

An Empirical Heuristic Algorithm for Solving the Student-Project Allocation Problem with Ties

Hoang Huu Bach¹, Nguyen Thi Thuong² and Son Thanh Cao² ^a

¹Faculty of Information Technology, University of Engineering and Technology (UET), Vietnam National University, Hanoi
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

²Faculty of Information Technology, School of Engineering and Technology, Vinh University
182 Le Duan, Vinh city, Nghe An, Vietnam
19020093@vnu.edu.vn, {205748020110145, sonct}@vinhuni.edu.vn

Keywords: Student-Project Allocation Problem, Matching Problem, Heuristic Search.

Abstract: In this paper, we propose a simple heuristic algorithm to deal with the Student-Project Allocation problem with lecturer preferences over Projects with Ties (SPA-PT). The aim of such a problem is to find a maximum stable matching of students and projects to meet the constraints on students' and lecturers' preferences over projects, and the maximum numbers of students given by lecturers for each lecturer and her/his projects. Our algorithm starts from an empty matching and iteratively constructs a maximum stable matching of students and projects. For each iteration, our algorithm chooses the most ranked project of an unassigned student to assign for her/him. If the assigned project or the lecturer who offers the assigned project is over-subscribed, our algorithm removes a worth student assigned the project, where a worth student is a person corresponding to the maximum value of the proposed heuristic function. Experimental results illustrate the outperformance of the proposed algorithm w.r.t. the execution time and solution quality for solving the problem.


1 INTRODUCTION

The problem of finding a suitable project together with a desired lecturer has been of great interest to students at universities. Many university departments assign a student to a random lecturer for doing the lecturer's project without any common interest between them. To deal with this problem, there are several approaches have been proposed over the years. In 2003, Abraham et. al. studied the problem of allocating students to projects called Student-Project Allocation problem (SPA) (Abraham et al., 2003), which is a generalization of the classical Hospitals-Residents problem (HR) (Gale and Shapley, 1962). In SPA, a lecturer will normally (i) offer a non-empty list of projects, and (ii) propose a preference list over students that he/she wants to supervise in a strict order, whilst each student proposes a strictly ordered preference list over available projects. There is no overlap of projects among lecturers. Moreover, there are also limitations on the number of students assigned to a given project as well as a given lecturer (i.e., capacity constrains). Solving this problem means to find a *matching*, which is an assignment between students

and projects w.r.t. given preference lists so that each student can only be assigned to at most one project without violating the capacity constraints for projects and lecturers. A property for a matching to satisfy stability (Roth, 1984). Informally, a matching is stable (i.e., stable matching) if there are no student and lecturer that would both prefer to be matched with each other instead of their current partners in the matching.

In 2007, Abraham et. al. (Abraham et al., 2007) presented two optimal linear-time algorithms for finding a stable matching of a given instance of SPA. The first algorithm is *student-oriented*, which finds the best-possible stable matching for all students. Besides, the second algorithm is *lecturer-oriented*, which finds the best-possible stable matching for all lecturers.

In 2008, Manlove et. al. (Manlove and O'Malley, 2008) considered a variant of SPA, referred to SPA-P (SPA with preferences over Projects), in which both students and lecturers have preference lists over projects in a strict order. The authors also showed that stable matchings for an SPA-P instance can have different sizes, and the problem of finding a maximum cardinality stable matching, referred to MAX-SPA-P, is NP-hard (even in the special case that each

^a  <https://orcid.org/0000-0002-4771-7856>

project and lecturer can accommodate only one student). Therefore, to solve the MAX-SPA-P problem effectively, researchers often focus on using the approximation solutions. In (Manlove and O'Malley, 2008), authors presented a polynomial-time 2-approximation algorithm for MAX-SPA-P, named SPA-P-approx. In 2012 (Iwama et al., 2012), Iwama et. al. proposed an $\frac{3}{2}$ -approximation algorithm, named SPA-P-approximation, which is a combination of ideas from Manlove's algorithm (Manlove and O'Malley, 2008) and the approximation algorithm of Király (Király, 2011) for MAX-SPA-P. In 2018 (Manlove et al., 2018), Manlove et. al. proposed $\frac{3}{2}$ -approximation algorithm to find stable matchings that are very close to having maximum cardinality for MAX-SPA-P by using Integer Programming model. Recently, Viet et. al. proposed a local search strategy based on the min-conflicts heuristic, named SPA-P-MCH algorithm, to find a maximum weakly stable matching for the SPA-P problem (Viet et al., 2020).

A variant of SPA that involves lecturer preferences over students, which is known as SPA-S (the SPA with lecturer preferences over Students). SPA-S requires ranking in strict order of preferences. However, this might not fair in practical applications. For example, a lecturer may be willing to rank two or more students equally in a tie. In 2022 (Olaosebikan and Manlove, 2022), Olaosebikan et. al. studied a variant of SPA-S with Ties, referred to SPA-ST. The authors also proposed a polynomial-time algorithm to find a supper-stable matching for an instance of SPA-ST or to report that no such matching exists.

As far as we are concerned, there is no research related to the problem of SPA-P with Ties. It is worth doing further research on the topic because this problem has many practical applications, especially for the cases with ties. In this work, we propose a simple heuristic algorithm to find a maximum stable matching for an instance of SPA-P with Ties, denoted by SPA-PT. The aim is to find a maximum stable matching of students and projects with the following constraints: (i) the ranking in a strict order of students' and lecturers' preferences list over projects, and (ii) the limitations on the number of students assigned to a given project and a given lecturer.

The rest of this paper is structured as follows. Section 2 reminds the SPA-PT problem. Section 3 describes our proposed algorithm. Section 4 presents our experimental results, and concluding remarks are given in Section 5.

2 SPA-P WITH TIES

In this section, we present the formal notions and definitions related to the *Student-Project Allocation problem with preferences over Project with Ties*, denoted by SPA-PT. For convenience, the notions used for SPA-PT are similar to the ones given in (Manlove and O'Malley, 2008). An instance of SPA-PT consists of the following sets: $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ of *students*, $\mathcal{P} = \{p_1, p_2, \dots, p_q\}$ of *projects*, and $\mathcal{L} = \{l_1, l_2, \dots, l_m\}$ of *lecturers*. Each lecturer l_k provides a non-empty set P_k of projects and ranks these projects in a preference list (with ties allowed). We assume that, there is no overlap of projects among lecturers (i.e., P_k partitions \mathcal{P} , where $k = 1, 2, \dots, m$). Also, each student $s_i \in \mathcal{S}$ suggests a set of projects $A_i \subseteq \mathcal{P}$ and ranks A_i in a preference list (with ties allowed). Moreover, each $l_k \in \mathcal{L}$ (resp. $p_j \in \mathcal{P}$) has an associated *capacity* to specify the maximum number of students that l_k can supervise (resp. that can be assigned to p_j), denoted by $d_k \in \mathbb{Z}^+$ (resp. $c_j \in \mathbb{Z}^+$).

We denote $A = \{(s_i, p_j) \in \mathcal{S} \times \mathcal{P}\}$ by a set of *acceptable* pairs, where p_j is given by l_k , s_i (resp. l_k) ranks p_j in the s_i 's (resp. l_k 's) preference list, denoted by $rank(s_i, p_j)$ (resp. $rank(l_k, p_j)$). If a student s_i strictly (resp. equally) prefers a project p_j to (resp. and) a project p_t , then we denote by $rank(s_i, p_j) < rank(s_i, p_t)$ (resp. $rank(s_i, p_j) = rank(s_i, p_t)$). We use notations for $rank(l_k, p_j)$ similarly.

An *assignment* M is a subset of A . If $(s_i, p_j) \in M$, then we say that s_i is *assigned* to p_j in M and vice versa. If s_i is assigned to p_j in M , then we denote $M(s_i) = p_j$, otherwise, we denote $M(s_i) = \emptyset$ (i.e., unassigned). Consequently, if l_k is the lecturer offering p_j and s_i is assigned to p_j , then we can also say that s_i is assigned to l_k and vice versa. Similarly, for any project $p_j \in \mathcal{P}$ (resp. lecturer $l_k \in \mathcal{L}$), we denote $M(p_j)$ (resp. $M(l_k)$) by the set of students assigned to p_j (resp. l_k) in M . We also say that, a project $p_j \in \mathcal{P}$ (resp. lecturer $l_k \in \mathcal{L}$) is *under-subscribed*, *full* or *over-subscribed* in M if $|M(p_j)| < c_j$, $|M(p_j)| = c_j$, or $|M(p_j)| > c_j$ (resp. $|M(l_k)| < d_k$, $|M(l_k)| = d_k$, or $|M(l_k)| > d_k$), respectively.

Definition 1 (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 $|M(l_k)| \leq d_l$ for each $l_k \in \mathcal{L}$ (i.e., each student is assigned to at most one project, and no project or lecturer is *over-subscribed* in M). \triangleleft

Definition 2 (Blocking Pair). An acceptable pair $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ is a *blocking pair* of a matching M , if all the following conditions are fulfilled:

1. $p_j \in A_i$ (i.e., s_i find p_j acceptable);
2. Either $M(s_i) = \emptyset$ or s_i prefers p_j to $M(s_i)$;

3. $|M(p_j)| < c_j$ 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 $|M(l_k)| < d_k$, or
 - (c) $s_i \notin M(l_k)$, $|M(l_k)| = d_k$, and l_k prefers p_j to l_k 's worst non-empty project, where l_k is the lecturer offering p_j . \triangleleft

We say that a matching M is *stable* if it admits no blocking pair, otherwise, it is *unstable*. Given a stable matching M , we denote by $|M|$ the number of students assigned in M . We also say that M is a *perfect matching* if $|M| = n$, otherwise, it is a *non-perfect matching*.

Example 1. To make these definitions clearer, in this example, we consider an instance of SPA-PT, which involves a set of six students $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, a set of two lecturers $\mathcal{L} = \{l_1, l_2\}$ with $P_1 = \{p_1, p_2, p_3, p_4\}$ and $P_2 = \{p_5, p_6\}$, and a set of six projects $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. Further details are described in Table 1, where ties in students' and lecturers' preference lists are presented in round brackets. In the lecturers' preference list, let's consider the first row for the lecturer l_1 , the meaning of $l_1: p_3 \ p_4 \ (p_1 \ p_2)$ is that, l_1 strictly prefers p_3 to p_4 , and p_4 to $(p_1$ and $p_2)$, also in this case, l_1 prefers p_1 and p_2 equally. In particular, $rank(l_1, p_3) = 1$, $rank(l_1, p_4) = 2$, and $rank(l_1, p_1) = rank(l_1, p_2) = 3$. We use the same notations in the students' preference list. Rank lists corresponding to the students' and lecturers' preference lists presented on the left-hand side of Table 1 are shown on the right-hand side of that table. From now on, we use the term students' list (resp. lecturers' list) instead of students' (resp. lecturers') rank list, for short. We say that, s_i 's list is *empty* if $rank(s_i, p_j) = 0$ for all $p_j \in \mathcal{P}$.

The matching $M = \{(s_1, p_2), (s_2, p_6), (s_3, p_5), (s_4, \emptyset), (s_5, p_5), (s_6, p_4)\}$ is unstable since there exists some blocking pairs $\{(s_2, p_2), (s_2, p_3), (s_4, p_2), (s_4, p_3)\}$ for M . The matching $M = \{(s_1, \emptyset), (s_2, p_6), (s_3, p_3), (s_4, p_2), (s_5, p_6), (s_6, p_4)\}$ is stable because there exists no blocking pair for M , and M is a non-perfect matching since $|M| = 5$. The matching $M = \{(s_1, p_3), (s_2, p_6), (s_3, p_5), (s_4, p_2), (s_5, p_4), (s_6, p_5)\}$ is stable since there exists no blocking pair for M . Additionally, in this case, M is a perfect matching since $|M| = 6$. \triangleleft

3 PROPOSED ALGORITHM FOR SPA-PT

In this section, we propose an algorithm for finding maximum stable matching of SPA-PT instances using a heuristic strategy, named SPA-PT-heuristic, which is described in Algorithm 1. For initialization, we set

Algorithm 1: SPA-PT-heuristic Algorithm.

Input : An instance of SPA-PT with

$$\begin{aligned} \mathcal{S} &= \{s_1, s_2, \dots, s_n\}; \\ \mathcal{P} &= \{p_1, p_2, \dots, p_q\}; \\ \mathcal{L} &= \{l_1, l_2, \dots, l_m\}; \\ M &:= \emptyset; \\ active(s_i) &:= 1, \forall s_i \in \mathcal{S}; \\ h(l_k, s_i) &:= 0, \forall l_k \in \mathcal{L} \text{ and } \forall s_i \in \mathcal{S}; \end{aligned}$$

Output: A maximum-size stable matching M for SPA-PT

```

1 while  $\exists s_i$  such that  $active(s_i) > 0$  do
2   if IsEmpty( $s_i$ 's list) then
3      $active(s_i) := 0$ ;
4     continue;
5   end
6    $A_i^+ := \operatorname{argmin}(rank(s_i, p_t) > 0), \forall p_t \in A_i$ ;
7    $p_j := \operatorname{argmax}(c_t - |M(p_t)|), \forall p_t \in A_i^+$ ;
8    $l_k :=$  lecturer offering  $p_j$ ;
9    $M := M \cup \{(s_i, p_j)\}$ ;
10   $active(s_i) := 0$ ;
11   $y(v) :=$  number of projects of rank  $v$  in  $s_i$ '
    list ( $v = 1, 2, \dots$ );
12   $h(l_k, s_i) := rank(l_k, p_j) + (q * sum(y) +$ 
     $max(y)) / (q * q + q + 1)$ ;
13  if  $p_j$  is over-subscribed then
14     $s_r := \operatorname{argmax}(h(l_k, s_r)), \forall s_r \in M(p_j)$ ;
15     $M := M \setminus \{(s_r, p_j)\}$ ;
16     $rank(s_r, p_j) := 0$ ;
17     $active(s_r) := 1$ ;
18     $h(l_k, s_r) := 0$ ;
19  end
20  if  $l_k$  is over-subscribed then
21     $s_r := \operatorname{argmax}(h(l_k, s_r)), \forall s_r \in M(l_k)$ ;
22     $p_t := M(s_r)$ ;
23     $M := M \setminus \{(s_r, p_t)\}$ ;
24     $rank(s_r, p_t) := 0$ ;
25     $active(s_r) := 1$ ;
26     $h(l_k, s_r) := 0$ ;
27  end
28 end
29 return  $M$ ;
```

$M = \emptyset$; $active(s_i) = 1$ for all $s_i \in \mathcal{S}$ with the meaning that all students are active for the first time; and an array $h(l_k, s_i) = 0$, for all $l_k \in \mathcal{L}$ ($1 \leq k \leq m$) and $s_i \in \mathcal{S}$ ($1 \leq i \leq n$), to store a heuristic value for the pair (l_k, s_i) at each iteration.

When there exists an active student s_i , if there is no more project in s_i 's list, then s_i becomes inactive forever (i.e., $active(s_i) = 0$, line 3), and therefore in this case s_i is unassigned in M . Otherwise, s_i is assigned to the project p_j with the minimum value of rank, and

Table 1: An instance of SPA-PT.

| Students' preference list | Lecturers' preference list | Students' rank list | Lecturers' rank list |
|---|--------------------------------|---|------------------------------|
| $s_1: (p_1 \ p_2) \ (p_3 \ p_5)$ | $l_1: p_3 \ p_4 \ (p_1 \ p_2)$ | $s_1: 1 \ 1 \ 2 \ 0 \ 2 \ 0$ | $l_1: 3 \ 3 \ 1 \ 2 \ 0 \ 0$ |
| $s_2: p_2 \ p_3 \ (p_5 \ p_6)$ | $l_2: p_6 \ p_5$ | $s_2: 0 \ 1 \ 2 \ 0 \ 3 \ 3$ | $l_2: 0 \ 0 \ 0 \ 0 \ 2 \ 1$ |
| $s_3: (p_1 \ p_4 \ p_5) \ p_2 \ p_3$ | | $s_3: 1 \ 2 \ 3 \ 1 \ 1 \ 0$ | |
| $s_4: (p_2 \ p_3) \ p_5$ | | $s_4: 0 \ 1 \ 1 \ 0 \ 2 \ 0$ | |
| $s_5: (p_4 \ p_5) \ p_1 \ (p_2 \ p_6)$ | | $s_5: 2 \ 3 \ 0 \ 1 \ 1 \ 3$ | |
| $s_6: p_5 \ p_4$ | | $s_6: 0 \ 0 \ 0 \ 2 \ 1 \ 0$ | |
| Lecturers' capacities: $d_1 = 3, d_2 = 3$; | | Projects' capacities: $c_1 = c_3 = c_4 = 1, c_2 = c_5 = 2, c_6 = 3$ | |

the maximum remaining capacity on the s_i 's list (i.e., the pair (s_i, p_j) is added to M , lines 6-9). In this case, s_i becomes inactive (line 10), and the heuristic value $h(l_k, s_i)$ is updated based on the following heuristic function, where y is an array and each $y(v)$ stores the number of project of rank v in s_i 's list (i.e., the frequency of rank v in s_i 's list with $v = 1, 2, \dots$):

$$h(l_k, s_i) := \text{rank}(l_k, p_j) + (q * \text{sum}(y) + \max(y)) / (q * q + q + 1).$$

The intuition behind the heuristic function $h(l_k, s_i)$ is that, the student s_i , who has the most opportunities to choose his/her projects based on s_i 's list, will be selected to remove from M for the cases either p_j is *over-subscribed*, or l_k is *over-subscribed*. Particularly, instead of choosing an arbitrary assigned student s_i to remove, the heuristic function is used to eliminate the student with the most opportunity and keep the students in M who have the least opportunity to be reassigned to some project in their lists at the next iterations.

In the case p_j is *over-subscribed*, the student $s_r \in M(p_j)$ corresponding to the maximum value of $h(l_k, s_r)$ is removed from M (lines 14-15), where l_k is the lecturer offering p_j . Also in this case, p_j is deleted from s_r 's list (line 16) and s_r is now active again (line 17).

Similarly, in the case l_k is *over-subscribed*, the student $s_r \in M(l_k)$ corresponding to the maximum value of $h(l_k, s_r)$ is removed from M (lines 21-23). Also in this case, p_t is deleted from s_r 's list (line 24) and s_r is now active again (line 25), where p_t is the project assigned to s_r in the earlier step. We set the heuristic value $h(l_k, s_r)$ to zero when a student s_r is removed from M (lines 18 and 26). If there is no student in the active state, the algorithm terminates and returns a maximum stable matching in M .

Example 2. This example illustrates the running of Algorithm 1 by using the SPA-PT instance given in Tables 1. Algorithm 1 starts with $M = \emptyset$, $\text{active}(s_i) = 1$, $\forall s_i \in \mathcal{S}$ (i.e., all students are assigned to active), and $h(l_k, s_i) = 0$, $\forall l_k \in \mathcal{L}$ and $\forall s_i \in \mathcal{S}$. The i^{th} iteration of running Algorithm 1 is performed in detail as follows:

- (1) s_1 is assigned to p_2 since p_2 has the minimum value of rank with the maximum remaining capacity on the s_1 's list (there are two projects p_1 and p_2 with $\text{rank}(s_1, p_1) = \text{rank}(s_1, p_2) = 1$, and the current capacity for p_1 is 1, whilst p_2 is 2). Consequently, $M = \{(s_1, p_2)\}$ and s_1 is now inactive (i.e., $\text{active}(s_1) = 0$).
- (2) s_2 is assigned to p_2 since p_2 is the unique minimum value of rank on the s_2 's list and p_2 is *under-subscribed*. As the result, $M = \{(s_1, p_2), (s_2, p_2)\}$ and s_2 is now inactive.
- (3) s_3 is assigned to p_5 since projects p_1 , p_4 and p_5 have the minimum value of rank on the s_3 's list, but p_5 has the maximum current capacity. Consequently, $M = \{(s_1, p_2), (s_2, p_2), (s_3, p_5)\}$ and s_3 becomes inactive.
- (4) Similarly, s_4 is assigned to his/her most preferred project p_3 and becomes inactive. Consequently, $M = \{(s_1, p_2), (s_2, p_2), (s_3, p_5), (s_4, p_3)\}$.
- (5) s_5 is assigned to his/her most preferred project p_4 and becomes inactive. However, in this case l_1 is *over-subscribed* and $h(l_1, s_1)$ is maximum, therefore the pair (s_1, p_2) is eliminated from M , s_1 is now active again. As the result, $M = \{(s_2, p_2), (s_3, p_5), (s_4, p_3), (s_5, p_4)\}$.
- (6) Similarly, s_6 is assigned to p_5 and becomes inactive. Consequently, $M = \{(s_2, p_2), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$. After this step, we have only s_1 in active state.
- (7) s_1 is assigned to p_1 and becomes inactive since p_1 is the unique minimum value of rank on the s_1 's list. However, in this case l_1 is *over-subscribed* and $h(l_1, s_2)$ is maximum, therefore the pair (s_2, p_2) is eliminated from M , s_2 is now active again. As the result, $M = \{(s_1, p_1), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$.
- (8) s_2 is assigned to his/her most preferred project p_3 and becomes inactive. However, in this case p_3 is *over-subscribed* and the pair $(s_2, p_3) \in M$ has the maximum value of $h(l_1, s_2)$, therefore the pair (s_2, p_3) is eliminated from M , and s_2 is now ac-

Table 2: The step-by-step of running Algorithm 1 for the instance given in Table 1.

| # | s_i | p_j | l_k | Matching M ($M = M \cup (s_i, p_j)$) | $M \setminus (s_r, p_t)$ |
|---|-------|-------|-------|--|--------------------------|
| 1 | s_1 | p_2 | l_1 | $\{(s_1, p_2)\}$ | |
| 2 | s_2 | p_2 | l_1 | $\{(s_1, p_2), (s_2, p_2)\}$ | |
| 3 | s_3 | p_5 | l_2 | $\{(s_1, p_2), (s_2, p_2), (s_3, p_5)\}$ | |
| 4 | s_4 | p_3 | l_1 | $\{(s_1, p_2), (s_2, p_2), (s_3, p_5), (s_4, p_3)\}$ | |
| 5 | s_5 | p_4 | l_1 | $\{(s_2, p_2), (s_3, p_5), (s_4, p_3), (s_5, p_4)\}$ | $\{(s_1, p_2)\}$ |
| 6 | s_6 | p_5 | l_2 | $\{(s_2, p_2), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ | |
| 7 | s_1 | p_1 | l_1 | $\{(s_1, p_1), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ | $\{(s_2, p_2)\}$ |
| 8 | s_2 | p_3 | l_1 | $\{(s_1, p_1), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$ | $\{(s_2, p_3)\}$ |
| 9 | s_2 | p_6 | l_2 | $\{(s_1, p_3), (s_2, p_6), (s_3, p_5), (s_4, p_2), (s_5, p_4), (s_6, p_5)\}$ | |

tive again. As the result, $M = \{(s_1, p_1), (s_3, p_5), (s_4, p_3), (s_5, p_4), (s_6, p_5)\}$.

- (9) Finally, s_2 is assigned to his/her most preferred project p_6 and becomes inactive. After this step, all students are inactive, the algorithm terminates and returns a stable matching $M = \{(s_1, p_3), (s_2, p_6), (s_3, p_5), (s_4, p_2), (s_5, p_4), (s_6, p_5)\}$. \triangleleft

The running steps of Algorithm 1 in Example 2 are summarized in Table 2.

4 EXPERIMENTS

In this section, we present several experiments to evaluate the performance of our SPA-PT-heuristic algorithm. We implemented our algorithm and the compared 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 run experiments, we extend the random problem generator algorithm for SMTI problem (Gent and Prosser, 2002) to generate random SPA-PT instances. Specifically, we generated random SPA-PT instances with four parameters (n, m, q, τ) , where n is the number of students, m is the number of lecturers, and q is the number of projects, and τ is the probability of ties in both preference lists of students and lecturers.

4.1 Experiments for SPA-P

In this section, we chose the well-known SPA-P-approx (Manlove and O'Malley, 2008) and SPA-P-approx-promotion (Iwama et al., 2012) (for short, we call it SPA-P-promotion) algorithms to compare their solution quality and execution time with those of SPA-

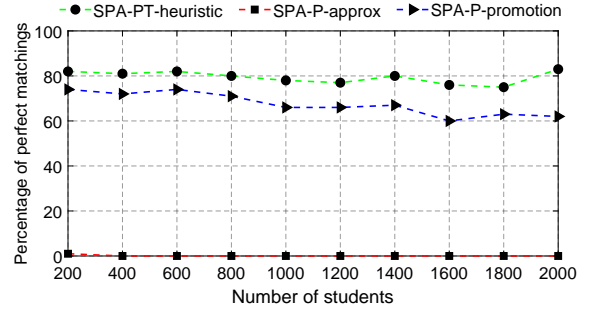


Figure 1: Percentage of perfect matchings found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms for the Experiment 1.

PT-heuristic algorithm. Since SPA-P-approx and SPA-P-promotion algorithm are used for SPA-P, so we chose $\tau = 0.0$, meaning that SPA-PT becomes SPA-P.

Experiment 1. In this experiment, we chose $n \in \{200, 400, \dots, 2000\}$. For each value of n , we generated 100 instances of SPA-PT of parameters (n, m, q) , where m and q are random integer numbers bound by $0.02n \leq m \leq 0.1n$ and $0.1n \leq q \leq 0.5n$, respectively. This means that the student-to-lecturer ratio is from 10 to 50, the student-to-project ratio is from 2 to 10, and each lecturer offers from 1 to 25 projects randomly. In each instance, we let each student randomly rank from 1 to 20 (i.e., $|A_i| \in [1, 20]$) projects in the set \mathcal{P} of projects offered by all lecturers. Besides, we generate the capacity c_j of each project $p_j \in \mathcal{P}$ such that $2 \leq c_j \leq 11$ and $\sum_{j=1}^q c_j = n$. Finally, we set the capacity d_k of each lecturer l_k to the total capacity of projects offered by l_k , i.e. $d_k = \sum c_j$, where c_j is the capacity of project $p_j \in \mathcal{P}_k$. These conditions mean that the total capacity of the projects proposed by all lecturers is the same as that with the total number of students (i.e., the total capacity of the projects is just enough to assign to the given number of students n).

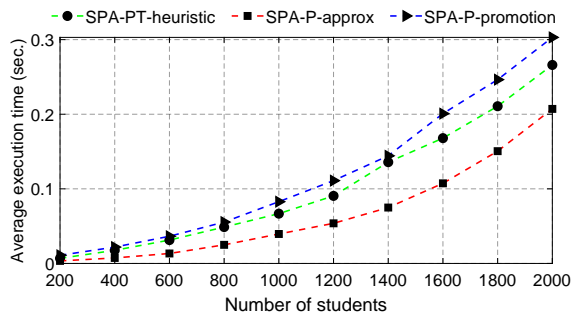


Figure 2: Execution time of SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms for the Experiment 1.

Figure 1 shows the percentage of perfect matchings found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms. SPA-PT-heuristic finds from 75% to 83% of perfect matchings, SPA-P-approx finds from 60% to 74% of perfect matchings, while SPA-P-promotion finds only 1% of perfect matchings at $n = 200$.

Table 3 shows the average of unassigned students found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms. For every n from 200 to 2000, SPA-PT-heuristic finds a smaller number of unassigned students in non-perfect matchings than SPA-P-promotion and SPA-P-approx do. Especially, when n is larger, the average of unassigned students in non-perfect matchings found by SPA-PT-heuristic is much smaller than that found by SPA-P-promotion and SPA-P-approx. This means that the stable matchings found by SPA-PT-heuristic are larger than those found by SPA-P-promotion and SPA-P-approx in terms of size.

Figure 2 shows the average execution time of found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms. We see that SPA-PT-heuristic runs faster than SPA-P-promotion, but slower than SPA-P-approx. As mentioned above, the problem of finding a maximum cardinality stable matching is NP-hard. By using the approximation solutions, these algorithms run approximately in 3 seconds with $n = 2000$. That is, they are possible for using in practical applications w.r.t. the execution time.

Experiment 2. In this experiment, we chose values of parameters n , m , and q as in Experiment 1. In each instance, we let each student randomly rank from 1 to 10 (i.e., $|A_i| \in [1, 10]$) projects in the set \mathcal{P} of projects offered by all lecturers. Besides, we generate the capacity c_j of each project $p_j \in \mathcal{P}$ such that $2 \leq c_j \leq 11$ and $n \leq \sum_{j=1}^q c_j \leq 1.5n$. Finally, we set the capacity d_k of each lecturer l_k to be a random integer number such that $0.8 \times \sum c_j \leq d_k \leq \sum c_j$, where c_j is the capacity of project $p_j \in \mathcal{P}_k$. This means that each lecturer l_k chooses from 80% to 100% students in the

Table 3: Average of unassigned students found by SPA-PT-heuristic (A1), SPA-P-promotion (A2), and SPA-P-approx (A3) algorithms for the Experiment 1.

| | Number of students n | | | | | | | | | |
|----|------------------------|------|------|-------|-------|-------|-------|-------|-------|-------|
| | 200 | 400 | 600 | 800 | 1000 | 1200 | 1400 | 1600 | 1800 | 2000 |
| A1 | 13.2 | 16.4 | 23.9 | 42.6 | 77.0 | 62.4 | 55.1 | 91.4 | 97.8 | 86.8 |
| A2 | 14.1 | 17.1 | 26.4 | 44.4 | 79.0 | 65.0 | 59.3 | 95.1 | 102.9 | 96.5 |
| A3 | 29.2 | 49.3 | 75.7 | 107.4 | 151.9 | 157.0 | 170.1 | 210.6 | 240.1 | 255.2 |

Table 4: Average of unassigned students found by SPA-PT-heuristic (A1), SPA-P-promotion (A2), and SPA-P-approx (A3) algorithms for the Experiment 2.

| | Number of students n | | | | | | | | | |
|----|------------------------|------|------|------|------|------|-------|-------|-------|-------|
| | 200 | 400 | 600 | 800 | 1000 | 1200 | 1400 | 1600 | 1800 | 2000 |
| A1 | 9.4 | 19.6 | 27.8 | 33.0 | 43.5 | 36.8 | 67.6 | 64.0 | 76.0 | 78.2 |
| A2 | 9.7 | 19.8 | 28.6 | 33.7 | 43.6 | 37.5 | 67.6 | 66.1 | 77.3 | 79.0 |
| A3 | 17.1 | 36.0 | 50.8 | 64.9 | 81.1 | 82.7 | 124.3 | 130.2 | 149.8 | 151.8 |

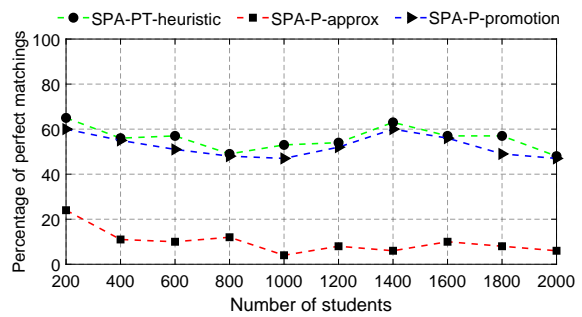


Figure 3: Percentage of perfect matchings found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms for the Experiment 2.

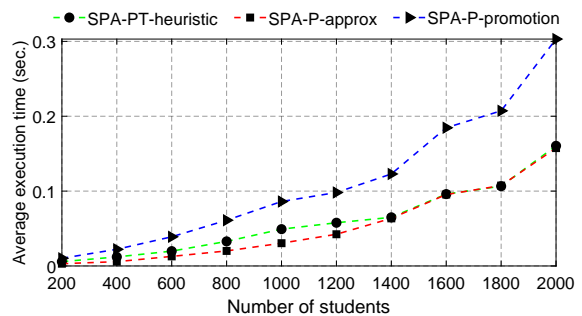


Figure 4: Execution time of SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms for the Experiment 2.

total of capacities of projects offered by her/him.

Figure 3 shows the percentage of perfect matchings found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms. Again, we see that SPA-PT-heuristic finds many perfect matchings than SPA-P-promotion and SPA-P-approx do.

Table 4 shows the average of unassigned students found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-

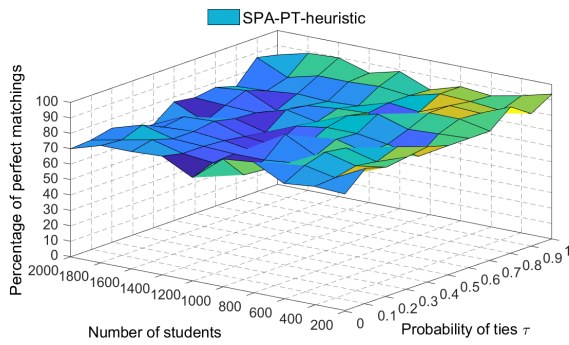


Figure 5: Percentage of perfect matchings found by SPA-PT-heuristic algorithm for the Experiment 3.

approx algorithms. Again, we see that SPA-PT-heuristic finds a smaller number of unassigned students in non-perfect matchings than SPA-P-promotion and SPA-P-approx. This means the stable matchings found by SPA-PT-heuristic is larger than those found by SPA-P-promotion and SPA-P-approx.

Figure 4 shows the average execution time of found by SPA-PT-heuristic, SPA-P-promotion, and SPA-P-approx algorithms. We see that when n is larger, SPA-PT-heuristic runs much faster than SPA-P-promotion, but equally fast as SPA-P-approx.

In brief, in the two above experiments, SPA-PT-heuristic not only outperforms both SPA-P-promotion and SPA-P-approx in terms of solution quality but also the execution time.

4.2 Experiments for SPA-PT

In this section, we present two experiments to consider the behavior of SPA-PT-heuristic for SPA-PT when the probability of ties τ varies from 0.0 to 1.0.

Experiment 3. In this experiment, we let all the settings be the same as in Experiment 1. However, we allow ties to appear in the preference lists of both students and lecturers with the probability of ties $\tau \in \{0.0, 0.1, \dots, 1.0\}$.

Figure 5 shows the percentage of perfect matchings found by SPA-PT-heuristic algorithm. For n from 200 to 2000, SPA-PT-heuristic found more than 70% of perfect matchings. Moreover, as n increases from 200 to 2000 the percentage of perfect matchings found by SPA-PT-heuristic decreases. As τ increases from 0.0 to 1.0 the percentage of perfect matchings found by SPA-PT-heuristic increases.

Figure 6 shows the execution time of SPA-PT-heuristic algorithm. As τ increases from 0.0 to 1.0, the execution time of SPA-PT-heuristic decreases. As n increases from 200 to 2000, the execution time of SPA-PT-heuristic increases. As mentioned earlier, with $n = 2000$, our algorithm runs approximately in 3 sec-

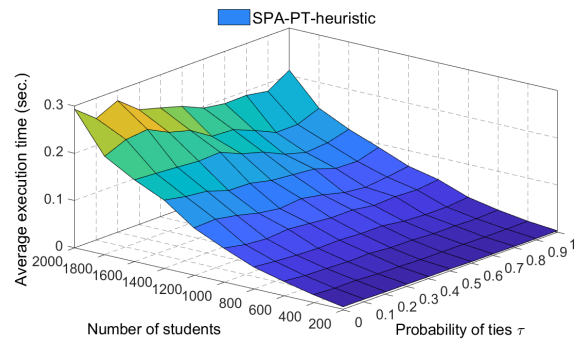


Figure 6: Execution time of SPA-PT-heuristic algorithm for the Experiment 3.

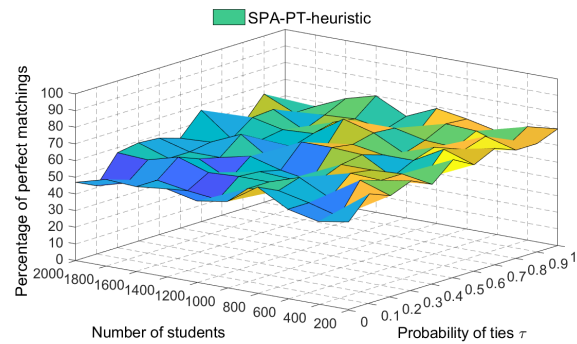


Figure 7: Percentage of perfect matchings found by SPA-PT-heuristic algorithm for the Experiment 4.

onds, thus, it can be applied easily in practical applications.

Experiment 4. In this experiment, we let all the settings be the same as in Experiment 2 and $\tau \in \{0.0, 0.1, \dots, 1.0\}$.

Figure 7 shows the percentage of perfect matchings found by SPA-PT-heuristic algorithm. As τ increases from 0.0 to 1.0 the percentage of perfect matchings found by SPA-PT-heuristic slightly increases. As n increases from 200 to 2000 the percentage of perfect matchings found by SPA-PT-heuristic decreases.

Figure 8 shows the execution time of SPA-PT-heuristic algorithm. As τ increases from 0.0 to 1.0, the execution time of SPA-PT-heuristic slightly increases. As n increases from 200 to 2000, the execution time of SPA-PT-heuristic increases.

5 CONCLUSIONS

In this paper, we proposed a heuristic algorithm to find maximum stable matching of SPA-PT instances. Starting at an empty matching, our algorithm iteratively constructs a maximum stable matching of students and projects. At each iteration, our algorithm

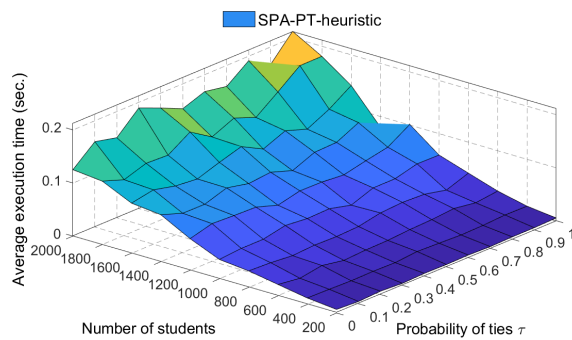


Figure 8: Execution time of SPA-PT-heuristic algorithm for the Experiment 4.

chooses the most ranked project of an unassigned student to assign for her/him. If the assigned project or the lecturer who offers the assigned project is over-subscribed, instead of choosing an arbitrary assigned student to remove, our algorithm removes the student corresponding to the maximum value of the proposed heuristic function (i.e., we want to keep the students who have the least opportunity to be re-assigned to some project in their lists at the next iterations). Experiment results showed that our algorithm not only outperforms the SPA-P-approx and SPA-P-approx-promotion algorithms for the SPA-P problem w.r.t. the execution time and solution quality, but also efficient for the SPA-PT problem.

REFERENCES

- Abraham, D. J., Irving, R. W., and Manlove, D. F. (2003). The Student-Project Allocation Problem. In *Proceeding of ISAAC'2003, Lecture Notes in Computer Science*, page 474–484.
- Abraham, D. J., Irving, R. W., and Manlove, D. F. (2007). Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5(1):73–90.
- Gale, D. and Shapley, L. S. (1962). College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15.
- Gent, I. P. and Prosser, P. (2002). An empirical study of the stable marriage problem with ties and incomplete lists. In *in Proceedings of the 15th European Conference on Artificial Intelligence*, pages 141–145, Lyon, France.
- Iwama, K., Miyazaki, S., and Yanagisawa, H. (2012). Improved approximation bounds for the student-project allocation problem with preferences over projects. *Journal of Discrete Algorithms*, 13(1):59–66.
- Király, Z. (2011). Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60:3–20.
- Manlove, D., Milne, D., and Olaosebikan, S. (2018). An Integer Programming Approach to the Student-Project Allocation Problem with Preferences over Projects. In *Proceeding of ISCO'2018, Lecture Notes in Computer Science*, page 313–325.
- Manlove, D. F. and O'Malley, G. (2008). Two algorithms for the student-project allocation problem. *Journal of Discrete Algorithms*, 5(4):553–560.
- Olaosebikan, S. and Manlove, D. (2022). Super-stability in the student-project allocation problem with ties. *Journal of Combinatorial Optimization*, 43:1203–1239.
- Roth, A. E. (1984). 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.
- Viet, H. H., Tan, L. V., and Cao, S. T. (2020). Finding Maximum Stable Matchings for the Student-Project Allocation Problem with Preferences Over Projects. In *Proceeding of FDSE'2020, Communications in Computer and Information Science*, page 411–422.