i,j,k} \leq 2 \ .} \tag{16}$$

A.3 Variables

We define a collective notation for each set of variables involved in J as follows:

$$A = \{\alpha_j : 1 \le j \le n_2\}, \qquad \Gamma = \{\gamma_j : 1 \le j \le n_2\},$$
$$B = \{\beta_k : 1 \le k \le n_3\}, \qquad \Delta = \{\delta_{i,k} : 1 \le i \le n_1, 1 \le k \le n_3\},$$
$$N = \{\eta_k : 1 \le k \le n_3\}, \qquad X = \{x_{i,j} : 1 \le i \le n_1, 1 \le j \le n_2\},$$
$$\Lambda = \{\lambda_{i,j,k} : 1 \le i \le n_1, 1 \le j \le n_2, 1 \le k \le n_3\}\ .$$

A.4 Objective function

On one hand, every super-stable matchings are of the same size, and thus nullifies the need for an objective function. On the other hand, optimization solvers require an objective function in addition to the variables and constraints in order to produce a solution. The objective function given below involves maximising the summation of all the $x_{i,j}$ binary variables.

$$\boxed{\max \sum_{i=1}^{n_1} \sum_{p_j \in A_i} x_{i,j}\ .} \tag{17}$$

Finally, we have constructed an IP model J of I comprising the set of integer-valued variables A, B, N, X, Γ, Δ, and Λ, the set of Inequalities (5) - (16) and an objective function (17). Note that J can then be used to construct a super-stable matching in I, should one exist.

A.5 Correctness of the IP model

Given an instance I of SPA- ST formulated as an IP model J using the above transformation, we present the following lemmas regarding the correctness of J.

Lemma 10 *A feasible solution S to J corresponds to a super-stable matching M in I.*

Proof Assume firstly that J has a feasible solution S. Let $M = \{(s_i, p_j) \in \mathcal{S} \times \mathcal{P} : x_{i,j} = 1\}$ be the assignment in I generated from S. We note that Inequality (5) ensures that each student is assigned in M to at most one project. Moreover, Inequalities (6) and (7) ensures that the capacity of each project and lecturer is not exceeded in M. Thus M is a matching. We will prove that Inequalities (8) - (16) ensures that M admits no blocking pair.

Suppose for a contradiction that there exists some acceptable pair (s_i, p_j) that forms a blocking pair for M, where l_k is the lecturer who offers p_j. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. Thus $\sum_{p_{j'} \in S_{i,j}} x_{i,j'} = 0$. Moreover, since s_i is not assigned to p_j in M, we have that $x_{i,j} = 0$. Thus $\theta_{i,j} = 1$.

Now suppose (s_i, p_j) forms a type (i) blocking pair for M. Then each of p_j and l_k is undersubscribed in M. Thus $\sum_{i'=1}^{n_1} x_{i',j} < c_j$ and $\sum_{i'=1}^{n_1} \sum_{p_{j'} \in P_k} x_{i',j'} < d_k$. This

implies that the RHS of Inequality (8) and the RHS of Inequality (9) is strictly greater than 0. Moreover, since S is a feasible solution to J, $\alpha_j = \beta_k = 1$. Hence, the LHS of Inequality (10) is strictly greater than 2, a contradiction to the feasibility of S.

Now suppose (s_i, p_j) forms a type (ii) blocking pair for M. Then p_j is undersubscribed in M and as explained above, $\alpha_j = 1$. Also, l_k is full in M and this implies that the RHS of Inequality (11) is strictly greater than 0. Since S is a feasible solution, we have that $\eta_k = 1$. Furthermore, either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or l_k is indifferent between them. In any of these cases, the RHS of Inequality (12) is strictly greater than 0. Thus $\delta_{i,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (13) is strictly greater than 3, a contradiction to the feasibility of S.

Finally, suppose (s_i, p_j) forms a type (iii) blocking pair for M. Then p_j is full in M and thus the RHS of Inequality (14) is strictly greater than 0. Since S is a feasible solution, we have that $\gamma_j = 1$. In addition, l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them. This implies that the RHS of Inequality (15) is strictly greater than 0. Thus $\lambda_{i,j,k} = 1$, since S is a feasible solution. Hence the LHS of Inequality (16) is strictly greater than 2, a contradiction to the feasibility of S. Hence M admits no blocking pair; and hence, M is a super-stable matching in I. □

Lemma 11 *A super-stable matching M in I corresponds to a feasible solution S to J.*

Proof Let M be a super-stable matching in I. First we set all the binary variables involved in J to 0. For each $(s_i, p_j) \in M$, we set $x_{i,j} = 1$. Since M is a matching, it is clear that Inequalities (5) - (7) is satisfied. For any acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$ such that s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them, we set $\theta_{i,j} = 1$. For any project $p_j \in \mathcal{P}$ such that p_j is undersubscibed in M, we set $\alpha_j = 1$ and thus Inequality (8) is satisfied. For any lecturer $l_k \in \mathcal{L}$ such that l_k is undersubscribed in M, we set $\beta_k = 1$ and thus Inequality (9) is satisfied.

Now, for Inequality (10) not to be satisfied, its LHS must be strictly greater than 2. This would only happen if there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1, \alpha_j = 1$ and $\beta_k = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them, and each of p_j and l_k is undersubscribed in M. Thus (s_i, p_j) forms a type (i) blocking pair for M, a contradiction to the super-stability of M. Hence, Inequality (10) is satisfied.

For any lecturer $l_k \in \mathcal{L}$ such that l_k is full in M, we set $\eta_k = 1$. Thus Inequality (11) is satisfied. Let (s_i, p_j) be an acceptable pair such that $p_j \in P_k$ and $(s_i, p_j) \notin M$. If $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them, we set $\delta_{i,k} = 1$. Thus Inequality (12) is satisfied. Suppose Inequality (13) is not satisfied. Then there exists $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$, where l_k is the lecturer who offers p_j, such that $\theta_{i,j} = 1$, $\alpha_j = 1$, $\eta_k = 1$ and $\delta_{i,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. In addition, p_j is undersubscribed in M, l_k is full in M and either $s_i \in M(l_k)$ or l_k prefers s_i to a worst student in $M(l_k)$ or is indifferent between them. Thus (s_i, p_j) forms a type (ii) blocking pair for M, a contradiction to the super-stability of M. Hence Inequality (13) is satisfied.

Finally, for any project $p_j \in \mathcal{P}$ such that p_j is full in M, we set $\gamma_j = 1$. Thus Inequality (14) is satisfied. Let l_k be the lecturer who offers p_j and let (s_i, p_j) be an acceptable pair. If l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between

them, we set $\lambda_{i,j,k} = 1$. Thus Inequality (15) is satisfied. Suppose Inequality (16) is not satisfied. Then there exists some $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$ such that $\theta_{i,j} = 1$, $\gamma_j = 1$ and $\lambda_{i,j,k} = 1$. This implies that either s_i is unassigned in M or s_i prefers p_j to $M(s_i)$ or is indifferent between them. In addition, p_j is full in M and l_k prefers s_i to a worst student in $M(p_j)$ or is indifferent between them. Thus (s_i, p_j) forms a type (iii) blocking pair for M, a contradiction to the super-stability of M. Hence, Inequality (16) is satisfied. Hence S, comprising the above assignments of values to the variables in $A \cup B \cup N \cup X \cup \Gamma \cup \Delta \cup \Lambda$, is a feasible solution to J. □

The following theorem is a consequence of Lemmas 10 and 11 .

Theorem 4 *Let I be an instance of* SPA- ST *and let J be the IP model for I as described above. A feasible solution to J corresponds to a super-stable matching in I. Conversely, a super-stable matching in I corresponds to a feasible solution to J.*

References

Abraham DJ, Irving RW, Manlove DF (2007) Two algorithms for the student-project allocation problem. J Discrete Algorithms 5(1):79–91

Abu El-Atta AH, Moussa MI (2009) Student project allocation with preference lists over (student,project) pairs. In: Proceedings of ICCEE 09: the 2nd international conference on computer and electrical engineering. IEEE, pp 375–379

Anwar AA, Bahaj AS (2003) Student project allocation using integer programming. IEEE Trans Educ 46(3):359–367

Brassard G, Bratley P (1996) Fundamentals of algorithmics. Prentice-Hall, London

Calvo-Serrano R, Guillén-Gosálbez G, Kohn S, Masters A (2017) Mathematical programming approach for optimally allocating students' projects to academics in large cohorts. Educ Chem Eng 20:11–21

Chiarandini M, Fagerberg R, Gualandi S (2019) Handling preferences in student-project allocation. Ann Oper Res 275(1):39–78

Cooper F, Manlove DF (2018a) A 3/2-approximation algorithm for the student-project allocation problem. In: Proceedings of SEA'18: the 17th international symposium on experimental algorithms, volume 103 of Leibniz international proceedings in informatics (LIPIcs), pp 8:1–8:13

Cooper F, Manlove DF (2018b) A 3/2-approximation algorithm for the student-project allocation problem. CoRR. arXiv:1804.02731

Harper PR, de Senna V, Vieira IT, Shahani AK (2005) A genetic algorithm for the project assignment problem. Comput Oper Res 32:1255–1265

Irving RW (1994) Stable marriage and indifference. Discrete Appl Math 48:261–272

Irving RW, Manlove DF, Scott S (2000) The hospitals/residents problem with Ties. In: Proceedings of SWAT '00: the 7th Scandinavian workshop on algorithm theory, volume 1851 of lecture notes in computer science. Springer, pp 259–271

Irving RW, Manlove DF, Scott S (2003) Strong stability in the hospitals/residents problem. In: Proceedings of STACS '03: the 20th annual symposium on theoretical aspects of computer science, volume 2607 of lecture notes in computer science. Springer, pp 439–450

Iwama K, Manlove D, Miyazaki S, Morita Y (1999) Stable marriage with incomplete lists and ties. In: Proceedings of ICALP'99: the 26th international colloquium on automata, languages, and programming, volume 1644 of lecture notes in computer science. Springer, pp 443–452

Iwama K, Miyazaki S, Yanagisawa H (2012) Improved approximation bounds for the student-project allocation problem with preferences over projects. J Discrete Algorithms 13:59–66

Ismaili A, Yahiro K, Yamaguchi T, Yokoo M (2019) Student-project-resource matching-allocation problems: two-sided matching meets resource allocation. In: Proceedings of AAMAS'19: the 18th international conference on autonomous agents and multiagent systems, pp 2033–2035

Kazakov D (2001) 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 12 April 2020)

Kwanashie A, Irving RW, Manlove DF, Sng CTS (2015) 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. Springer, pp 213–225

Manlove DF (2002) The structure of stable marriage with indifference. Discrete Appl Math 122(1–3):167–181

Manlove DF (2013) Algorithmics of matching under preferences. World Scientific, Singapore

Manlove DF, Irving RW, Iwama K, Miyazaki S, Morita Y (2002) Hard variants of stable marriage. Theor Comput Sci 276(1–2):261–279

Manlove DF, Milne D, Olaosebikan S (2018) An integer programming approach to the student-project allocation problem with preferences over projects. Lecture notes in computer science, vol 10856. Springer, pp 313–325

Manlove DF, O'Malley G (2008) Student project allocation with preferences over projects. J Discrete Algorithms 6:553–560

Olaosebikan S, Manlove D (2018) Super-stability in the student-project allocation problem with ties. In: Proceedings of COCOA'18: the 12th annual international conference on combinatorial optimization and applications, volume 11346 of lecture notes in computer science. Springer, pp 357–371

Pittel BG, Irving RW (1994) An upper bound for the solvability probability of a random stable roommates instance. Random Struct Algorithms 5:465–486

Roth AE (1984) The evolution of the labor market for medical interns and residents: a case study in game theory. J Polit Econ 92(6):991–1016

Roth AE (1990) New physicians: a natural experiment in market organization. Science 250:1524–1528

Roth AE (1991) A natural experiment in the organization of entry levellabor markets: regional markets for new physicians and surgeons inthe U.K. Am Econ Rev 81:415–440

Spieker B (1995) The set of super-stable marriages forms a distributive lattice. Discrete Appl Math 58:79–84