The Student-Project Allocation Problem

David J. Abraham, Robert W. Irving, and David F. Manlove*

Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK
Email: {dabraham,rwi,davidm}@dcs.gla.ac.uk. *Fax:* +44 141 330 4913.

Abstract. We study the *Student-Project Allocation problem* (SPA), a generalisation of the classical Hospitals / Residents problem (HR). An instance of SPA involves a set of students, projects and lecturers. Each project is offered by a unique lecturer, and both projects and lecturers have capacity constraints. Students have preferences over projects, whilst lecturers have preferences over students. We present an optimal linear-time algorithm for allocating students to projects, subject to these preferences and capacities. In particular, the algorithm finds a *stable matching* of students to projects. Here, the concept of stability generalises the stability definition in the HR context. The stable matching produced by our algorithm is simultaneously best-possible for all students. The SPA problem model that we consider is very general and has applications to a range of different contexts besides student-project allocation.

1 Introduction

In many university departments, students seek a project in a given field of speciality as part of the upper level of their degree programme. Usually, a project can be filled by at most one student, though in some cases a project is suitable for more than one student to work on simultaneously. To give students something of a choice, there should be as wide a range of available projects as possible, and in any case the total number of project places should not be less than the total number of students. Typically a lecturer will also offer a range of projects, but does not necessarily expect that all will be taken up.

Each student has preferences over the available projects that he/she finds acceptable, whilst a lecturer will normally have preferences over the students that he/she is willing to supervise. There may also be upper bounds on the number of students that can be assigned to a particular project, and the number of students that a given lecturer is willing to supervise. In this paper we consider the problem of allocating students to projects based on these preference lists and capacity constraints – the so-called *Student-Project Allocation problem* (SPA).

SPA is an example of a *two-sided matching problem* [10], a large and very general class of problems in which the input set of participants can be partitioned into two disjoint sets A and B (in this case A is the set of students and B is the set of projects), and we seek to match members of A to members of B, i.e. to find a subset of $A \times B$, subject to various criteria. These criteria usually involve

* Supported by award NUF-NAL-02 from the Nuffield Foundation, and grant GR/R84597/01 from the Engineering and Physical Sciences Research Council.

capacity constraints, and/or preference lists, for example.

Both historical evidence (see e.g. [4, pp.3-4], [7]) and economic analysis [10] indicate that participants involved in two-sided matching problems should not be allowed to construct an allocation by approaching one another directly in order to make ad hoc arrangements. Instead, the allocation process should be automated by means of a centralised matching scheme. Moreover, it has been convincingly argued [9] that the key property that a matching constructed by such schemes should satisfy is *stability*. A formal definition of stability follows, but informally, a stable matching M guarantees that no two participants who are not matched together in M would rather be matched to one another than remain with their assignment in M. Such a pair of participants would form a private arrangement and would undermine the integrity of the matching.

The National Resident Matching Program (NRMP) [8] in the US is perhaps the largest and best-known example of a centralised matching scheme. It has been in operation since 1952, and currently handles the allocation of some 20,000 graduating medical students, or *residents*, to their first hospital posts, based on the preferences of residents over available hospital posts, and the preferences of hospital consultants over residents. The NRMP employs at its heart an efficient algorithm that essentially solves a variant of the classical Hospitals / Residents problem (HR) [3, 4]. The algorithm finds a stable matching of residents to hospitals that is *resident-optimal*, in that each resident obtains the best hospital that he/she could obtain in any stable matching.

There are many other examples of centralised matching schemes, both in educational and vocational contexts. Many university departments in particular seek to automate the allocation of students to projects. However, as we discuss in greater detail later, an optimal linear-time algorithm for this setting cannot be obtained by simply reducing an instance of SPA to an instance of HR. Thus, a specialised algorithm is required for the SPA problem.

In this paper we present a linear-time algorithm for finding a stable matching, given an instance of SPA. This algorithm is *student-oriented*, in that it finds a *student-optimal* stable matching. In this matching, each student obtains the best project that he/she could obtain in any stable matching. Our algorithm is applicable for any context that fits into the SPA model, for example where applicants seek posts at large organisations, each split into several departments.

The remainder of this paper is structured as follows. In Section 2, a formal definition of the SPA problem is given. Then, in Section 3, the algorithm for SPA is presented, together with correctness proofs and an analysis of its complexity. Finally, Section 4 contains some conclusions and open problems.

2 Definition of the Student-Project Allocation Problem

An instance of the *Student-Project Allocation problem* (SPA) may be defined as follows. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of *students*, let $P = \{p_1, p_2, \ldots, p_m\}$ be a set of *projects*, and let $L = \{l_1, l_2, \ldots, l_q\}$ be a set of *lecturers*. Each student s_i supplies a preference list, ranking a subset of P in strict order. If project p_j appears on s_i's preference list, we say that s_i finds p_j *acceptable*. Denote by A_i

Student preferences	Lecturer preferences	
$s_1 : p_1 \;\; p_7$	$l_1 : s_7 \;\; s_4 \;\; s_1 \;\; s_3 \;\; s_2 \;\; s_5 \;\; s_6$	l_1 offers p_1, p_2, p_3
$s_2 : p_1 \;\; p_2 \;\; p_3 \;\; p_4 \;\; p_5 \;\; p_6$	$l_2 : s_3 \;\; s_2 \;\; s_6 \;\; s_7 \;\; s_5$	l_2 offers p_4, p_5, p_6
$s_3 : p_2 \;\; p_1 \;\; p_4$	$l_3 : s_1 \;\; s_7$	l_3 offers p_7, p_8
$s_4 : p_2$		
$s_5 : p_1 \;\; p_2 \;\; p_3 \;\; p_4$		
$s_6 : p_2 \;\; p_3 \;\; p_4 \;\; p_5 \;\; p_6$	Project capacities: $c_1 = 2$, $c_i = 1$ $(2 \leq i \leq 8)$	
$s_7 : p_5 \;\; p_3 \;\; p_8$	Lecturer capacities: $d_1 = 3$, $d_2 = 2$, $d_3 = 2$	

Fig. 1. An instance of the Student-Project Allocation problem.

the set of projects that s_i finds acceptable.

Each lecturer l_k *offers* a non-empty set of projects P_k, where $P_1, P_2, \ldots, P_q$ partitions P. Let $B_k = \{s_i \in S : P_k \cap A_i \neq \emptyset\}$ (i.e. B_k is the set of students who find acceptable a project offered by l_k). Lecturer l_k supplies a preference list, denoted by $\mathcal{L}_k$, ranking B_k in strict order. For any $p_j \in P_k$, we denote by $\mathcal{L}_k^j$ the *projected preference list of* l_k *for* p_j – this is obtained from $\mathcal{L}_k$ by deleting those students who do not find p_j acceptable. In this way, the ranking of $\mathcal{L}_k^j$ is inherited from $\mathcal{L}_k$. Also, l_k has a capacity constraint d_k, indicating the maximum number of students that he/she is willing to supervise. Similarly, each project p_j carries a capacity constraint c_j, indicating the maximum number of students that could be assigned to p_j. We assume that $max\{c_j : p_j \in P_k\} \leq d_k$.

An example SPA instance is shown in Figure 1. Here the set of students is $S = \{s_1, s_2, \ldots, s_7\}$, the set of projects is $P = \{p_1, p_2, \ldots, p_8\}$ and the set of lecturers is $L = \{l_1, l_2, l_3\}$. Lecturers offer projects as indicated, and the preference lists and capacity constaints are also shown. As an example, the projected preference list of l_1 for p_1 comprises s_1, s_3, s_2, s_5, ranked in that order.

An *assignment* M is a subset of $S \times P$ such that:

1. $(s_i, p_j) \in M$ implies that $p_j \in A_i$ (i.e. s_i finds p_j acceptable).
2. For each student $s_i \in S$, $|\{(s_i, p_j) \in M : p_j \in P\}| \leq 1$.

If $(s_i, p_j) \in M$, we say that s_i is *assigned to* p_j, and p_j is *assigned* s_i. Hence Condition 2 states that each student is assigned to at most one project in M. For notational convenience, if s_i is assigned in M to p_j, we may also say that s_i *is assigned to* l_k, and l_k *is assigned* s_i, where $p_j \in P_k$.

For any student $s_i \in S$, if s_i is assigned in M to some project p_j, we let $M(s_i)$ denote p_j; otherwise we say that s_i is *unmatched* in M. For any project $p_j \in P$, we denote by $M(p_j)$ the set of students assigned to p_j in M. Project p_j is *under-subscribed*, *full* or *over-subscribed* according as $|M(p_j)|$ is less than, equal to, or greater than c_j, respectively. Similarly, for any lecturer $l_k \in L$, we denote by $M(l_k)$ the set of students assigned to l_k in M. Lecturer l_k is *under-subscribed*, *full* or *over-subscribed* according as $|M(l_k)|$ is less than, equal to, or greater than d_k respectively.

A *matching* M is an assignment such that:

3. For each project $p_j \in P$, $|\{(s_i, p_j) \in M : s_i \in S\}| \leq c_j$.
4. For each lecturer $l_k \in L$, $|\{(s_i, p_j) \in M : s_i \in S \wedge p_j \in P_k\}| \leq d_k$.

Hence Condition 3 stipulates that p_j is assigned at most c_j students in M, whilst Condition 4 requires that l_k is assigned at most d_k students in M.

A *blocking pair* relative to a matching M is a (student,project) pair $(s_i, p_j) \in (S \times P) \backslash M$ such that:

1. $p_j \in A_i$ (i.e. s_i finds p_j acceptable).
2. Either s_i is unmatched in M, or s_i prefers p_j to $M(s_i)$.
3. Either
   (a) p_j is under-subscribed and l_k is under-subscribed, or
   (b) p_j is under-subscribed, l_k is full, and either l_k prefers s_i to the worst student s' in $M(l_k)$ or $s_i = s'$, or
   (c) p_j is full and l_k prefers s_i to the worst student in $M(p_j)$,
   where l_k is the lecturer who offers p_j.

A matching is *stable* if it admits no blocking pair. We now give some intuition for the definition of a blocking pair. Suppose that (s_i, p_j) forms a blocking pair with respect to matching M, and let l_k be the lecturer who offers p_j.

In general we assume that s_i prefers to be matched to an acceptable project rather than to remain unmatched. Hence Condition 2 indicates the means by which a student could improve relative to M. Suppose now that this condition is satisfied. To explain Condition 3(a), matching M cannot be stable if each of project p_j and lecturer l_k has a free place to take on s_i (or to let s_i change projects offered by l_k). We now consider Condition 3(b). If p_j is under-subscribed, l_k is full, and s_i was not already matched in M to a project offered by l_k, then l_k cannot take on s_i without first rejecting at least one student. Lecturer l_k would only agree to this switch if he/she prefers s_i to the worst student assigned to l_k in M. In this case, project p_j has room for s_i. Alternatively, if s_i was already matched in M to a project offered by l_k, then the total number of students assigned to l_k remains the same, and l_k agrees to the switch since p_j has room for s_i. Finally, we consider Condition 3(c). If p_j is full, then l_k cannot take on s_i without first rejecting at least one student assigned to p_j. Lecturer l_k would only agree to this switch if he/she prefers s_i to the worst student assigned to p_j in M. Notice that if s_i was already matched in M to a project offered by l_k, then the number of students assigned to l_k would decrease by 1 after the switch. However we argue that this is the "correct" definition of a blocking pair in this case, also having the side-effects of avoiding issues of strategy and maintaining useful structural properties. For a full discussion of this point, we refer the reader to Section 4.1 of [1].

We remark that HR is a special case of SPA in which $m = q$, $c_j = d_j$ and $P_j = \{p_j\}$ $(1 \leq j \leq m)$. Essentially the projects and lecturers are indistinguishable in this case. In the HR setting, lecturers / projects are referred to as *hospitals*, and students are referred to as *residents*. Linear-time algorithms are known for finding a stable matching, given an instance of HR. The *resident-oriented* algorithm [4, Section 1.6.3] finds a *resident-optimal stable matching*. In this stable matching, each matched resident is assigned to the best hospital that he/she could obtain in any stable matching, whilst each unmatched resident is unmatched in every stable matching. On the other hand, the *hospital-oriented*

algorithm [4, Section 1.6.2] finds a *hospital-optimal stable matching*. In this stable matching, each full hospital is assigned the best set of residents that it could obtain in any stable matching, whilst each under-subscribed hospital is assigned the same set of residents in every stable matching.

It is worth drawing attention to a special case of HR (and hence of SPA). This is the classical Stable Marriage problem with Incomplete lists (SMI), where $c_j = 1$ $(1 \leq j \leq m)$ [3], [4, Section 1.4.2]. In this setting, residents are referred to as *men* and hospitals are referred to as *women*. There exists a reduction from HR to SMI using the method of 'cloning' hospitals. That is, replace each hospital h_j, of capacity c_j, with c_j women, denoted by $h_j^1, h_j^2, \ldots, h_j^{c_j}$. The preference list of h_j^k is identical to the preference list of h_j. Any occurrence of h_j in a resident's preference list should be replaced by $h_j^1, h_j^2, \ldots, h_j^{c_j}$ in that order. Hence in theory, the Gale / Shapley algorithm for SMI [4, Section 1.4.2] could be used to solve an HR instance. However in practice direct algorithms are applied to HR instances [4, Section 1.6], because the cloning technique increases the number of hospitals (women) in a given HR instance by a potentially significant factor of C/m, where $C = \sum_{j=1}^{j=m} c_j$.

On the other hand there is no straightforward reduction involving cloning from an instance of SPA to an instance of HR, due to the projects and lecturers being distinct entities, each having capacity constraints. Even if such a reduction were possible, again it would typically increase the number of lecturers (hospitals) by a significant factor. This justifies the approach of this paper, in which we consider a direct algorithm for SPA.

The running time of our algorithm is $O(L)$, where L is the total length of the input preference lists, and hence is linear in the size of the problem instance. This algorithm is optimal, since the Stable Marriage problem (SM) – the special case of SMI in which $m = n$ and each student finds every project acceptable – is a special case of SPA. A lower bound of $\Omega(L)$ is known for SM [6], and hence this also applies to SPA.

3 The algorithm for SPA

3.1 Overview

The algorithm for finding a student-optimal stable matching involves a sequence of *apply* operations (i.e. students *apply* to projects). An apply operation is similar to a *proposal* in the context of the Gale / Shapley algorithm for SM [3]. These operations lead to provisional assignments between students, projects and lecturers; such assignments can subsequently be broken during the algorithm's execution. Also, throughout the execution, entries are possibly deleted from the preference lists of students, and from the projected preference lists of lecturers. We use the abbreviation *delete the pair* (s_i, p_j) to denote the operation of deleting p_j from the preference list of s_i, and deleting s_i from $\mathcal{L}_k^j$, where l_k is the lecturer who offers p_j.

Initially all students are free, and all projects and lecturers are totally unsubscribed. As long as there is some student s_i who is free and who has a non-empty

```
assign each student to be free;
assign each project and lecturer to be totally unsubscribed;
while (some student s_i is free) and (s_i has a non-empty list) {
    p_j = first project on s_i's list;
    l_k = lecturer who offers p_j;
    /* s_i applies to p_j */
    provisionally assign s_i to p_j;                    /* and to l_k */
    if (p_j is over-subscribed) {
        s_r = worst student assigned to p_j;            /* according to L_k^j */
        break provisional assignment between s_r and p_j;
    }
    else if (l_k is over-subscribed) {
        s_r = worst student assigned to l_k;
        p_t = project assigned s_r;
        break provisional assignment between s_r and p_t;
    }
    if (p_j is full) {
        s_r = worst student assigned to p_j;            /* according to L_k^j */
        for (each successor s_t of s_r on L_k^j)
            delete the pair (s_t, p_j);
    }
    if (l_k is full) {
        s_r = worst student assigned to l_k;
        for (each successor s_t of s_r on L_k)
            for (each project p_u ∈ P_k ∩ A_t)
                delete the pair (s_t, p_u);
    }
}
```

Fig. 2. Algorithm SPA-student for finding a student-optimal stable matching

list, s_i applies to the first project p_j on his/her list. We let l_k be the lecturer who offers p_j. Immediately, s_i becomes provisionally assigned to p_j (and to l_k).

If p_j is over-subscribed, then l_k rejects the worst student s_r assigned to p_j. The pair (s_r, p_j) will be deleted by the subsequent conditional that tests for p_j being full. Similarly, if l_k is over-subscribed, then l_k rejects his/her worst assigned student s_r. The pair (s_r, p_t) will be deleted by either of the two subsequent conditionals, where p_t was the project formerly assigned to s_r.

Regardless of whether any rejections occurred as a result of the two situations described in the previous paragraph, we have two further (possibly non-disjoint) cases in which deletions may occur. If p_j is full, we let s_r be the worst student assigned to p_j (according to $\mathcal{L}_k^j$) and delete the pair (s_t, p_j) for each successor s_t of s_r on $\mathcal{L}_k^j$. Similarly if l_k is full, we let s_r be the worst student assigned to l_k, and delete the pair (s_t, p_u) for each successor s_t of s_r on $\mathcal{L}_k$, and for each project p_u offered by l_k that s_t finds acceptable.

The algorithm is described in pseudocode form in Figure 2 as Algorithm SPA-student. We will prove that, once the main loop terminates, the assigned pairs constitute a student-optimal stable matching.

3.2 Correctness of Algorithm SPA-student

We firstly remark that Algorithm SPA-student terminates with a matching. The correctness of the algorithm, together with the optimality property of the constructed matching, may be established by the following sequence of lemmas.

Lemma 1. *No pair deleted during an execution of Algorithm SPA-student can block the constructed matching.*

Proof. Let E be an arbitrary execution of the algorithm in which some pair (s_i, p_j) is deleted. Suppose for a contradiction that (s_i, p_j) blocks M, the matching generated by E. Now, (s_i, p_j) is deleted in E because either (i) p_j becomes full, or (ii) l_k becomes full, where l_k is the lecturer offering p_j. In Case (i), it turns out that (s_i, p_j) fails (a), (b) and (c) of Condition 3 of a blocking pair, a contradiction. The details for each of these sub-cases are omitted here for space reasons, but may be found in [1]. Case (ii) is easier: (s_i, p_j) cannot block M, since once full, a lecturer never becomes under-subscribed, and is only ever assigned more preferable students. □

Lemma 2. *Algorithm SPA-student generates a stable matching.*

Proof. Let M be the matching generated by an arbitrary execution E of the algorithm, and let (s_i, p_j) be any pair blocking M. We will show that (s_i, p_j) must be deleted in E, thereby contradicting Lemma 1. For, suppose not. Then s_i must be matched to some project $M(s_i) \neq p_j$, for otherwise s_i is free with a non-empty preference list (containing p_j), thereby contradicting the fact that the algorithm terminates. Now, when s_i applies to $M(s_i)$, $M(s_i)$ is the first project on his/her list. Hence, (s_i, p_j) must be deleted, since for (s_i, p_j) to block M, s_i must prefer p_j to $M(s_i)$. □

Lemma 3. *No stable pair (i.e. (student,project) pair belonging to some stable matching) is deleted during an execution of Algorithm SPA-student.*

Proof. Suppose for a contradiction that (s_i, p_j) is the first stable pair deleted during an arbitrary execution E of the algorithm. Let M be the matching immediately after the deletion in E, and let M' be any stable matching containing (s_i, p_j). Now, (s_i, p_j) is deleted in E because either (i) p_j becomes full, or (ii) l_k becomes full, where l_k is the lecturer offering p_j. We consider each case in turn.

(i) Suppose (s_i, p_j) is deleted because p_j becomes full during E. Immediately after the deletion, p_j is full, and l_k prefers all students in $M(p_j)$ to s_i. Now, $s_i \in M'(p_j) \backslash M(p_j)$, and since p_j is full in M, there must be some $s \in M(p_j) \backslash M'(p_j)$. We will show that (s, p_j) forms a blocking pair, contradicting the stability of M'.
Firstly, since (s_i, p_j) is the first stable pair deleted in E, s prefers p_j to any of his/her stable partners (except possibly for p_j itself). Additionally, since $(s_i, p_j) \in M'$ and l_k prefers s to s_i, it follows that l_k prefers s to both the worst student in $M'(p_j)$ and $M'(l_k)$. Clearly then, for any combination of l_k and p_j being full or under-subscribed, (s, p_j) satisfies all the conditions to block M'.

(ii) Suppose that (s_i, p_j) is deleted because l_k becomes full during E. Immediately after the deletion, l_k is full, and l_k prefers all students in $M(l_k)$ to s_i. We consider two cases: $|M'(p_j)| > |M(p_j)|$ and $|M'(p_j)| \leq |M(p_j)|$.
Suppose firstly $|M'(p_j)| > |M(p_j)|$. Since l_k is full in M, and $(s_i, p_j) \notin M$, there must be some project $p \in P_k \backslash \{p_j\}$ such that $|M'(p)| < |M(p)|$. We remark that p is therefore under-subscribed in M'. Now, let s be any student in $M(p) \backslash M'(p)$. Since (s_i, p_j) is the first stable pair deleted, s prefers p to any of his/her stable partners (except possibly for p itself). Also, l_k prefers s to s_i, and hence to the worst student in $M'(l_k)$. So, in either case that l_k is full or under-subscribed, (s, p) blocks M'.
Now suppose $|M'(p_j)| \leq |M(p_j)|$. Then there is some $s \neq s_i \in M(p_j) \backslash M'(p_j)$. Now, p_j is under-subscribed in M, for otherwise (s_i, p_j) is deleted because p_j becomes full, contradicting the assumption that deletion occurs because l_k becomes full. Therefore, p_j is under-subscribed in M'. As above, s prefers p_j to any of his/her stable partners (except possibly for p_j itself), since (s_i, p_j) is the first stable pair deleted. Also, l_k prefers s to s_i, and hence to the worst pair in $M'(l_k)$. So, in either case that l_k is full or under-subscribed, (s, p_j) blocks M'. □

The following theorem collects together Lemmas 1-3.

Theorem 1. *For a given instance of SPA, any execution of Algorithm SPA-student constructs the student-optimal stable matching.*

Proof. Let M be a matching generated by an arbitrary execution E of the algorithm. In M, each student is assigned to the first project on his/her reduced preference list, if any. By Lemma 2, M is stable, and so each of these (student, project) pairs is stable. Also, by Lemma 3, no stable pair is deleted during E. It follows then that in M, each student is assigned to the best project that he/she can obtain in any stable matching. □

For example, in the SPA instance given by Figure 1, the student-optimal stable matching is $\{(s_1, p_1), (s_2, p_5), (s_3, p_4), (s_4, p_2), (s_7, p_3)\}$.

We now state a result similar to the 'Rural Hospitals Theorem' for HR [4, Theorem 1.6.3]. In particular, the following theorem indicates that, in no other stable matching could we match a different set of students than that matched by Algorithm SPA-student. The proof is omitted here for space reasons, but may be found in [1].

Theorem 2. *For a given SPA instance:*

(i) each lecturer has the same number of students in all stable matchings;
(ii) exactly the same students are unmatched in all stable matchings;
(iii) a project offered by an under-subscribed lecturer has the same number of students in all stable matchings.

However it turns out that an under-subscribed lecturer need not obtain the same set of students in all stable matchings, and in addition, a project offered by a full lecturer need not obtain the same number of students in all stable matchings. Example SPA instances illustrating these remarks are given in [1].

3.3 Analysis of Algorithm SPA-student

The algorithm's time complexity depends on how efficiently we can execute 'apply' operations and deletions, each of which occur at most once for any (student, project) pair. It turns out that both operations can be implemented to run in constant time, giving an overall time complexity of $\Theta(L)$, where L is the total length of all the preference lists. We briefly outline the non-trivial aspects of such an implementation.

For each student s_i, build an array, $rank_{s_i}$, where $rank_{s_i}(p_j)$ is the index of project p_j in s_i's preference list. Represent s_i's preference list by embedding doubly linked lists in an array, $preference_{s_i}$. For each project $p_j \in A_i$, $preference_{s_i}(rank_{s_i}(p_j))$ 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 list (after deletions, this project may not located at the next array position), and another pointer to the next project p' in s_i's list, where p' and p_j are both offered by the same lecturer. Construct this 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 [2, p.149]) for these arrays, since the overall $\Theta(nq)$ initialisation cost may be super-linear in L. Clearly, using these data structures, we can find and delete a project from a given student in constant time, as well as efficiently delete all projects offered by a given lecturer.

Represent each lecturer l_k's preference list $\mathcal{L}_k$ by an array $preference_{l_k}$, with an additional pointer, $last_{l_k}$. Initially, $last_{l_k}$ stores the index of the last position in $preference_{l_k}$. However, once l_k is full, make $last_{l_k}$ equivalent to l_k's worst assigned student through the following method. Perform a backwards linear traversal through $preference_{l_k}$, starting at $last_{l_k}$, and continuing until l_k's worst assigned student is encountered (each student stores a pointer to their assigned project, or a special null value if unassigned). All but the last student on this traversal must be deleted, and so the cost of the traversal may be attributed to the cost of the deletions in the student preference lists.

For each project p_j offered by l_k, construct a preference array corresponding to $\mathcal{L}_k^j$. These project preference arrays are used in much the same way as the lecturer preference array, with one exception. When a lecturer l_k becomes over-subscribed, the algorithm frees l_k's worst assigned student s_i and breaks the assignment of s_i to some project p_j. If p_j was full, then it is now under-subscribed, and $last_{p_j}$ is no longer equivalent to p_j's worst assigned student. Rather than update $last_{p_j}$ immediately, which could be expensive, wait until p_j is full again. The update then involves the same backwards linear traversal described above for l_k, although we must be careful not to delete pairs already deleted in one of l_k's traversals. Since we only visit a student at most twice during these backwards traversals, once for the lecturer and once for the project, the asymptotic running time remains linear.

The implementation issues discussed above lead to the following conclusion.

Theorem 3. *Algorithm SPA-student may be implemented to use $\Theta(L)$ time and space, where L is the total length of the preference lists in a given SPA instance.*

4 Conclusions and open problems

In this paper we have presented a student-oriented algorithm for a SPA instance. This produces the student-optimal stable matching, in which each student obtains the best project that he/she could obtain in any stable matching. We remark that we have also formulated a lecturer-oriented counterpart, which we omit for space reasons. This second algorithm produces the lecturer-optimal stable matching, in which each lecturer obtains the best set of students that he/she could obtain in any stable matching.

A number of interesting open problems remain. These include:

- The SPA model may be extended to the case where lecturers have preferences over (student,project) pairs. However in this setting it is an open problem to formulate an acceptable stability definition that avoids issues of strategy. For example, a student could deliberately shorten his/her preference list in order to obtain a better project, rather than submitting his/her true preferences. These strategic issues are described in more detail in [1].
- If we allow ties in the preference lists of students and lecturers, different stability definitions are possible. These can be obtained by extending stability definitions that have been applied to the Hospitals / Residents problem with Ties [5]. It remains open to construct algorithms for SPA where preference lists contain ties, under each of these stability criteria.

References

1. D.J. Abraham, R.W. Irving, and D.F. Manlove. *The Student-Project Allocation Problem*. Technical Report TR-2003-141 of the Computing Science Department of Glasgow University, 2003.
2. G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.
3. D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
4. D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.
5. R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS 2003: the 20th International Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer-Verlag, 2003.
6. C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.
7. National Resident Matching Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.
8. National Resident Matching Program. Why the Match? Web document available at `http://www.nrmp.org/whythematch.pdf`.
9. 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.
10. A.E. Roth and M.A.O. Sotomayor. *Two-sided matching: a study in game-theoretic modeling and analysis*, volume 18 of *Econometric Society Monographs*. Cambridge University Press, 1990.