Two Algorithms for the Student-Project Allocation Problem*

David J. Abraham[1†], Robert W. Irving[2], and David F. Manlove[2‡]

[1] *Computer Science Department, Carnegie-Mellon University, 5000 Forbes Ave, Pittsburgh PA 15213-3890, USA. Email:* dabraham@cs.cmu.edu.

[2] *Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK. Email:* {rwi,davidm}@dcs.gla.ac.uk.

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 two optimal linear-time algorithms for allocating students to projects, subject to the preference and capacity constraints. In particular, each 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 the first algorithm is simultaneously best-possible for all students, whilst the one produced by the second algorithm is simultaneously best-possible for all lecturers. We also prove some structural results concerning the set of stable matchings in a given instance of SPA. 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 to undertake a project in a given field of speciality as part of the upper level of their degree programme. Typically a wide range of available projects is offered, and usually the total number of project places exceeds the number of students, to provide something of a choice. Also, typically each lecturer will offer a variety 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* [25, 1], a large and very general class of problems whose input includes a set of participants that can be partitioned into

*A preliminary version of this paper was presented at ISAAC 2003 [2].

†Work done whilst at Department of Computing Science, University of Glasgow.

‡Supported by Engineering and Physical Sciences Research Council grant GR/R84597/01, Nuffield Foundation award NUF-NAL-02, and Royal Society of Edinburgh/Scottish Executive Personal Research Fellowship.

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

Both historical evidence (see e.g. [12, pp.3-4], [20]) and game-theoretic analysis [23, 25] indicate that participants involved in two-sided matching problems should not be allowed to construct an allocation by approaching one another directly and making ad hoc arrangements. Rather, the allocation process should be automated by means of a centralised matching scheme. Moreover, it has been convincingly argued [23] that, when preference lists exist on both sides, the key property that a matching constructed by such schemes should satisfy is that of *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 could come to a private arrangement that would undermine the integrity of the matching.

The National Resident Matching Program (NRMP) [19] 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 30,000 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) [10, 12]. 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 (e.g. allocating pupils to secondary schools in Singapore [26], school-leavers to universities in Spain [22] and trainee teachers to probationary posts in Scotland). Many university departments in particular seek to automate the allocation of students to projects [27, 5, 3]. 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 two linear-time algorithms for finding a stable matching, given an instance of SPA. The first algorithm is *student-oriented*, in that it finds the stable matching in which each student obtains the best project that he/she could obtain in any stable matching. The second algorithm is *lecturer-oriented*, in that it constructs the stable matching in which each lecturer has as good a set of students (in a precise sense, to be defined) as in any other stable matching. Our algorithms are applicable in any context that fits into the SPA model, for example where applicants seek posts at large organisations, each split into several departments.

As alluded to above, the centralised allocation of students to projects has been considered previously in the literature. Various models have been constructed that allow student preferences over projects, but do not permit lecturer preferences [21, 27, 3], so stability is not relevant in these contexts. However an automated system for allocating students to projects at the Department of Computer Science, University of York is described [5, 15, 28] which takes into account student preferences over projects and lecturer preferences over students. In this model, preference lists may include ties, each project has capacity 1, and lecturer capacities are unbounded. Constraint programming techniques are utilised in order to find a stable matching with additional properties, such as balancing the supervision load among lecturers as evenly as possible. The underlying algorithms do not, in general, run in polynomial time.

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: $d_1 = 3$, $d_2 = 2$, $d_3 = 2$	

Figure 1: An instance of the Student-Project Allocation problem.

The remainder of this paper is structured as follows. In Section 2, a formal definition of the SPA problem is given, followed by some consequences of this definition and a discussion of relationships between SPA and existing models in the literature. Then, in Section 3, the student-oriented algorithm for SPA is presented, together with correctness proofs and an analysis of its complexity. In Section 4 we present some properties of the set of stable matchings in a given SPA instance and consider the issue of load balancing students among lecturers. Then in Section 5, we give the lecturer-oriented algorithm, also establishing its correctness and time complexity. Finally, Section 6 contains a discussion of the SPA problem model considered in this paper, and also presents some open problems.

2 The Student-Project Allocation problem model

2.1 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 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 $\max\{c_j : p_j \in P_k\} \leq d_k \leq \sum\{c_j : p_j \in P_k\}$.

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 $L = \{l_1, l_2, l_3\}$. 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 con-

venience, 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 *unassigned* 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 $p_j \in P$, $|M(p_j)| \leq c_j$.
4. For each lecturer $l_k \in L$, $|M(l_k)| \leq d_k$.

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 (student,project) pair $(s_i, p_j) \in (S \times P) \backslash M$ *blocks* a matching M if:

1. $p_j \in A_i$ (i.e. s_i finds p_j acceptable).
2. Either s_i is unassigned 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 $s_i \in M(l_k)$ or l_k prefers s_i to the worst student in $M(l_k)$, 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.

We call (s_i, p_j) a *blocking pair* of M. A matching is *stable* if it admits no blocking pair.

2.2 Consequences of the SPA problem definition

Our blocking pair definition in this paper attempts to encapsulate the various practical scenarios in which s_i and l_k could both simultaneously improve relative to M by permitting an assignment between s_i and p_j. For this to occur, s_i must find p_j acceptable (Condition 1), and either be unassigned in M or prefer p_j to $M(s_i)$ (Condition 2). We now consider l_k's perspective. In Condition 3(a), l_k will co-operate if there was already a free place for s_i. Similarly, in Condition 3(b), if l_k is full and s_i was already assigned in M to a project offered by l_k, then l_k agrees to the switch since the total number of students assigned to l_k remains the same, and p_j has room for s_i. Alternatively, if l_k is full and s_i was not already assigned in M to a project offered by l_k, then l_k cannot take on s_i without first rejecting some student assigned to l_k. 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, and project 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 some 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 assigned 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; we revisit this point in Section 6.1.

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 [12, Section 1.6.3] finds the *resident-optimal* stable matching, in which each assigned resident is assigned to the best hospital that he/she could obtain in any stable matching, whilst each unassigned resident is unassigned in every stable matching. On the other hand, the *hospital-oriented* algorithm [12, Section 1.6.2] finds the *hospital-optimal* stable matching M. Such a matching M satisfies the property that there is no stable matching M' and hospital h for which h prefers a resident in $M'(h) \backslash M(h)$ to the worst resident in $M(h)$.

The set of stable matchings in a given instance of HR satisfy several interesting properties that together form the *Rural Hospitals Theorem* [12, Theorem 1.6.4].

Theorem 2.1 (Rural Hospitals) *For a given instance of HR, the following holds.*

(i) Each hospital is assigned the same number of residents in all stable matchings [11].

(ii) Exactly the same set of residents are unassigned in all stable matchings [11].

(iii) Any hospital that is under-subscribed in one stable matching is assigned precisely the same set of residents in all stable matchings [24].

In Section 4, we generalise parts of the Rural Hospitals Theorem to the SPA case, although as we demonstrate, not all of the above properties carry over to SPA.

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 $c_j = 1$ $(1 \leq j \leq m)$ [10], [12, 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 [12, Section 1.4.2] could be used to solve an HR instance. However in practice direct algorithms are applied to HR instances [12, 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 $C = \sum_{j=1}^{m} c_j$.

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 direct algorithms for SPA.

The two algorithms that we present are generalisations of the resident-oriented and hospital-oriented algorithms for HR. The running time of each algorithm is $O(\lambda)$, where λ is the total length of the input preference lists, and hence is linear in the size of the problem instance. This time complexity is optimal, since the Stable Marriage problem (SM) – the special case of SMI in which $m = n$ and each man finds every woman acceptable – is a special case of SPA. A lower bound of $\Omega(\lambda)$ is known for SM [18], and hence this also applies to SPA.

2.3 Related models in the literature

We now consider similarities between the SPA problem model and existing models for two-sided matching problems that have been proposed in the literature.

Recently Fleiner [7, 8] developed a matroid-theoretic characterisation of stable matchings in bipartite matching models. This is based on imposing two ordered partition matroids, $\mathcal{M}_A, \mathcal{M}_B$, one on each side of a bipartite graph G. A matching is an independent set that is common to both $\mathcal{M}_A$ and $\mathcal{M}_B$. Moreover a stable matching corresponds to an $\mathcal{M}_A\mathcal{M}_B$-*kernel*, and it is shown that such a structure is bound to exist [7, 8]. Fleiner [9] noted that the SPA problem model may be included in this characterisation by imposing a student matroid as a partition matroid, and a lecturer matroid as the truncation of a direct sum of uniform matroids (thus ensuring that all project and lecturer capacities are satisfied). Here the vertices on one side of G correspond to students, the vertices on the other side correspond to lecturers, and the edges correspond to acceptable (student,project) pairs (so that G is in general a multigraph).

Also Eguchi et al. [6] formulated a model for two-sided matching problems in which preferences are based on $M^\natural$-*concave functions*, which arise in discrete convex analysis. They gave an algorithm for finding a stable matching in such a context, however the algorithm does not, in general, run in polynomial time for an arbitrary $M^\natural$-concave function. Their model includes the possibility of capacities and multiple partners; moreover since linear orders gives rise to $M^\natural$-concave functions, it follows that the model of Eguchi et al. [6] includes SPA as a special case.

Our approach in this paper is to give specialised linear-time algorithms for SPA. As the algorithms are described directly in terms of the SPA problem model, they should be more intuitive and easier to implement in practical applications. Also, some structural and optimality properties of the SPA problem model are derived (see Theorems 3.5, 4.1 and 5.5) which do not necessarily hold in the more general models mentioned.

3 Student-oriented algorithm for SPA

3.1 Overview of Algorithm SPA-student

We now present our first algorithm for SPA, starting with an overview of its operation. The student-oriented algorithm for an instance of SPA 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 [10]. 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* (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 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 most recently assigned 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

```
SPA-student(I) {
    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 (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 (s_t, p_u);
        }
    }
    return {(s_i, p_j) ∈ S × P : s_i is provisionally assigned to p_j};
}
```

Figure 2: Pseudocode of Algorithm SPA-student.

to $\mathcal{L}_k^j$) and delete (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 (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 the stable matching that is simultaneously best-possible for all students.

3.2 Correctness of Algorithm SPA-student

The correctness of the algorithm, together with the optimality property of the constructed matching, may be established by the following sequence of lemmas.

Lemma 3.1 *Algorithm SPA-student terminates with a matching.*

Proof: Each loop iteration involves a free student s_i applying to the first project p_j on his/her preference list. No student can apply to the same project twice, since, for example, once s_i is freed from p_j, the pair (s_i, p_j) is deleted. The total number of iterations is therefore bounded by the overall length of the student preference lists. Finally, it is clear that, once the main loop terminates, the assigned pairs constitute a matching. ■

Lemma 3.2 *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. We will show that in Case (i), (s_i, p_j) fails (a), (b) and (c) of Condition 3 of a blocking pair. 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. We now deal with Case (i), and further consider the three sub-cases of Condition 3 of a blocking pair.

(a) p_j is under-subscribed and l_k is under-subscribed.
Condition (a) requires that p_j subsequently becomes under-subscribed – something that can only happen if l_k becomes over-subscribed and one of his/her assignments involving p_j is broken. However, it is not possible for l_k to subsequently become under-subscribed, contradicting the first clause of Condition (a).

(b) p_j is under-subscribed, l_k is full, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student s' in $M(l_k)$.
Condition (b) requires that p_j becomes under-subscribed at some point after the deletion of (s_i, p_j). Let (s, p_j) be the pair whose deletion by the over-subscribed l_k results in p_j becoming under-subscribed. Now l_k prefers s to s_i, and by Condition (b), l_k either prefers s_i to s', or $s_i = s'$. It follows then that l_k prefers s to s', and so, immediately after (s, p_j) is deleted, the algorithm will ensure that $(s', M(s'))$ is also deleted. This is a contradiction, since M is a matching of undeleted pairs.

(c) p_j is full and l_k prefers s_i to the worst student s' in $M(p_j)$.
Condition (c) gives us that l_k prefers s_i to s', and since (s_i, p_j) is deleted, (s', p_j) must also be deleted. This is a contradiction, since M is a matching of undeleted pairs. ■

Lemma 3.3 *Algorithm SPA-student generates a stable matching.*

Proof: By Lemma 3.1, 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 3.2. For, suppose not. Then s_i must be assigned 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 $M(s_i)$. ■

For a given instance of SPA, we define a *stable pair* to be a (student,project) pair that belongs to some stable matching. The next lemma shows that Algorithm SPA-student never deletes a stable pair.

Lemma 3.4 *No stable pair 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 that (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 that $|M'(p_j)| > |M(p_j)|$. Since l_k is full in 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 that $|M'(p_j)| \leq |M(p_j)|$. Then since $(s_i, p_j) \notin M$, 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 student 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 3.1-3.4.

Theorem 3.5 *For a given instance of SPA, any execution of Algorithm SPA-student constructs the stable matching in which each assigned student is assigned to the best project that he/she could obtain in any stable matching, whilst each unassigned student is unassigned in any stable matching.*

Proof: By Lemma 3.3, let M be the stable 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. Also, by Lemma 3.4, no stable pair is deleted during E. It follows then that in M, each assigned student is assigned to the best project that he/she could obtain in any stable matching, whilst any unassigned student is unassigned in any stable matching. ■

Given the optimality property established by Theorem 3.5, we define the stable matching returned by Algorithm SPA-student to be the *student-optimal* stable matching. For example, in the SPA instance given by Figure 1, the student-optimal stable matching is $\{(s_1, p_1), (s_2, p_5), (s_3, p_4), (s_4, p_2), (s_7, p_3)\}$.

In the next subsection, we show how to implement Algorithm SPA-student so that it runs in linear time.

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 $\Theta(\lambda)$, where λ 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 be 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 [4, p.149]) for these arrays, since the overall $\Theta(nq)$ initialisation cost may be super-linear in λ. 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 backward 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 backward 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 backward 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 3.6 *Algorithm SPA-student may be implemented to run in $\Theta(\lambda)$ time and $O(mn)$ space, where λ is the total length of the preference lists, and n, m are the numbers of students and projects respectively, in a given SPA instance.*

4 Properties of stable matchings in an instance of SPA

In this section we consider properties of the set of stable matchings in a given instance of SPA. We begin by proving a result similar to Theorem 2.1, the Rural Hospitals Theorem for HR, in the context of a given SPA instance.

Theorem 4.1 *For a given SPA instance, the following holds.*

(i) Each lecturer has the same number of students in all stable matchings.

(ii) Exactly the same students are unassigned in all stable matchings.

Student preferences	Lecturer preferences	
$s_1:\ p_3\ p_1\ p_2\ p_4$	$l_1:\ s_1\ s_2$	l_1 offers p_1, p_2
$s_2:\ p_1\ p_3\ p_2\ p_4$	$l_2:\ s_2\ s_1$	l_2 offers p_3, p_4

Project capacities: $c_i = 1$ $(1 \leq i \leq 4)$
Lecturer capacities: $d_i = 2$ $(1 \leq i \leq 2)$

Figure 3: Instance I_1 of the Student-Project Allocation problem.

(iii) A project offered by an under-subscribed lecturer has the same number of students in all stable matchings.

Proof: Let M be the student-optimal stable matching, and let M' be any other stable matching.

(i) Suppose $|M'(l_k)| < |M(l_k)|$ for some lecturer l_k. There must be some project $p_j \in P_k$ such that $|M'(p_j)| < |M(p_j)|$. So, l_k and p_j are both under-subscribed in M'. Also, there exists $s_i \in M(p_j) \backslash M'(p_j)$ who is unassigned in M' or prefers p_j to $M'(s_i)$, since M is student-optimal. Hence, (s_i, p_j) blocks M', and, therefore, $|M'(l_k)| \geq |M(l_k)|$ for all l_k. It follows that $|M'| \geq |M|$. However, $|M'| \leq |M|$, since M is student-optimal and therefore matches the maximum number of students of any stable matching. Hence $|M'| = |M|$, and for all l_k, $|M'(l_k)| = |M(l_k)|$.

(ii) Let U and U' be the sets of students unassigned in M and M' respectively. By Theorem 3.5, $U \subseteq U'$, since no student unassigned in M can be assigned in M'. But $|U| = |U'|$, by (i), and so it follows that $U = U'$.

(iii) Let l_k be any lecturer who is under-subscribed in M'. Suppose there is some project $p_j \in P_k$ such that $|M'(p_j)| < |M(p_j)|$. Then p_j is under-subscribed in M', and there exists $s_i \in M(p_j) \backslash M'(p_j)$ who is unassigned in M' or prefers p_j to $M'(s_i)$. Hence, (s_i, p_j) blocks M', and, therefore, $|M'(p_j)| \geq |M(p_j)|$. Now, by (i) above, $|M'(l_k)| = |M(l_k)|$, and so $|M'(p_j)| = |M(p_j)|$ for all $p_j \in P_k$. ■

It turns out that two key properties of the Rural Hospitals Theorem for HR have no analogue for SPA. Firstly, we give an instance of SPA illustrating that a lecturer who is under-subscribed in one stable matching need not be assigned the same set of students in all stable matchings. Note that the Rural Hospitals Theorem for HR states that any hospital that is under-subscribed in one stable matching is assigned the same set of residents in all stable matchings. However consider the SPA instance I_1 shown in Figure 3. Instance I_1 admits the stable matchings $M = \{(s_1, p_3), (s_2, p_1)\}$ and $M' = \{(s_1, p_1), (s_2, p_3)\}$. Lecturer l_1 is under-subscribed in M (and hence in M' by Part (i) of Theorem 4.1). However $M(l_1) = \{s_2\}$ whilst $M'(l_1) = \{s_1\}$.

Secondly, we give an instance of SPA illustrating that a project offered by a lecturer who is full in one stable matching need not be assigned the same number of students in all stable matchings. Note that the Rural Hospitals Theorem for HR states that each hospital is assigned the same number of residents in all stable matchings. However consider the SPA instance I_2 shown in Figure 4. Instance I_2 admits the stable matchings $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_3), (s_4, p_3)\}$ and $M' = \{(s_1, p_3), (s_2, p_4), (s_3, p_1), (s_4, p_2)\}$. Lecturer l_1 is full in M (and hence in M' by Part (i) of Theorem 4.1). However $M(p_1) = \{s_1, s_2\}$ whilst $M'(p_1) = \{s_3\}$.

Student preferences	Lecturer preferences	
s_1 : p_1 p_3 p_2 p_4	l_1 : s_3 s_4 s_1 s_2	l_1 offers p_1, p_2
s_2 : p_1 p_4 p_3 p_2	l_2 : s_1 s_2 s_3 s_4	l_2 offers p_3, p_4
s_3 : p_3 p_1 p_2 p_4		
s_4 : p_3 p_2 p_1 p_4		

Project capacities: $c_1 = 2$, $c_2 = 1$, $c_3 = 2$, $c_4 = 1$
Lecturer capacities: $d_1 = 2$, $d_2 = 2$

Figure 4: Instance I_2 of the Student-Project Allocation problem.

Finally we consider the issue of load balancing project supervision among lecturers. Theorem 4.1(i) might seem to imply that this issue is not relevant in our model. However load balancing can be achieved if each $l_k \in L$ is constrained (for example by the Head of Department) to set $d_k = \lceil \frac{n}{q} \rceil$ (recall our assumption from Section 2 that $\sum\{c_j : p_j \in P_k\} \geq d_k$). Once Algorithm SPA-student has been run, the matched students would be removed, any unmatched student would be invited to submit a longer preference list chosen from the remaining under-subscribed projects, and the capacities of the under-subscribed projects and lecturers would be suitably adjusted. Then a "second round" of the algorithm could be used to assign the remaining students, ensuring that the students are distributed equitably among the lecturers.

5 Lecturer-oriented algorithm for SPA

5.1 Overview of Algorithm SPA-lecturer

We now present the lecturer-oriented counterpart of Algorithm SPA-student. The lecturer-oriented algorithm for an instance of SPA begins with the empty assignment, in which all students are free, and every project and lecturer is totally unsubscribed. The algorithm then enters a loop, each iteration of which involves an under-subscribed lecturer l_k offering a project $p_j \in P_k$ to a student s_i. Student s_i must be the first student on l_k's list who is currently free or prefers an under-subscribed project in P_k to his/her current provisional assignment. Additionally, p_j must be the first such under-subscribed project from P_k on s_i's preference list. This offer is always accepted, and after breaking any existing assignment involving s_i, s_i is provisionally assigned to p_j and l_k. Following this assignment, any pair (s_i, p), where s_i prefers p_j to p is *deleted*, which means that p is removed from s_i's preference list, and s_i is removed from the projected preference list of l_r for p, where l_r is the lecturer who offers p. The algorithm continues in this loop until no such l_k, p_j and s_i can be found.

The algorithm is described in pseudocode form in Figure 5 as Algorithm SPA-lecturer. We will prove that, once the main loop terminates, the assigned pairs constitute the stable matching that is simultaneously best-possible for all lecturers.

5.2 Correctness of Algorithm SPA-lecturer

The correctness of the algorithm, together with the optimality property of the constructed matching, may be established by the following sequence of lemmas.

Lemma 5.1 *Algorithm SPA-lecturer terminates with a matching.*

```
SPA-lecturer(I) {
    assign each student, project and lecturer to be free;
    while (some lecturer l_k is under-subscribed and
        there is some (student, project) pair (s_i, p_j) where
        s_i is not provisionally assigned to p_j and
        p_j ∈ P_k is under-subscribed and s_i ∈ L_k^j)
    {
        s_i = first such student on l_k's list;
        p_j = first such project on s_i's list;
        if (s_i is provisionally assigned to some project p)
            break the provisional assignment between s_i and p;
        /* l_k offers p_j to s_i */
        provisionally assign s_i to p_j; /* and to l_k*/
        for each successor p of p_j on s_i's list
            delete (s_i, p);
    }
    return {(s_i, p_j) ∈ S × P : s_i is provisionally assigned to p_j};
}
```

Figure 5: Pseudocode of Algorithm SPA-lecturer.

Proof: Each loop iteration involves a provisional assignment: either the first assignment for a student, or an assignment that the student prefers to his/her previous assignment. Therefore, the number of iterations is bounded by the total length of the student preference lists, which is linear in the size of the input. Finally, it is clear that, once the main loop terminates, the assigned pairs constitute a matching. ■

Lemma 5.2 *No pair deleted during an execution of Algorithm SPA-lecturer 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 because s_i is provisionally assigned to some project p, where s_i prefers p to p_j. On subsequent iterations, s_i can only improve his/her assignment, and so, by transitivity, s_i prefers his/her final assignment to p_j. Therefore, (s_i, p_j) cannot form a blocking pair. ■

Lemma 5.3 *A matching generated by Algorithm SPA-lecturer is stable.*

Proof: By Lemma 5.1, let M be the matching generated by an arbitrary execution E of the algorithm. Suppose for a contradiction that M is blocked by the pair (s_i, p_j), where l_k is the lecturer offering p_j. By Lemma 5.2, (s_i, p_j) is not deleted, and so, upon termination of E, $s_i \in \mathcal{L}_k^j$. Also, we have that (s_i, p_j) must satisfy (a), (b) or (c) of Condition 3 for a blocking pair. We show a contradiction in each case.

(a) p_j is under-subscribed and l_k is under-subscribed.
Student s_i, project p_j and lecturer l_k satisfy the loop condition, contradicting the termination property established in Lemma 5.1.

(b) p_j is under-subscribed, l_k is full, and either $s_i \in M(l_k)$ or l_k prefers s_i to the worst student s' in $M(l_k)$.
Let T be the point in the execution immediately after s' obtains his/her final assignment $p' \in P_k$, and all subsequent deletions involving s' have occurred. Let M'

be the matching at time T, and let $B = \{s'\} \cup \{s \in B_k : l_k \text{ prefers } s \text{ to } s'\}$. Define also the following set:

$$F = \left\{ p \in P_k : \begin{array}{l} \text{there exists a student } s_l \in B \text{ such that } p \in A_l, \\ (s_l, p) \notin M' \text{ and } (s_l, p) \text{ is not deleted before time } T \end{array} \right\}.$$

The following properties of F must hold.

1. Any assignment in M involving l_k that was made after time T must involve a project from F, since s' is the worst student in $M(l_k)$.
2. Every $p \in F$ is full at time T, otherwise l_k would not have offered p' to s'.
3. $p_j \in F$, since $s_i \in B$ by Condition (b), and (s_i, p_j) is not deleted by Lemma 5.2, which implies that $(s_i, p_j) \notin M'$, since $(s_i, p_j) \notin M$.

Now since $p_j \in F$, the number of students assigned to l_k in M' is given by

$$|M'(l_k)| \;=\; \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M'(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)| \;\leq\; d_k. \tag{1}$$

Similarly, the number of students assigned to l_k in M is given by

$$|M(l_k)| \;=\; \sum_{p_f \in F \setminus \{p_j\}} |M(p_f)| + |M(p_j)| + \sum_{p_g \in P_k \setminus F} |M(p_g)|.$$

Now, since all assignments in M involving l_k that were made after time T only involve projects from F (Property 1) and all projects in F are full in M' (Property 2), we have that

$$|M(l_k)| \;\leq\; \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)|.$$

Finally, we are given that p_j is under-subscribed at the termination of E (Condition (b)). Therefore

$$\begin{aligned} |M(l_k)| \;&<\; \sum_{p_f \in F \setminus \{p_j\}} |M'(p_f)| + |M'(p_j)| + \sum_{p_g \in P_k \setminus F} |M'(p_g)| \\ &=\; |M'(l_k)| \leq d_k \end{aligned}$$

by Equation 1. So, l_k is under-subscribed at the termination of E, contradicting Condition (b).

(c) p_j is full and l_k prefers s_i to the worst student s' assigned to p_j.
We have that l_k prefers s_i to s', and so at the time l_k offered p_j to s', (s_i, p_j) must have been deleted (otherwise l_k would have offered p_j to s_i). This is a contradiction, since by Lemma 5.2, (s_i, p_j) blocks M only if it is not deleted. ■

Lemma 5.4 *No stable pair is deleted during an execution of Algorithm SPA-lecturer.*

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 a stable matching containing (s_i, p_j). The deletion of (s_i, p_j) during E occurs because s_i is provisionally assigned to a project

Student preferences	Lecturer preferences	
s_1 : p_3 p_1	l_1 : s_1 s_2 s_3 s_4	l_1 offers p_1, p_2
s_2 : p_1 p_3	l_2 : s_2 s_1 s_4 s_3	l_2 offers p_3, p_4
s_3 : p_4 p_2	Project capacities: $c_i = 1$ $(1 \leq i \leq 4)$	
s_4 : p_2 p_4	Lecturer capacities: $d_i = 2$ $(1 \leq i \leq 2)$	

Figure 6: A SPA instance to illustrate that lecturer optimality differs from that in HR.

p', where s_i prefers p' to p_j. Let l' be the lecturer offering p', and let c' and d' be the capacities of p' and l' respectively.

The number of stable pairs (s', p') for which l' prefers s' to s_i must be less than c', for otherwise, one of these pairs must be deleted before s_i is assigned to p' during E, contradicting the assumption that (s_i, p_j) is the first stable pair deleted in E. Therefore in M, since $(s_i, p') \notin M$, either (i) p' is under-subscribed, or (ii) p' is full and assigned a student inferior to s_i.

We will prove that (s_i, p') blocks M. Firstly, we have that s_i prefers p' to p_j, and so (s_i, p') satisfies Conditions 1 and 2 of a blocking pair. It remains to show that (s_i, p') satisfies Condition 3(a), (b) or (c) of a blocking pair. If (ii) above holds, then (s_i, p') satisfies Condition 3(c). Otherwise, (i) holds, and p' is under-subscribed in M.

If l' is under-subscribed in M, then (s_i, p') satisfies Condition 3(a). Otherwise l' is full in M, and the only way (s_i, p') cannot satisfy Condition 3(b) is if l' is assigned d' students in M, each of whom he/she prefers to s_i. We will show a contradiction for this case.

Since M is a stable matching, each of these d' assignments forms a stable pair. Now, for l' to offer p' to s_i in E, only $0 \leq z < d'$ of these stable pairs are assigned (since l' must be under-subscribed to make an offer). However, none of the d' stable pairs is deleted before the offer to s_i in E, for otherwise (s_i, p_j) is not the first stable pair deleted. So, it must be the case that for the $d' - z$ unassigned stable pairs in E, each of the projects in these pairs is full (otherwise, the next offer from l' in E would involve one of the unassigned stable pairs, not s_i and p'). But then l' is full when the offer of p' is made to s_i in E, giving the required contradiction. ■

In the Hospitals / Residents problem, the hospital-oriented algorithm generates a stable matching M that is unequivocally optimal for the hospitals – as mentioned in Section 2.2, M satisfies the property that there is no stable matching M' and hospital h for which h prefers a resident in $M'(h) \backslash M(h)$ to the worst resident in $M(h)$. On the other hand, M is the worst possible stable matching for the residents – no stable matching assigns any resident to a worse hospital.

In our context, the stable matching produced by Algorithm SPA-lecturer is again unequivocally student-pessimal. However, the optimality situation is a little different. It can again be viewed as lecturer-optimal in a precise, if somewhat less emphatic sense.

In the example instance of Figure 6, the matching $M = \{(s_1, p_1), (s_2, p_3), (s_3, p_2), (s_4, p_4)\}$ is produced by an execution of Algorithm SPA-lecturer. However the matching $M' = \{(s_1, p_3), (s_2, p_1), (s_3, p_4), (s_4, p_2)\}$ is also stable. In M', each lecturer is assigned a student whom he/she prefers to one of the students whom he/she is assigned in M. Hence it is not the case that, in M, either lecturer is assigned the best two students that he/she can be assigned in any stable matching.

The somewhat weaker form of optimality that applies in this context can be described as follows. Let M and M' be two stable matchings for a given instance of SPA. By Theorem 4.1, we know that $|M| = |M'|$ and $|M(l_k)| = |M'(l_k)|$. For a given lecturer l_k

who is assigned different sets of students in M and M', let

$$M(l_k) \setminus M'(l_k) = \{s_1, \ldots, s_r\}$$

and

$$M'(l_k) \setminus M(l_k) = \{s'_1, \ldots, s'_r\},$$

where, in each case, the students are enumerated in the order in which they appear in $\mathcal{L}_k$. If l_k prefers s_i to s'_i for all i $(1 \leq i \leq r)$ we say that l_k *prefers* M to M'. Alternatively, and equivalently, l_k *prefers* M to M' if there is a one-to-one mapping f from $M'(l_k) \setminus M(l_k)$ to $M(l_k) \setminus M'(l_k)$ with the property that l_k prefers $f(s)$ to s for all s.

The following theorem summarises the key properties of the stable matching resulting from any execution of Algorithm SPA-lecturer.

Theorem 5.5 *For a given instance of SPA, any execution of Algorithm SPA-lecturer constructs the stable matching M satisfying the following properties.*
(i) Each student is unassigned or is assigned to the worst project he/she has in any stable matching.
(ii) Each project p_j is assigned the first x_j students not deleted from the projected preference list for p_j (where x_j is an integer independent of execution).
(iii) Each lecturer prefers M to any stable matching in which he/she has a different set of assigned students.

Proof:

(i) Let s_i be any student assigned in M. Algorithm SPA-lecturer deletes all successors of $M(s_i)$ from s_i's preference list. Now, by Lemma 5.4, no stable pair is deleted, and so s_i can have no worse partner than $M(s_i)$ in any stable matching. Hence, each student is either unassigned in M, and therefore in any stable matching (Theorem 4.1), or assigned to the worst project that he/she has in any stable matching.

(ii) Suppose there is some (student, project) pair $(s_i, p_j) \notin M$, such that s_i is not deleted from $\mathcal{L}_k^j$, where l_k is the lecturer offering p_j, and l_k prefers s_i to the worst student s' in $M(p_j)$. Since (s_i, p_j) is not deleted in E, s_i is either unassigned in M, or prefers p_j to $M(s_i)$. So, (s_i, p_j) is a blocking pair, contradicting the stability of M.

(iii) Let M' be any other stable matching, and let l_k be an arbitrary lecturer who is assigned different sets of students in M and M'. We construct a one-to-one mapping f from $M'(l_k) \setminus M(l_k)$ to $M(l_k) \setminus M'(l_k)$ with the required property.

Define a student $s \in M'(l_k) \setminus M(l_k)$ to be a *dominated* student in $\mathcal{L}_k$ if l_k prefers every student in $M(l_k)$ to s. For each such student s we have a free choice for $f(s)$. So we can complete the one-to-one mapping arbitrarily by dealing with these students once f has been defined for the other students in $M'(l_k) \setminus M(l_k)$.

So let s_{i_1} be a student in $M'(l_k) \setminus M(l_k)$ who is preferred by l_k to at least one of the students in $M(l_k)$, and suppose that $(s_{i_1}, p_{j_1}) \in M'$, where l_k offers p_{j_1}. By part (i) above, s_{i_1} prefers p_{j_1} to the project to which he/she is assigned in M. So to avoid (s_{i_1}, p_{j_1}) being a blocking pair for M, either

- p_{j_1} is fully subscribed in M with students whom l_k prefers to s_{i_1}; let s_{i_2} be such a student who is in $M(p_{j_1}) \setminus M'(p_{j_1})$; or
- p_{j_1} is under-subscribed in M but l_k is fully subscribed in M with students preferable to s_{i_1}.

However, we may reject the second possibility in view of our assumption that s_{i_1} is not dominated in $\mathcal{L}_k$, so we can consider the student s_{i_2}.

If $s_{i_2} \notin M'(l_k)$ we let $f(s_{i_1}) = s_{i_2}$. Otherwise, by (i) above, there is a project p_{j_2} offered by l_k such that $(s_{i_2}, p_{j_2}) \in M' \setminus M$, where s_{i_2} prefers p_{j_2} to p_{j_1}. To avoid (s_{i_2}, p_{j_2}) being a blocking pair for M, p_{j_2} must be fully subscribed in M with students whom l_k prefers to s_{i_2}; let s_{i_3} be such a student who is in $M(p_{j_2}) \setminus M'(p_{j_2})$. (Note that, since l_k prefers s_{i_2} to s_{i_1}, and therefore cannot prefer all the students in $M(l_k)$ to s_{i_2}, we may again ignore the other possibility.)

If $s_{i_3} \notin M'(l_k)$ we let $f(s_{i_1}) = s_{i_3}$. Otherwise we continue in this way to generate a sequence $s_{i_4}, \ldots, s_{i_t}$ of students, each member of which is preferred by l_k to its predecessor. Since the number of students is finite, this sequence must terminate with a student $s_{i_t} \in M(l_k) \setminus M'(l_k)$ such that l_k prefers s_{i_t} to s_{i_1}, and we set $f(s_{i_1}) = s_{i_t}$.

The sequence $s_{i_1}, \ldots, s_{i_t}$ is such that

- $s_{i_1} \in M'(l_k) \setminus M(l_k)$
- $s_{i_r} \in M'(l_k) \cap M(l_k)$ for $1 < r < t$
- $s_{i_t} \in M(l_k) \setminus M'(l_k)$
- l_k prefers s_{i_r} to $s_{i_{r-1}}$ for $1 < r \leq t$.

Additionally, we need to ensure that, if the same project occurs in the sequences originating with more than one of the students in $M'(l_k) \setminus M(l_k)$, we can choose a *unique* student matched to that project in $M' \setminus M$ on each occasion. For, suppose p_x is such a project that arises r times from pairs $(s_{u_1}, p_x), \ldots, (s_{u_r}, p_x) \in M' \setminus M$. So, arguing as before, p_x must be fully subscribed in M with students whom l_k prefers to all of $s_{u_1}, \ldots, s_{u_r}$. Clearly there must be at least r such students from which to choose, and a different one can be chosen in each sequence.

Finally, as indicated earlier, we complete the one-to-one mapping f in an arbitrary way for the dominated students. Hence f is, as claimed, a one-to-one mapping from $M'(l_k) \setminus M(l_k)$ to $M(l_k) \setminus M'(l_k)$ with the property that l_k prefers $f(s)$ to s for all s. Hence l_k prefers M to M'. The argument may be repeated for each relevant lecturer, and the result follows. ■

In the next subsection, we show how to implement Algorithm SPA-lecturer so that it runs in linear time.

5.3 Analysis of Algorithm SPA-lecturer

Even with the specialised data structures described in Section 3.3, it is not immediately clear whether Algorithm SPA-lecturer can be implemented to run in linear time. Consider for example the execution trace in Figure 7.

The sequence of offers made by l_1, i.e. $\langle (s_2, p_1),\ (s_1, p_2),\ (s_4, p_3),\ (s_1, p_1),\ (s_3, p_2) \rangle$, highlights two major differences between Algorithm SPA-lecturer and the hospital-oriented algorithm for HR. Firstly, a lecturer can make more than one offer to the same student (l_1 offers both p_2 and p_1 to s_1), and secondly, a lecturer may not make offers in order of preference (l_1 offers p_3 to s_4 before s_3 is made an offer).

Of course, the reason for these differences is that a project and lecturer can become under-subscribed after being full. For example, in step (ii) of the execution, p_1 is full, and so s_1 is assigned to his/her second preference, p_2. This makes p_2 full, which means that l_1 is not immediately able to make an offer to s_3. However, in step (iv), s_2 accepts a more

Student preferences	Lecturer preferences	
s_1 : p_1 p_2	l_1 : s_2 s_1 s_3 s_4 s_5	l_1 offers p_1, p_2, p_3
s_2 : p_4 p_1	l_2 : s_2	l_2 offers p_4
s_3 : p_2		
s_4 : p_3	Project capacities: $c_i = 1$ $(1 \leq i \leq 4)$	
s_5 : p_1 p_2 p_3	Lecturer capacities: $d_1 = 3$, $d_2 = 1$	

(i) l_1 offers p_1 to s_2; p_1 becomes full;

(ii) l_1 offers p_2 to s_1; p_2 becomes full;

(iii) l_1 offers p_3 to s_4; l_1 and p_3 become full;

(iv) l_2 offers p_4 to s_2; l_2 and p_4 become full; s_2 is freed from p_1; l_1 and p_1 become under-subscribed; (s_2, p_1) is deleted;

(v) l_1 offers p_1 to s_1; p_1 becomes full; s_1 is freed from p_2; p_2 becomes under-subscribed; (s_1, p_2) is deleted;

(vi) l_1 offers p_2 to s_3; l_1 and p_2 become full.

Figure 7: A SPA instance I, together with an execution trace of Algorithm SPA-lecturer as applied to I.

preferable project, thereby freeing p_1 for s_1, which in turn frees p_2 for s_3. Also l_1 was full just after step (iii), but under-subscribed just after step (iv).

The main problem then is how to efficiently find the next student to whom a given lecturer l_k can make an offer (since P_k may contain *several* projects that have become under-subscribed). It turns out that we can overcome this problem by restricting the non-deterministic choice of l_k in the main loop of Algorithm SPA-lecturer. Consider the implementation given in Figure 8.

For each lecturer l_k, we maintain a pointer $next_{l_k}$, which initially refers to the first student in $\mathcal{L}_k$. Also, for each project $p_j \in P_k$, we maintain a pointer $next_{p_j}$, which initially refers to the first student in $\mathcal{L}_k^j$. In the implementation, l_k repeatedly makes an offer to $next_{l_k}$, which moves left to right through $\mathcal{L}_k$, advancing one student at a time. If $next_{l_k}$ is already provisionally assigned to some project p' and lecturer l', project p' may become under-subscribed having previously been full (since $next_{l_k}$ leaves p'). At this point, the next offer made by l' can only involve one of two students - $next_{l'}$, or $next_{p'}$, where $next_{p'}$ is maintained as the first student not assigned to p' on the projected preference list of l' for p'. If $next_{p'}$ is defined, and either $next_{l'}$ is undefined or l' prefers $next_{p'}$ to $next_{l'}$, we require that l' makes an offer to $next_{p'}$ in the next loop iteration. Alternatively, if l' prefers $next_{l'}$ to $next_{p'}$, student $next_{p'}$ is in the scope of the variable $next_{l'}$, and so l' can revert to making offers to $next_{l'}$ as usual. By making this decision when there are only two possibilities ($next_{p'}$ and $next_{l'}$), we avoid the problem of choosing among several students when l' next makes an offer.

We briefly outline the other data structures used in the linear-time implementation. For each student s_i, we construct an array $preference_{s_i}$, the kth element of which is the kth-ranked post in s_i's preference list. Additionally, as in Algorithm SPA-student, we use next and previous pointers to link together the projects in this list that are offered by a given lecturer. When a lecturer l_k makes an offer to s_i, we can follow these pointers to

```
SPA-lecturer(I) {
    assign each student, project and lecturer to be free;
    assign p to be undefined;
    let S be a stack consisting of all lecturers;
    while (S ≠ ∅) {
        /* we seek a provisional assignment of student s_i to project p_j and lecturer l_k */
        if (p is defined) {      /* p has just become under-subscribed */
            p_j = p;
            l_k = lecturer offering p_j;
            s_i = next_{p_j};
            assign p to be undefined;
        } else {
            l_k = S.pop();
            if (l_k is full or next_{l_k} is undefined)
                continue;      /* to the next loop iteration */
            s_i = next_{l_k};
            advance next_{l_k} by one student;
            S.push(l_k);
            if (s_i's list contains no under-subscribed project in P_k)
                continue;      /* to the next loop iteration */
            p_j = first such project on s_i's list;
        }
        advance next_{p_j} by one student;
        if (s_i is provisionally assigned to some project p' and lecturer l') {
            if (p' is full and next_{p'} is defined and
              (next_{l'} is undefined or l' prefers next_{p'} to next_{l'}))
                p = p';
            break provisional assignment between s_i, p' and l';
        }
        provisionally assign s_i to p_j and l_k;
        for (each successor p' of p_j on s_i's list) {
            if (s_i == next_{p'})
                advance next_{p'} by one student;
            delete (s_i, p');
        }
    }
    return {(s_i, p_j) ∈ S × P : s_i is provisionally assigned to p_j};
}
```

Figure 8: Implementation of Algorithm SPA-lecturer.

find the first under-subscribed project from P_k in s_i's list. Such a traversal happens at most once for each lecturer, and so no project in s_i's preference list is visited more than a constant number of times. We also remark that all successors of a given project can be deleted from s_i's preference list simply by traversing the underlying array.

For each lecturer l_k, we build an array, $rank_{l_k}$, where $rank_{l_k}(s_i)$ is the index of student s_i in $\mathcal{L}_k$. We represent $\mathcal{L}_k$ by an array, $preference_{l_k}$, where $preference_{l_k}(rank_{l_k}(s_i))$ stores student s_i. Each lecturer l_k also stores a count of the number of students whom they are provisionally assigned, and a pointer $next_{l_k}$ into $preference_{l_k}$, as described earlier.

For each project p_j offered by l_k, we build an array $rank_{p_j}$, where $rank_{p_j}(s_i)$ is the index of student s_i in $\mathcal{L}_k^j$. We represent $\mathcal{L}_k^j$ by embedding a doubly-linked list in an array $preference_{p_j}$. For each student $s_i \in \mathcal{L}_k^j$, $preference_{p_j}(rank_{p_j}(s_i))$ stores the list node containing s_i. This node has a pointer to the next student in $\mathcal{L}_k^j$, and one to the previous student in $\mathcal{L}_k^j$. Each project also stores a count of the number of students whom it is

provisionally assigned, and a pointer $next_{p_j}$ into $preference_{p_j}$, as described earlier.

Using these data structures, we can find and delete a student from a project's preference list in constant time. For each preference list, we can also compare the ranks of any two students, and efficiently traverse through the sequence of students, missing out any students that have been deleted.

The above discussion therefore leads to the following result.

Theorem 5.6 *Algorithm SPA-lecturer may be implemented to run in $\Theta(\lambda)$ time and $O(mn)$ space, where λ is the total length of the preference lists, and n, m are the numbers of students and projects respectively, in a given SPA instance.*

6 Concluding remarks

6.1 Discussion of the SPA problem model

The definition of a blocking pair (s_i, p_j) considered in this paper includes the possibility that s_i was already assigned in M to a project offered by l_k, where l_k is the lecturer offering p_j, and seeks to become assigned to a preferred project p_j offered by the same lecturer. Such a switch obviously cannot alter the total number of students assigned to l_k.

If p_j is under-subscribed in M, then p_j has room for s_i, and l_k is implicitly indifferent about the switch, so would not prevent it from taking place. However if p_j is full in M, then the only way that the switch could occur is if l_k rejects a student from p_j. Moreover, l_k would agree to such a switch only if l_k prefers s_i to the worst student s' assigned to p_j in M. But this implies that, following the rejection of s', the number of students assigned to l_k would decrease by 1 if s' is unable to move to a worse project offered by l_k.

The following small SPA instance illustrates this phenomenon. We have two students, s_1, s_2, two projects, p_1, p_2, and one lecturer, l_1. Each of p_1 and p_2 has capacity 1, whilst l_1 has capacity 2. Student s_1 prefers p_1 to p_2, whilst s_2 finds only p_1 acceptable. Lecturer l_1 prefers s_1 to s_2. Clearly then, the matching $M_1 = \{(s_1, p_2), (s_2, p_1)\}$ admits the blocking pair (s_1, p_1), whilst $M_2 = \{(s_1, p_1)\}$ is the only stable matching.

In going from M_1 to M_2, we satisfy the blocking pair (s_1, p_1); however in doing so, l_1 loses a student, so in practice he/she may not agree to the switch. Hence one could alter Condition 3(c) of the blocking pair definition to prevent a change such as this from taking place. However we make two counter-arguments.

Firstly, by allowing a matching such as M_1 to be stable, we introduce an element of strategy into the problem. That is, it could be in a student's interest to submit a shorter preference list in order to obtain a more preferable project, rather than to submit his/her true preferences. For example, in the above instance, s_1 could list only p_1. In doing so, s_1 would be assigned to p_1 under either definition of Condition 3(c). On the other hand, by not listing every project that he/she finds acceptable, a student assumes a greater risk of being unassigned in the final matching.

Secondly, allowing both M_1 and M_2 to be stable would imply that this instance admits stable matchings of different sizes. Hence, to match as many students as possible, we would seek a maximum cardinality stable matching. However we conjecture that this problem is NP-hard. Evidence for this is given by the apparent lack of structure in this case: if both M_1 and M_2 are stable, there is no student-optimal stable matching. This is in contrast with the structure that follows from the definition of Condition 3(c) as adopted in this paper. In this context we have been able to prove several desirable properties of SPA, including an analogue of the Rural Hospitals Theorem (Theorem 4.1), and the existence of student-optimal and lecturer-optimal stable matchings (Theorems 3.5 and 5.5 respectively). With stable matching problems in general, it is often the case [12] that

the existence of structural properties and efficient algorithms are closely related to one another.

6.2 Open problems

In this paper we introduced SPA, and studied this problem from an algorithmic and structural point of view. A number of interesting open problems remain, including the following:

- Clearly many different formulations of the SPA model are possible. If only students supply preference lists, then a matching that optimises the students' satisfaction may be constructed using network flow techniques (see [1] for further details). Additionally, as in this paper, lecturers may supply preference lists, but over the projects that they offer rather than over students. We have considered this model from an algorithmic viewpoint – see [17] for further details. Finally, lecturers may have preferences over (student,project) pairs. In this setting, Fleiner [9] noted that the matroid-theoretic characterisation as described in Section 2.3 is applicable if a certain stability definition is imposed. However with this definition of stability there are strategic issues similar to those outlined in the previous subsection. It remains open as to whether a stability definition for this case can be formulated that leads to efficient algorithms, whilst avoiding such issues of strategy.

- If we allow ties in the preference lists of students and lecturers, different stability definitions are possible. These can be obtained by extending three stability definitions that have been applied to the Hospitals / Residents problem with Ties [13, 14]. Under the weakest of these stability criteria, so-called *weak stability*, every instance of SPA with ties admits a stable matching (this follows by breaking the ties arbitrarily and applying Algorithm SPA-student to the resulting instance of SPA, for example). However such matchings could be of different sizes for a given SPA instance with ties, and the problem of finding a maximum weakly stable matching is NP-hard (this follows by restriction, since the same problem has been shown to be NP-hard in the case of SMI with ties [16]). Under the two stronger stability criteria, namely *strong stability* and *super-stability*, an instance of SPA with ties need not admit a matching satisfying either criterion. However it remains open to construct algorithms for finding such a matching in each case, or reporting that none exists, for a given instance of SPA with ties.

- A further natural extension arises when each project p_j carries a lower bound $x_j \geq 0$. That is, p_j cannot run unless at least x_j students are assigned to it. Clearly a stable matching need not exist that satisfies all the projects' lower bounds. If lower bounds were present for hospitals in the case of HR, the problem of deciding whether a stable matching exists that satisfies them would be trivial, in view of the Rural Hospitals Theorem (i.e. find one stable matching, and if it does not satisfy the lower bounds, then no other stable matching does). However in view of the discussion following Theorem 4.1, the same is not true in the case of SPA. It is open as to whether there exists a polynomial-time algorithm for finding a stable matching if one exists, given an instance of SPA with lower bounds for the projects.

Acknowledgements

We would like to thank Tamás Fleiner for helpful discussions concerning the SPA problem model, and also the anonymous referees for their detailed comments, which have helped

to improve the presentation of this paper.

References

[1] D.J. Abraham. Algorithmics of two-sided matching problems. Master's thesis, University of Glasgow, Department of Computing Science, 2003.

[2] D.J. Abraham, R.W. Irving, and D.F. Manlove. The Student-Project Allocation Problem. In *Proceedings of ISAAC 2003: the 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 474–484. Springer-Verlag, 2003.

[3] A.A. Anwar and A.S. Bahaj. Student project allocation using integer programming. *IEEE Transactions on Education*, 46(3):359–367, 2003.

[4] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.

[5] J. Dye. A constraint logic programming approach to the stable marriage problem and its application to student-project allocation. BSc Honours project report, University of York, Department of Computer Science, 2001.

[6] A. Eguchi, S. Fujishige, and A. Tamura. A generalized Gale-Shapley algorithm for a discrete-concave stable marriage model. In *Proceedings of ISAAC 2003: the 14th Annual International Symposium on Algorithms and Computation*, volume 2906 of *Lecture Notes in Computer Science*, pages 495–504. Springer-Verlag, 2003.

[7] T. Fleiner. A matroid generalization of the stable matching polytope. In *Proceedings of IPCO '01: the 8th Conference on Integer Programming and Combinatorial Optimization*, volume 2081 of *Lecture Notes in Computer Science*, pages 105–114, 2001.

[8] T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.

[9] T. Fleiner. Personal communication, 2004.

[10] D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[11] D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

[12] D. Gusfield and R.W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, 1989.

[13] R.W. Irving, D.F. Manlove, and S. Scott. The Hospitals/Residents problem with Ties. In *Proceedings of SWAT 2000: the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pages 259–271. Springer-Verlag, 2000.

[14] R.W. Irving, D.F. Manlove, and S. Scott. Strong stability in the Hospitals/Residents problem. In *Proceedings of STACS 2003: the 20th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2607 of *Lecture Notes in Computer Science*, pages 439–450. Springer-Verlag, 2003.

[15] D. Kazakov. Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science, 2002.

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

[17] D.F. Manlove and G. O'Malley. Student project allocation with preferences over projects. In *Proceedings of ACID 2005: the 1st Algorithms and Complexity in Durham workshop*, volume 4 of *Texts in Algorithmics*, pages 69–80. KCL Publications, 2005.

[18] C. Ng and D.S. Hirschberg. Lower bounds for the stable marriage problem and its variants. *SIAM Journal on Computing*, 19:71–77, 1990.

[19] National Resident Matching Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.

[20] National Resident Matching Program. Why the Match? Web document available at `http://www.nrmp.org/whythematch.pdf`.

[21] L.G. Proll. A simple method of assigning projects to students. *Operational Research Quarterly*, 23(2):195–201, 1972.

[22] A. Romero-Medina. Implementation of stable solutions in a restricted matching market. *Review of Economic Design*, 3(2):137–147, 1998.

[23] 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.

[24] A.E. Roth. On the allocation of residents to rural hospitals: a general property of two-sided matching markets. *Econometrica*, 54:425–427, 1986.

[25] 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.

[26] C.-P. Teo, J. Sethuraman, and W.-P. Tan. Gale-Shapley stable marriage problem revisited: strategic issues and applications. In *Proceedings of IPCO '99: the 7th Conference on Integer Programming and Combinatorial Optimization*, volume 1610 of *Lecture Notes in Computer Science*, pages 429–438, 1999.

[27] C.Y. Teo and D.J. Ho. A systematic approach to the implementation of final year project in an electrical engineering undergraduate course. *IEEE Transactions on Education*, 41(1):25–30, 1998.

[28] M. Thorn. A constraint programming approach to the student-project allocation problem. BSc Honours project report, University of York, Department of Computer Science, 2003.