Finding Maximum Stable Matchings for the Student-Project Allocation Problem with Preferences Over Projects

Hoang Huu Viet, Le Van Tan, and Son Thanh Cao(✉)

School of Engineering and Technology, Vinh University, Vinh City, Vietnam
{viethh,tandhv,sonct}@vinhuni.edu.vn

Abstract. This paper proposes an efficient algorithm to find a maximum weakly stable matching for the Student-Project Allocation Problem with Preferences over Projects. We consider the problem as a constraint satisfaction problem and solve it using a local search approach based on the min-conflicts algorithm. By choosing a student generated by a fixed-increment rule and removing the undominated blocking pair formed by the student, we aim to remove all the blocking pairs formed by the student at each iteration of our algorithm. This allows our algorithm to obtain a solution of the problem as quickly as possible. Experimental results show that our algorithm is efficient in terms of both performance and solution quality for solving the problem.

Keywords: Student-Project Allocation Problem · Matching problem · Stable matching · Blocking pair · Local search

1 Introduction

In many undergraduate courses of universities, students have to undertake projects offered by lecturers. To do this, students firstly need to be assigned to projects such that both students and lecturers meet their preference and capacity constraints. This problem originally described by Abraham et al. [7] and known as the *Student-Project Allocation problem* (SPA). In the setting of SPA, each lecturer offers a set of projects and ranks a subset of students in strict order that he/she intends to supervise, whilst each student ranks a subset of the available projects that he/she finds acceptable in strict order. There exist capacity constraints on the maximum number of students that can be assigned to each project and lecturer. The aim of SPA is to allocate projects to students to satisfy the constraints on these preferences and capacities. Abraham et al. [7] proposed two linear-time algorithms to find a *stable matching* of students to projects in SPA. The first one returns a *student-optimal* stable matching in which each student gets the best project that he/she could get in any stable matching, while the second one returns a *lecturer-optimal* stable matching in which each lecturer gets the best set of students that he/she could get in any stable matching.

T. K. Dang et al. (Eds.): FDSE 2020, CCIS 1306, pp. 411–422, 2020.
https://doi.org/10.1007/978-981-33-4370-2_29

In SPA requiring each lecturer to rank a subset of students in a strict order is unfair for both lecturers and students. For example, lecturers often strongly prefer to supervise students with good academic results rather than students with poor academic results. This sometimes leads to conflicts among lecturers and students. Manlove and Malley [2] proposed a variant of SPA, called SPA *with Preferences over Projects* (SPA-P), where lecturers rank a subset of projects they offer in strict order rather than a subset of students. Given an SPA-P instance, Manlove et al. showed that stable matchings can have different sizes and the problem of finding a maximum cardinality stable matching is NP-hard.

Both SPA and SPA-P have recently received a great deal of attention from the research community in building automated applications for allocating students to projects. Examples may include the School of Computing Science, University of Glasgow [10], the Faculty of Science, University of Southern Denmark [1], the Department of Computing Science, University of York [8].

In the last few years, there are several researchers focused on designing efficient approximation algorithms to consider the lower and upper bounds for SPA-P. An algorithm is called r-approximation algorithm for SPA-P if it always finds a stable matching M with $|M| \geq |M_{opt}|/r$, where M_{opt} is a stable matching of maximum size. Manlove and Malley [2] extended the well-known Gale-Shapley algorithm [3] to find an 2-approximation algorithm, namely SPA-P-approx. This algorithm consists of a sequence of apply operations, in which an unassigned student with a non-empty list applies to the first project on his/her list to form pairs of a matching such that the lecturers and projects satisfy their capacity constraints. The algorithm returns a stable matching in a finite number of iterations. Iwama et al. [6] modified SPA-P-approx using Király's idea [9] to find an $\frac{3}{2}$- approximation algorithm. Recently, Manlove et al. [11] investigated an integer programming approach to SPA-P and proposed an $\frac{3}{2}$-approximation algorithm to find stable matchings that are very close to having maximum cardinality.

In this paper, we propose an algorithm to find maximum weakly stable matchings of SPA-P instances of large sizes. Our approach is based on a local search strategy applied for constraint satisfaction problems [12,13]. The local search strategy uses very little memory and can quickly find solutions in large state spaces and therefore, it is used for solving SPA-P of large sizes. Our experimental results show that our algorithm is much efficient than the SPA-P-approx algorithm [2] in terms of performance and solution quality for SPA-P instances of large sizes.

The rest of this paper is organized as follows. Section 2 describes preliminaries of SPA-P. Section 3 presents our proposed algorithm. Section 4 discusses the experiments and evaluations, and Sect. 5 concludes our work.

2 Preliminaries

This section recalls the SPA-P problem given in [2,4,5]. The SPA-P is defined consisting of a set $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ of *students*, a set $\mathcal{P} = \{p_1, p_2, \cdots, p_q\}$ of *projects* and a set $\mathcal{L} = \{l_1, l_2, \cdots, l_m\}$ of *lecturers*. Each lecturer l_k offers a set

P_k $(k = 1, 2, \cdots, m)$ of projects ranked in strict order of preference. We assume that $P_1, P_2, \cdots, P_k$ partitions $\mathcal{P}$ and each project $p_j \in \mathcal{P}$ is offered by a unique lecturer $l_k \in \mathcal{L}$. Also, each student s_i ranks a set of projects $A_i \subseteq \mathcal{P}$ in strict order of preference. If project $p_j \in \mathcal{P}$ is ranked by student s_i, we say that s_i finds p_j *acceptable*. Finally, each lecturer l_k has a capacity d_k, indicating the maximum number of students that can be supervised by l_k, and each project p_j has a capacity c_j, indicating the maximum number of students that can be assigned to p_j.

Definition 1 (assignment). *An assignment M is a subset of $S \times P$ such that $(s_i, p_j) \in M$ implies that $p_j \in A_i$. If $(s_i, p_j) \in M$, we say that s_i is assigned to p_j, p_j is assigned to s_i and we also say that s_i is assigned to l_k, l_k is assigned to s_i, where l_k is the lecturer who offers p_j.*

For any student $s_i \in \mathcal{S}$, if s_i is assigned to p_j in M, we let $M(s_i)$ denote p_j, otherwise, we say that s_i is *unassigned* in M or $M(s_i) = \emptyset$. For any project $p_j \in \mathcal{P}$, we let $M(p_j)$ denote the set of students assigned to p_j in M. We say that project p_j is *under-subscribed*, *full* or *over-subscribed* if $|M(p_j)| < c_j$, $|M(p_j)| = c_j$ or $|M(p_j)| > c_j$, respectively. Similarly, for any lecturer $l_k \in \mathcal{L}$, we let $M(l_k)$ denote the set of students assigned to l_k in M. We also say that lecturer l_k is *under-subscribed*, *full* or *over-subscribed* if $|M(l_k)| < d_k$, $|M(l_k)| = d_k$ or $|M(l_k)| > d_k$, respectively.

Definition 2 (matching). *A matching M is an assignment such that $|M(s_i)| \leq 1$ for each $s_i \in \mathcal{S}$, $|M(p_j)| \leq c_j$ for each $p_j \in \mathcal{P}$, and $|M(l_k)| \leq d_k$ for each $l_k \in \mathcal{L}$.*

Definition 3 (blocking pair). *A pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \backslash M$ is a blocking pair of a matching M, or blocks M, if the following three conditions are met:*

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. *p_j is under-subscribed and either*
   - *(a) $s_i \in M(l_k)$ and l_k prefers p_j to $M(s_i)$, or*
   - *(b) $s_i \notin M(l_k)$ and l_k is under-subscribed, or*
   - *(c) $s_i \notin M(l_k)$, l_k is full, and l_k prefers p_j to l_k's worst non-empty project, where l_k is the lecturer who offers p_j.*

Definition 4 (dominated blocking pair). *A blocking pair (s_i, p_j) dominates a blocking pair (s_i, p_k) if s_i prefers p_j to p_k.*

Definition 5 (undominated blocking pair). *A blocking pair (s_i, p_j) is undominated if there is no other blocking pair that dominates (s_i, p_j).*

Definition 6 (stable matching). *A matching M is called weakly stable if it admits no blocking pair, otherwise it is called unstable. The size of a weakly stable matching M, denoted $|M|$, is the number of students assigned in M.*

Table 1. An instance of SPA-P

Student preferences	Lecturer preferences	Project capacities
s_1: p_1 p_2 p_6	l_1: p_3 p_1 p_2 p_4	$c_1 = 1$
s_2: p_1 p_4	l_2: p_5 p_6	$c_2 = 2$
s_3: p_1 p_2 p_5		$c_3 = 2$
s_4: p_3	Lecturer capacities	$c_4 = 1$
s_5: p_3 p_5	$d_1 = 3$	$c_5 = 1$
s_6: p_5 p_3 p_6	$d_2 = 3$	$c_6 = 2$

In this paper, we consider only weakly stable matchings and therefore, we simply call a weakly stable matching a stable matching. The aim of SPA-P is to find a stable matching of maximum size, i.e. the stable matching admits the smallest number of unassigned students.

Definition 7 (perfect matching). *A stable matching M is called perfect if all students are assigned in M (i.e. $|M| = n$), otherwise it is called non-perfect.*

An instance of SPA-P consists of $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ and $\mathcal{L} = \{l_1, l_2\}$ is shown in Table 1, where $P_1 = \{p_3, p_1, p_2, p_4\}$, $P_2 = \{p_5, p_6\}$, $A_1 = \{p_1, p_2, p_6\}$, $A_2 = \{p_1, p_4\}$, $A_3 = \{p_1, p_2, p_5\}$, $A_4 = \{p_3\}$, $A_5 = \{p_3, p_5\}$, $A_6 = \{p_5, p_3, p_6\}$. The matching $M = \{(s_1, p_6), (s_2, p_4), (s_3, p_2), (s_5, p_3), (s_6, p_6)\}$ is unstable since there exist blocking pairs $\{(s_1, p_1), (s_1, p_2), (s_2, p_1), (s_3, p_1), (s_6, p_5)\}$ of M. Moreover, blocking pair (s_1, p_1) dominates blocking pair (s_1, p_2) since s_1 prefers p_1 to p_2 and blocking pair (s_1, p_1) is undominated. The matchings $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_5), (s_4, p_3), (s_6, p_6)\}$ and $M = \{(s_1, p_6), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_5), (s_6, p_6)\}$ are stable with sizes 5 and 6, respectively.

3 Algorithm for SPA-P

In this section, we propose an algorithm to find a maximum stable matching for SPA-P. We consider SPA-P as a constraint satisfaction problem (CSP), in which students are variables, projects ranked in each student's preference list is the domain of each variable and constraints are conditions of the blocking pair definition. Accordingly, a stable matching is an assignment of projects to students such that it does not violate constraints. Our key idea is that we adapt the min-conflicts heuristic for the CSP [12,13], in which at each iteration we select a project $p_j \in A_i$ to assign for a student s_i that results in the minimum number of conflicts with other students in terms of the number of blocking pairs. This means that we have to remove blocking pairs to improve stability of an unstable matching. However, some blocking pairs removed may be useless since the student remains involved in other blocking pairs. We thus focus on the

concept of undominated blocking pairs applied for the stable marriage problem with ties and incomplete lists [4,5].

Algorithm 1: SPA-P-MCH Algorithm

Input: - An instance I of SPA-P.
- A maximum number of iterations max_iters.

Output: A matching M.

```
1.  function Main(I)
2.      M := a randomly generated matching;
3.      M_best := M;
4.      s_i := a random student;
5.      iter := 0;
6.      while (iter ≤ max_iters) do
7.          iter := iter + 1;
8.          for (r = 1 ··· n) do
9.              s_i := mod(s_i, n) + 1;
10.             p_j := Find_Project(s_i, M);
11.             if (p_j ≠ ∅) then
12.                 break;
13.             end
14.         end
15.         if (p_j = ∅) then
16.             if (|M_best| < |M|) then
17.                 M_best := M;
18.             end
19.             if (|M_best| = n) then
20.                 break;
21.             else
22.                 M:= a randomly generated matching;
23.                 continue;
24.             end
25.         end
26.         M := M ∪ {(s_i, p_j)}, where p_j is offered by l_k;
27.         if (p_j is over-subscribed) then
28.             s_w := p_j's worst non-empty student;
29.             M := M\{(s_w, p_j)};
30.         end
31.         if (l_k is over-subscribed) then
32.             p_z := l_k's worst non-empty project;
33.             s_w := p_z's worst non-empty student;
34.             M := M\{(s_w, p_z)};
35.         end
36.     end
37.     return M_best;
38. end function
```

Our algorithm based on the min-conflicts heuristic for the SPA-P, so-called SPA-P-MCH, is shown in Algorithm 1. Initially, the algorithm assigns the best matching, M_{best}, to a randomly generated matching, M, and takes a random

student $s_i \in \mathcal{S}$. At each iteration, the algorithm finds a project $p_j \in A_i$ so that the pair (s_i, p_j) is an undominated blocking pair of the current matching, as shown in Algorithm 2. If there exists no such project p_j for every student, the algorithm has rearched a stable maching. If so, the algorithm checks if the current matching is better than M_{best} in terms of its size, it assigns the current matching to M_{best}. Moreover, if M_{best} is a perfect matching, the algorithm returns M_{best}, otherwise, it restarts at a randomly generated matching. However, if there exists a project p_j such that (s_i, p_j) is an undominated blocking pair, the algorithm removes (s_i, p_j) of M by assigning p_j to s_i, i.e. $M(s_i) = p_j$. Next, the algorithm checks if p_j is over-subscribed then it removes the pair $(s_w, p_j) \in M$ such that p_j is full, where s_w is the worst student assigned to p_j. Since p_j is assigned to s_i, this means l_k is assigned to s_i, where l_k is the lecturer who offers p_j. Therefore, the algorithm has to check the capacity d_k of l_k. Specifically, if l_k is over-subscribed, the pair $(s_w, p_z) \in M$ is removed such that l_k is full, where p_z is the worst project offered by l_k and s_w is the worst student assigned to project p_z. The algorithm repeats until either M_{best} is a perfect matching or a maximum number of iterations is reached. In the latter case, the algorithm returns either a maximum stable matching or an unstable matching.

Given a student $s_i \in \mathcal{S}$, the function to find a project $p_j \in A_i$ such that the pair (s_i, p_j) is an undominated blocking pair of a matching M is shown in Algorithm 2. The function performs an iteration for each project p_j in an ascending order of ranks in A_i and stops at the first blocking pair encountered, then (s_i, p_j) is an undominated blocking pair. If a blocking pair is found, the function returns p_j, otherwise, it returns an empty set.

Algorithm 2: Find pj such that (si, pj) is an undominated blocking pair

Input: A student $s_i \in \mathcal{S}$ and a matching M.
Output: A project $p_j \in A_i$ or empty.

```
1.  function p_j = Find_Project(s_i, M)
2.      p_j := ∅;
3.      sort s_i's rank list in ascending order;
4.      for (each p_k ∈ A_i such that rank(s_i, p_k) < rank(s_i, M(s_i))) do
5.          if ((s_i, p_k) is a blocking pair) then
6.              p_j := p_k;
7.              break;
8.          end
9.      end
10.     return p_j;
11. end function
```

An execution of the algorithm for the SPA-P instance shown in Table 1 is illustrated as in Table 2, where we initialize $M = \{(s_1, p_2), (s_2, p_4), (s_3, p_2), (s_5, p_5), (s_6, p_6)\}$ and the algorithm starts from s_1. After 5 iterations, the algorithm terminates and returns a stable matching $M = \{(s_1, p_6), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_5), (s_6, p_6)\}$, where every student is assigned to one project.

Table 2. An execution of the algorithm for the SPA-P instance shown in Table 1

Iter.	s_i	p_j	*Matching* M	*Blocking pairs*
0	s_1	-	$\{(s_1,p_2), (s_2,p_4), (s_3,p_2), (s_5,p_5), (s_6,p_6)\}$	$\{(s_1,p_1), (s_2,p_1), (s_3,p_1), (s_4,p_3), (s_5,p_3), (s_6,p_3)\}$
1	s_2	p_1	$\{(s_1,p_2), (s_2,p_1), (s_3,p_2), (s_5,p_5), (s_6,p_6)\}$	$\{(s_1,p_1), (s_3,p_1), (s_4,p_3), (s_5,p_3), (s_6,p_3)\}$
2	s_3	p_1	$\{(s_1,p_2), (s_2,p_1), (s_3,p_1), (s_5,p_5), (s_6,p_6)\}$	$\{(s_4,p_3), (s_5,p_3), (s_6,p_3)\}$
3	s_4	p_3	$\{(s_2,p_1), (s_3,p_1), (s_4,p_3), (s_5,p_5)\}, (s_6,p_6)\}$	$\{(s_1,p_6)\}$
4	s_1	p_6	$\{(s_1,p_6), (s_2,p_1), (s_3,p_1), (s_4,p_3), (s_5,p_5), (s_6,p_6)\}$	$\{\}$
5	$\forall s_i$	$\emptyset$	$\|M\| = 6 \rightarrow$ return $M = \{(s_1,p_6), (s_2,p_1), (s_3,p_1), (s_4,p_3), (s_5,p_5), (s_6,p_6)\}$.	

4 Experiments

In this section, we present experiments to evaluate the efficiency of our SPA-P-MCH algorithm. To do so, we compared the execution time and matching quality found by SPA-P-MCH with those found by SPA-P-approx [2]. We implemented both SPA-P-MCH and SPA-P-approx algorithms by Matlab R2017a software on a laptop computer with Core i7-8550U CPU 1.8 GHz and 16 GB RAM, running on Windows 10.

Datasets. To compare the efficiency of SPA-P-MCH and SPA-P-approx algorithms, we randomly generated SPA-P instances by varying parameters such as the number of students, lecturers and projects; the total capacities of the lecturers and projects; the number of projects ranked by each student in his/her preference list. Table 3 shows the number of students (n), lecturers (m) and projects (q) in our experiments. For each n varying from 500 to 5000 with steps 500, we randomly generated 100 instances of parameters (n, m, q) such that $0.02n \leq m \leq 0.1n$ (i.e., the student-to-lecturer ratio is from 10 to 50) and $0.1n \leq q \leq 0.5n$ (i.e., the student-to-project ratio is from 2 to 10 and each lecturer offers from 1 to 25 projects). In addition, we set a probability of incompleteness, $|A_i|/q$, indicating that, on average, each student s_i ranks $|A_i|$ projects in his/her preference list, where $|A_i| = 10, 20$ and 30.

Experiment 1. In this experiment, we set the total capacity, C, of projects offered by all the lecturers: $C = 1.1n$. Then, we distributed C to the capacity c_j of each project p_j such that $2 \leq c_j \leq 11$ (since the number of projects is $0.1n \leq q \leq 0.5n$). Next, we set the capacity of each lecturer l_k to be $d_k = \sum_{j=1}^{|P_k|} c_j$, where c_j is the capacity of project p_j offered by lecturer l_k. Figure 1(a) shows the average execution time of SPA-P-MCH and SPA-P-approx algorithms. The average execution time of both SPA-P-MCH and SPA-P-approx increases when n increases. When students increase the number of projects, $|A_i|(i = 1, 2, \cdots, n)$,

Table 3. Parameter values for experiments

ID	n	Number of lecturers ($0.02n \leq m \leq 0.1n$)		Number of projects ($0.1n \leq q \leq 0.5n$)		Number of projects offered by lecturer l_k (i.e. $\|P_k\|$)	
		Min	*Max*	*Min*	*Max*	*Min*	*Max*
1	500	10	50	50	250	1	25
2	1000	20	100	100	500	1	25
3	1500	30	150	150	750	1	25
4	2000	40	200	200	1000	1	25
5	2500	50	250	250	1250	1	25
6	3000	60	300	300	1500	1	25
7	3500	70	350	350	1750	1	25
8	4000	80	400	400	2000	1	25
9	4500	90	450	450	2250	1	25
10	5000	100	500	500	2500	1	25

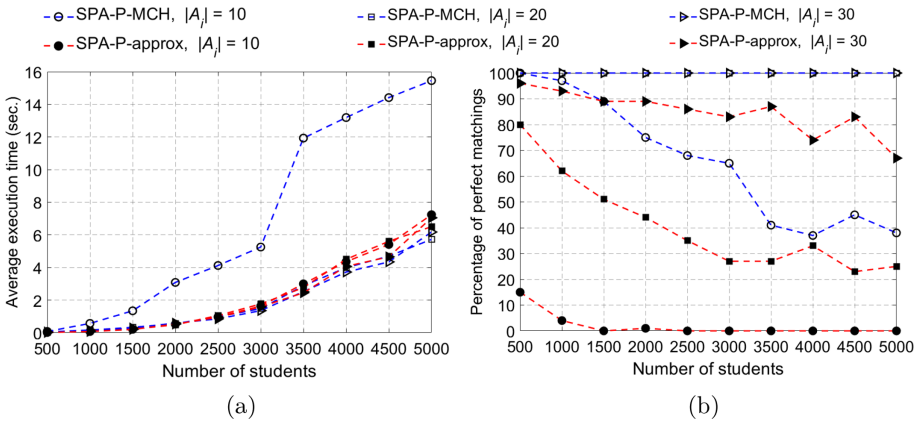

Fig. 1. Results for Experiment 1: (a) the average execution time and (b) the percentage of perfect matchings

in their preference lists, the average execution time of SPA-P-MCH is almost the same as that of SPA-P-approx. However, when $|A_i| = 10$, the average execution time of SPA-P-MCH is larger than that of SPA-P-approx. This is because when a found matching is non-perfect, SPA-P-MCH applies a restart strategy from a new random matching to find a better one in terms of matching size. Figure 1(b) shows the percentage of perfect matchings found by SPA-P-MCH and SPA-P-approx. When n increases, the percentage of perfect matchings found by both SPA-P-MCH and SPA-P-approx decreases. However, the percentage of perfect matchings found by SPA-P-MCH is always higher than that found by SPA-P-

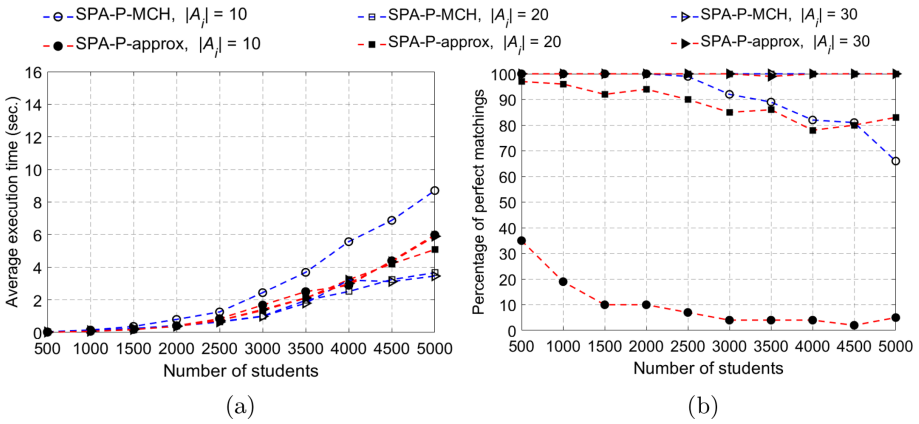

Fig. 2. Results for Experiment 2: (a) the average execution time and (b) the percentage of perfect matchings

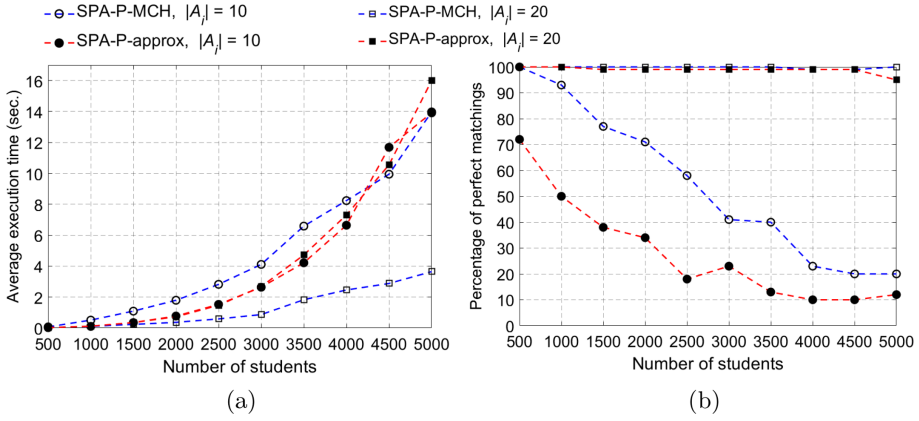

Fig. 3. Results for Experiment 3: (a) the average execution time and (b) the percentage of perfect matchings

approx. In particularly, when $|A_i| = 20$ or $|A_i| = 30$, SPA-P-MCH always finds 100% of perfect matchings.

Experiment 2. In this experiment, we increase the total capacity of projects offered by all the lecturers: $C = 1.2n$. Then, we distributed C to the capacity c_j of each project p_j such that $2 \leq c_j \leq 12$. The other parameters are set the same as those in Experiment 1. Figure 2 shows our experimental results. When the total capacity of projects is increased, i.e. the capacity of each project is increased, the average execution time of both SPA-P-MCH and SPA-P-approx decreases, while the percentage of perfect matchings found by both SPA-P-MCH and SPA-P-approx increases. This is because when the capacity of each project is increased, the opportunity to assign a project for students is increased.

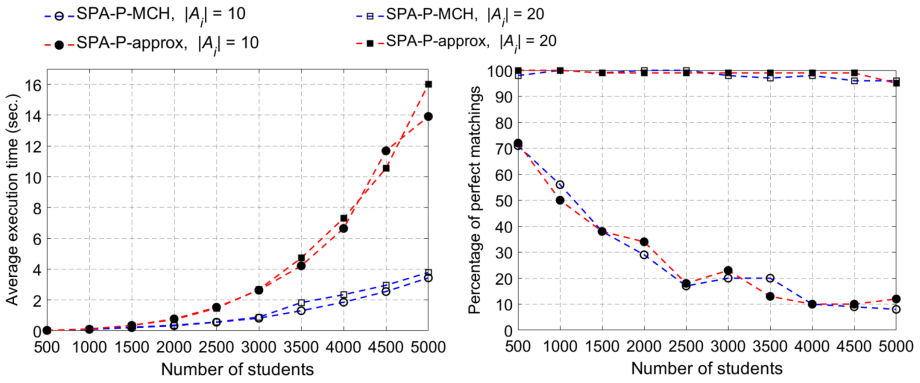

Fig. 4. Results for Experiment 4: (a) the average execution time and (b) the percentage of perfect matchings

Experiment 3. In the above two experiments, we set the capacity of each lecturer to be equal to the total capacities of projects offered by him/her. But, in practice, the capacity of each lecturer often is smaller than the total capacity of projects offered by him/her. Therefore, in this experiment, we set the capacity d_k of each lecturer l_k to be a random number between $0.6C_k$ and $0.85C_k$, where C_k is the total capacity of the projects offered by l_k. In addition, we set the total capacity of projects to be $C = 1.5n$ and distributed C to the capacity c_j of each project p_j such that $3 \leq c_j \leq 15$. The other parameter values are the same as those in Experiment 2. Figure 3(a) shows the average execution time of SPA-P-MCH and SPA-P-approx. The average execution time of SPA-P-MCH is much smaller than that of SPA-P-approx when the number of projects in students' preference lists is about 10, but it is almost the same of that of SPA-P-approx when the number of projects in students' preference lists is 20. Figure 3(b) shows the percentage of perfect matchings found by SPA-P-MCH and SPA-P-approx. SPA-P-MCH always finds 100% of perfect matchings when $|A_i| = 20$ and finds more than 65% of perfect matchings when $|A_i| = 10$. SPA-P-approx always finds the percentage of perfect matchings that is much lower than that found by SPA-P-MCH.

Experiment 4. In the above experiments, the maximum number of iterations in SPA-P-MCH is 20000. It should be noted that, when the umber of iterations in SPA-P-MCH is increased, the opportunity to find perfect matchings increases, but the execution time is also increased. In this experiment, we run SPA-P-MCH in which it returns the first stable matching found in iterations. Figure 4 shows the average execution time and matching quality found by SPA-P-MCH and SPA-P-approx, where the parameter values of SPA-P-MCH are the same as those in Experiment 3. It can be seen that the average execution time of SPA-P-MCH is much smaller than that of SPA-P-approx, while the percentage of perfect matchings found by SPA-P-MCH is almost the same as that found by

SPA-P-approx. In other words, SPA-P-MCH outperforms SPA-P-approx in terms of execution time in this case.

5 Conclusions

This paper proposed a SPA-P-MCH algorithm to find a maximum weakly stable matching for the SPA-P problem. Our algorithm starts to search a solution of the problem from a random matching. At each iteration, the algorithm finds a project for a student such that the pair (student, project) is an undominated blocking pair. If there exists such a project, the algorithm removes the pair (student, project). Otherwise, the algorithm checks if a perfect matching is found, it returns the found matching, otherwise, it continues searching a solution of the problem from a random matching. Our algorithm repeats until it finds a perfect matching or reaches a maximum number of search steps. In the latter case, the found matching is either a maximum stable matching or an approximate maximum matching. Experiments showed that our algorithm outperforms the SPA-P-approx in terms of execution time and matching quality for the SPA-P problem.

References

1. Chiarandini, M., Fagerberg, R., Gualandi, S.: Handling preferences in student-project allocation. Ann. Oper. Res. **275**(1), 39–78 (2017). https://doi.org/10.1007/s10479-017-2710-1
2. Manlove, D.F., O'Malley, G.: Student-project allocation with preferences over projects. J. Discrete Algorithms **6**(4), 553–560 (2008)
3. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Monthly **9**(1), 9–15 (1962)
4. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Local search for stable marriage problems with ties and incomplete lists. In: Proceedings of 11th Pacific Rim International Conference on Artificial Intelligence. pp. 64–75. Daegu, Korea (August 2010)
5. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Local search approaches in stable matching problems. Algorithms **6**(4), 591–617 (2013)
6. Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation bounds for the student-project allocation problem with preferences over projects. J. Discrete Algorithms **13**(1), 59–66 (2012)
7. Abraham, D.J., Irving, R.W., Manlove, D.F.: Two algorithms for the student-project allocation problem. J. Discrete Algorithms **5**(1), 73–90 (2007)
8. Kazakov, D.: Co-ordination of student-project allocation. Manuscript, University of York, Department of Computer Science http://www-users.cs.york.ac.uk/kazakov/papers/proj.pdf (2001)
9. Király, Z.: Better and simpler approximation algorithms for the stable marriage problem. Algorithmica **60**(1), 3–20 (2011). https://doi.org/10.1007/s00453-009-9371-710.1007/s00453-009-9371-7

10. Kwanashie, A., Irving, R.W., Manlove, D.F., Sng, C.T.S.: Profile-based optimal matchings in the student/project allocation problem. In: Proceedings of 25th International Workshop on Combinatorial Algorithms, pp. 213–225. Duluth, USA (15–17 October 2014)
11. Manlove, D., Milne, D., Olaosebikan, S.: An integer programming approach to the student-project allocation problem with preferences over projects. In: Proceedings of 5th International Symposium on Combinatorial Optimization, pp. 313–325. Morocco (April 2018)
12. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artif. Intell. **58**(1–3), 161–205 (1992)
13. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River (2009)