An Algorithm for Strong Stability in the Student-Project Allocation Problem with Ties

Sofiat Olaosebikan* and David Manlove**

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

Abstract. We study a variant of the *Student-Project Allocation problem with lecturer preferences over Students* where ties are allowed in the preference lists of students and lecturers (SPA-ST). We investigate the concept of *strong stability* in this context. Informally, a matching is *strongly stable* if there is no student and lecturer l such that if they decide to form a private arrangement outside of the matching via one of l's proposed projects, then neither party would be worse off and at least one of them would strictly improve. We describe the first polynomial-time algorithm to find a strongly stable matching or to report that no such matching exists, given an instance of SPA-ST. Our algorithm runs in $O(m^2)$ time, where m is the total length of the students' preference lists.

1 Introduction

Matching problems, which generally involve the assignment of a set of agents to another set of agents based on preferences, have wide applications in many real-world settings, including, for example, allocating junior doctors to hospitals [25] and assigning students to projects [15]. In the context of assigning students to projects, each project is proposed by one lecturer and each student is required to provide a preference list over the available projects that she finds acceptable. Also, lecturers may provide a preference list over the students that find their projects acceptable, and/or over the projects that they propose. Typically, each project and lecturer have a specific capacity denoting the maximum number of students that they can accommodate. The goal is to find a *matching*, i.e., an assignment of students to projects that respects the stated preferences, such that each student is assigned at most one project, and the capacity constraints on projects and lecturers are not violated — the so-called *Student-Project Allocation problem* (SPA) [1,6,19].

Two major models of SPA exist in the literature: one permits preferences only from the students [15], while the other permits preferences from the students and lecturers [1,14]. In the latter case, three different variants have been studied

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

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

based on the nature of the lecturers' preference lists. These include SPA with lecturer preferences over (i) students [14], (ii) projects [12,21,22], and (iii) (student, project) pairs [2]. Outwith assigning students to projects, applications of each of these three variants can be seen in multi-cell networks where the goal is to find a stable assignment of users to channels at base-stations [3,4,5].

In this work, we will concern ourselves with variant (i), i.e., the *Student-Project Allocation problem with lecturer preferences over Students* (SPA-S). In this context, it has been argued in [25] that a natural property for a matching to satisfy is that of *stability*. Informally, a *stable matching* ensures that no student and lecturer would have an incentive to deviate from their current assignment. Abraham *et al.* [1] described two linear-time algorithms to find a stable matching in an instance of SPA-S where the preference lists are strictly ordered. In their paper, they also proposed an extension of SPA-S where the preference lists may include ties, 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 are possible [8,9,10]. We give an informal definition in what follows. Suppose that M is a matching in an instance of SPA-ST. Then M is (i) weakly stable, (ii) strongly stable, or (iii) super-stable, if there is no student and lecturer l such that if they decide to become assigned outside of M via one of l's proposed projects, respectively,

(i) both of them would strictly improve,
(ii) one of them would be better off, and the other would not be worse off
(iii) neither of them would be worse off.

Existing results in SPA-ST. Every instance of SPA-ST admits a weakly stable matching, which could be of different sizes [20]. Moreover, the problem of finding a maximum size weakly stable matching (MAX-SPA-ST) is NP-hard [11,20], even for the so-called *Stable Marriage problem with Ties and Incomplete lists* (SMTI). Cooper and Manlove [7] described a $\frac{3}{2}$-approximation algorithm for MAX-SPA-ST. On the other hand, Irving *et al.* argued in [9] that super-stability is a natural and most robust solution concept to seek in cases where agents have incomplete information. Recently, Olaosebikan and Manlove [23] showed that if an instance of SPA-ST admits a super-stable matching M, then all weakly stable matchings in the instance are of the same size (equal to the size of M), and match exactly the same set of students. The main result of their paper was a polynomial-time algorithm to find a super-stable matching or report that no such matching exists, given an instance of SPA-ST. Their algorithm runs in $O(L)$ time, where L is the total length of all the preference lists.

It was motivated in [10] that weakly stable matching may be undermined by bribery or persuasion, in practical applications of the *Hospitals-Residents problem with Ties* (HRT). In what follows, we give a corresponding argument for an instance I of SPA-ST. Suppose that M is a weakly stable matching in I, and suppose that a student s_i prefers a project p_j (where p_j is offered by lecturer l_k) to her assigned project in M, say $p_{j'}$ (where $p_{j'}$ is offered by a lecturer different from l_k). Suppose further that p_j is full and l_k is indifferent between s_i and one of the worst student/s assigned to p_j in M, say $s_{i'}$. Clearly, the pair (s_i, p_j) does not constitute a blocking pair for the weakly stable matching M, as l_k would not improve by taking on s_i in the place of $s_{i'}$. However, s_i might be overly invested in p_j that she is even ready to persuade or even bribe l_k to reject $s_{i'}$ and accept her instead; l_k being indifferent between s_i and $s_{i'}$ may decide to accept s_i's proposal. We can reach a similar argument if the roles are reversed. However, if M is strongly stable, it cannot be potentially undermined by this type of (student, project) pair.

Henceforth, if a SPA-ST instance admits a strongly stable matching, we say that such an instance is solvable. Unfortunately not every instance of SPA-ST is solvable. To see this, consider the case where there are two students, two projects and two lecturers, the capacity of each project and lecturer is 1, the students have exactly the same strictly ordered preference list of length 2, and each of the lecturers preference list is a single tie of length 2 (any matching will be undermined by a student and lecturer that are not matched together). However, it should be clear from the discussions above that in cases where a strongly stable matching exists, it should be preferred over a matching that is merely weakly stable. Previous results for strong stability in the literature include [8,10,13,16,18].

Our contribution. We present the first polynomial-time algorithm to find a strongly stable matching or report that no such matching exists, given an instance of SPA-ST – thus solving an open problem given in [1,23]. Our algorithm is student-oriented, which implies that if the given instance is solvable then our algorithm will output a solution in which each student has at least as good a project as she could obtain in any strongly stable matching. We note that our algorithm is a non-trivial extension of the strong stability algorithms for SMT (*Stable Marriage problem with Ties*), SMTI and HRT described in [8,10,18] (we discuss this further in [24, Sect. 4.3]).

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 three stability concepts in Sect. 2. We describe our algorithm for SPA-ST under strong stability in Sect. 3. Further, in Sect. 3, we also illustrate an execution of our algorithm with respect to an instance of SPA-ST before moving on to present the algorithm's correctness and complexity results (all omitted proofs can be found in [24, Sect. 4.5]). Finally, we present some potential direction for future work in Sect. 4.

2 Preliminary definitions

In this section, we give a formal definition of SPA-S as described in the literature [1,23]. We also give a formal definition of SPA-ST as described in [23], which is a generalisation of SPA-S in which preference lists can include ties.

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 her preference list. We say that s_i finds p_j *acceptable* if p_j appears on s_i's preference list. 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 $\mathcal{P}$, and l_k provides a preference list, denoted by $\mathcal{L}_k$, ranking in strict order of preference those students who find at least one project in P_k acceptable. Also l_k has a capacity $d_k \in \mathbb{Z}^+$, indicating the maximum number of students she is willing to supervise. Similarly each project $p_j \in \mathcal{P}$ has a capacity $c_j \in \mathbb{Z}^+$ indicating the maximum number of students that it can accommodate.

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

An *assignment* M is a subset of $\mathcal{S} \times \mathcal{P}$ such that $(s_i, p_j) \in M$ implies that s_i finds p_j acceptable. If $(s_i, p_j) \in M$, we say that s_i *is assigned to* p_j, and p_j *is assigned* s_i. For convenience, if s_i is assigned in M to p_j, where p_j is offered by l_k, we may also say that s_i *is assigned to* l_k, and l_k *is assigned* s_i. For any project $p_j \in \mathcal{P}$, we denote by $M(p_j)$ the set of students assigned to p_j in M. Project p_j is *undersubscribed*, *full* or *oversubscribed* according as $|M(p_j)|$ is less than, equal to, or greater than c_j, respectively. Similarly, for any lecturer $l_k \in \mathcal{L}$, we denote by $M(l_k)$ the set of students assigned to l_k in M. Lecturer l_k is *undersubscribed*, *full* or *oversubscribed* according as $|M(l_k)|$ is less than, equal to, or greater than d_k, respectively. A *matching* M is an assignment such that $|M(s_i)| \leq 1$, $|M(p_j)| \leq c_j$ and $|M(l_k)| \leq d_k$. If s_i is assigned to some project in M, we let $M(s_i)$ denote that project; otherwise $M(s_i)$ is undefined.

2.2 Ties in the preference lists

We now give a formal definition, similar to the one given in [23], for 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, in what follows we consider a non-tied entry in a preference list as a tie of length one. We denote by SPA-ST the generalisation of SPA-S in which the preference list of each student (respectively lecturer) comprises a strict ranking of ties, each comprising one or more projects (respectively students). An example SPA-ST instance I_1 is given in Fig. 1, which involves the set of students $\mathcal{S} = \{s_1, s_2, s_3\}$, the set of projects $\mathcal{P} = \{p_1, p_2, p_3\}$ and the set of lecturers $\mathcal{L} = \{l_1, l_2\}$. Ties in the preference lists are indicated by round brackets.

In the context of SPA-ST, we assume that all notation and terminology carries over from SPA-S with the exception of stability, which we now define. When ties appear in the preference lists, three types of stability arise, namely *weak stability, strong stability and super-stability* [9,10]. For our purpose in this paper, we only

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

Fig. 1. An example SPA-ST instance I_1.

give a formal definition of strong stability in the context of SPA-ST. Henceforth, 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.

Definition 1 (strong stability). We say that M is *strongly stable* in I if it admits no blocking pair, where a *blocking pair* of M is an acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ such that either (1a and 1b) or (2a and 2b) holds as follows:

(1a) either s_i is unassigned in M, or s_i prefers p_j to $M(s_i)$;
(1b) either (i), (ii), or (iii) holds as follows:
- (i) p_j is undersubscribed and l_k is undersubscribed;
- (ii) p_j is undersubscribed, l_k is full, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student/s in $M(l_k)$ or is indifferent between them;
- (iii) p_j is full and l_k prefers s_i to the worst student/s in $M(p_j)$ or is indifferent between them.

(2a) s_i is indifferent between p_j and $M(s_i)$;
(2b) either (i), (ii), or (iii) holds as follows:
- (i) p_j is undersubscribed, l_k is undersubscribed and $s_i \notin M(l_k)$;
- (ii) p_j is undersubscribed, l_k is full, $s_i \notin M(l_k)$, and l_k prefers s_i to the worst student/s in $M(l_k)$;
- (iii) p_j is full and l_k prefers s_i to the worst student/s in $M(p_j)$.

Some intuition for the strong stability definition is given in [24, Sect. 3]. In the remainder of this paper, any usage of the term *blocking pair* refers to the version of this term for strong stability as defined above.

3 An algorithm for SPA-ST under strong stability

In this section we present our algorithm for SPA-ST under strong stability, which we will refer to as `Algorithm SPA-ST-strong`. In Sect. 3.1, we give some definitions relating to the algorithm. In Sect. 3.2, we give a description of our algorithm and present it in pseudocode form. We illustrate an execution of our algorithm with respect to a SPA-ST instance in Sect. 3.3. Finally, we present the algorithm's correctness results in Sect. 3.4.

3.1 Definitions relating to the algorithm

Given a pair $(s_i, p_j) \in M$, for some strongly stable matching M in I, we call (s_i, p_j) a *strongly stable pair*. During the execution of the algorithm, students become *provisionally assigned* to projects (and implicitly to lecturers), and it is possible for a project (and lecturer) to be provisionally assigned a number of students that exceeds its capacity. We describe a project (respectively lecturer) as *replete* if at any time during the execution of the algorithm it has been full or oversubscribed. We say that a project (respectively lecturer) is *non-replete* if it is not replete.

The *provisional assignment graph* is an undirected bipartite graph $G = (S \cup P, E)$, with $S \subseteq \mathcal{S}$ and $P \subseteq \mathcal{P}$ such that there is an edge $(s_i, p_j) \in E$ if and only if s_i is provisionally assigned to p_j. During the execution of the algorithm, it is possible for a student to be adjacent to more than one project in G. Thus, we denote by $G(s_i)$ the set of projects adjacent to s_i in G. Given a project $p_j \in P$, we denote by $G(p_j)$ the set of students who are provisionally assigned to p_j in G and we let $d_G(p_j) = |G(p_j)|$. Similarly, we denote by $G(l_k)$ the set of students who are provisionally assigned to a project offered by l_k in G and we let $d_G(l_k) = |G(l_k)|$.

As stated earlier, for a project p_j, it is possible that $d_G(p_j) > c_j$ at some point during the algorithm's execution. Thus, we denote by $q_j = \min\{c_j, d_G(p_j)\}$ the *quota of* p_j in G, which is the minimum between p_j's capacity and the number of students provisionally assigned to p_j in G. Similarly, for a lecturer l_k, it is possible that $d_G(l_k) > d_k$ at some point during the algorithm's execution. At this point, we denote by $\alpha_k = \sum\{q_j : p_j \in P_k \cap P\}$ the total quota of projects offered by l_k that is provisionally assigned to students in G and we denote by $q_k = \min\{d_k, d_G(l_k), \alpha_k\}$ the *quota of* l_k in G.

The algorithm proceeds by deleting from the preference lists certain (s_i, p_j) pairs that are not strongly 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 $(s_i, p_j) \in E$ we delete the edge from G. By the *head* and *tail* of a preference list at a given point we mean the first and last tie respectively on that list after any deletions might have occurred (recalling that a tie can be of length 1). Given a project p_j, we say that a student s_i is *dominated in* $\mathcal{L}_k^j$ if s_i is worse than at least c_j students who are provisionally assigned to p_j. The concept of a student becoming dominated in a lecturer's preference list is defined in a slightly different manner.

Definition 2 (Dominated in $\mathcal{L}_k$). At a given point during the algorithm's execution, let α_k and $d_G(l_k)$ be as defined above. We say that a student s_i is *dominated in* $\mathcal{L}_k$ if $\min\{d_G(l_k), \alpha_k\} \geq d_k$, and s_i is worse than at least d_k students who are provisionally assigned in G to a project offered by l_k.

Definition 3 (Lower rank edge). We define an edge $(s_i, p_j) \in E$ as a *lower rank edge* if s_i is in the tail of $\mathcal{L}_k$ and $\min\{d_G(l_k), \alpha_k\} > d_k$.

Definition 4 (Bound). Given an edge $(s_i, p_j) \in E$, we say that s_i is *bound to* p_j if (i) p_j is not oversubscribed or s_i is not in the tail of $\mathcal{L}_k^j$ (or both), and (ii)

(s_i, p_j) is not a lower rank edge or s_i is not in the tail of $\mathcal{L}_k$ (or both). If s_i is bound to p_j, we may also say that (s_i, p_j) is a *bound edge*. Otherwise, we refer to it as an *unbound edge*.[1]

We form a *reduced assignment graph* $G_r = (S_r, P_r, E_r)$ from a provisional assignment graph G as follows. For each edge $(s_i, p_j) \in E$ such that s_i is bound to p_j, we remove the edge (s_i, p_j) from G_r and we reduce the quota of p_j in G_r (and intuitively l_k[2]) by one. Further, we remove all other unbound edges incident to s_i in G_r. Each isolated student vertex is then removed from G_r. Finally, if the quota of any project is reduced to 0, or p_j becomes an isolated vertex, then p_j is removed from G_r. For each surviving p_j in G_r, we denote by q_j^* the *revised quota of* p_j, where q_j^* is the difference between p_j's quota in G (i.e., q_j) and the number of students that are bound to p_j. Similarly, we denote by q_k^* the *revised quota of* l_k in G_r, where q_k^* is the difference between l_k's quota in G (i.e., q_k) and the number of students that are bound to a project offered by l_k. Further, for each l_k who offers at least one project in G_r, we let $n = \sum\{q_j^* : p_j \in P_k \cap P_r\} - q_k^*$, where n is the difference between the total revised quota of projects in G_r that are offered by l_k and the revised quota of l_k in G_r. Now, if $n \leq 0$, we do nothing; otherwise, we extend G_r as follows. We add n dummy student vertices to S_r. For each of these dummy vertex, say s_{d_i}, and for each project $p_j \in P_k \cap P_r$ that is adjacent to a student vertex in S_r via a lower rank edge, we add the edge (s_{d_i}, p_j) to E_r.[3]

Given a set $X \subseteq S_r$ of students, define $\mathcal{N}(X)$, the *neighbourhood of* X, to be the set of project vertices adjacent in G_r to a student in X. If for all subsets X of S_r, each student in X can be assigned to one project in $\mathcal{N}(X)$, without exceeding the revised quota of each project in $\mathcal{N}(X)$ (i.e., $|X| \leq \sum\{q_j^* : p_j \in \mathcal{N}(X)\}$ for all $X \subseteq S_r$); then we say G_r admits a *perfect matching* that saturates S_r.

Definition 5 (Critical set). It is well known in the literature [17] that if G_r does not admit a perfect matching that saturates S_r, then there must exist a *deficient* subset $Z \subseteq S_r$ such that $|Z| > \sum\{q_j^* : p_j \in \mathcal{N}(Z)\}$. To be precise, the *deficiency* of Z is defined by $\delta(Z) = |Z| - \sum\{q_j^* : p_j \in \mathcal{N}(Z)\}$. The *deficiency* of G_r, denoted $\delta(G_r)$, is the maximum deficiency taken over all subsets of S_r. Thus, if $\delta(Z) = \delta(G_r)$, we say that Z is a *maximally deficient* subset of S_r, and we refer to Z as a *critical set*.

We denote by P_R the set of replete projects in G and we denote by P_R^* a subset of projects in P_R which is obtained as follows. For each project $p_j \in P_R$, let l_k

[1] An edge $(s_i, p_j) \in E$ can change state from *bound* to *unbound*, but not vice versa.

[2] If s_i is bound to more than one projects offered by l_k, for all the bound edges involving s_i and these projects that we remove from G_r, we only reduce l_k's quota in G_r by one.

[3] An intuition as to why we add dummy students to G_r is as follows. Given a lecturer l_k whose project is provisionally assigned to a student in G_r. If $q_k^* < \sum\{q_j^* : p_j \in P_k \cap P_r\}$, then we need n dummy students to offset the difference between $\sum\{q_j^* : p_j \in P_k \cap P_r\}$ and q_k^*, so that we don't oversubscribe l_k in any maximum matching obtained from G_r.

be the lecturer who offers p_j. For each student s_i such that (s_i, p_j) has been deleted, we add p_j to P_R^* if (i) and (ii) holds as follows:

(i) either s_i is unassigned in G, or $(s_i, p_{j'}) \in G$ where s_i prefers p_j to $p_{j'}$, or $(s_i, p_{j'}) \in G$ and s_i is indifferent between p_j and $p_{j'}$ where $p_{j'} \notin P_k$;
(ii) either l_k is undersubscribed, or l_k is full and either $s_i \in G(l_k)$ or l_k prefers s_i to some student assigned to l_k in G.

Definition 6 (Feasible matching). A *feasible matching* in the final provisional assignment graph G is a matching M obtained as follows:

1. Let G^* be the subgraph of G induced by the students who are adjacent to a project in P_R^*. First, find a maximum matching M^* in G^*;
2. Using M^* as an initial solution, find a maximum matching M in G.

3.2 Description of the algorithm

`Algorithm SPA-ST-strong`, described in Algorithm 1, begins by initialising an empty bipartite graph G which will contain the provisional assignments of students to projects (and intuitively to lecturers). We remark that such assignments (i.e., edges in G) can subsequently be broken during the algorithm's execution.

The `while` loop of the algorithm involves each student s_i who is not adjacent to any project in G and who has a non-empty list applying in turn to each project p_j at the head of her list. Immediately, s_i becomes provisionally assigned to p_j in G (and to l_k). If, by gaining a new provisional assignee, project p_j becomes full or oversubscribed then we set p_j as replete. Further, for each student s_t in $\mathcal{L}_k^j$, such that s_t is dominated in $\mathcal{L}_k^j$, we delete the pair (s_t, p_j). As we will prove later, such pairs cannot belong to any strongly stable matching. Similarly, if by gaining a new provisional assignee, l_k becomes full or oversubscribed then we set l_k as replete. For each student s_t in $\mathcal{L}_k$, such that s_t is dominated in $\mathcal{L}_k$ and for each project $p_u \in P_k$ that s_t finds acceptable, we delete the pair (s_t, p_u). This continues until every student is provisionally assigned to one or more projects or has an empty list. At the point where the `while` loop terminates, we form the reduced assignment graph G_r and we find the critical set Z of students in G_r (we describe how to find Z on Page 9). As we will see later, no project $p_j \in \mathcal{N}(Z)$ can be assigned to any student in the tail of $\mathcal{L}_k^j$ in any strongly stable matching, so all such pairs are deleted.

At the termination of the inner `repeat-until` loop in line 21, i.e., when Z is empty, if some project p_j that is replete ends up undersubscribed, we carry out some certain deletions[4]. 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

[4] This type of deletion was also carried out in `Algorithm SPA-ST-super` for super-stability [23].

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

At the termination of the outer `repeat-until` loop in line 30, if a student is adjacent in G to a project p_j via a bound edge, then we may potentially carry out extra deletions. First, we let l_k be the lecturer that offers p_j and we let U be the set of projects that are adjacent to s_i in G via an unbound edge. For each project $p_u \in U \setminus P_k$, it turns out that the pair (s_i, p_u) cannot belong to any strongly stable matching, thus we delete all such pairs. Finally, we let M be any feasible matching in the provisional assignment graph G. If M is strongly stable relative to the given instance I then M is output as a strongly stable matching in I. Otherwise, the algorithm reports that no strongly stable matching exists in I. We present `Algorithm SPA-ST-strong` in pseudocode form in Algorithm 1.

Finding the critical set. Consider the reduced assignment graph $G_r = (S_r, P_r, E_r)$ formed from G at a given point during the algorithm's execution (at line 15). To find the critical set of students in G_r, first we need to construct a maximum matching M_r in G_r, with respect to the revised quota q_j^*, for each $p_j \in P_r$. In this context, a *matching* $M_r \subseteq E_r$ is such that $|M_r(s_i)| \leq 1$ for all $s_i \in S_r$, and $|M_r(p_j)| \leq q_j^*$ for all $p_j \in P_r$. We describe how to construct M_r as follows:

1. Let G_r' be the subgraph of G_r induced by the dummy students adjacent to a project in G_r. First, find a maximum matching M_r' in G_r'.
2. Using M_r' as an initial solution, find a maximum matching M_r in G_r.[5]

Given a maximum matching M_r in the reduced assignment graph G_r, the critical set Z consists of the set U of unassigned students together with the set U' of students reachable from a student in U via an alternating path (see [24, Lemma 1] for a proof).

3.3 Example algorithm execution

In this section, we illustrate an execution of `Algorithm SPA-ST-strong` with respect to the SPA-ST instance I_3 shown in Fig. 2 (Page 10), which involves the set of students $\mathcal{S} = \{s_i : 1 \leq i \leq 8\}$, the set of projects $\mathcal{P} = \{p_j : 1 \leq j \leq 6\}$ and the set of lecturers $\mathcal{L} = \{l_k : 1 \leq k \leq 3\}$. The algorithm starts by initialising the bipartite graph $G = \{\}$, which will contain the provisional assignment of students to projects. We assume that the students become provisionally assigned to each project at the head of their list in subscript order. Figs. 3, 4 and 5 illustrate how this execution of `Algorithm SPA-ST-strong` proceeds with respect to I_3.

[5] By making sure that all the dummy students are matched in step 1, we are guaranteed that no lecturer is oversubscribed with non-dummy students in G_r.

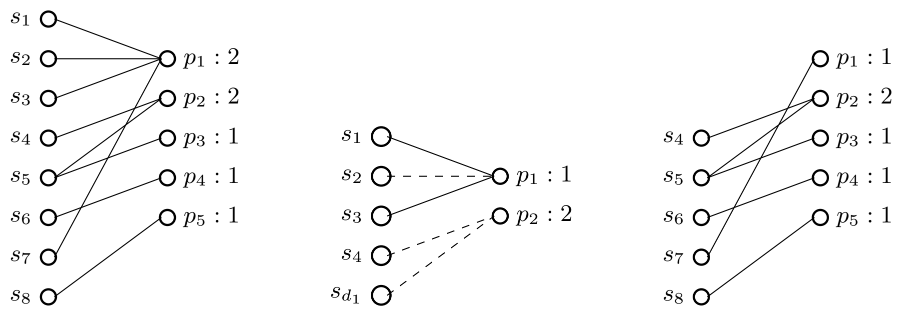
Algorithm 1 `Algorithm SPA-ST-strong`

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

```
G ← ∅
repeat
    repeat
        while some student s_i is unassigned and has a non-empty list do
            for each project p_j at the head of s_i's list do
                l_k ← lecturer who offers p_j
                add the edge (s_i, p_j) to G
                if p_j is full or oversubscribed then
                    for each student s_t dominated in L_k^j do
                        delete (s_t, p_j)
                if l_k is full or oversubscribed then
                    for each student s_t dominated in L_k do
                        for each project p_u ∈ P_k ∩ A_t  do
                            delete (s_t, p_u)
        form the reduced assignment graph G_r
        find the critical set Z of students
        for each project p_u ∈ N(Z) do
            l_k ← lecturer who offers p_u
            for each student s_t at the tail of L_k^u  do
                delete (s_t, p_u)
    until Z is empty
    for each p_j ∈ P do
        if p_j is replete and p_j is undersubscribed then
            l_k ← lecturer who offers p_j
            s_r ← most preferred student rejected from p_j in L_k^j {any if > 1}
            if the students at the tail of L_k are no better than s_r then
                for each student s_t at the tail of L_k do
                    for each project p_u ∈ P_k ∩ A_t  do
                        delete (s_t, p_u)
until every unassigned student has an empty list
for each student s_i in G  do
    if s_i is adjacent in G to a project p_j via a bound edge then
        l_k ← lecturer who offers p_j
        U ← unbound projects adjacent to s_i in G
        for each p_u ∈ U \ P_k do
            delete (s_i, p_u)
M ← a feasible matching in G
if M is a strongly stable matching in I then
    return M
else
    return "no strongly stable matching exists in I"
```

`Student preferences`	`Lecturer preferences`	offers
s_1: p_1 p_6	$\{3\}$ l_1: s_8 s_7 $(s_1$ s_2 $s_3)$ $(s_4$ $s_5)$ s_6	p_1, p_2
s_2: p_1 p_2	$\{2\}$ l_2: s_6 s_5 $(s_7$ $s_3)$	p_3, p_4
s_3: $(p_1$ $p_4)$	$\{3\}$ l_3: $(s_1$ $s_4)$ s_8	p_5, p_6
s_4: p_2 $(p_5$ $p_6)$		
s_5: $(p_2$ $p_3)$		
s_6: $(p_2$ $p_4)$		
s_7: p_3 p_1	Project capacities: $c_1 = c_2 = c_6 = 2,\ c_3 = c_4 = c_5 = 1$	
s_8: p_5 p_1	Lecturer capacities: $d_1 = d_3 = 3,\ d_2 = 2$	

Fig. 2. An instance I_3 of SPA-ST.



(a) The provisional assignment graph $G^{(1)}$ at the end of the `while` loop, with the quota of each project labelled beside it.

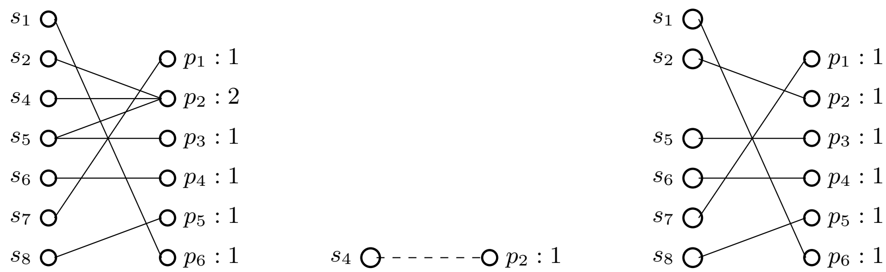
(b) The reduced assignment graph $G_r^{(1)}$, with the revised quota of each project labelled beside it. The collection of the dashed edges is the maximum matching $M_r^{(1)}$.

(c) The provisional assignment graph $G^{(1)}$ at the termination of iteration (1).

Fig. 3. Iteration (1).

Iteration 1: At the termination of the `while` loop during the first iteration of the inner `repeat-until` loop, every student, except s_3, s_6 and s_7, is provisionally assigned to every project in the first tie on their preference list. Edge $(s_3, p_4) \notin G^{(1)}$ because (s_3, p_4) was deleted as a result of s_6 becoming provisionally assigned to p_4, causing s_3 to be dominated in $\mathcal{L}_2^4$. Also, edge $(s_6, p_2) \notin G^{(1)}$ because (s_6, p_2) was deleted as a result of s_4 becoming provisionally assigned to p_2, causing s_6 to be dominated in $\mathcal{L}_1$ (at that point in the algorithm, $\min\{d_G(l_1), \alpha_1\} = \min\{4, 3\} = 3 = d_1$ and s_6 is worse than at least d_1 students who are provisionally assigned to l_1). Finally, edge $(s_7, p_3) \notin G^{(1)}$ because (s_7, p_3) was deleted as a result of s_5 becoming provisionally assigned to p_5, causing s_7 to be dominated in $\mathcal{L}_2^3$.

To form $G_r^{(1)}$, the bound edges $(s_5, p_3), (s_6, p_4), (s_7, p_1)$ and (s_8, p_5) are removed from the graph. We can verify that edges (s_4, p_2) and (s_5, p_2) are unbound, since they are lower rank edges for l_1. Also, since p_1 is oversubscribed, and each of s_1, s_2 and s_3 is at the tail of $\mathcal{L}_1^1$, edges (s_1, p_1), (s_2, p_1) and (s_3, p_1) are unbound. Further, the revised quota of l_1 in $G_r^{(1)}$ is 2, and the total revised quota of projects offered by l_1 (i.e., p_1 and p_2) is 3. Thus, we add one dummy student vertex s_{d_1} to G_r^1, and we add an edge between s_{d_1} and p_2 (since p_2 is the only project in $G_r^{(1)}$ adjacent to a student in the tail of $\mathcal{L}_1$ via a lower rank edge). With respect to the maximum matching $M_r^{(1)}$, it is clear that the critical set $Z^{(1)} = \{s_1, s_2, s_3\}$, thus we delete the edges (s_1, p_1), (s_2, p_1) and (s_3, p_1) from $G^{(1)}$; and the inner `repeat-until` loop is reactivated.



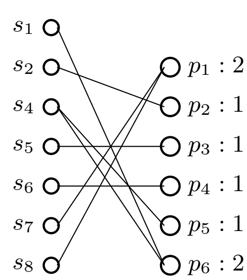
(a) The provisional assignment graph $G^{(2)}$ at the end of the `while` loop.

(b) The reduced assignment graph $G_r^{(2)}$.

(c) The provisional assignment graph $G^{(2)}$ at the termination of iteration (2).

Fig. 4. Iteration (2).

Iteartion 2: At the beginning of this iteration, each of s_1 and s_2 is unassigned and has a non-empty list; thus we add edges (s_1, p_6) and (s_2, p_2) to the provisional assignment graph obtained at the termination of iteration (1) to form $G_r^{(2)}$. It can be verified that every edge in $G_r^{(2)}$, except (s_4, p_2) and (s_5, p_2), is a bound edge. Clearly, the critical set $Z^{(2)} = \emptyset$, thus the inner `repeat-until` loop terminates. At this point, project p_1, which was replete during iteration (1), is undersubscribed in iteration (2). Moreover, the students at the tail of $\mathcal{L}_1$ (i.e., s_4 and s_5) are no better than s_3, where s_3 is one of the most preferred students rejected from p_1 according to $\mathcal{L}_1^1$; thus we delete edges (s_4, p_2) and (s_5, p_2). The outer `repeat-until` loop is then reactivated (since s_4 is unassigned and has a non-empty list).



(a) The provisional assignment graph $G^{(3)}$ at the end of the `while` loop.

Fig. 5. Iteration (3).

Iteration 3: At the beginning of this iteration, the only student that is unassigned and has a non-empty list is s_4; thus we add edges (s_4, p_5) and (s_4, p_6) to the provisional assignment graph obtained at the termination of iteration (2) to form $G_r^{(3)}$. The provisional assignment of s_4 to p_5 led to p_5 becoming oversubscribed; thus (s_8, p_5) is deleted (since s_8 is dominated on $\mathcal{L}_3^5$). Further, s_8 becomes provisionally assigned to p_1. It can be verified that all the edges in $G_r^{(3)}$ are bound edges. Moreover, the reduced assignment graph $G_r^{(3)} = \emptyset$.

Again, every unassigned students has an empty list. We also have that a project p_2, which was replete in iteration (2), is undersubscribed in iteration (3). However, no further deletion is carried out in line 29 of the algorithm, since the student at the tail of $\mathcal{L}_1$ (i.e., s_2) is better than s_4 and s_5, where s_4 and s_5 are the most preferred students rejected from p_2 according to $\mathcal{L}_1^2$. Hence, the `repeat-until` loop terminates. We observe that $P_R^* = \{p_5\}$, since (s_8, p_5) has been deleted, s_8 prefers p_5 to her provisional assignment in G and l_3 is undersubscribed. Thus we need to ensure p_5 fills up in the feasible matching M constructed from G, so as to avoid (s_8, p_5) from blocking M. Finally, the algorithm outputs the feasible matching $M = \{(s_1, p_6), (s_2, p_2), (s_4, p_5), (s_5, p_3), (s_6, p_4), (s_7, p_1), (s_8, p_1)\}$ as a strongly stable matching.

3.4 Correctness of the algorithm

The correctness of `Algorithm SPA-ST-strong` is established via a sequence of lemmas, namely Lemmas 4-14 in [24, Sect. 4.5]. These are omitted here for space reasons, but may be summarised as follows:

1. no strongly stable pair is ever deleted during the execution of the algorithm;
2. no strongly stable matching exists if some:
   (a) non-replete lecturer l_k has fewer assignees in the feasible matching M than provisional assignees in the final assignment graph G, or
   (b) replete lecturer is not full in M, or
   (c) student is bound to two or more projects that are offered by different lecturers, or
   (d) pair (s_i, p_j) was deleted where p_j is offered by l_k, each of p_j and l_k is undersubscribed in M, and for any $p_{j'} \in P_k$ such that s_i is indifferent between p_j and $p_{j'}$, $(s_i, p_{j'}) \notin M$;
3. if the algorithm outputs "no strongly stable matching" then at least one of the properties in (2) above must hold;
4. `Algorithm SPA-ST-strong` may be implemented to run in $O(m^2)$ time, where m is the total length of the students' preference lists.

The following theorem collects together Lemmas 4-14 in [24] and establishes the correctness of `Algorithm SPA-ST-strong`.

Theorem 1. *For a given instance I of* SPA-ST*, `Algorithm SPA-ST-strong` determines in $O(m^2)$ time whether or not a strongly 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 at least as good a project as she could obtain in any strongly stable matching, and each unassigned student is unassigned in every strongly stable matchings.*

Given the optimality property established by Theorem 1, we define the strongly stable matching found by `Algorithm SPA-ST-strong` to be *student-optimal*. For example, in the SPA-ST instance illustrated in Fig. 1, the student-optimal strongly stable matching is $\{(s_1, p_1), (s_2, p_2), (s_3, p_3)\}$.

4 Conclusion

We leave open the formulation of a lecturer-oriented counterpart to `Algorithm SPA-ST-strong`. From an experimental perspective, an interesting direction would be to carry out an empirical analysis of `Algorithm SPA-ST-strong`, to investigate how various parameters (e.g., the density and position of ties in the preference lists, the length of the preference lists, or the popularity of some projects) affect the existence of a strongly stable matching, based on randomly generated and/or real instances of SPA-ST.

Acknowledgement

The authors would like to convey their sincere gratitude to Adam Kunysz for valuable discussions concerning `Algorithm SPA-ST-strong`. They would also like to thank the anonymous reviewers for their helpful suggestions.

References

1. D.J. Abraham, R.W. Irving, and D.F. Manlove. Two algorithms for the Student-Project allocation problem. *Journal of Discrete Algorithms*, 5(1):79–91, 2007.
2. A.H. Abu El-Atta and M.I. Moussa. Student project allocation with preference lists over (student,project) pairs. In *Proceedings of ICCEE 09*, pp. 375–379, 2009.
3. M. Baidas, Z. Bahbahani, E. Alsusa. User association and channel assignment in downlink multi-cell NOMA networks: A matching-theoretic approach. *EURASIP Journal on Wireless Communications and Networking*, 2019:220, 2019.
4. M. Baidas, M. Bahbahani, E. Alsusa, K. Hamdi and Z. Ding. D2D Group Association and Channel Assignment in Uplink Multi-Cell NOMA Networks: A Matching Theoretic Approach. *To appear in IEEE Transactions on Communications*, 2019.
5. M. Baidas, Z. Bahbahani, N. El-Sharkawi, H. Shehada and E. Alsusa. Joint relay selection and max-min energy-efficient power allocation in downlink multi-cell NOMA networks: A matching-theoretic approach. *Transactions on Emerging Telecommunications Technologies*, 30:5, 2019.
6. M. Chiarandini, R. Fagerberg, and S. Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research*, 275(1):39 – 78, 2019.
7. Frances Cooper and David Manlove. A 3/2-Approximation Algorithm for the Student-Project Allocation Problem. In *Proceedings of SEA '18*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1 – 8:13, 2018.

8. R.W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.
9. R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT '00*, vol. 1851 of *LNCS*, pp. 259–271, 2000.
10. R. Irving, D. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS '03*, vol. 2607 of *LNCS*, pp. 439–450, 2003.
11. K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Proceedings of ICALP '99*, vol. 1644 of *LNCS*, pp. 443–452, 1999.
12. K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation bounds for the student-project allocation problem with preferences over projects. *Journal of Discrete Algorithms*, 13:59–66, 2012.
13. T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. Strongly stable matchings in time $O(nm)$ and extension to the Hospitals-Residents problem. In *Proceedings of STACS '04*, volume 2996 of *LNCS*, pages 222–233. Springer, 2004.
14. D. Kazakov. Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science. Available from `http://www-users.cs.york.ac.uk/kazakov/papers/proj.pdf` (last accessed 25 November 2019), 2001.
15. A. Kwanashie, R.W. Irving, D.F. Manlove, and C.T.S. Sng. Profile-based optimal matchings in the Student–Project Allocation problem. In *Proceedings of IWOCA '14*, volume 8986 of *LNCS*, pages 213–225. Springer, 2015.
16. A. Kunysz. An Algorithm for the Maximum Weight Strongly Stable Matching Problem. In *Proceedings of ISAAC '18*, vol. 123 of *LIPIcs*, pages 42:1–42:13, 2018.
17. C.L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, NY, 1968.
18. D. Manlove. Stable marriage with ties and unacceptable partners. Technical Report TR-1999-29, University of Glasgow, Department of Computing Science, Jan 1999.
19. D. Manlove. *Algorithmics of Matching Under Preferences*. World Scientific, 2013.
20. D. Manlove, R.W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.
21. D. Manlove, D. Milne, and S. Olaosebikan. An Integer Programming Approach to the Student-Project Allocation Problem with Preferences over Projects. In *Proceedings of ISCO '18*, volume 10856 of *LNCS*, pp. 313 – 325. Springer, 2018.
22. D. Manlove and G. O'Malley. Student project allocation with preferences over projects. *Journal of Discrete Algorithms*, 6:553–560, 2008.
23. S. Olaosebikan and D. Manlove. Super-Stability in the Student-Project Allocation Problem with Ties. In *Proceedings of COCOA '18*, volume 11346 of *Lecture Notes in Computer Science*, pages 357 – 371. Springer, 2018.
24. S. Olaosebikan and D.F. Manlove. An Algorithm for Strong Stability in the Student-Project Allocation problem with Ties. *CoRR*, abs/1911.10262, 2019. Available from `http://arxiv.org/abs/1911.10262`.
25. A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.