An efficient algorithm to find a maximum weakly stable matching for SPA-ST problem

Nguyen Thi Uyen[1](✉) and Tran Xuan Sang[2]

[1] School of Engineering and Technology, Vinh University, Vietnam
uyennt@vinhuni.edu.vn
[2] Cyber School, Vinh University, Vietnam
sangtx@vinhuni.edu.vn

Abstract. This paper presents a heuristic algorithm to seek a maximum *weakly stable* matching for the *Student-Project Allocation with lecturer preferences over Students containing Ties* (SPA-ST) problem. We extend Gale-Shapley's idea to find a stable matching and propose two new heuristic search strategies to improve the found stable matching in terms of maximum size. The experimental results show that our algorithm is more effective than AP in terms of solution quality and execution time for solving the MAX-SPA-ST problem of large sizes.

Keywords: SPA · Heuristic Search · Weakly Stable Matching · MAX-SPA-ST · Undominated Blocking Pairs.

1 Introduction

The *Student-Project Allocation problem with lecturer preferences over Students containing Ties* (SPA-ST) is an extension of the Student-Project Allocation problem (SPA) [7,18,17,6,20,9]. This extension makes the original SPA problem more practical because lecturers have preference lists over students, and students also have preference lists over projects with allowing ties in order. The goal of SPA-ST is to seek a *stable matching* like SPA, which includes pairs of students and projects based on their preference lists. Note that each student is eligible for only one project, and the capacity constraints of both projects and lecturers meet requirements and satisfactions. According to ties given in the SPA-ST problem, there are three stability criteria of matching consists of *weakly stable*, *strongly stable*, and *super-stable* matching [22,21].

Recently, several researchers have focused on solving the SPA-ST problem because of its applications to large-scale matching schemes in university departments around the world, such as Glasgow University [14], Southern Denmark University [20], York University [15], and elsewhere [10,4,3,2,8]. Several algorithms have been proposed to solve the SPA-ST problem. Cooper et al. [6] presented a 3/2- approximation algorithm, called AP, to find a *weakly stable* matching based on Király's idea for the HRT problem [16,13]. Besides, they also modeled the SPA-ST problem as an Integer Programming (IP) problem. Olaosebikan et al. [21] described the polynomial-time algorithm to find a *strongly* *stable* matching, and they proved that it might not exist for SPA-ST problem. Their algorithm runs in $O(m^2)$ time, where m is the total length of the students' preference lists. In addition, Olaosebikan et al. [22] proposed an approximation algorithm for solving SPA-ST problem in terms of finding a *super-stable* matching.

Practically, the problem of finding *weakly stable* matching is the most suitable for real-life applications. Irving et al. [11] showed that *weakly stable* matchings always exist and have different sizes [19]. This research aims to find a *weakly stable* matching with maximum size, called a MAX-SPA-ST problem, meaning that as many students as possible are assigned to projects. However, the MAX-SPA-ST problem is known as NP-hard, and therefore, finding an efficient algorithm to solve the MAX-SPA-ST of large sizes is a challenge for the research community.

Our contribution. This paper presents an effective heuristic algorithm to solve the MAX-SPA-ST problem of large sizes. Our main idea is to start from a stable matching, then define two heuristic strategies promoting *unmatched students* and *under-subscribed* lecturers to improve the matching size by breaking stable pairs. Our algorithm terminates when it finds a *perfect matching* or reaches a maximum number of iterations. The experimental results show that our proposed algorithm is more efficient than the AP algorithm [6] in terms of solution quality and execution time.

The rest of this paper is organized as follows: Section 2 presents preliminaries of SPA-ST, Section 3 describes our proposed algorithm, Section 4 discusses our experimental results, and Section 5 concludes our work.

2 Preliminaries

An SPA-ST instance consists of a set of students, denoted by $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$, a set of projects, denoted by $\mathcal{P} = \{p_1, p_2, \cdots, p_q\}$, and a set of lecturers, denoted by $\mathcal{L} = \{l_1, l_2, \cdots, l_m\}$. Each lecturer l_k offers a set of projects and ranks a set of students in her/his preference list. Each student s_i ranks a set of projects in her/his preference list. Both lecturers' and students' preference lists allow ties in order. Each lecturer has a capacity $d_k \in \mathbb{Z}^+$ indicating the maximum number of students that can be matched to l_k. Each project is offered by one lecturer and has a capacity $c_j \in \mathbb{Z}^+$ indicating the maximum number of students that can be matched to p_j. For any pair $(s_i, p_j) \in \mathcal{S} \times \mathcal{P}$ where p_j is offered by l_k, we consider (s_i, p_j) as an *acceptable pair* if s_i and p_j both find each other acceptable, i.e. p_j is ranked by a student s_i and s_i is ranked by a lecturer l_k who offers p_j. We denote the rank of p_j in s_i's preference list by $R_{s_i}(p_j)$ and the rank of s_i in l_k's preference list by $R_{l_k}(s_i)$. Note that we will use the term rank list instead of the preference list in the implementation process.

A matching M of a SPA-ST instance is a set of acceptable pairs (s_i, p_j) or $(s_i, \varnothing)$ such that $|M(s_i)| \leq 1$ for all $s_i \in \mathcal{S}$, $|M(p_j)| \leq c_j$ for all $p_j \in \mathcal{P}$, and $|M(l_k)| \leq d_k$ for all $l_k \in \mathcal{L}$. A project p_j is *under-subscribed*, *full* or *over-subscribed* according as $|M(p_j)| < c_j$, $|M(p_j)| = c_j$, or $|M(p_j)| > c_j$, respectively.

Similarly, lecturer l_k is *under-subscribed*, *full* or *over-subscribed* according as $|M(l_k)| < d_k$, $|M(l_k)| = d_k$, or $|M(l_k)| > d_k$, respectively. If $(s_i, p_j) \in M$, then s_i is matched to p_j, denoted by $M(s_i) = p_j$. If $M(s_i) = \varnothing$, then s_i is *unmatched* in M.

Let $(s_i, p_j) \in (\mathcal{S} \times \mathcal{P}) \setminus M$ be a blocking pair for a weakly stable matching M if the following conditions are satisfied:

1. s_i and p_j find accept each other;
2. s_i prefers p_j to $M(s_i)$ or $M(s_i) = \varnothing$;
3. either (a), (b) or (c) holds as follows:
   (a) $|M(p_j)| < c_j$ and $|M(l_k)| < d_k$;
   (b) $|M(p_j)| < c_j$, $|M(l_k)| = d_k$ and;
      i. either $s_i \in M(l_k)$ or;
      ii. l_k prefers s_i to the worst student in $M(l_k)$;
   (c) $|M(p_j)| = c_j$ and l_k prefers s_i to the worst student in $M(p_j)$.

Suppose that we have two blocking pairs (s_i, p_j) and (s_i, p_k), we say that (s_i, p_j) *dominates* (s_i, p_k) from the student's point of view if s_i prefers p_j to p_k. A pair (s_i, p_j) is *undominated* if there are no blocking pairs that dominate it from the student's point of view.

A matching M is called *weakly stable* if it admits no blocking pair, otherwise it is called *unstable*. In this paper, we consider a *weakly stable* matching as a stable matching. The size of a stable matching M, denoted by $|M|$, is the number of matched students in M. If $|M| =$ n, then M is a *perfect* matching, otherwise, M is a *non-perfect* matching.

3 Proposed algorithm

3.1 HA algorithm

This section describes our heuristic algorithm for the MAX-SPA-ST problem, called HA, in Algorithm 1. Our main idea is to start stable matching, which is adapted from Gale-Shapely's idea [7]. Then, if a stable matching is *non-perfect*, we improve its size by proposing two heuristic search strategies with two tasks as follows:

Task 1: Algorithm 1 selects a random *unmatched* student s_i from a stable matching M. Then, the algorithm considers one by one project $p_j \in \mathcal{P}$ in order of s_i's rank list in which p_j is offered by l_k. The algorithm finds a student s_t which is the same ties with s_i in l_k's rank list, i.e. $R_{l_k}(s_t) = R_{l_k}(s_i)$. Then, the algorithm satisfies either condition in case (1) or (2) as follows: *Case* (1): p_j is *under-subscribed* and $v(s_i) \geq v(s_t)$. *Case* (2): p_j is full, $s_t \in M(p_j)$ and $v(s_i) \geq v(s_t)$, then the algorithm replaces (s_t, p_z) where $p_z = M(s_t)$ by $(s_i,\, p_j)$ in M and increases the value of $v(s_i)$. It should be noted that the condition $v(s_i) \geq v(s_t)$ means that the number of replacements of s_i is higher than s_t, meaning that s_i is prioritized to match with p_j. If p_z is removed and became *under-subscribed*, we call the function *Repair*(p_z, l_k) to break blocking pairs for M.

Algorithm 1: HA Algorithm for MAX-SPA-ST problem

Input: - An SPA-ST instance I.
- max_iter is the maximum number of iterations.

Output: A maximum stable matching M.

function HA(I)
  $M :=$ `EGS(I)`; ▷ `generate a stable matching`
  $v(s_i) := 0$, $(1 \leq i \leq n)$; ▷ `mark the replacing time of` s_i
  $v(p_i) := 0$, $(1 \leq i \leq q)$; ▷ `mark the replacing time of` p_i
  $iter := 0$;
  while $(iter \leq max\_iter)$ **do**
    $iter := iter + 1$;
    if $|M| = n$ **then** break ;
    $s_i :=$ a random unmatched student in M; ▷ `Task 1`
    $R'_{s_i} := s_i$'s ranks list;
    while R'_{s_i} *is non-empty* **do**
      $p_j := \text{argmin}(R'_{s_i} > 0), \forall p_j \in \mathcal{P}$;
      $l_k :=$ a lecturer who offers p_j;
      for $(each\ s_t \in M(l_k)|\ R_{l_k}(s_t) = R_{l_k}(s_i))$ **do**
        if $(|M(p_j)| < c_j)$ *or* $(s_t \in M(p_j)$ *and* $|M(p_j)| = c_j)$ **then**
          if $v(s_i) \geq v(s_t)$ *or a small probability* **then**
            $M := M \setminus \{(s_t, p_z)\} \cup \{(s_i, p_j)\}|\ p_z = M(s_t)$;
            $v(s_i) := v(s_i) + 1$;
            $Repair(p_z, l_k)$;
            $M :=$ `Break_Student(`M, s_t`)`;
            break;
      if $M(s_i) \neq \varnothing$ **then** break;
      else $R'_{s_i}(p_j) := 0$;
    $p_i :=$ a random *under-subscribed* project in M; ▷ `Task 2`
    $l_k :=$ a lecturer who offers p_i;
    for *each* $s_j \in \mathcal{S}|R_{s_j}(p_t) = R_{s_j}(p_i)|p_t = M(s_j)$ **do**
      if $(|M(l_k)| < d_k)$ *or* $(|M(l_k)| = d_k$ *and* $s_j \in M(l_k))$ **then**
        if $(v(p_i) \geq v(p_t)$ *or a small probability* **then**
          $M := M \setminus \{(s_j, p_t)\} \cup \{(s_j, p_i)\}$;
          $v(p_i) := v(p_i) + 1$;
          $M :=$ `Break_Lecturer(`M, p_t`)`;
          break;
  return M;
end function

To avoid a local minimum with a small probability, we prioritize s_i without considering the value of $v(s_i)$. As a result, s_t is now *unmatched*, thus the algo-

Algorithm 2: Breaking blocking pair from the student of M

Input: A matching M.
Output: A stable matching M.

function `Break_Student`(M, s_t)
  while *(there exists blocking pairs)* **do**
    (s_t, p_u) := an undominated blocking pair from s_t;
    l_k := a lecturer who offers p_u;
    $M := M \cup \{(s_t, p_u)\}$;
    if p_u *is over-subscribed* **then**
      s_w := a worst student of p_u;
      $M := M \setminus \{(s_w, p_u)\}$;
      $s_t := s_w$;
    else if l_k *is over-subscribed* **then**
      s_r := a worst student of l_k;
      $M := M \setminus \{(s_r, p_z)\}$, where $p_z = M(s_r)$;
      Repair(p_z, l_k); ▷ `repair blocking pair of type` *(3bi)*
      $s_t := s_r$;
  return M;
end function

rithm calls the Algorithm 2 to break blocking pairs for M. Finally, Algorithm 1 returns a stable matching that is equal to or greater in size than the current matching M.

Task 2: Algorithm 1 selects a random *under-subscribed* project p_i from a stable matching M. Then, the algorithm finds a student s_j which ranks p_i at the same rank as $M(s_j)$ in s_j's rank list, i.e. $R_{s_j}(p_i) = R_{s_j}(p_t)$ where $M(s_j) = p_t$. Then, the algorithm satisfies either condition in case (1) or (2) as follows: *Case* (1): l_k is *under-subscribed* and $v(p_i) \geq v(p_t)$. *Case* (2): l_k is full, $s_j \in M(l_k)$, and $v(p_i) \geq v(p_t)$, then the algorithm replaces $(s_j,\ p_t)$ by $(s_j,\ p_i)$ in M and increases the value of $v(p_i)$. Note that $v(p_i) \geq v(p_t)$ means that the number of replacements of p_i is higher than p_t, i.e. p_i is prioritized to match with s_j. To avoid a local minimum, with a small probability, we always prioritize p_i without considering the value of $v(p_i)$. As a result, p_t and l_w who offers p_t are *under-subscribed*, thus the algorithm calls the Algorithm 3 to break blocking pairs for M. Finally, the Algorithm 1 returns a stable matching that is equal to or greater in size than the current matching M. Our HA algorithm stops if a perfect matching is found or it is reached to the maximum number of iterations.

We use Algorithm 2 to break blocking pairs when a student s_t is removed and becomes an *unmatched* student. The algorithm finds an undominated blocking pair (s_t, p_u) from s_t's point of view. If there exists, then we add (s_t, p_u) into M, where p_u is offered by l_k. This process repeats until there are no existing blocking pairs for only *unmatched* students who have just been removed.

Algorithm 3: Breaking blocking pair from the lecturer of M

Input: A matching M.
Output: A stable matching M.

function `Break_Lecturer`(M, p_t)
  while *(there exists blocking pairs)* **do**
    $l_w :=$ a lecturer who offers p_t;
    if p_t *is under-subscribed* **then**
      Repair(p_t, l_w); ▷ `repair blocking pair of type (3bi)`
    $(s_z, p_u) :=$ an undominated blocking pair from l_w;
    if *there exists pair* (s_z, p_u) **then**
      $M := M \setminus \{(s_z, p_h)\} \cup \{(s_z, p_u)\}$, where $p_h = M(s_z)$;
      if p_u *is over-subscribed* **then**
        $s_w :=$ a worst student of p_u;
        $M := M \setminus \{(s_w, p_u)\}$;
        $M :=$ `Break_Student`(M, s_w);
      else if l_w *is over-subscribed* **then**
        $s_r :=$ a worst student of l_w;
        $M := M \setminus \{(s_r, p_z)\}$, where $p_z = M(s_r)$;
        Repair(p_z, l_w);
        $M :=$ `Break_Student`(M, s_r);
      $p_t := p_h$;
    else
      break;
  return M;
end function

We use Algorithm 3 to break blocking pairs when a project p_t is replaced and becomes *under-subscribed*. The algorithm uses the function *Repair*(p_t, l_w) to remove blocking pairs type of *(3bi)*, then we find an undominated blocking pair (s_z, p_u) from l_w's point of view. If there exists, then we remove (s_z, p_h) where $p_h = M(s_z)$ and add (s_z, p_u) into M. This process repeats until there are no existing blocking pairs for only p_h which have just been removed.

3.2 Example

This section presents an example execution of our HA algorithm for the SPA-ST instance consisting of seven students, eight projects, and three lecturers in Table 1. Starting from a stable matching $M = \{(s_1, p_1), (s_2, \varnothing), (s_3, p_4), (s_4, p_2), (s_5, \varnothing), (s_6, p_5), (s_7, p_3)\}$ of size $|M| = 5$, HA runs as follows: HA algorithm takes a random *unmatched* student s_2 and finds s_6 such that $R_{l_2}(s_2) = R_{l_2}(s_6)$ from l_2's rank list. Then, HA removes (s_6, p_5) and adds (s_2, p_5) into M, thus s_6 becomes *unmatched*. Next, the algorithm calls Alg. 2 to break blocking pairs.

The algorithm finds an undominated blocking pair (s_6, p_8) from s_6's point of view and adds (s_6, p_8) into M to generate a new stable matching $M = \{(s_1, p_1), (s_2, p_5), (s_3, p_4), (s_4, p_2), (s_5, \varnothing), (s_6, p_8), (s_7, p_3)\}$ of size $|M| = 6$. Next, HA considers a random *under-subscribed* project p_7 and seeks a project p_1 which has $R_{s_1}(p_7) = R_{s_1}(p_1)$ from s_1's rank list. Then, the algorithm removes (s_1, p_1) and adds (s_1, p_7) into M. Next, the algorithm calls the Alg. 3 to break blocking pairs for M. Finally, HA returns a perfect matching $M = \{(s_1, p_7), (s_2, p_5), (s_3, p_1), (s_4, p_2), (s_5, p_4), (s_6, p_8), (s_7, p_3)\}$ of size $|M| = 7$.

Table 1. An instance of SPA-ST

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

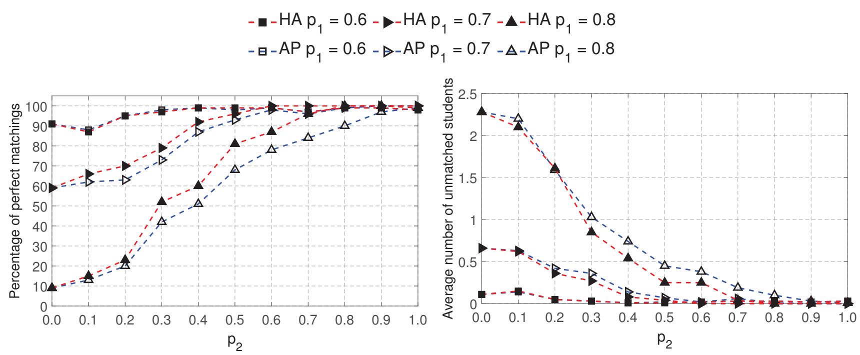
4 Experimental Results

In this section, we compared the solution quality and execution time of HA with those of AP which is an approximation algorithm [6] for the MAX-SPA-ST problem. We implemented these algorithms by Matlab R2019a software on a Xeon-R Gold 6130 CPU 2.1 GHz computer with 16 GB RAM. To perform the experiments, we generated randomly SPA-ST instances with five parameters (n, m, q, p_1, p_2), where n is the number of students, m is the number of lecturers, q is the number of projects, p_1 is the probability of incompleteness, and p_2 is the probability of ties. By this setting, on average, each student ranks about $q \times (1 - p_1)$ projects. In our experiments, we set the total capacity of projects and lecturers as $C = 1.2n$ and $D = 1.1n$, respectively.

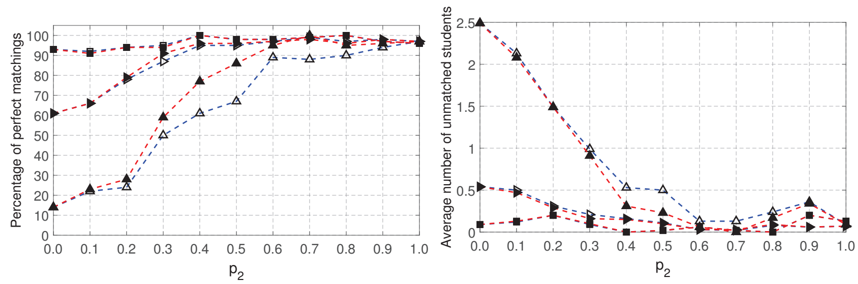
4.1 Comparison of solution quality

This section presents two experiments to compare the solution quality found by HA with that found by AP [6].

Experiment 1. Firstly, we randomly generated 100 instances of SPA-ST for parameters (n, m, q, p_1, p_2) with $n \in \{100, 200\}$, $m = 0.05n$, $q = 0.1n$, $p_1 \in [0.1, 0.8]$ with step 0.1, and $p_2 \in [0.0, 1.0]$ with step 0.1. Then, we ran HA and AP, averaged results, and compared the percentage of perfect matchings and the average number of *unmatched* students found by these two algorithms. Our experimental results show that when $p_1 \in [0.1, 0.6]$ with every the value of p_2, both our HA and AP obtain approximately 100% of perfect matchings, so we do not show the experiment results here. Figures 1(a) and 1(c) show the percentage of perfect matchings found by HA and AP. When $p_1 = 0.7$ or $p_1 = 0.8$, HA finds a much higher percentage of perfect matchings than AP does. Figures 1(b) and 1(d) show the average number of *unmatched* students found by HA and AP. When $p_1 = 0.7$ or $p_1 = 0.8$, HA finds a fewer number of *unmatched* students in stable matchings than AP does.



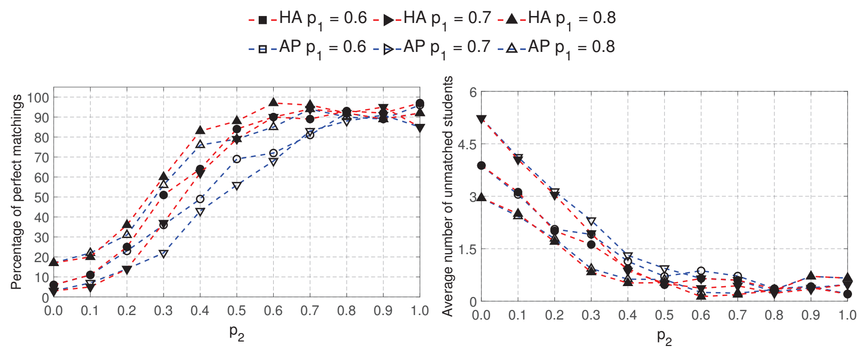
(a) – Solution quality for $n = 100$, $m = 5$, $q = 10$ – (b)



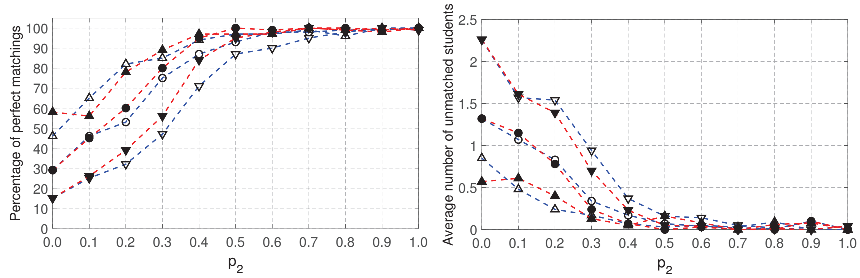
(c) – Solution quality for $n = 200$, $m = 10$, $q = 20$ – (d)

Fig. 1. Percentage of perfect matching and average number of unmatched students

Experiment 2. As we saw in Experiment 1, when p_1 increases, both HA and AP are hard to find perfect matchings since the number of projects ranked in students' preference lists decreases. In this experiment, we changed $n \in \{300, 400\}$, $p_1 \in \{0.82, 0.84, 0.86\}$ and kept the values of m, q, and p_2 as in Experiment 1. Figure 2 shows the percentage of perfect matchings found by HA and AP. Again, we see that HA finds a much higher percentage of perfect matchings than AP does.



(a) – Solution quality for $n = 300$, $m = 15$, $q = 30$ – (b)



(c) – Solution quality for $n = 400$, $m = 20$, $q = 40$ – (d)

Fig. 2. Percentage of perfect matching and average number of unmatched students

4.2 Comparison of execution time

In the above experiments, n is small, and therefore, the execution time of HA and AP is almost the same. This section presents two experiments to compare the execution time of HA and AP for SPA-ST instances of large sizes.

Experiment 3. We randomly generated 100 instances of SPA-ST for parameters (n, m, q, p_1, p_2) with $n \in \{1000, 2000\}$, $m = 0.05n$, $q = 0.4n$, $p_1 \in [0.1, 0.8]$ with step 0.1, and $p_2 \in [0.0, 1.0]$ with step 0.1. Figures 3 (a) and 3(b) show the average execution time over p_1 of HA and AP. When p_2 increases from 0.0 to 1.0, the execution time of AP almost remains unchanged, while HA slightly decreases, except for $p_2 = 1.0$, the execution time of HA significantly increases. When $n = 1000$, HA runs about 9 times faster than AP. When $n = 2000$, HA runs about 12.5 times faster than AP.

Experiment 4. Finally, we kept the values of n, m, q, and p_2 as in Experiment 3, increased the values of $p_1 \in [0.81, 0.89]$ with step 0.01, and randomly generated 100 instances of SPA-ST for each combination of values (p_1, p_2). By increasing the values of p_1, we aim to reduce the number of projects ranked by each student compared to Experiment 3. Figures 3(c) and 3(d) show the average execution time over p_1 of HA and AP. As in Experiment 3, we saw that when p_2 increases from 0.0 to 1.0, the execution time of AP almost remains unchanged, while HA slightly decreases, except for $p_2 = 1.0$, the execution time

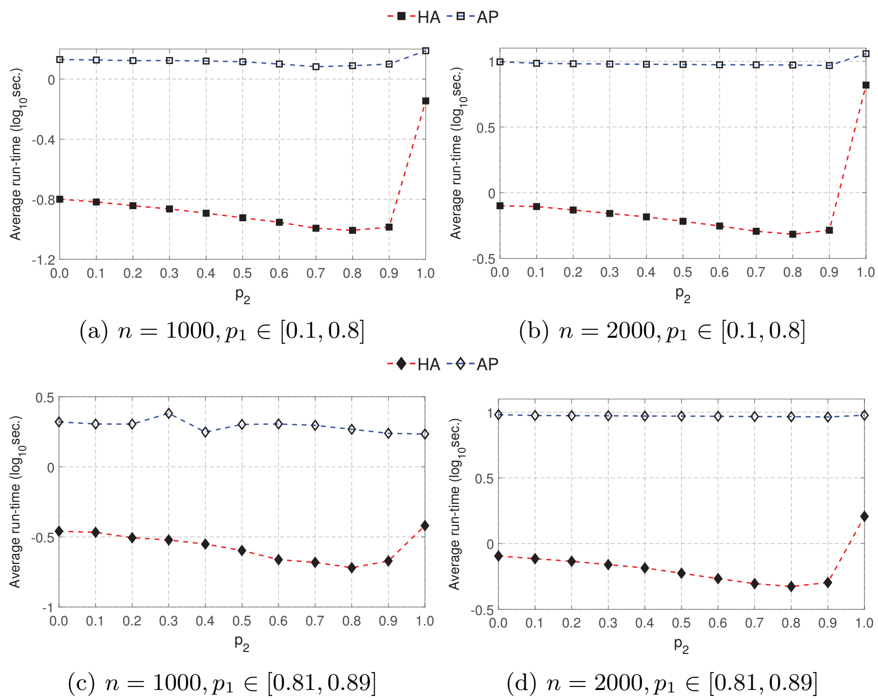


Fig. 3. Average of execution time for $n = 1000$, 2000

of HA increases. When $n = 1000$, HA ran faster about 5 times than AP. When $n = 2000$, HA runs faster about 12.5 times than AP.

5 Conclusions

In this study, we presented a heuristic algorithm for solving the MAX-SPA-ST problem. We started with a stable matching and improved the matching size by defining two heuristic strategies to pair the *unmatched* students and *under-subscribed* projects. The experimental results showed that our proposed algorithm is efficient in terms of solution quality and execution time for the MAX-SPA-ST problem of large sizes. In the future, we will extend this proposed approach to solve the other variants of the SPA problem [1,5,20,12].

References

1. Abraham, D., Irving, R., Manlove, D.: Two algorithms for the student-project allocation problem. Journal of Discrete Algorithms **5**(1), 73–90 (2007)
2. Aderant, F., Amosa, R., Oluwatobiloba, A.: Development of student project allocation system using matching algorithm. In: ICONSEET. vol. 1, pp. 153–160 (2016)

3. Binong, J.: Solving student project allocation with preference through weights. In: COMSYS. pp. 423–430 (2021)
4. Calvo-Serrano, R., Guillén-Gosálbez, C., Simon, K., Andrew, M.: Mathematical programming approach for optimally allocating students' projects to academics in large cohorts. Education for Chemical Engineers **20**, 11–21 (2017)
5. El-Atta, A.A., Ibrahim, M.M.: Student project allocation with preference lists over (student, project) pairs. In: ICCEE. vol. 1, pp. 375–379 (2009)
6. Frances, C., Manlove, D.: A 3/2-approximation algorithm for the student-project allocation problem. In: SEA. vol. 103, pp. 8:1–8:13 (2018). https://doi.org/10.4230/LIPIcs.SEA.2018.8
7. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. The American Mathematical Monthly **9**(1), 9–15 (1962)
8. Gani, M., AHamid, R., et al.: Optimum allocation of graduation projects: Survey and proposed solution. Journal of Al-Qadisiyah for Computer Science and Mathematics **13**(1), 58–66 (2021)
9. Hamada, K., Iwama, K., Miyazaki, S.: The hospitals-residents problem with lower quotas. Algorithmica **74**(1), 440–465 (2016)
10. Harper, R., Senna, V., Vieira, I., Shahani, A.: A genetic algorithm for the project assignment problem. Computers & Operations Research **32**(5), 1255–1265 (2005)
11. Irving, R., Manlove, D., Sandy, S.: The hospitals/residents problem with ties. In: SWAT. pp. 259–271. Bergen, Norway (Jul 2000)
12. Ismaili, A., Yahiro, K., Yamaguchi, T., Yokoo, M.: Student-project-resource matching-allocation problems: two-sided matching meets resource allocation. In: AAMAS. pp. 2033–2035 (2019)
13. Iwama, K., Miyazaki, S., Yanagisawa, H.: Improved approximation bounds for the student-project allocation problem with preferences over projects. Journal of Discrete Algorithms **13**, 59–66 (2012)
14. K.Augustine, Irving, R., Manlove, D., S.Colin: Profile-based optimal matchings in the student/project allocation problem. In: IWOCA. pp. 213–225 (2014)
15. Kazakov, D.: Co-ordination of student-project allocation. Manuscript (2002)
16. Király, Z.: Linear time local approximation algorithm for maximum stable marriage. Algorithms **6**(1), 471–484 (2013)
17. Kwanashie, K., Manlove, D.: An integer programming approach to the hospitals/residents problem with ties. In: GOR. pp. 263–269 (2013)
18. Manlove, D., Gregg, O.: Student-project allocation with preferences over projects. Journal of Discrete Algorithms **6**(4), 553–560 (2008)
19. Manlove, D., Irving, R., Iwama, K., Miyazaki, S., Morita, Y.: Hardvariants of stable marriage. Theoretical Computer Science **276**(1), 261–279 (2002)
20. Marco, C., Rolf, F., Stefano, G.: Handling preferences in student-project allocation. Annals of Operations Research **275**(1), 39–78 (2019)
21. Olaosebikan, S., Manlove, D.: An algorithm for strong stability in the student-project allocation problem with ties. In: CALDAM. pp. 384–399 (2020)
22. Olaosebikan, S., Manlove, D.: Super-stability in the student-project allocation problem with ties. Journal of Combinatorial Optimization pp. 1–37 (2020)