Tongliang Liu
Geoff Webb
Lin Yue
Dadong Wang (Eds.)

LNAI 14471

AI 2023: Advances in Artificial Intelligence

36th Australasian Joint Conference on Artificial Intelligence, AI 2023
Brisbane, QLD, Australia, November 28 – December 1, 2023
Proceedings, Part I

Springer

MOREMEDIA

Finding Maximum Weakly Stable Matchings for Hospitals/Residents with Ties Problem via Heuristic Search

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

Faculty of Information Technology, Vinh University, Vinh City, Vietnam
{sonct,thanhlv,viethh}@vinhuni.edu.vn

Abstract. The Hospitals/Residents with Ties Problem is a many-one stable matching problem, in which residents need to be assigned to hospitals to meet their constraints. In this paper, we propose a simple heuristic algorithm but solve this problem efficiently. Our algorithm starts from an empty matching and gradually builds up a maximum stable matching of residents to hospitals. At each iteration, we propose a heuristic function to choose the best hospital for an active resident to form a resident-hospital pair for the matching. If the chosen hospital overcomes its offered capacity, we propose another heuristic function to remove the worst resident among residents assigned to the hospital in the matching. Our algorithm returns a stable matching if it finds no active resident. Experimental results show that our algorithm is efficient in execution time and solution quality for solving the problem.

Keywords: Gale-Shapley algorithm · Hospitals/Residents with Ties · Heuristic algorithm · Weakly stable matching

1 Introduction

The Hospitals/Residents problem, as defined by Gale and Shapley in 1962, was initially called the "College Admissions Problem" [3]. This classic problem addresses the issue of matching medical residents to hospitals in a way that is both stable and satisfactory for all parties involved. In the original formulation of the problem, there are a set of hospitals and a set of residents. Each hospital has a ranking of the residents based on their preferences, while each resident has a ranking of the hospitals. The goal is to find a *stable matching*, where no two hospitals and residents would prefer each other over their current assignments.

Since its introduction, the Gale-Shapley (GS) algorithm has inspired further research and variations on the Hospitals/Residents (HR) problem, including the consideration of ties in preference rankings and the exploration of different optimization objectives. These developments continue to shape the field and contribute to improving the allocation of medical residents to hospitals and other related matching problems. We can find HR applications in various contexts and systems worldwide. Notable examples include the National Resident Matching

Program (NRMP) in the United States, the Scottish Pre-registration House Officer Allocations (SPA) matching scheme, and the Canadian Resident Matching Service (CaRMS) in Canada [9].

The Hospitals/Residents problem with Ties (HRT) is an extension of the classic Hospitals/Residents problem that allows ties in the preferences of both residents and hospitals, in which participants can rank a subset of the other set with equal preference, indicating that they consider those options equally desirable [9]. Stability in HRT implies that there are no resident-hospital pairs who both prefer each other over their current assignments, considering the ties in their preferences. There are various criteria for stability, such as *weak-stability*, *strong-stability*, or *super-stability* [8,9], which determine the level of preference satisfaction and the absence of *blocking pairs*.

Solving the HRT problem poses computational challenges, as including ties in preferences increases the complexity of finding stable matchings. The problem of finding a weakly stable matching with the maximum number of residents assigned to hospitals, known as MAX-HRT, has been proven to be NP-hard [9]. Researchers have proposed different algorithms and approaches to tackle the HRT problem, including integer programming [1,12], local and adaptive search [4,13], approximation [11], and heuristic repair [2] algorithms. However, finding efficient solutions for large-sized instances of HRT remains an area of ongoing research.

This paper proposes a heuristic algorithm to deal with the MAX-HRT problem. Our algorithm starts from an empty matching and proceeds iteratively to achieve a maximum stable matching of residents and hospitals. At each iteration, we design a heuristic function for choosing a hospital to assign to an active resident such that the hospital has not only the minimum remaining capacity but also the minimum remaining preference list. If the chosen hospital is *over-subscribed*, we design another heuristic function for removing a resident from the hospital such that the removed resident has not only the maximum rank among residents assigned to the hospital but also the maximum remaining preference list. The algorithm repeats until it finds no active resident and returns a stable matching. Experimental results show that our algorithm is efficient in solving the large-sized MAX-HRT problem.

The remaining sections of this paper are organized as follows. Section 2 provides a brief background on HRT. Section 3 outlines the details of our algorithm. Section 4 shows the results obtained from our experiments. Finally, Sect. 5 presents the concluding remarks of our work.

2 Preliminaries

In this section, we remind a brief background on HRT taken from [9,12]. An instance I of HRT consists of two sets, a set $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ of residents and a set $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$ of hospitals. Each resident $r_i \in \mathcal{R}$, $1 \leq i \leq n$, has a preference list that ranks a subset of hospitals in her/his preference with tie allowed. Similarly, each hospital $h_j \in \mathcal{H}$, $1 \leq j \leq m$, has a preference list that ranks a subset of residents in its preference with tie allowed. Each hospital h_j is assigned a *capacity* $c_j \in \mathbb{Z}^+$, indicating the maximum number of residents it can accommodate.

We denote the rank of $h_j \in \mathcal{H}$ (resp. $r_i \in \mathcal{R}$) in r_i's (resp. h_j's) preference list by $rank(r_i, h_j)$ (resp. $rank(h_j, r_i)$). Accordingly, if r_i (resp. h_j) ranks h_j (resp. r_i) in her/his (resp. its) preference list, then $1 \leq rank(r_i, h_j) \leq n$ (resp. $1 \leq rank(h_j, r_i) \leq n$), otherwise, $rank(r_i, h_j) = 0$ (resp. $rank(h_j, r_i) = 0$). If r_i (resp. h_j) strictly prefers h_j (resp. r_i) to h_k (resp. r_t), then we denote by $rank(r_i, h_j) < rank(r_i, h_k)$ (resp. $rank(h_j, r_i) < rank(h_j, r_t)$). If r_i (resp. h_j) prefers h_j (resp. r_i) and h_k (resp. r_t) equally, i.e. r_i (resp. h_j) ranks h_j (resp. r_i) and h_k (resp. r_t) with the same tie in her/his (resp. its) preference list, then we denote by $rank(r_i, h_j) = rank(r_i, h_k)$ (resp. $rank(h_j, r_i) = rank(h_j, r_t)$).

A pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ is called an *acceptable* pair in I if $rank(r_i, h_j) \geq 1$ and $rank(h_j, r_i) \geq 1$, i.e. r_i and h_j rank each other in their preference lists. A set of *acceptable* pairs is denoted by $\mathcal{A} = \{(r_i, h_j) \in \mathcal{R} \times \mathcal{H} | rank(r_i, h_j) \geq 1$ and $rank(h_j, r_i) \geq 1\}$, i.e., both the resident r_i and the hospital h_j must rank each other in their preference lists.

An *assignment* M in I is defined as a subset of $\mathcal{A}$, i.e., $M = \{(r_i, h_j) \in \mathcal{A}\}$. If $(r_i, h_j) \in M$, we say that r_i is *assigned* to h_j and vice versa. For any hospital $h_j \in \mathcal{H}$, we denote $M(h_j)$ by the set of residents assigned to h_j (i.e., $M(h_j) = \{r_i | (r_i, h_j) \in M\}$) and $M(r_i)$ by the hospital h_j assigned to r_i (i.e., $M(r_i) = h_j$), respectively. We denote by $M(r_i) = \varnothing$ if r_i is unassigned in M. A hospital $h_j \in \mathcal{H}$ is referred to as *under-subscribed*, *full-subscribed*, or *over-subscribed* depending on the conditions $|M(h_j)| < c_j$, $|M(h_j)| = c_j$, or $|M(h_j)| > c_j$, respectively.

A *matching* M in I is an assignment such that $|M(r_i)| \leq 1$, $\forall r_i \in \mathcal{R}$, and $|M(h_j)| \leq c_j$, $\forall h_j \in \mathcal{H}$. These conditions ensure that each resident is assigned to at most one hospital, and no hospital exceeds its capacity. As we mentioned above, this paper only considers the problem of finding maximum weakly stable matchings in I. Hereafter, we use the term *matching* to refer to weak matching.

A pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ is a *blocking pair* for a matching M if the following conditions hold: (i) $(r_i, h_j) \in \mathcal{A}$; (ii) $M(r_i) = \varnothing$ or $rank(r_i, h_j) < rank(r_i, M(r_i))$; and (iii) $|M(h_j)| < c_j$ or $rank(h_j, r_i) < rank(h_j, r_w)$, where r_w is the worst resident in $M(h_j)$. When a blocking pair exists, it implies that M is *unstable* because there are participants who would both prefer each other if given a chance, otherwise, M is called *stable*. The number of residents assigned to hospitals in M is denoted by $|M|$. If $|M| = n$, then M is called *perfect*, otherwise, M is called *non-perfect*.

Example 1. Let's consider the following example to provide greater clarity to mentioned notations. Given an instance of HRT consisting of a set $\mathcal{R} = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ of six residents, and a set $\mathcal{H} = \{h_1, h_2, h_3\}$ of three hospitals. Additional details can be found in Table 1, which presents the preference lists of residents and hospitals, including any ties indicated within round brackets. In the hospitals' preference lists of Table 1, let's focus on the second row for the hospital h_2. The notation "$h_2 : r_2\ r_1\ r_6\ (r_4\ r_5)$" indicates that h_2 strictly prefers r_2 to r_1, r_1 to r_6, and r_6 to both r_4 and r_5. Also, in this particular case, h_2 considers r_4 and r_5

Table 1. An instance of HRT

Residents' preference list	Hospitals' preference list	Residents' rank list	Hospitals' rank list
r_1: h_1 h_2	h_1: r_1 r_2 r_3 r_6	r_1: 1 2 0	h_1: 1 2 3 0 0 4
r_2: h_1	h_2: r_2 r_1 r_6 $(r_4$ $r_5)$	r_2: 1 0 0	h_2: 2 1 0 4 4 3
r_3: h_1 h_3	h_3: r_5 $(r_3$ $r_4)$	r_3: 1 0 2	h_3: 0 0 2 2 1 0
r_4: h_2		r_4: 0 1 0	
r_5: h_2 $(h_1$ $h_3)$		r_5: 2 1 2	
r_6: h_1 h_2		r_6: 1 2 0	
Hospitals' capacities: $c_1 = c_2 = c_3 = 2$			

to be equally preferred. Accordingly, we have $rank(h_2, r_2) = 1$, $rank(h_2, r_1) = 2$, $rank(h_2, r_6) = 3$, and $rank(h_2, r_4) = rank(h_2, r_5) = 4$. The same notations are utilized for the residents' preference lists. The rank lists corresponding to both the residents' and hospitals' preference lists are presented on the right-hand side of Table 1. The matching $M = \{(r_1, h_2), (r_2, h_1), (r_3, h_3), (s_4, h_2), (r_5, h_3), (r_6, h_1)\}$ is unstable due to the presence of blocking pairs $\{(r_1, h_2), (r_3, h_3)\}$ for M. Particularly, we have $rank(r_1, h_1) < rank(r_1, h_2)$, indicating that (r_1, h_2) forms a blocking pair. A similar explanation holds for the pair (r_3, h_3). The matching $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, \varnothing), (r_5, h_2), (r_6, h_2)\}$ is stable because there are no blocking pairs for M. However, this matching is considered non-perfect due to its cardinality $|M| = 5$. Similarly, the matching $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_3), (r_6, h_2)\}$ is stable since no blocking pair can be formed for M. In this case, M is a perfect matching since $|M| = 6$. $\lhd$

3 Heuristic Algorithm for MAX-HRT

In this section, we propose a heuristic algorithm to deal with the MAX-HRT problem. We consider the resident-oriented GS algorithm for HR problem given in [7]. At the beginning, a matching M is initialized to be empty, meaning that every resident is unassigned to any hospital ranked by her/him. At each iteration, an unassigned resident $r_i \in \mathcal{R}$ with a non-empty preference list is provisionally assigned to the most preferred hospital h_j in her/his preference list to form a pair $(r_i, h_j) \in M$. If the hospital h_j is *over-subscribed*, then the worst resident r_w assigned to h_j in $M(h_j)$ becomes free and h_j deletes r_w in its preference list (i.e. $rank(h_j, r_w) = 0$). If the hospital h_j is full, then h_j deletes the successor residents of the worst resident r_w in its preference list to accelerate finding a stable matching. The algorithm terminates with a resident-optimal stable matching M.

It is evident that we can apply the resident-oriented GS algorithm to find a stable matching for an instance of HRT. However, the stable matchings of an instance of HRT may be of different sizes [7]. Therefore, we are to extend the resident-oriented GS algorithm to find maximum stable matchings of HRT instances. Since the preference lists of both residents and hospitals in HRT include ties, we need to solve two issues in iterations of the resident-oriented GS algorithm: (i) an unassigned resident $r_i \in \mathcal{R}$ with a non-empty preference list is provisionally assigned to which most preferred hospital if there exist at least two most preferred hospitals with the same ties in her/his preference list; and (ii) if the hospital h_j is *over-subscribed*, then which worst resident assigned to h_j in $M(h_j)$ should be removed if there exist at least two worst residents with the same ties in its preference list.

For the first issue, we propose a heuristic function for all the hospital h_j in the preference list of an unassigned resident r_i as follows:

$$f(h_j) = rank(r_i, h_j) + 0.5 \times (|M(h_j)|/(c_j + 1) + |rank(h_j, r_k)|/(n + 1)), \quad (1)$$

where $|rank(h_j, r_k)|$ is the number of residents $r_k \in \mathcal{R}$ ranked by h_j in its preference list. Then, a hospital h_j is chosen to assign to r_i as follows:

$$h_j = \underset{h_j}{\operatorname{argmin}}\ f(h_j), \forall rank(r_i, h_j) > 0. \quad (2)$$

We can see $f(h_j)$ in Eq. (1) that $rank(r_i, h_j)$ is a positive integer and $0.5 \times (|M(h_j)|/(c_j + 1) + |rank(h_j, r_k)|/(n + 1)) < 1$. If a hospital h_j is chosen such that $f(h_j)$ is minimum, meaning that three following conditions are satisfied: (i) r_i prefers h_j most; (ii) h_j is being assigned to the least residents; and (iii) h_j ranks least residents in its preference list. These conditions ensure that the pair (r_i, h_j) is not only a stable pair in M, but also h_j with the least opportunities is prioritized to assign to r_j.

For the second issue, we propose another heuristic function for all the residents r_w being assigned to an *over-subscribed* hospital h_j as follows:

$$g(r_w) = rank(h_j, r_w) + |rank(r_w, h_t)|/(m + 1)), \forall r_w \in M(h_j), \quad (3)$$

where $|rank(r_w, h_t)|$ is the number of hospitals $h_t \in \mathcal{H}$ ranked by r_w in her/his preference list. Then, a resident r_w is chosen to remove from h_j as follows:

$$r_w = \underset{r_w}{\operatorname{argmax}}\ g(r_w), \forall r_w \in M(h_j). \quad (4)$$

We can see $g(r_w)$ in Eq. (3) that $rank(h_j, r_w)$ is a positive integer and $|rank(r_w, h_t)|/(m + 1) < 1$. If a resident $r_w \in M(h_j)$ is chosen such that $g(r_w)$ is maximum, meaning that two following conditions are satisfied: (i) r_w is the worst resident assigned to h_j; and (ii) r_w ranks most hospitals in her/his preference list. In other words, r_w is not only the worst resident assigned to h_j, but also has the highest number of opportunities to choose a hospital from her/his preference list during the next iterations.

Based on two heuristic functions above, we propose a simple heuristic algorithm shown in Algorithm 1. Initially, we set $M = \varnothing$ and $active(r_i) = 1, \forall r_i \in \mathcal{R}$, meaning that all residents are unassigned to any hospitals and r_i is marked as active. At each iteration, our algorithm checks if some active resident $r_i \in \mathcal{R}$

Algorithm 1: Heuristic Algorithm for MAX-HRT

```
Input  : An instance $I$ of HRT
Output : A stable matching $M$ for HRT
1  $M := \varnothing$;
2  $active(r_i) := 1, \forall r_i \in \mathcal{R}$;
3  while $\exists r_i$ such that $active(r_i) > 0$ do
4      if ($r_i$'s preference list is empty) then
5          $active(r_i) := 0$;
6          continue;
7      end
8      $f(h_j) = rank(r_i, h_j) + 0.5(|M(h_j)|/(c_j + 1) + |rank(h_j, r_k)|/(n + 1))$;
9      $h_j = \underset{h_j}{\text{argmin}}\ f(h_j), \forall rank(r_i, h_j) > 0$;
10     $M := M \cup (r_i, h_j)$;
11     $active(r_i) := 0$;
12     if $h_j$ is over-subscribed then
13         $g(r_w) := rank(h_j, r_w) + |rank(r_w, h_t)|/(m + 1)), \forall r_w \in M(h_j)$;
14         $r_w := \underset{r_w}{\text{argmax}}\ g(r_w), \forall r_w \in M(h_j)$;
15         $M := M \setminus (r_w, h_j)$;
16         $rank(r_w, h_j) := 0$;
17         $rank(h_j, r_w) := 0$;
18         $active(r_w) := 1$;
19     end
20 end
21 return $M$;
```

has no remaining hospital in her/his preference list, i.e. $rank(r_i, h_t) = 0$ for $\forall h_t \in \mathcal{H}$, then r_i is marked as inactive permanently, and the algorithm runs the next iteration (lines 4–7). Otherwise, r_i remains unassigned. Next, the algorithm finds a hospital h_j such that $f(h_j)$ is minimum, assigns h_j to r_i to form a pair $(r_i, h_j) \in M$, and marks r_i as inactive (lines 8–11). If h_j is *over-subscribed*, the algorithm computes $g(r_w)$ for $\forall r_w \in M(h_j)$, finds a resident r_w such that $g(r_w)$ is maximum, and removes the pair (r_w, h_j) from M. If so, r_w deletes h_j from her/his preference list and vice versa, r_w is marked as active again (lines 12–19). The algorithm repeats until there exists no active resident and it returns a stable matching.

We consider the time complexity of the proposed algorithm. In the best case, if each resident proposes the first preferred hospital in her/his preference list and she/he is assigned to this hospital to form a stable matching, then our algorithm takes $O(n)$ time, where n is the number of residents. In the worst case, if each resident ranks m hospitals and proposes to all the hospitals in her/his preference list, then our algorithm takes $O(nm)$ time, which is a linear time complexity.

Example 2. This example shows how Algorithm 1 operates using the HRT instance given in Table 1. Algorithm 1 initializes with $M = \varnothing$, $active(r_i) = 1$ for

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

Iter.	r_i	h_j	*Matching* M $(M = M \cup \{r_i, h_j\})$	*Over-subscribed*	$M = M \setminus \{r_i, h_j\}$
1	r_1	h_1	$\{(r_1, h_1)\}$		
2	r_2	h_1	$\{(r_1, h_1), (r_2, h_1)\}$		
3	r_3	h_1	$\{(r_1, h_1), (r_2, h_1)\}$	h_1	$M = M \setminus \{r_3, h_1\}$
4	r_3	h_3	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3)\}$		
5	r_4	h_2	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2)\}$		
6	r_5	h_2	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2)\}$		
7	r_6	h_1	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2)\}$	h_1	$M = M \setminus \{r_6, h_1\}$
8	r_6	h_2	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_6, h_2)\}$	h_2	$M = M \setminus \{r_5, h_2\}$
9	r_5	h_3	$\{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_3), (r_6, h_2)\}$		

all $r_i \in \mathcal{R}, 1 \leq i \leq n$, and runs iterations shown in Table 2. At the i^{th} iteration, the algorithm executes as follows:

(1) r_1 is assigned to h_1 since $f(h_1)$ is minimum. We have $M = \{(r_1, h_1)\}$ and r_1 is marked as inactive (i.e., *active*$(r_1) = 0$).
(2) r_2 is assigned to h_1 since $f(h_1)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1)\}$ and r_2 is marked as inactive.
(3) r_3 is assigned to h_1 since $f(h_1)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_1)\}$ and r_3 is marked as inactive. However, h_1 is *over-subscribed* and $g(r_3)$ is maximum, therefore the pair (r_3, h_1) is removed from M and r_3 is active again. So, we have $M = \{(r_1, h_1), (r_2, h_1)\}$.
(4) r_3 is assigned to h_3 since $f(h_3)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3)\}$ and r_3 is marked as inactive.
(5) r_4 is assigned to h_2 since $f(h_2)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2)\}$ and r_4 is marked as inactive.
(6) r_5 is assigned to h_2 since $f(h_2)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2)\}$ and r_5 is marked as inactive.
(7) r_6 is assigned to h_1 since $f(h_1)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2), (r_6, h_1)\}$ and r_6 is marked as inactive. However, h_1 is *over-subscribed* and $g(r_6)$ is maximum, therefore the pair (r_6, h_1) is removed from M and r_6 is active again. So, we have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2)\}$.
(8) r_6 is assigned to h_2 since $f(h_2)$ is minimum. We have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_2), (r_6, h_2)\}$ and r_6 is marked as inactive. However, h_2 is *over-subscribed* and $g(r_5)$ is maximum, therefore the pair (r_5, h_2) is removed from M and r_5 is active again. So, we have $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_6, h_2)\}$.
(9) Finally, r_5 is assigned to h_3 since $f(h_3)$ is minimum and r_5 becomes inactive. After this step, all residents are inactive, the algorithm returns a stable matching $M = \{(r_1, h_1), (r_2, h_1), (r_3, h_3), (r_4, h_2), (r_5, h_3), (r_6, h_2)\}$. In this case, M is also a perfect matching. $\triangleleft$

4 Experiments

In this section, we present experiments to evaluate the performance of our heuristic algorithm, namely HA, for finding maximum stable matchings of HRT instances. We chose the heuristic repair (HR) algorithm [2] to compare with HA since HR outperformed LS [4,5] in terms of the execution time and solution quality as mentioned in [2].

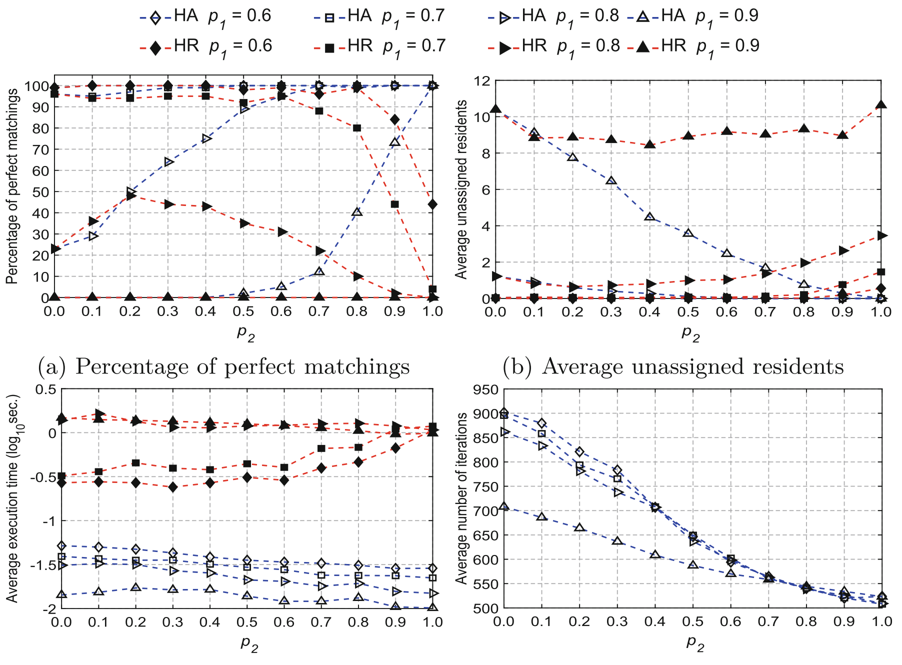
Datasets: we adapted the SMTI generator given in [6] to generate random HRT instances $I(n, m, p_1, p_2, \{c_1, c_2, \cdots, c_m\})$, where n is the number of residents, m is the number of hospitals, p_1 is the probability of incompleteness in preference lists of residents and hospitals, and p_2 is the probability of ties in the preference lists of residents and hospitals, and c_j is the capacity of each hospital $h_j \in H, 1 \leq j \leq m$. This means that on average, each resident ranks about $m(1 - p_1)$ hospitals and each hospital ranks about $n(1 - p_1)$ residents in each HRT instance. Since stable matchings of HRT instances consist of acceptable pairs, we generated HRT instances in which the residents' and hospitals' preference lists of each instance have only acceptable pairs. We implemented HA and HR algorithms by Matlab 2019a on a computer with a Core i7-8550U CPU 1.8 GHz and 16 GB memory.

4.1 Experiment 1

In this experiment, we randomly generated 100 HRT instances for each combination of values (n, m, p_1, p_2), where $n = 500$, $m = 50$, $p_1 \in \{0.0, 0.1, \cdots, 0.9\}$, and $p_2 \in \{0.0, 0.1, \cdots, 1.0\}$. In each instance, we set $c_j = n/m$ for all $h_j \in H, 1 \leq j \leq m$, meaning that the total capacities of hospitals are equal to n. It is evident that this is a hard experiment for algorithms to find perfect matching in HRT instances since each resident has only a post to be assigned to each hospital in her/his preference list. We set the maximum number of iterations of HR to 500.

Figure 1(a) shows the percentage of perfect matchings found by HA and HR when p_1 varies from 0.6 to 0.9. It should be noted that when p_1 varies from 0.0 to 0.5, both HA and HR find 100% of perfect matchings and therefore, we do not depict in Fig. 1(a). We see that when p_2 varies from 0.0 to 1.0, the percentage of perfect matchings found by HA increases, while that found by HR decreases, meaning that HA found perfect matchings easier than HR when ties in the preference lists of residents and hospitals increase. When $p_1 = 0.6$, the percentage of perfect matchings found by HA is approximate to that found by HR. When $p_1 \in \{0.7, 0.8, 0.9\}$, the percentage of perfect matchings found by HA is much higher than that found by HR.

Figure 1(b) shows the average number of unassigned residents in stable matchings. When p_2 varies from 0.1 to 1.0, HA finds a much smaller number of unassigned residents than HR in stable matchings, meaning that HA finds much larger stable matchings than HR. It should be noted that when $p_2 = 0.0$, meaning that the residents' and hospitals' preference lists do not contain ties, both HA and HR find the same percentage of perfect matchings as well as the same number of unassigned residents since all stable matchings have the same size [14].



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

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

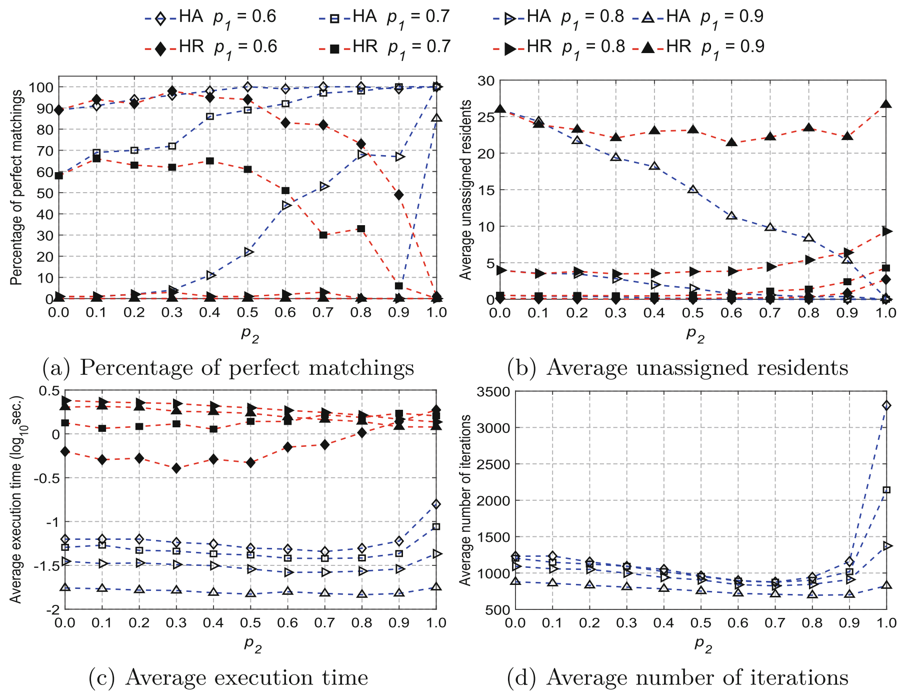
Fig. 1. Comparing solution quality and execution time of HA and HR algorithms

Figure 1(c) shows the average execution time of HA and HR. When p_1 increases from 0.6 to 0.9, the average execution time of HA and HR slightly changed. When p_2 increases from 0.0 to 1.0, the average execution time of HA and HR slightly decreases. We can see that the average execution time of HA increases from about $10^{-1.99} = 0.01$ to $10^{-1.28} = 0.05$ s, while that of HR increases from about $10^{-0.62} = 0.24$ to $10^{0.22} = 1.66$ s for every value of p_1 and p_2. This means that HA runs from about 24 to 33 times faster than HR.

Figure 1(d) shows the average number of iterations used by HA. HA used about 900 down to 500 iterations when p_2 increases from 0.0 to 1.0. It should be noted that we do not show the average number of iterations found by HR since the iteration mechanism of HR is different from that of HA.

4.2 Experiment 2

In this experiment, we randomly generated 100 HRT instances for each combination of values (n, m, p_1, p_2) given in Experiment 1. In each instance, we set the total capacity of hospitals to n and randomly distributed n to the capacity c_j of each hospital h_j such that $1 \leq c_j \leq 20$. Besides, we set the maximum number of iterations of HR to 500.



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

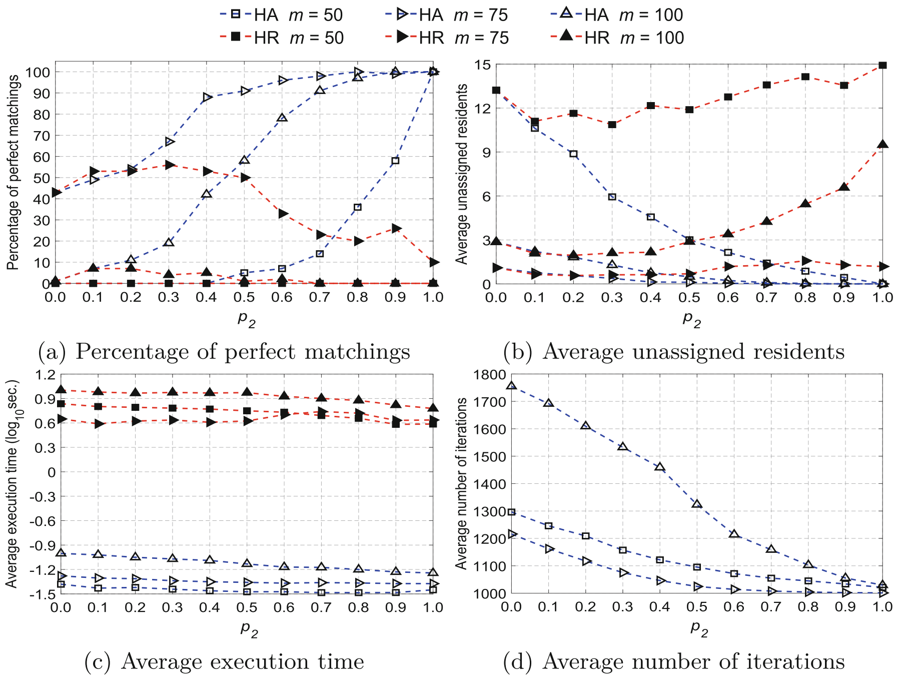
(c) Average execution time (d) Average number of iterations

Fig. 2. Comparing solution quality and execution time of HA and HR algorithms

Figure 2(a) shows the percentage of perfect matchings found by HA and HR. Again, HA finds a much higher percentage of perfect matchings than HR. Moreover, both HA and HR find stable matchings more difficult when $c_j = n/m$ as shown in Experiment 1. Figure 2(b) shows the average number of unassigned residents in stable matchings. When p_2 varies from 0.1 to 1.0, HA finds a much smaller number of unassigned residents in stable matchings than HR, meaning that HA finds much larger stable matchings than HR.

Figure 2(c) shows the average execution time of HA and HR. When p_2 varies from 0.0 to 1.0, the average execution time of HA and HR slightly changed. When p_1 varies from 0.6 to 0.9, the average execution time of HA decreases from about $10^{-1.3} = 0.050$ down to $10^{-1.8} = 0.016$ s, while that of HR increases from about $10^{-0.2} = 0.631$ to $10^{0.4} = 2.512$ s. On average over p_1, the average execution time of HA and HR is with respective to $10^{-1.5} = 0.032$ and $10^0 = 1.0$ s, meaning that HA runs about 31 times faster than HR.

Figure 2(d) shows the average number of iterations used by HA. When p_2 varies from 0.0 to 0.9, HA runs from about 800 to 1200 iterations. When $p_2 = 1.0$, the average number of iterations used by HA increases significantly, but the average execution time of HA increases sightly. This indicates that at each iteration, HA used a small amount of time to find a resident-hospital pair for a stable matching.



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

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

Fig. 3. Comparing solution quality and execution time of HA and HR algorithms

4.3 Experiment 3

In this experiment, we varied the number of hospitals. Specifically, we randomly generated 100 HRT instances for each combination of parameters (n, m, p_1, p_2), where $n = 1000$, $m \in \{50, 75, 100\}$, $p_1 = 0.9$, and $p_2 \in \{0.0, 0.1, \cdots, 1.0\}$. This means that each resident ranks on average at 5, 7.5, and 10 hospitals when $m =$ 50, 75, and 100, respectively, and each hospital ranks about 100 residents in the instances. In each instance, we set $c_j = n/m$ for all $h_j \in H, 1 \leq j \leq m$, meaning that each resident has one post assigned to each hospital. We set the maximum number of iterations of HR to 1000. Figure 3 shows the results of this experiment. Again, we see that our HA not only outperforms HR in matching quality shown by the percentage of perfect matchings and the number of unassigned residents, but also runs much faster than HR shown by the average execution time.

5 Conclusions

In this paper, we proposed a simple heuristic algorithm but it efficiently solved the MAX-HRT problem. Our algorithm starts from an empty matching to find a maximum stable matching of an HRT instance. At each iteration, the algorithm finds a hospital to assign to an active resident based on a heuristic function such that the hospital has not only the minimum remaining capacity but also the minimum preference list. If the chosen hospital overcomes its offered capacity, the algorithm finds a resident to remove her/him from the hospital based on another heuristic function such that the resident has not only the maximum rank among the residents being assigned to the hospital but also the maximum preference list. The algorithm repeats until it finds no active resident and returns a stable matching. Experiments showed that our algorithm is efficient in execution time and solution quality for large-sized HRT instances. In the future, we plan to extend this approach to find strongly- or super-stable matchings for HRT [9,10].

References

1. Biró, P., Manlove, D.F., McBride, I.: The hospitals/residents problem with couples: complexity and integer programming models. In: Proceeding of SEA 2014: 13th International Symposium on Experimental Algorithms, Copenhagen, Denmark, pp. 10–21 (2014)
2. Cao, S.T., Anh, L.Q., Viet, H.H.: A heuristic repair algorithm for the hospitals/residents problem with ties. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2022. LNCS, vol. 13588, pp. 340–352. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-23492-7_29
3. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **9**(1), 9–15 (1962)
4. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Local search for stable marriage problems with ties and incomplete lists. In: Proceedings of 11th Pacific Rim International Conference on Artificial Intelligence, Daegu, Korea, pp. 64–75 (2010)
5. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Local search approaches in stable matching problems. Algorithms **6**(4), 591–617 (2013)
6. Gent, I.P., Prosser, P.: An empirical study of the stable marriage problem with ties and incomplete lists. In: Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France, pp. 141–145 (2002)
7. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press Cambridge, New York (1989)
8. Irving, R.W.: Stable marriage and indifference. Discret. Appl. Math. **48**, 261–272 (1974)
9. Irving, R.W., Manlove, D.F., Scott, S.: The hospitals/residents problem with ties. In: Proceedings of the 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, pp. 259–271 (2000)
10. Irving, R.W., Manlove, D.F., Scott, S.: Strong stability in the hospitals/residents problem. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 439–450. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36494-3_39
11. Király, Z.: Linear time local approximation algorithm for maximum stable marriage. Algorithms **6**(1), 471–484 (2013)
12. Kwanashie, A., Manlove, D.F.: An integer programming approach to the hospitals/residents problem with ties. In: Huisman, D., Louwerse, I., Wagelmans, A.P.M. (eds.) Operations Research Proceedings 2013. ORP, pp. 263–269. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07001-8_36

13. Munera, D., Diaz, D., Abreu, S., Rossi, F., Saraswat, V., Codognet, P.: A local search algorithm for SMTI and its extension to HRT problems. In: Proceedings of the 3rd International Workshop on Matching Under Preferences, pp. 66–77. University of Glasgow (2015)
14. Irving, R.W., Manlove, D.F.: Finding large stable matchings. J. Exp. Algorithmics **14**(1), 1–2 (2009)