Undefined 0 (2024) 1–0
IOS Press

An efficient two-heuristic algorithm for the student-project allocation with preferences over projects

Hoang Huu Viet [a,*], Nguyen Thi Uyen [a], Son Thanh Cao [a], and Long Giang Nguyen [b]
[a] *Faculty of Information Technology, Vinh University, Nghe An, Vietnam*
E-mail: {viethh,uyennt,sonct}@vinhuni.edu.vn
[b] *Institute of Information Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam*
E-mail: nlgiang@ioit.ac.vn

Abstract. The Student-Project Allocation with preferences over Projects problem is a many-to-one stable matching problem that aims to assign students to projects in project-based courses so that students and lecturers meet their preference and capacity constraints. In this paper, we propose an efficient two-heuristic algorithm to solve this problem. Our algorithm starts from an empty matching and iteratively constructs a maximum stable matching of students to projects. At each iteration, our algorithm finds an unassigned student and assigns her/his most preferred project to her/him to form a student-project pair in the matching. If the project or the lecturer who offered the project is over-subscribed, our algorithm uses two heuristic functions, one for the over-subscribed project and the other for the over-subscribed lecturer, to remove a student-project pair in the matching. To reach a stable matching of a maximum size, our two heuristics are designed such that the removed student has the most opportunities to be assigned to some project in the next iterations. Experimental results show that our algorithm is efficient in execution time and solution quality for solving the problem.

Keywords: Approximation algorithm, heuristic search, matching problem, student-project allocation problem

1. Introduction

In project-based courses, students must be assigned to projects to build projects together. To do this, either lecturers can propose a list of students for their projects or departments can assign students to lecturers for doing projects. If doing so, it is evident that there exist conflicts not only among lecturers but also among students since lecturers usually choose good academic students for their projects, while students typically choose projects based on their preferences. The question for this problem is how to allocate students to projects to meet the preference requirements of both students and lecturers. To solve this problem, Abraham et al. introduced the *Student-Project Allocation problem* (SPA) in 2003 [1]. In particular, an instance of SPA involves three non-empty sets of students, projects, and lecturers. Each lecturer offers a set of projects and ranks a subset of students in strict order of preference in their lists to whom they intend to supervise, while each student ranks a subset of the available projects in a strict order of preference in their lists that they find acceptable. Moreover, each project is offered by a unique lecturer, both projects and lecturers have capacity constraints to indicate the maximum number of students assigned to projects and lecturers. In this context, a stable matching refers to an assignment of students to projects such that there exist no student-project unstable pairs, i.e., the student and project are not matched together in the matching, but they prefer each other to their current assigned partners in the matching. Then, they proposed a student-

*Corresponding author. E-mail: viethh@vinhuni.edu.vn

oriented algorithm running in a linear time to find a student-optimal stable matching for a given instance of SPA, in which each student is assigned to the most preferred project they could get in any stable matching. In 2007, Abraham et al. [2] extended their work and introduced two algorithms for SPA. The first one is a student-oriented algorithm described in their previous work [1], while the second one is a lecturer-oriented algorithm running in a linear time to find a lecturer-optimal stable matching for a given instance of SPA, in which each lecturer is assigned to the most preferred students to whom they could get in any stable matching.

In practical applications, requiring each lecturer to rank a subset of students in a strict order as in SPA 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. Moreover, if there are many students in SPA, it is difficult for lecturers to rank students in their lists. For these reasons, in 2008, Manlove and O'Malley [12] proposed a variant of SPA, called SPA *with preferences over Projects* (SPA-P), where lecturers rank a subset of projects in strict order rather than a subset of students in their lists. Given an instance of SPA-P, Manlove and O'Malley [12] showed that stable matchings could have different sizes, and the problem of finding a maximum stable matching is NP-hard even if each project and lecturer has a capacity 1.

In the last few years, several researchers have proposed efficient approximation algorithms and provided the lower and upper bounds for SPA-P. An algorithm is said to be an r-approximation algorithm for SPA-P if it results in a stable matching M with $|M_{opt}|/|M| \leq r$ for all SPA-P instances, where M_{opt} is the stable matching of maximum size. In 2008, Manlove and O'Malley [12] extended the well-known Gale-Shapley algorithm [5] to propose a 2-approximation algorithm with linear time complexity, namely SPA-P-approx. This algorithm consists of a sequence of proposals, in which an unassigned student with a non-empty list proposes the most preferred project on her/his list to form student-project 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. In 2012, Iwama et al. [6] extended the SPA-P-approx [12] using Király's idea [8] and proposed a $\frac{3}{2}$-approximation algorithm with a linear time complexity, namely SPA-P-approx-promotion, to find a stable matching for instances of SPA-P. In 2020, Viet et al. [16] considered SPA-P as a constraint satisfaction problem and proposed a local search approach based on the min-conflicts algorithm [13,15] to solve it. Recently, Manlove et al. [10,11] investigated an integer programming approach to SPA-P and proposed a $\frac{3}{2}$-approximation algorithm to find stable matchings that are very close to having maximum cardinality over their tested instances.

So far, both SPA and SPA-P have received significant attention from the research community for their roles in developing automated systems for student project allocation. Several examples can be found at various institutions, such as the School of Computing Science at the University of Glasgow [9], the Faculty of Science at the University of Southern Denmark [3], and the Department of Computing Science at the University of York [7].

Our contribution: In this paper, we first analyze the weaknesses of the SPA-P-approx [12] and the SPA-P-approx-promotion [6] algorithms for solving the SPA-P problem. Accordingly, we propose two heuristic functions to improve the drawbacks of the SPA-P-approx and SPA-P-approx-promotion algorithms. We then propose an efficient two-heuristic algorithm, namely SPA-P-heuristic, for solving the SPA-P problem. We also show that our algorithm returns a stable matching after a finite number of iterations. To conduct our experiments, we propose a method to generate random SPA-P instances. Our experimental results over all the tested scenarios show that our proposed algorithm is much more efficient than the SPA-P-approx [12] and SPA-P-approx-promotion [6] algorithms in terms of execution time and solution quality for SPA-P instances of large sizes. Therefore, our algorithm can be applied to develop intelligent systems for student project allocation.

The remainder of this paper is structured as follows. Section 2 gives a formal definition of SPA-P, Section 3 presents our proposed algorithm, Section 4 discusses the experiments, and Section 5 concludes our work.

2. SPA-P Problem

In this section, we remind the SPA-P problem given in [12,6]. An instance I of the SPA-P problem comprises a set $S = \{s_1, s_2, \cdots, s_n\}$ of *students*, a set $P = \{p_1, p_2, \cdots, p_q\}$ of *projects*, and a set $L = \{l_1, l_2, \cdots, l_m\}$ of *lecturers*, where (*i*) Each lecturer $l_k \in L$ offers a non-empty set P_k of projects in strict order of preference in their lists, subject to $P_1 \cup P_2 \cup \cdots \cup P_m = P$ and $P_{k_1} \cap P_{k_2} = \emptyset$, $\forall k_1 \neq k_2$; (*ii*) Each student $s_i \in S$ ranks a non-empty set $A_i \subseteq P$ of projects in strict order of preference in their lists; (*iii*) Each lecturer $l_k \in L$ has a *capacity* $d_k \in \mathbb{Z}^+$ to indicate the maximum number of students to whom they can supervise; and (*iv*) Each project $p_j \in P$ has a *capacity* $c_j \in \mathbb{Z}^+$ to indicate the maximum number of students who can work p_j together. Hereafter, we use the list of notations shown in Table 1 for reader convenience.

For $\forall s_i \in S$, $\forall l_k \in L$, and $\forall p_j \in P$, we denote $rank(s_i, p_j)$ and $rank(l_k, p_j)$ by the rank of p_j in s_i's and l_k's lists, respectively. If s_i and l_k prefer p_j as the α^{th} and β^{th} choices in their lists $(1 \leq \alpha, \beta \leq q)$, then $rank(s_i, p_j) = \alpha$ and $rank(l_k, p_j) = \beta$, respectively. For $\forall p_j \in A_i$ and $\forall p_t \in A_i$, if s_i prefers p_j to p_t, we denote by $rank(s_i, p_j) < rank(s_i, p_t)$. For $\forall p_j \in P_k$ and $\forall p_t \in P_k$, if l_k prefers p_j to p_t, we denote by $rank(l_k, p_j) < rank(l_k, p_t)$. Moreover, we denote $rank(s_i, p_j) = 0$ for $\forall p_j \in P \backslash A_i$ and $rank(l_k, p_j) = 0$ for $\forall p_j \in P \setminus P_k$.

An *assignment* M in I is a subset of $S \times P$ such that if $(s_i, p_j) \in M$, then $p_j \in A_i$. If l_k offers p_j and $(s_i, p_j) \in M$, then we say that s_i is assigned to p_j and l_k in M, p_j is assigned to s_i in M, and l_k is assigned to s_i in M.

For $\forall s_i \in S$, we denote $M(s_i)$ by the set of projects assigned to s_i in M and $|M(s_i)|$ by the number of projects in $M(s_i)$. If $M(s_i) = \varnothing$, then we say that s_i is unassigned in M. For $\forall p_j \in P$, we denote $M(p_j)$ by the set of students assigned to p_j in M and $|M(p_j)|$ by the number of students in $M(p_j)$. If $|M(p_j)| > c_j$, $|M(p_j)| < c_j$, or $|M(p_j)| = c_j$, then we say that p_j is *over-subscribed*, *under-subscribed*, or *full*, respectively. For $\forall l_k \in L$, we denote $M(l_k)$ by the set of students assigned to l_k in M and $|M(l_k)|$ by the number of students in $M(l_k)$. If $|M(l_k)| > d_k$, $|M(l_k)| < d_k$, or $|M(l_k)| = d_k$, then we say that l_k is *over-subscribed*, *under-subscribed*, or *full*, respectively.

A *matching* M in I is an assignment such that $|M(s_i)| \leq 1$, $|M(p_j)| \leq c_j$, and $|M(l_k)| \leq d_k$ for $\forall s_i \in S$, $\forall p_j \in P$, and $\forall l_k \in L$. If $|M(s_i)| = 1$, we denote $M(s_i)$ by the project assigned to s_i in M.

A pair $(s_i, p_j) \in (S \times P) \setminus M$ is a *blocking pair* for a matching M if all the following conditions are met, where $p_t = M(s_i)$:

1. $p_j \in A_i$, i.e., s_i finds p_j acceptable;
2. either $p_t = \varnothing$ or $rank(s_i, p_j) < rank(s_i, p_t)$, i.e., s_i prefers p_j to p_t;
3. $|M(p_j)| < c_j$ and either
   (a) $s_i \in M(l_k)$ and $rank(l_k, p_j) < rank(l_k, p_t)$, or
   (b) $s_i \notin M(l_k)$ and $|M(l_k)| < d_k$, or
   (c) $s_i \notin M(l_k)$, $|M(l_k)| = d_k$, and $rank(l_k, p_j) < rank(l_k, p_z)$, where p_z is l_k's worst non-empty project, i.e., l_k ranks p_z with the lowest priority in $M(l_k)$, where $M(p_z) \neq \varnothing$.

Table 1

List of some notations

I	Instance of the SPA-P problem
S	Set of students
L	Set of lecturers
P	Set of projects
A_i	Set of projects ranked by student $s_i \in S$
P_k	Set of projects offered by lecturer $l_k \in L$
M	Matching
$M(s_i)$	Set of projects assigned to student s_i in M
$M(p_j)$	Set of students assigned to project p_j in M
$M(l_k)$	Set of students assigned to lecturer l_k in M
s_i	Student
l_k	Lecturer
p_j	Project
c_j	Capacity of project $p_j \in P$
d_k	Capacity of lecturer $l_k \in L$
n	Number of students
m	Number of lecturers
q	Number of projects

The concept of blocking pair refers to a situation where a student s_i finds a project p_j acceptable and s_i prefers p_j to her/his currently assigned project, then s_i and p_j have the potential to form a better matching than the current matching by replacing their current assigned partners.

A matching M in I is called *stable* if no blocking pair exists for M; otherwise, M is called *unstable*. Given a stable matching M, we denote $|M|$ by the number of students assigned in M, i.e., the size of M. If $|M| = n$, then M is called *perfect*; otherwise, M is called *non-perfect*. The SPA-P problem aims to find a stable matching with maximum size, known as MAX-SPA-P [12].

An instance I of the SPA-P problem is given in Table 2, where $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, $P = \{p_1, p_2, p_3, p_4, p_5\}$, $L = \{l_1, l_2\}$. The students' and lecturers' lists columns show the preference lists of students and lecturers over projects, respectively, i.e., $A_1 = \{p_1, p_2, p_5\}$, $A_2 = \{p_1, p_4\}$, $A_3 = \{p_2, p_1, p_4\}$, $A_4 = \{p_3\}$,

Table 2
An instance of the SPA-P problem

Students' lists	Lecturers' lists	Students' ranks	Lecturers' ranks
s_1: p_1 p_2 p_5	l_1: p_3 p_1 p_2	s_1: 1 2 0 0 3	l_1: 2 3 1 0 0
s_2: p_1 p_4	l_2: p_4 p_5	s_2: 1 0 0 2 0	l_2: 0 0 0 1 2
s_3: p_2 p_1 p_4		s_3: 2 1 0 3 0	
s_4: p_3		s_4: 0 0 1 0 0	
s_5: p_3 p_4		s_5: 0 0 1 2 0	
s_6: p_5 p_3 p_4		s_6: 0 0 2 3 1	
Projects' capacities: $c_1 = 2$, $c_2 = 2$, $c_3 = 1$, $c_4 = 1$, $c_5 = 2$.			
Lecturers' capacities: $d_1 = 3$, $d_2 = 3$.			

$A_5 = \{p_3, p_4\}$, $A_6 = \{p_5, p_3, p_4\}$, $P_1 = \{p_3, p_1, p_2\}$, and $P_2 = \{p_4, p_5\}$. The students' and lecturers' ranks columns show the rank of projects in the students' and lecturers' lists, respectively, where if s_i and l_k prefer p_j as the α^{th} and β^{th} choices in their lists, then $rank(s_i, p_j) = \alpha$ and $rank(l_k, p_j) = \beta$. For example, in the students' lists, we write "s_1: p_1 p_2 p_5", meaning that s_1 prefers p_1 as the first choice, p_2 as the second choice, and p_5 as the third choice and therefore, $rank(s_1, p_1) = 1$, $rank(s_1, p_2) = 2$, and $rank(s_1, p_5) = 3$ in the students' ranks. We use similar notations in the lecturers' lists. The matching $M = \{(s_1, p_5), (s_2, p_1), (s_3, p_2), (s_4, p_3), (s_5, p_4)\}$ is unstable because there exist two blocking pairs consisting of (s_1, p_1) and (s_6, p_5) for M. Specifically, $(s_1, p_1) \notin M$ and s_1 prefers p_1 to p_5, so s_1 should be assigned to p_1. Meanwhile, $(s_6, \varnothing) \notin M$, s_6 prefers the most p_5, and $|M(p_5)| < c_5$, so s_6 should be assigned to p_5. The matching $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_4), (s_4, p_3), (s_6, p_5)\}$ is a stable matching of size $|M| = 5$. On the contrary, the matching $M = \{(s_1, p_5), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ is a maximum stable matching and it is also perfect since its size is $|M| = 6$.

3. Proposed algorithm

In this section, we first propose two heuristic functions used in our algorithm. Then, we propose an algorithm to find stable matchings of maximum size for the SPA-P problem. Finally, we give an execution of our algorithm for the SPA-P instance given in Table 2.

3.1. Heuristic definitions

We consider the SPA-P-approx [12] and SPA-P-approx-promotion [6] algorithms for finding maximum stable matchings of SPA-P instances. The main principle of SPA-P-approx is as follows. At the beginning, the algorithm initializes a matching M to be empty, meaning that every student is unassigned to any project offered by lecturers. At each iteration, the algorithm finds the first project p_j of an unassigned student s_i with a non-empty list. If p_j is full, meaning that p_j does not have a slot for s_i, then the algorithm deletes p_j from s_i's list so that it does not choose p_j for s_i in the next iterations. Otherwise, the algorithm provisionally assigns p_j to s_i to form a stable pair $(s_i, p_j) \in M$. When p_j is assigned to s_i, the lecturer l_k who offered p_j is assigned to s_i. If l_k is over-subscribed, the algorithm removes an arbitrary student s_r from $M(p_z)$, where p_z is l_k's worst non-empty project, and deletes p_z in s_r's list. Meanwhile, SPA-P-approx-promotion [6] algorithm runs as follows. At the beginning, the algorithm marks all students as unpromoted. At each iteration, the algorithm chooses the first project p_j of an unassigned student s_i with a non-empty list. If p_j is full, the algorithm removes an arbitrary student s_r from $M(p_j)$ and adds (s_i, p_j) to M. If l_k is over-subscribed, the algorithm removes an arbitrary student s_r from $M(p_z)$ such that l_k is full, where l_k is the lecturer who offered p_j and p_z is l_k's worst non-empty project in $M(l_k)$.

We found that removing an arbitrary student s_r in $M(p_j)$ or $M(p_z)$ is a weak point of both SPA-P-approx and SPA-P-approx-promotion algorithms. For example, we consider a specific case where the three following conditions are met: (*i*) $M(p_z)$ consists of at least two students s_r and s_w; (*ii*) s_r ranks only one project; and (*iii*) s_w ranks more than one project. If we remove s_r from M, then s_r is unassigned in M forever, while if we remove s_w from M, then s_w can be assigned to some project in her/his list at the next iterations.

In the general case, we find in each iteration of the above algorithms that when a project $p_j \in A_i$ is assigned to a student $s_i \in S$, if p_j is over-subscribed, we need to remove a student from $M(p_j)$ so that p_j is full. We recognize that the student removed from $M(p_j)$ should have the most remaining projects in her/his list since she/he has the most opportunity to be assigned to some project at the next iterations. Moreover, when a project p_j is assigned to a student s_i, the lecturer l_k who offered p_j can be over-subscribed. If l_k is over-subscribed, we need to remove a student from $M(l_k)$ so that l_k is full. To keep M stable, the student removed from $M(l_k)$ must be a student assigned to a project p_z, which is l_k's worst non-empty project, so that condition $(3c)$ in the definition of a blocking pair is not violated. Similar to the case where p_j is over-subscribed, the student removed from $M(p_z)$ should have the most remaining projects in her/his list since she/he has the most opportunity to be assigned to some project at the next iterations. To solve these two issues, we propose two heuristic functions as follows:

Case 1: When a project p_j is over-subscribed, we propose a heuristic function for every student $s_r \in M(p_j)$ as follows:

$$f(s_r) = rank(l_k, p_j) + |A_r|/(q+1), \forall s_r \in M(p_j), \quad (1)$$

where $|A_r|$ is the number of projects in A_r and q is the number of projects in P. Then, the student is chosen to be removed from M as follows:

$$s_w = argmax(f(s_r)), \forall s_r \in M(p_j). \quad (2)$$

It is evident that a student s_w determined by Eq. (2) means that $rank(l_k, p_j)$ is maximum and $|A_w|$ is maximum. If we remove s_w from $M(p_j)$, then we keep the students in $M(p_j)$ who have the least opportunity to be reassigned to projects in their lists and remove the student s_w in $M(p_j)$ who has the most opportunity to be reassigned to some project in her/his list at the next iterations, since the student s_w ranks the most projects in her/his list.

Case 2: When a lecturer l_k is over-subscribed, we propose a heuristic function for every student $s_r \in M(l_k)$ as follows:

$$g(s_r) = rank(l_k, p_z) + |A_r|/(q+1), \forall s_r \in M(l_k), \quad (3)$$

where $p_z = M(s_r)$ is a project offered by l_k. Then, the student is chosen to be removed from M as follows:

$$s_w = argmax(g(s_r)), \forall s_r \in M(l_k). \quad (4)$$

Similarly, a student s_w determined by Eq. (4) means that $rank(l_k, p_z)$ is maximum and $|A_w|$ is maximum, where $p_z = M(s_w)$. Since $rank(l_k, p_z)$ is a positive integer number and $0 < |A_w|/(q + 1) < 1$, if we remove such a student s_w meaning that p_z is l_k's worst non-empty project and s_w ranks the most projects in her/his list. By doing so, we not only keep M stable but also keep in $M(l_k)$ the students who have the least opportunity to be reassigned to projects in their lists and remove the student s_w in $M(l_k)$ who has the most opportunity to be reassigned to some project in her/his list at the next iterations

Since the student s_w removed from $M(p_j)$ and $M(l_k)$ corresponds to the maximum $f(s_w)$ and $g(s_w)$ values given in Eq. (2) and Eq. (4), we call such a student s_w the *worst student* removed from either $M(p_j)$ or $M(l_k)$.

3.2. Algorithm for SPA-P

Our heuristic algorithm for finding maximum stable matchings of SPA-P instances, namely SPA-P-heuristic, is shown in Algorithm 1. During the execution of the algorithm, each student is marked active (i.e., $a(s_i) = 1$) or inactive (i.e., $a(s_i) = 0$). At the beginning, every student $s_i \in S$ is active and unassigned in M. At each iteration, if there exists an active student s_i with an empty list, then she/he becomes inactive forever (i.e., $a(s_i) = 0$). Therefore, she/he is unassigned in M and the algorithm runs the next iteration (lines 5–8). Otherwise, she/he is assigned to her/his most preferred project p_j to form a pair (s_i, p_j) in M, and she/he becomes inactive (lines 9–12). If p_j is over-subscribed, then the worst student s_w in $M(p_j)$ determined by Eq. (2) is removed from M (lines 14–15), s_w deletes p_j in her/his list (line 16), and she/he becomes active again (line 17). If l_k is over-subscribed, then the worst student s_w in $M(l_k)$ determined by Eq. (4) is removed from M (lines 20–22). If so, s_w deletes p_z in her/his list, where p_z is offered by l_k and assigned to s_w (line 23), and she/he becomes active again (line 24). The algorithm repeats until all students are inactive and returns a stable matching.

Lemma 3.1 *SPA-P-heuristic terminates after a finite number of iterations.*

Proof. Given an instance I of SPA-P, we have each student $s_i \in S$ ranked $|A_i|$ projects. Initially, every student $s_i \in S$ is marked active. At each iteration, a student s_i is assigned to her/his most preferred project p_j and she/he becomes inactive. When s_i is assigned

Algorithm 1: SPA-P-heuristic Algorithm

function `SPA-P-heuristic(`I`)`
  $M := \varnothing$;
  $a(s_i) := 1, \forall s_i \in S$;
  while *there exists an active* s_i **do**
    if s_i*'s list is empty* **then**
      $a(s_i) := 0$;
      continue;
    end
    $p_j :=$ the most prefered project on s_i's list;
    $l_k :=$ lecturer who offers p_j;
    $M := M \cup \{(s_i, p_j)\}$;
    $a(s_i) := 0$;
    if p_j *is over-subscribed* **then**
      $s_w := \text{argmax}(f(s_r))$, $f(s_r)$ is Eq.(1);
      $M := M \setminus \{(s_w, p_j)\}$;
      $rank(s_w, p_j) := 0$;
      $a(s_w) := 1$;
    end
    if l_k *is over-subscribed* **then**
      $s_w := \text{argmax}(g(s_r))$, $g(s_r)$ is Eq.(3);
      $p_z := M(s_w)$;
      $M := M \setminus \{(s_w, p_z)\}$;
      $rank(s_w, p_z) := 0$;
      $a(s_w) := 1$;
    end
  end
  return M;
end function

to p_j, the lecturer l_k who offered p_j is assigned to s_i. If both p_j and l_k are not over-subscribed, then the algorithm terminates after n iterations. If p_j or l_k is over-subscribed, then some student $s_w \in M(p_j)$ or $s_w \in M(l_k)$, respectively, is removed from M. If so, s_w deletes $M(s_w)$ from her/his list and becomes active again. Although s_w becomes active, since s_w deletes $M(s_w)$ in her/his list, s_w is not reassigned to $M(s_w)$. Thus, if some student s_r is not assigned to any project, then s_r deletes every project $p_t \in A_r$, making s_r's list empty and s_r inactive forever. We let $S = S_1 \cup S_2$, where S_1 is a set of students assigned to projects and S_2 is a set of students not assigned to any project. If so, we have (*i*) $S_1 \cap S_2 = \emptyset$; (*ii*) every $s_i \in S_1$ is inactive since s_i is assigned to some project in her/his list; and (*iii*) every $s_r \in S_2$ is inactive since s_r's list is empty after deleting $|A_r|$ projects in her/his list. This shows that all students are inactive after a finite number of iterations and therefore, the algorithm terminates since it runs when there exists an active student $s_i \in S$. □

Lemma 3.2 *SPA-P-heuristic finds a solution of SPA-P in $O(n \times q^2)$ time.*

Proof. It follows by the proof of Lemma 3.1 that in the best case, where every student is assigned to the first preferred project in their lists to form a stable matching, our algorithm takes $O(n)$ time. In the worst case, where every student ranks all the projects in P and proposes the last preferred project in their lists, our algorithm takes $O(n \times q)$ time. When a project p_j is over-subscribed, our algorithm finds the worst student s_w in $M(p_j)$ to remove from M, so it takes $O(q)$ time in the worst case. When a lecturer l_k is over-subscribed, our algorithm finds the worst student s_w in $M(l_k)$ to remove from M, so it takes $O(m)$ time in the worst case. Therefore, our algorithm takes $O(n \times q \times (O(q) + O(m))) = O(n \times q \times max(q, m))$ time to find a solution of SPA-P. Since each lecturer must propose at least one project, we have $q \geq m$ or $max(q, m) = q$. This shows that our algorithm takes $O(n \times q \times max(q, m)) = O(n \times q^2)$ time to find a solution of SPA-P. □

Lemma 3.3 *SPA-P-heuristic returns a stable matching.*

Proof. We assume that the algorithm returns a matching M and there exists a blocking pair $(s_r, p_t) \in (S \times P) \setminus M$ for M. During the execution of the algorithm, we consider two cases:

Case 1: If p_t is not deleted from s_r's list, then s_r's list is not empty. If s_r is unassigned in M, then s_r is marked active. If so, the while loop would not have terminated and we get a contradiction with Lemma 3.1. Hence, s_r is assigned in M. Since we assume that (s_r, p_t) blocks M, i.e., s_r prefers p_t to $p_z = M(s_r)$. However, when s_r proposes p_z, p_z was the first project on s_r's list and we get a contradiction. Hence, M is stable.

Case 2: If p_t is deleted from s_r's list, then this occurs when (*i*) p_t is over-subscribed. If so, p_t becomes full and (s_r, p_t) cannot block M; or (*ii*) l_k is over-subscribed, where l_k is the lecturer who offered p_t. If so, $g(s_r)$ is maximum as shown in Eq. (4), i.e., $rank(l_k, p_t)$ is maximum and $|A_r|$ is maximum. Since $rank(l_k, p_t)$ is a positive integer number and $0 < A_r/(q+1) < 1$, meaning that p_t is l_k's worst non-empty project in $M(l_k)$. Now l_k becomes full in M and l_k's worst non-empty project in $M(l_k)$ is better than p_t. Hence, (s_r, p_t) cannot block M. Since we assume that (s_r, p_t) blocks M, we get a contradiction. Hence, M is stable. □

Table 3
An execution of SPA-P-heuristic for the instance given in Table 2

Iter.	s_i	p_j	l_k	$M \cup (s_i, p_j)$	*Over-subscribed*	$M \setminus (s_r, p_t)$	M
1	s_1	p_1	l_1	(s_1, p_1)			$\{(s_1, p_1)\}$
2	s_2	p_1	l_1	(s_2, p_1)			$\{(s_1, p_1), (s_2, p_1)\}$
3	s_3	p_2	l_1	(s_3, p_2)			$\{(s_1, p_1), (s_2, p_1), (s_3, p_2)\}$
4	s_4	p_3	l_1	(s_4, p_3)	l_1	(s_3, p_2)	$\{(s_1, p_1), (s_2, p_1), (s_4, p_3)\}$
5	s_5	p_3	l_1	(s_5, p_3)	p_3	(s_5, p_3)	$\{(s_1, p_1), (s_2, p_1), (s_4, p_3)\}$
6	s_6	p_5	l_2	(s_6, p_5)			$\{(s_1, p_1), (s_2, p_1), (s_4, p_3), (s_6, p_5)\}$
7	s_3	p_1	l_1	(s_3, p_1)	p_1	(s_1, p_1)	$\{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_6, p_5)\}$
8	s_5	p_4	l_2	(s_5, p_4)			$\{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$
9	s_1	p_2	l_1	(s_1, p_2)	l_1	(s_1, p_2)	$\{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$
10	s_1	p_5	l_2	(s_1, p_5)			$\{(s_1, p_5), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$

3.3. Example

In this section, we consider the execution of our algorithm for the SPA-P instance given in Table 2. First, our algorithm sets all students to be active and unassigned in a matching M, i.e., $M = \{\}$. Then, it runs the iterations shown in Table 3, where $p_t = M(s_r)$. Specifically, the iterations are as follows:

1. At the 1st, 2nd, and 3rd iterations, s_1, s_2, and s_3 are assigned to their most preferred project p_1, p_1, and p_2, respectively, offered by l_1. Therefore, we have $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_2)\}$ and s_1, s_2, and s_3 are marked inactive.
2. At the 4th iteration, s_4 is assigned to her/his most preferred project p_3 offered by l_1. Therefore, we have $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_2), (s_4, p_3)\}$ and s_4 is marked inactive. Since l_1 is over-subscribed, from Eq. (3), we have $M(l_1) = \{s_1, s_2, s_3, s_4\}$, $g(s_1) = 2.50$, $g(s_2) = 2.33$, $g(s_3) = 3.50$, and $g(s_4) = 1.17$, i.e., $g(s_3)$ is maximum. From Eq. (4), (s_3, p_2) is removed from M. So, we have $M = \{(s_1, p_1), (s_2, p_1), (s_4, p_3)\}$, s_3 deletes p_2 from her/his list and she/he is active again.
3. At the 5th iteration, s_5 is assigned to her/his most preferred project p_3 offered by l_1. Therefore, we have $M = \{(s_1, p_1), (s_2, p_1), (s_4, p_3), (s_5, p_3)\}$ and s_5 is marked inactive. Since p_3 is over-subscribed, from Eq. (1), we have $M(p_3) = \{s_4, s_5\}$, $f(s_4) = 1.17$, and $f(s_5) = 1.33$, i.e., $f(s_5)$ is maximum. From Eq. (2), (s_5, p_3) is removed from M. So, we have $M = \{(s_1, p_1), (s_2, p_1), (s_4, p_3)\}$, s_5 deletes p_3 from her/his list and she/he is active again.
4. At the 6th iteration, s_6 is assigned to her/his most preferred project p_5 offered by l_2. Therefore, we have $M = \{(s_1, p_1), (s_2, p_1), (s_4, p_3), (s_6, p_5)\}$ and s_6 is marked inactive.
5. At the 7th iteration, s_3 is assigned to her/his most preferred project p_1 offered by l_1. Therefore, we have $M = \{(s_1, p_1), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_6, p_5)\}$ and s_3 is marked inactive. Since p_1 is over-subscribed, from Eq. (1), we have $M(p_1) = \{s_1, s_2, s_3\}$, $f(s_1) = 2.50$, $f(s_2) = 2.33$, and $f(s_3) = 2.33$, i.e., $f(s_1)$ is maximum. From Eq. (2), (s_1, p_1) is removed from M. So, we have $M = \{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_6, p_5)\}$, s_1 deletes p_1 from her/his list and she/he is active again.
6. At the 8th iteration, s_5 is assigned to her/his most preferred project p_4 offered by l_2. Therefore, we have $M = \{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ and s_5 is marked inactive.
7. At the 9th iteration, s_1 is assigned to her/his most preferred project p_2 offered by l_1. Therefore, we have $M = \{(s_1, p_2), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ and s_1 is marked inactive. Since l_1 is over-subscribed, from Eq. (3), we have $M(l_1) = \{s_1, s_2, s_3, s_4\}$, $g(s_1) = 3.33$, $g(s_2) = 2.33$, $g(s_3) = 2.33$, and $g(s_4) = 1.17$, i.e., $g(s_1)$ is maximum. From Eq. (4), (s_1, p_2) is removed from M. So, we have $M = \{(s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$, s_1 deletes p_2 from her/his list and she/he is active again.
8. At the 10th iteration, s_1 is assigned to her/his most preferred project p_5 offered by l_2. Therefore, we have $M = \{(s_1, p_5), (s_2, p_1), (s_3, p_1), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ and s_1 is marked inactive.

Since all students are inactive, the algorithm returns a stable matching $M = \{(s_1, p_5), (s_2, p_1), (s_3, p_1),$

(s_4, p_3), (s_5, p_4), $(s_6, p_5)\}$ of size $|M| = 6$, which is a perfect matching.

4. Experiments

In this section, we present some experiments to evaluate the performance of our SPA-P-heuristic algorithm. We chose the SPA-P-approx [12] and SPA-P-approx-promotion [6] (for short, we call it SPA-P-promotion) algorithms to compare their solution quality and execution time with those of our SPA-P-heuristic algorithm since both SPA-P-approx and SPA-P-promotion are approximation algorithms with a linear time complexity. We implemented three algorithms by Matlab R2019a software on a laptop computer with Core i7-8550U CPU 1.8 GHz and 16 GB RAM, running on Windows 11.

Datasets. To conduct our experiments, we generated random SPA-P instances with four parameters (n, m, q, σ), where n is the number of students, m is the number of lecturers, q is the number of projects, and σ is the total capacity of q projects offered by all the lecturers, i.e., $\sigma = \sum_{j=1}^{q} c_j$. Given four parameters (n, m, q, σ), our method to generate a random SPA-P instance is as follows:

1. Generate a set $S = \{1, 2, \cdots, n\}$ of students, a set $P = \{1, 2, \cdots, q\}$ of projects, and a set $L = \{1, 2, \cdots, m\}$ of lecturers.
2. Generate randomly non-empty sets $P_1, P_2, \cdots, P_m$ of projects such that $P_1 \cup P_2 \cup \cdots \cup P_m = P$ and $P_i \cap P_j = \emptyset$ for $i = 1, 2, \cdots m$, $j = 1, 2, \cdots m$, and $i \neq j$. We consider P_k as a set of projects offered by lecturer $l_k \in L$ $(k = 1, 2, \cdots, m)$.
3. Iterate for each $l_k \in L$ and each $p_j \in P$, if p_j is at the position β^{th} in P_k, then we set $rank(l_k, p_j) = \beta$; otherwise, we set $rank(l_k, p_j) = 0$. By doing so, we have a rank matrix of all the lecturers.
4. Distribute the total capacity σ of all the projects randomly to the capacity c_j of each project $p_j \in P$ $(j = 1, 2, \cdots, q)$ such that $0 < c_j < \sigma$ and $\sum_{j=1}^{q} c_j = \sigma$.
5. Calculate the total capacity ρ_k of all the projects $p_j \in P_k$ and generate the capacity d_k of each lecturer $l_k \in L$ by setting d_k to be some percentage of ρ_k (e.g., $d_k = 100\%\rho_k$, or d_k is a random integer number such that $80\%\rho_k \leq d_k \leq 100\%\rho_k$).
6. Generate randomly non-empty sets $A_1, A_2, \cdots, A_n$ of projects such that $A_i \subseteq P$. We consider A_i as a set of projects proposed by student $s_i \in S$ $(i = 1, 2, \cdots, n)$.
7. Iterate for each $s_i \in S$ and each $p_j \in P$, if p_j is at the position α^{th} in A_i, then we set $rank(s_i, p_j) = \alpha$; otherwise, we set $rank(s_i, p_j) = 0$. By doing so, we have a rank matrix of all the students.

Table 4
Parameter values for datasets in Experiments 1 and 2

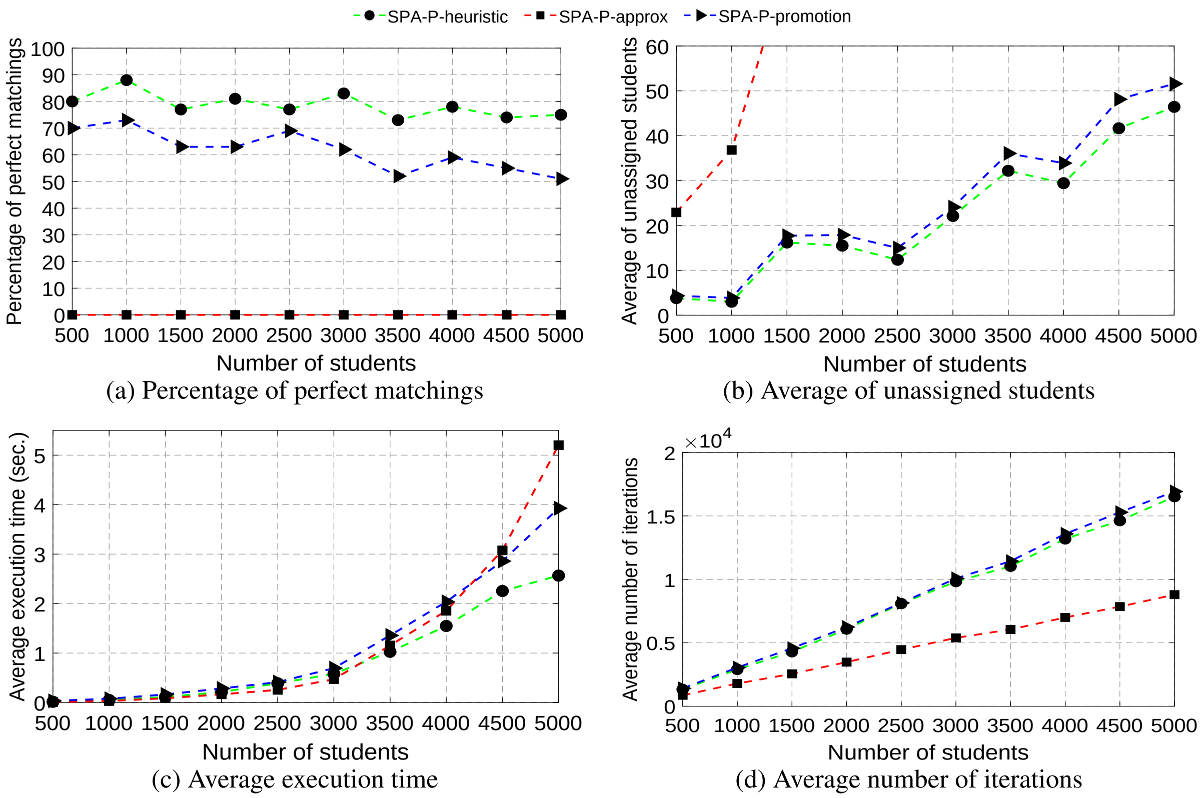
ID	n	Number of lecturers $(0.02n \leq m \leq 0.1n)$		Number of projects $(0.1n \leq q \leq 0.4n)$	
		Min	*Max*	*Min*	*Max*
1	500	10	50	50	200
2	1000	20	100	100	400
3	1500	30	150	150	600
4	2000	40	200	200	800
5	2500	50	250	250	1000
6	3000	60	300	300	1200
7	3500	70	350	350	1400
8	4000	80	400	400	1600
9	4500	90	450	450	1800
10	5000	100	500	500	2000

As a result, our method represents an instance of SPA-P by a rank matrix of students, a rank matrix of lecturers, a capacity list of projects, and a capacity list of lecturers, which are inputs for our algorithm.

In our experiments, we generated 100 instances of SPA-P for each value of n. In each instance, we chose the values of m and q to make the student-to-lecturer ratio and the student-to-project ratio suitable for real applications. Besides, σ is chosen based on the value of n. To compare the performance of our SPA-P-heuristic algorithm with that of SPA-P-approx and SPA-P-promotion algorithms for SPA-P instances, we ran SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms for each instance to find their solution and execution time. Then, we determined the percentage of perfect matchings, the average of unassigned students, and the average execution time found by each algorithm run on 100 instances of SPA-P to compare their performance.

4.1. Experiment 1

In this experiment, we chose the values of parameters n, m, and q as shown in Table 4. For each value of n varying from 500 to 5000 with steps 500, we generated 100 instances of SPA-P of parameters n, m, and q, where m and q are random numbers constrained by $0.02n \leq m \leq 0.1n$ and $0.1n \leq q \leq 0.4n$, respectively. The constraints of m and q mean that the



(a) Percentage of perfect matchings

(b) Average of unassigned students

(c) Average execution time

(d) Average number of iterations

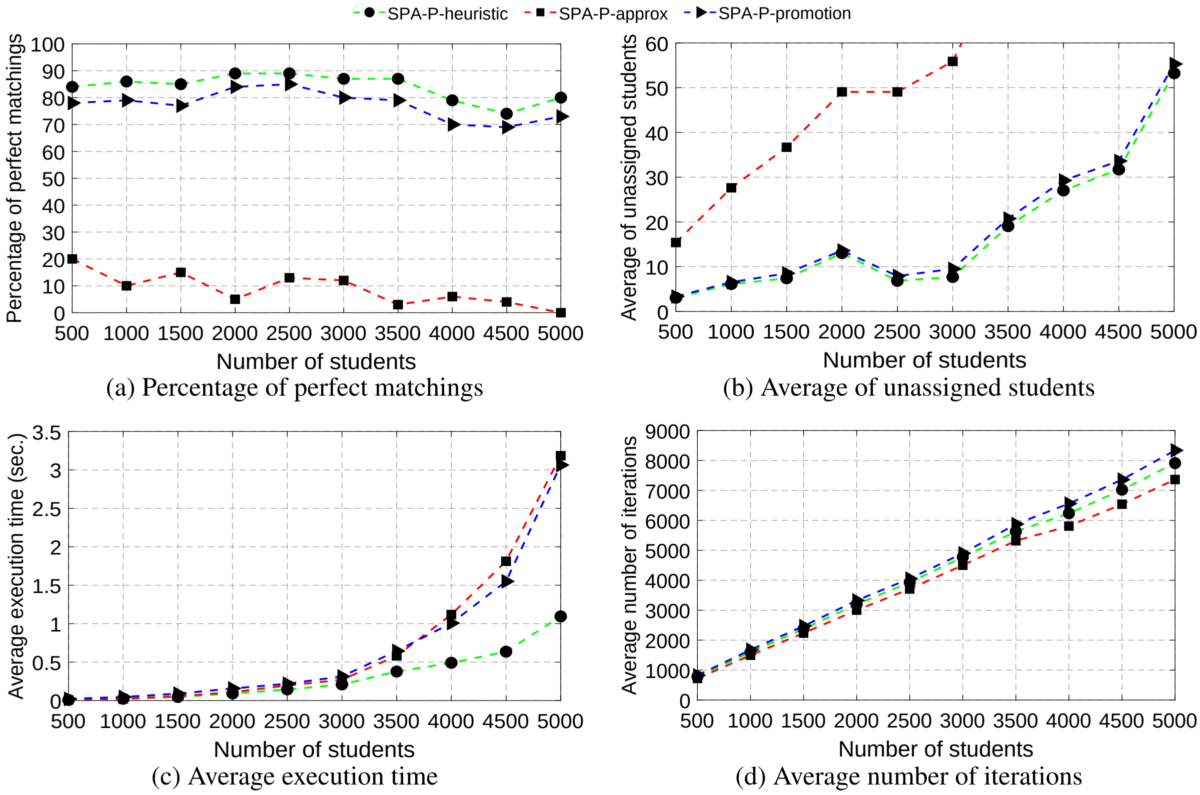
Fig. 1. Comparing solution quality and execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms

student-to-lecturer ratio is from 10 to 50 and each lecturer offers from 1 to 20 projects. In each instance, we let each student randomly rank from 1 to 20 projects in the set of projects offered by all lecturers. We set the total capacity σ of projects offered by all lecturers as n, i.e., $\sigma = n$. Then, we distributed σ randomly to the capacity c_j of each project $p_j \in P$ to ensure that $\sum_{j=1}^{q} c_j = \sigma$ and $1 \leq c_j \leq 100$. Besides, we set the capacity d_k of each lecturer l_k to the total capacity of projects offered by l_k, i.e., $d_k = \sum c_t$, where c_t is the capacity of projects $p_t \in P_k$. By setting so, this scenario is a challenging experiment for the algorithms to find perfect matchings in SPA-P instances since each student has only a slot to be assigned to each project in their lists.

Figure 1(a) shows the percentage of perfect matchings found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n increases from 500 to 5000 with steps 500, SPA-P-heuristic finds a much higher percentage of perfect matchings than SPA-P-promotion and SPA-P-approx. Specifically, SPA-P-heuristic finds from 73% to 88% of perfect matchings, SPA-P-promotion finds from 51% to 73% of perfect matchings, while SPA-P-approx fails to find any perfect matchings of SPA-P instances.

Figure 1(b) shows the average number of unassigned students found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n increases from 500 to 5000 with steps 500, SPA-P-approx finds stable matchings with more than 22 unassigned students. Meanwhile, SPA-P-heuristic finds fewer unassigned students in stable matchings than SPA-P-promotion. This means that the stable matchings found by SPA-P-heuristic are larger than those found by SPA-P-promotion in terms of size.

Figure 1(c) shows the average execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n increases from 500 to 5000 with steps 500, the average execution time of SPA-P-approx increases from 0.0097 seconds to 5.2027 seconds, the average execution time of SPA-P-promotion increases from 0.0327 seconds to 3.9276 seconds, and the average execution time of SPA-P-heuristic increases from 0.0150 seconds to 2.5655 seconds. We see that when $n \geq 4000$, SPA-P-heuristic runs about two times faster than SPA-P-promotion and SPA-P-approx.



(a) Percentage of perfect matchings (b) Average of unassigned students

(c) Average execution time (d) Average number of iterations

Fig. 2. Comparing solution quality and execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms

Figure 1(d) shows the average number of iterations found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. We see that the average number of iterations found by SPA-P-heuristic is slightly smaller than that found by SPA-P-promotion, but larger than that found by SPA-P-approx. However, the average execution time of SPA-P-heuristic is much smaller than that found by SPA-P-promotion and SPA-P-approx, meaning that at each iteration, SPA-P-heuristic needs a smaller computation than SPA-P-promotion and SPA-P-approx.

4.2. *Experiment 2*

In this experiment, we chose the values of parameters n, m, and q as those constraints in Experiment 1. In each randomly generated instance of SPA-P, we set each student to rank randomly from 1 to 20 projects in the set of projects offered by all lecturers. Moreover, we set the total capacity σ of projects offered by all lecturers as $1.1n$, i.e., $\sigma = 1.1n$, and distributed σ randomly to the capacity c_j of each project $p_j \in P$ such that $\sum_{j=1}^{q} c_j = \sigma$ and $1 \leq c_j \leq 100$. Besides, we set the capacity d_k of each lecturer $l_k \in L$ to a random integer number in $[0.9\rho_k, \rho_k]$, where ρ_k is the total capacity of projects offered by l_k. This means that $\sigma = \sum_{k=1}^{m} \rho_k = \sum_{j=1}^{q} c_j = 1.1n$. Since $0.9\rho_k \leq d_k \leq \rho_k$, we have $0.9\sum_{k=1}^{m} \rho_k \leq \sum_{k=1}^{m} d_k \leq \sum_{k=1}^{m} \rho_k$, i.e., $0.99n \leq \sum_{k=1}^{m} d_k \leq 1.1n$. Therefore, if some generated instances that $0.99n \leq \sum_{k=1}^{m} d_k < n$, then they have not any perfect matching.

Figure 2(a) shows the percentage of perfect matchings found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n varies from 500 to 5000 with steps 500, SPA-P-heuristic finds from 74% to 89% of perfect matchings, SPA-P-promotion finds from 69% to 85% of perfect matchings, while SPA-P-approx finds only from 0% to 20% of perfect matchings. It is obvious that SPA-P-heuristic finds a higher percentage of perfect matchings than SPA-P-promotion and SPA-P-approx. Compared to Experiment 1, we can see that when the total capacity of projects increases, i.e., the capacity of each project increases, it is easy for these algorithms to find perfect matchings in SPA-P instances.

Figure 2(b) shows the average number of unassigned students found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n increases from 500 to 5000 with steps 500, SPA-P-approx results in stable matchings with more than 15 unassigned students. In contrast, SPA-P-heuristic yields stable matchings with fewer unassigned students than SPA-P-promotion. This means that the stable matchings found by SPA-P-heuristic are larger than those generated by SPA-P-promotion in terms of size.

Figure 2(c) shows the average execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When n varies from 500 to 5000 in increments of 500, the average execution time of SPA-P-approx increases from 0.0083 seconds to 3.1844 seconds, the average execution time of SPA-P-promotion increases from 0.0220 seconds to 3.0629 seconds, and the average execution time of SPA-P-heuristic increases from 0.0083 seconds to 1.0946 seconds. This shows that SPA-P-promotion and SPA-P-approx exhibit similar execution time, while SPA-P-heuristic runs approximately three times faster than both SPA-P-promotion and SPA-P-approx.

Figure 2(d) shows the average number of iterations used by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. As in Experiment 1, we can see that SPA-P-heuristic used a smaller number of iterations than SPA-P-promotion, but a larger number of iterations than SPA-P-approx. Moreover, we can see that when the total capacity of projects increases, all these three algorithms not only find stable matchings faster than those, but also use a smaller number of iterations compared to those in Experiment 1.

4.3. Experiment 3

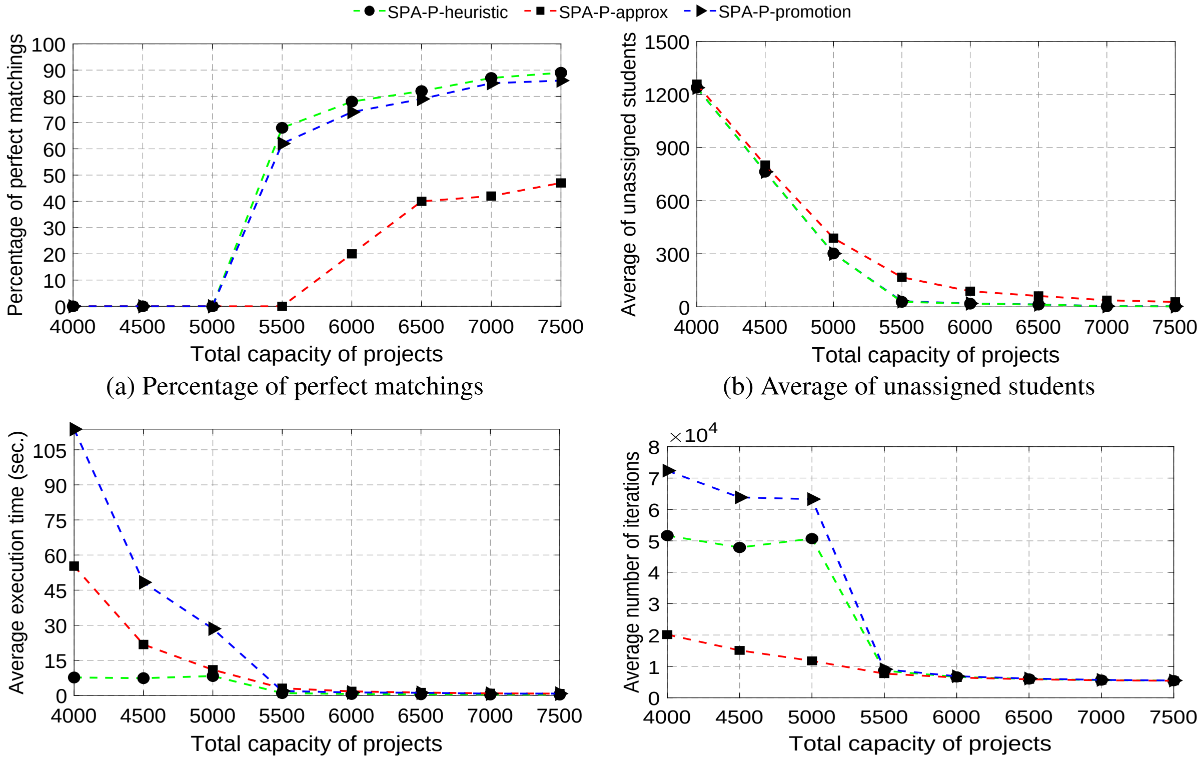
In this experiment, we chose $n = 5000$ and varied the total capacity σ of projects offered by all lecturers from $0.8n$ to $1.5n$ with steps $0.1n$, i.e., σ varied from 4000 to 7500 with steps 500. For each combination of parameter values n and σ, we generated 100 instances of SPA-P, in which other parameters were set as follows: (*i*) m and q were random integer numbers constrained by $0.02 \leq m \leq 0.1n$ and $0.1n \leq q \leq 0.4n$, i.e., $100 \leq m \leq 500$ and $500 \leq q \leq 2000$; (*ii*) σ was distributed randomly to the capacity c_j of each project $p_j \in P$ such that $\sum_{j=1}^{q} c_j = \sigma$ and $1 \leq c_j \leq 120$; and (*iii*) d_k of each lecturer $l_k \in L$ was a random integer number such that $0.8\rho_k \leq d_k \leq 1.2\rho_k$, where ρ_k is the total capacity of projects offered by l_k. As mentioned in Experiment 1, the constraints of m and q mean that the student-to-lecturer ratio was chosen from 10 to 50 and each lecturer offered from 1 to 20 projects.

Figure 3(a) shows the percentage of perfect matchings found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. We see that when the total capacity $\sigma \in \{4000, 4500\}$, all these three algorithms cannot find any perfect matching since we have $\sigma < n$, i.e., the total capacity σ of projects is not enough slots for n students. However, when $\sigma = 5000$, i.e., each project has only a slot for each student, all these three algorithms cannot find any perfect matching. When σ increases, the percentage of perfect matchings found by these algorithms increases since the capacity of projects and lecturers increases. However, SPA-P-heuristic finds a higher percentage of perfect matchings than SPA-P-approx and SPA-P-promotion.

Figure 3(b) shows the average of unassigned students found by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When σ increases, the average of unassigned students found by these algorithms decreases, meaning that the sizes of stable matchings increase. Moreover, we see that SPA-P-heuristic finds stable matchings whose sizes approximate those of SPA-P-promotion (i.e., the green line overlaps the blue line) but are larger than those of SPA-P-approx.

Figure 3(c) shows the average execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. When σ increases, the average execution time found by these algorithms decreases since the capacity of projects and lecturers increases, making these algorithms find stable matchings easier. When σ increases from 4000 to 7500, the average execution time of SPA-P-heuristic decreases from 7.66 seconds to 0.36 seconds, the average execution time of SPA-P-approx decreases from 53.28 seconds to 0.75 seconds, and the average execution time of SPA-P-promotion decreases from 113.82 seconds to 0.78 seconds. When $\sigma = 4000$, SPA-P-heuristic runs about 15 times faster than SPA-P-promotion and about 7 times faster than SPA-P-approx. When $\sigma \geq 5500$, SPA-P-heuristic runs about 2 times faster than SPA-P-promotion and SPA-P-approx. In particular, when σ decreases from 5000 to 4000, the execution time of SPA-P-approx and SPA-P-promotion significantly increases, while that of SPA-P-heuristic almost remains unchanged.

Figure 3(d) shows the average number of iterations used by SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms. As in Experiments 1 and 2, we see that SPA-P-heuristic used a smaller number of iterations than SPA-P-promotion, but a larger number of iterations than SPA-P-approx.



(a) Percentage of perfect matchings (b) Average of unassigned students

(c) Average execution time (d) Average number of iterations

Fig. 3. Comparing solution quality and execution time of SPA-P-heuristic, SPA-P-approx, and SPA-P-promotion algorithms

4.4. Remarks

In summary, we see from the three experiments above that SPA-P-heuristic outperforms SPA-P-approx and SPA-P-promotion in solution quality and execution time. This can be explained as follows:

1. In SPA-P-approx, there are two main reasons to show that this algorithm performed poorly in finding maximum matchings for SPA-P instances. Firstly, when an unassigned student s_i with a non-empty list chooses the first project p_j in her/his list, if p_j is full, then p_j is not assigned to s_i. However, if s_i has only a project p_j in her/his list and if the algorithm does not assign p_j to s_i, then s_i is single. Since $M(p_j)$ is a set of students assigned to p_j, if the algorithm removes some student from $M(p_j)$ and assigns p_j to s_i, then s_i is not single. Secondly, when a student s_i is assigned to a project p_j in her/his list, meaning that s_i is assigned to a lecturer l_k who offered p_j. If l_k is over-subscribed, the algorithm removes an arbitrary student s_r from $M(p_z)$, where p_z is l_k's worst non-empty project, and deletes p_z in s_r's list. If s_r remains only a project p_z in her/his list, then s_r becomes single. Since $M(p_z)$ is a set of students assigned to p_z and in this case, the algorithm should remove another student from $M(p_z)$ rather than s_r. Moreover, removing an arbitrary student s_r from $M(p_z)$ makes the algorithm find a stable matching difficult, leading to inefficient execution time.
2. In SPA-P-approx-promotion, If a project p_j is full, the algorithm removes an arbitrary student s_r from $M(p_j)$ and adds (s_i, p_j) to M. If a lecturer l_k is over-subscribed, the algorithm removes an arbitrary student s_r from $M(p_z)$, where l_k is the lecturer who offered p_j and p_z is l_k's worst non-empty project in $M(l_k)$. Similar to SPA-P-approx, removing an arbitrary student s_r in $M(p_j)$ or $M(p_z)$ is a weak point of SPA-P-approx-promotion. Moreover, when a student s_i with a non-empty list is unpromoted, she/he is allowed to recover her/his original list once again to find a project again in her/his list. This makes the algorithm inefficient in execution time.
3. In SPA-P-heuristic, our heuristic functions $f(x)$ and $g(x)$ given in Eqs. (1) and (3) are used to keep the students in M who have the least opportunity to be reassigned to projects in their lists and remove the

students in M who have the most opportunity to be reassigned to projects in their lists. Therefore, our SPA-P-heuristic solves the weaknesses of SPA-P-approx and SPA-P-promotion algorithms.

Finally, the scenarios of our experiments, where $n = 5000$, m ranges from 100 to 500, and q ranges from 500 to 2000, show that our SPA-P-heuristic results in maximum stable matchings in approximately 1.0 to 7.0 seconds. This underscores the remarkable efficiency of SPA-P-heuristic for dealing with large SPA-P instances.

5. Conclusions

In this paper, we propose a SPA-P-heuristic algorithm to find maximum stable matchings of SPA-P instances. At the beginning, our algorithm initializes a matching to be empty and sets all the students to be active. At each iteration, our algorithm finds an active student with a non-empty list. If such a student exists, our algorithm assigns to her/him the most preferred project in her/his list to form a student-project pair in the matching. If the assigned project overcomes its capacity, our algorithm uses a heuristic function to remove the worst student among students assigned to the project in the matching. If the lecturer who offered the project overcomes her/his capacity, our algorithm uses another heuristic function to remove the worst student among students assigned to the lecturer in the matching. When a student is assigned to a project, she/he becomes inactive. When a student removes a project assigned to her/him, she/he deletes the project from her/his list and becomes active again. Our algorithm repeats until all the students are inactive. We show that our algorithm returns a stable matching after a finite number of iterations. Our experimental results over all the tested scenarios show that our SPA-P-heuristic algorithm outperforms SPA-P-approx and SPA-P-promotion algorithms regarding solution quality and execution time for SPA-P instances of large sizes.

The SPA-P problem consists of variants such as the Student-Project Allocation with preferences over Projects with Ties (SPA-PT), the Student-Project Allocation problem with lecturer preferences over Students (SPA-S) [2], or the Student-Project Allocation problem with lecturer preferences over Students with Ties (SPA-ST) [4,14]. Therefore, our approach can be extended by defining suitable heuristic functions to solve these problems efficiently.

References

[1] David J. Abraham, Robert W. Irving, and David F. Manlove. The student-project allocation problem. In *Proceedings of the 14th International Symposium on Algorithms and Computation*, pages 474–484, Kyoto, Japan, Dec. 15–17 2003.

[2] David J. Abraham, Robert W. Irving, and David F. Manlove. Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5(1):73–90, 2007.

[3] Marco Chiarandini, Rolf Fagerberg, and Stefano Gualandi. Handling preferences in student-project allocation. *Annals of Operations Research*, 257(1):39–78, 2019.

[4] Frances Cooper and David Manlove. A 3/2-approximation algorithm for the student-project allocation problem. In *Proceedings of the 17th International Symposium on Experimental Algorithms*, pages 8:1–8:13, L'Aquila,Italy, Jun. 2018.

[5] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 9(1):9–15, 1962.

[6] Kazuo Iwama, Shuichi Miyazaki, and Hiroki Yanagisawa. Improved approximation bounds for the student-project allocation problem with preferences over projects. *Journal of Discrete Algorithms*, 13(1):59–66, 2012.

[7] Dimitar Kazakov. *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.

[8] Zoltán Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011.

[9] Augustine Kwanashie, Robert W. Irving, David F. Manlove, and Colin T. S. Sng. Profile-based optimal matchings in the student/project allocation problem. In *Proceedings of the 25th International Workshop on Combinatorial Algorithms*, pages 213–225, Duluth, USA, Oct. 15–17 2014.

[10] David Manlove, Duncan Milne, and Sofiat Olaosebikan. An integer programming approach to the student-project allocation problem with preferences over projects. In *Proceedings of the 5th International Symposium on Combinatorial Optimization*, pages 313–325, Morocco, Apr. 2018.

[11] David Manlove, Duncan Milne, and Sofiat Olaosebikan. Student-project allocation with preferences over projects: Algorithmic and experimental results. *Discrete Applied Mathematics*, 308(1):220–234, 2022.

[12] David F. Manlove and Gregg O'Malley. Student-project allocation with preferences over projects. *Journal of Discrete Algorithms*, 6(4):553–560, 2008.

[13] Steven Minton, Mark D.Johnston, Andrew B.Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[14] Sofiat Olaosebikan and David Manlove. Super-stability in the student-project allocation problem with ties. *Journal of Combinatorial Optimization*, 43(1):1203–1239, 2022.

[15] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[16] Hoang Huu Viet, Le Van Tan, and Son Thanh Cao. Finding maximum stable matchings for the student-project allocation problem with preferences over projects. In *Proceedings of the 7th International Conference on Future Data and Security Engineering*, pages 411–422, Quy Nhon, Vietnam, Nov. 2020.