A Heuristic Repair Algorithm for the Hospitals/Residents Problem with Ties

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

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

Abstract. The Hospitals/Residents problem with Ties is a many-to-one stable matching problem and it has several practical applications. In this paper, we present a heuristic repair algorithm to find a stable matching with maximal size for this problem. Our approach is to apply a random-restart algorithm used commonly to deal with constraint satisfaction problems. At each iteration, our algorithm finds and removes the conflicted pairs in terms of preference ranks between hospitals and residents to improve rapidly the stability of the matching. Experimental results show that our approach is efficient in terms of execution time and solution quality for the problem of large sizes.

Keywords: Hospitals/residents with ties · Heuristic repair · Undominated blocking pair · Weakly stable matching

1 Introduction

In 1962, Gale and Shapley introduced the Hospitals/Residents problem (HR) under the name "*College Admissions Problem*" [3]. An instance of the HR involves a set of residents and a set of hospitals, in which each of them ranks a subset of the other set in a strict order of preference and each hospital has a *capacity* to indicate the maximum number of residents that can be assigned to it. Solving such a problem is to find a *matching* of residents and hospitals, in which each resident is assigned to at most one hospital and each hospital does not exceed its capacity. Moreover, the matching must be *stable* or it admits no *blocking pair*, where a blocking pair (r, h) for the matching is a resident r and a hospital h such that (i) r and h rank each other; (ii) r either is unassigned or prefers h to the hospital assigned to it; and (iii) h either is under-subscribed or prefers r to the worst resident assigned to it. HR can be found in applications such as the National Resident Matching Program (NRMP) in the US [18], the Scottish Pre-registration house officer Allocations (SPA) matching scheme [7], or the Canadian Resident Matching Service (CaRMS) in Canada [1].

Recently, there are several variations of HR have been proposed by researchers [2,8,13,15]. The most popular one is a natural generalization of HR

L. Rutkowski et al. (Eds.): ICAISC 2022, LNAI 13588, pp. 340–352, 2023.
https://doi.org/10.1007/978-3-031-23492-7_29

known as the Hospitals/Residents problem with Ties (HRT) [8,13], where both residents and hospitals can rank a subset of the other set with ties. Accordingly, there are three criteria of stable matchings consisting of *weak stability*, *strong stability*, and *super-stability* [8]. Among these criteria, the problem of finding weakly stable matchings has been an active field of researchers for several years since its practical applications. Irving et al. [8] showed that an instance of HR may have more than one stable matching and every stable matching is the same size, while an instance of HRT may have more than one weakly stable matching with different sizes. The problem of finding a weakly stable matching with the maximum number of residents assigned to hospitals is known as MAX-HRT and shown to be NP-hard [8].

In the last few years, several algorithms to solve MAX-HRT were introduced in the literature. Manlove et al. [14] proved that the size of the largest stable matching was at most twice the size of the smallest one for any HRT instance. Kwanashie et al. [12] presented an integer programming approach to find a stable matching. Munera et al. [16] proposed an adaptive search algorithm for the stable matching with ties and incomplete lists (SMTI) [10,14] and its extension to deal with MAX-HRT. Kir'aly [11] described ingenious approximation algorithms for MAX-HRT. However, all the algorithms mentioned above are inefficient to solve MAX-HRT of large sizes.

In this paper, we propose a heuristic repair algorithm to solve MAX-HRT. For brevity, hereinafter, we refer to a weakly stable matching as a stable matching and MAX-HRT as HRT. Our idea is to improve the stability of a randomly generated matching. At each iteration, our algorithm finds a set of undominated blocking pairs of a matching from the residents' point of view, then it removes the best blocking pair for each hospital such that it does not only remove as many blocking pairs from the residents' point of view as possible but also removes as many blocking pairs as possible from the hospitals' point of view. Experimental results show that our algorithm is efficient in solving HRT of large sizes.

The remainder of this paper is structured as follows. Section 2 reminds the main definitions for HRT, Sect. 3 presents our proposed algorithm, Sect. 4 discusses our experimental results, and Sect. 5 concludes our work.

2 Background

In this section, we remind the background for HRT [4,8]. An instance I of HRT involves a set of residents, denoted by $\mathcal{R} = \{r_1, r_2, \cdots, r_n\}$, and a set of hospitals, denoted by $\mathcal{H} = \{h_1, h_2, \cdots, h_m\}$, in which each $r_i \in \mathcal{R}$ ranks a subset of $\mathcal{H}$ in its preference list and each $h_j \in \mathcal{H}$ ranks a subset of $\mathcal{R}$ in its preference list. Moreover, each h_j has a *capacity* $c_j \in \mathbb{Z}^+$ to indicate the maximum number of residents that can be assigned to it. We denote a set of *acceptable* pairs by $\mathcal{A} = \{(r_i, h_j) \in \mathcal{R} \times \mathcal{H}\}$, where r_i and h_j must rank each other.

An *assignment* M is a subset of $\mathcal{A}$. If $(r_i, h_j) \in M$, we say that r_i is assigned to h_j and h_j is assigned r_i, and we denote $M(h_j)$ by the set of residents assigned to h_j and $M(r_i) = h_j$, respectively. If r_i is unassigned in M, then we denote by

$M(r_i) = \varnothing$. A hospital $h_j \in \mathcal{H}$ is called *under-subscribed*, *full*, or *over-subscribed* if $|M(h_j)| < c_j$, $|M(h_j)| = c_j$, or $|M(h_j)| > c_j$, respectively.

Definition 1 (matching). *A matching is an assignment M such that $|M(r_i)| \leq 1$ for each $r_i \in \mathcal{R}$, and $|M(h_j)| \leq c_j$ for each $h_j \in \mathcal{H}$, meaning that each resident is assigned to at most one hospital, and no hospital is over-subscribed.*

Given a matching M and a pair $(r_i, h_j) \in \mathcal{A}$, if r_i strictly prefers h_j to $M(r_i)$, then we denote by $h_j \prec_{r_i} M(r_i)$; if h_j strictly prefers r_i to the worst resident in $M(h_j)$, then we denote by $r_i \prec_{h_j} M(h_j)$.

Definition 2 (blocking pair). *A pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ is a blocking pair for a matching M if (i) $(r_i, h_j) \in \mathcal{A}$; (ii) $M(r_i) = \varnothing$ or $h_j \prec_{r_i} M(r_i)$; and (iii) $|M(h_j)| < c_j$ or $r_i \prec_{h_j} M(h_j)$.*

Definition 3 (stable matching). *A matching M is called stable if it admits no blocking pairs, otherwise, it is called unstable.*

Definition 4 (matching size). *The size of a stable matching M, denoted by $|M|$, is the number of residents assigned to hospitals in M. If $|M| = n$, then M is called perfect. Otherwise, M is called non-perfect.*

Definition 5 (dominated blocking pair). *A blocking pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ dominates a blocking pair $(r_i, h_k) \in \mathcal{R} \times \mathcal{H}$ from the residents' point of view if r_i prefers h_j to h_k.*

Definition 6 (undominated blocking pair). *A blocking pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ is called an undominated blocking pair (*UBP*) if there exists no other blocking pair that dominates it from the residents' point of view.*

The concepts of the dominated and undominated blocking pairs were given in [4] and then they were applied to solve efficiently the SMTI problem [5,17]. In this paper, we apply these concepts to solve HRT. Given a matching M and a blocking pair $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ for M, we call an operation of removing (r_i, h_j) for M means that r_i is assigned to h_j, or $M(r_i) = h_j$. We assume that there exist two blocking pairs, denoted by $(r_i, h_j) \in \mathcal{R} \times \mathcal{H}$ and $(r_i, h_k) \in \mathcal{R} \times \mathcal{H}$, for M, where (r_i, h_j) dominates (r_i, h_k) from the residents' point of view. If we remove (r_i, h_j) for M to obtain a matching M' from M, i.e. $M'(r_i) = h_j$, and the other pairs of M' are the same as those of M, except if $M'(h_j) > c_j$, then the worst resident in $M'(h_j)$ becomes unassigned. As a result, the blocking pair (r_i, h_k) is removed for M'. Otherwise, if we remove (r_i, h_k) for M to obtain a matching M' from M, then the blocking pair (r_i, h_j) still remains for M'. This follows that if we remove an UBP (r_i, h_j) for a matching M, then all the blocking pairs formed by r_i from the residents' point of view will be removed for M. We have equivalent concepts of the dominated and undominated blocking pairs from the hospitals' point of view. Accordingly, if we remove an UBP (r_i, h_j) from the hospitals' point of view, then all the blocking pairs formed by h_j will be removed for M.

Table 1. An instance of HRT of eight residents and four hospitals

Residents	Preference lists	Hospitals	Preference lists	Capacities
r_1	h_1 $(h_2$ $h_3)$ h_4	h_1	r_8 r_2 r_7 r_1 r_6 r_5 r_3 r_4	$c_1 = 3$
r_2	h_4 h_1 h_2 h_3	h_2	r_6 r_2 r_1 r_4 r_3 r_7	$c_2 = 6$
r_3	h_1 h_3 h_4 h_2	h_3	r_6 r_2 r_1 r_4 r_5 r_8 r_7 r_3	$c_3 = 3$
r_4	$(h_1$ $h_4)$ h_2 h_3	h_4	r_2 r_5 r_4 $(r_7$ $r_8)$ r_1 r_3	$c_4 = 4$
r_5	h_3 h_1 h_4			
r_6	h_2 h_1 h_3			
r_7	h_2 h_4 h_1 h_3			
r_8	h_1 h_3 h_4			

We consider an HRT instance consisting of 8 residents and 4 hospitals shown in Table 1. In residents' preference lists, for example, the notation r_1: h_1 $(h_2$ $h_3)$ h_4 means r_1 strictly prefers h_1 to h_2 and h_3, which are equally preferred. We have similar notations in the hospitals' preference lists. The matching $M = \{(r_1, \varnothing), (r_2, \varnothing), (r_3, h_1), (r_4, h_1), (r_5, h_3), (r_6, h_1), (r_7, h_3), (r_8, \varnothing)\}$ is unstable because there exist blocking pairs such as (r_1, h_1), (r_1, h_2), (r_1, h_3), (r_1, h_4), (r_2, h_1) for M. The blocking pair (r_1, h_1) dominates the blocking pair (r_1, h_4) from the residents' point of view and the blocking pair (r_1, h_1) is undominated since there exists no blocking pairs dominating it from the residents' point of view. If we remove (r_1, h_1) for M to obtain a matching M', i.e. $M' = \{(r_1, h_1), (r_2, \varnothing), (r_3, h_1), (r_4, \varnothing), (r_5, h_3), (r_6, h_1), (r_7, h_3), (r_8, \varnothing)\}$, then all the UBPs formed by r_1 from the residents' point of view are removed for M'. The matching $M = \{(r_1, h_1), (r_2, h_4), (r_3, h_1), (r_4, h_4), (r_5, h_3), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$ is perfect since M is stable and $|M| = 8$.

3 Algorithm for HRT

In this section, we propose an algorithm of repairing undominated blocking pairs, called heuristic repair algorithm, to solve MAX-HRT. Given an arbitrary matching M of an instance I of HRT, we assume that there exists a set $X = \{(r_i, h_j) | (r_i, h_j) \in \mathcal{R} \times \mathcal{H}\}$ of UBPs from the residents' point of view for M. As we mentioned above, if we remove only an UBP $(r_i, h_j) \in X$ for M (i.e. $M(r_i) = h_j$), then all the blocking pairs formed by r_i will be removed for M. If so, we were wasted time in finding the remaining pairs in X. Obviously, we cannot remove every pair $(r_i, h_j) \in X$, since if there exist two pairs $(r_i, h_j) \in X$ and $(r_k, h_j) \in X$, then we remove (r_i, h_j) or (r_k, h_j) for M (i.e. $M(r_i) = h_j$ or $M(r_k) = h_j$)? Our question is that which pairs $(r_i, h_j) \in X$ should be removed in M such that we can rapidly obtain the stability of M. To answer this question, we first analyze the instance of HRT given in Table 1. We assume that given an unstable matching $M = \{(r_1, \varnothing), (r_2, \varnothing), (r_3, h_1), (r_4, h_1), (r_5, h_3), (r_6, h_1), (r_7, h_3), (r_8, \varnothing)\}$, then the set of UBPs from the residents' point of view for M

is $X = \{(r_1, h_1), (r_2, h_4), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$. Since X is a set of UBPs from the residents' point of view, each $r_i \in X$ belongs to only one element of X, while each $h_j \in X$ is not so. This means that we can partition $X = X_1 \cup X_2 \cup X_3$, where $X_1 = \{(r_1, h_1), (r_8, h_1)\}$, $X_2 = \{(r_2, h_4)\}$, and $X_3 = \{(r_6, h_2), (r_7, h_2)\}$. If we remove a pair $(r_i, h_j) \in X_t (t = 1, 2, 3)$ that (r_i, h_j) dominates all the other $(r_k, h_j) \in X_t$ from the hospitals' point of view, then all (r_k, h_j) formed by h_j from the hospitals' point of view are removed for M.

As with the analysis above, our idea to solve HRT is that at each iteration of our algorithm, we do the following: (i) finding a set X of UBPs for an unstable matching M from the residents' point of view; (ii) partitioning $X = X_1 \cup X_2 \cup \cdots \cup X_l$ such that each $X_t (t = 1, 2, \cdots, l)$ consists of blocking pairs $(r_i, h_j) \in X$ formed by a unique $h_j \in X$; and (iii) removing a pair $(r_i, h_j) \in X_t (t = 1, 2, \cdots, l)$ that (r_i, h_j) dominates all the other $(r_k, h_j) \in X_t$ from the hospitals' point of view. By doing so, our idea is not to remove all the blocking pairs formed by r_i from the residents' point of view but also reject as many blocking pairs formed by h_j from the hospitals' point of view as possible to obtain a stable matching of an HRT instance as quickly as possible.

Our algorithm is shown in Algorithm 1. To avoid getting stuck in local maxima, we use the mechanism of the random-restart hill climbing algorithm [19]. Specifically, our algorithm finds a maximum stable matching, denoted by M_{best}, from a randomly generated matching M. At each iteration, our algorithm runs as follows. First, the algorithm finds a set X of UBPs for M from the residents' point of view (line 4). Second, the algorithm checks if X is empty, then if M_{best} is worse than M in terms of the matching size, M is assigned to M_{best} (lines 6-8). Next, the algorithm checks if M_{best} is perfect, then it returns M_{best} (lines 9-11), otherwise, it restarts at a randomly generated matching M and continues the next iteration (lines 12-13). Third, the algorithm checks if a small probability of p is accepted, it chooses a random pair $(r_i, h_j) \in X$ and removes it for M (lines 15-22). Otherwise, it iterates for each $h_j \in X$ to select a pair $(r_i, h_j) \in X$ that h_j prefers r_i to r_k for all $(r_k, h_j) \in X$ and removes (r_i, h_j) for M (lines 24-25). When the algorithm removes a blocking pair (r_i, h_j) for M, i.e. $M(r_i) = h_j$, and if h_j is over-subscribed, then it removes the pair $(r_z, h_j) \in M$ such that h_j is full, where r_z is the worst resident assigned to h_j in M (lines 26-29). Finally, the algorithm repeats until either M_{best} is a perfect matching or a maximum number of iterations is reached. In the latter case, the algorithm returns either a maximum stable matching found so far or an unstable matching. We note that to find an UBP $(r_i, h_j) \in X$ from the residents' point of view for M, the algorithm runs an iteration for each hospital h_j in ascending order of ranks in r_i's preference list and returns the first blocking pair encountered, then (r_i, h_j) is an undominated blocking pair.

An execution of our algorithm for the HRT instance shown in Table 1 is illustrated as in Table 2. We assume that the probability to choose a random pair in X is $p = 0$ and the algorithm starts from a random matching $M_0 = \{(r_1, \varnothing), (r_2, \varnothing), (r_3, h_1), (r_4, h_1), (r_5, h_3), (r_6, h_1), (r_7, h_3), (r_8, \varnothing)\}$. At the first iteration, the algorithm finds a set $X_0 = \{(r_1, h_1), (r_2, h_4), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$

Algorithm 1: Heuristic Repair Algorithm

```
Input: - An HRT instance I of size n × m
       - A small probability p.
       - The maximum iterations max_iters.
Output: A matching M_best.
M := a randomly generated matching;
M_best := M;
for iter := 1 to max_iters do
    X := a set of undominated blocking pairs for M;
    if (X = ∅) then
        if (|M_best| < |M|) then
            M_best := M;
        end
        if (|M_best| = n) then
            break;
        end
        M := a randomly generated matching;
        continue;
    end
    if (a small probability of p) then
        take a random pair (r_i, h_j) ∈ X;
        M(r_i) := h_j;
        if (h_j is over-subscribed ) then
            r_z := worst resident in M(h_j);
            M(r_z) := ∅;
        end
    else
        for (each h_j ∈ X) do
            select (r_i, h_j) ∈ X such that h_j prefers r_j to r_k, ∀(r_k, h_j) ∈ X;
            M(r_i) := h_j;
            if (h_j is over-subscribed ) then
                r_z := worst resident in M(h_j);
                M(r_z) := ∅;
            end
        end
    end
end
return M_best;
```

of UBPs from the residents' point of view for M_0. Since (r_8, h_1) dominates (r_1, h_1) from the hospitals' point of view (i.e. h_1 prefers r_8 to r_1) and (r_6, h_2) dominates (r_7, h_2) from the hospitals' point of view (i.e. h_2 prefers r_6 to r_7), the algorithm removes (r_2, h_4), (r_6, h_2) and (r_8, h_1) to obtain a matching $M_1 = \{(r_1, \varnothing), (r_2, h_4), (r_3, h_1), (r_4, \varnothing), (r_5, h_3), (r_6, h_2), (r_7, h_3), (r_8, h_1)\}$. At the second iteration, the algorithm finds a set $X_1 = \{(r_1, h_1), (r_4, h_1), (r_7, h_2)\}$ of UBPs from the residents' point of view for M_1. It should be noted that at the first iteration, (r_8, h_1) dominated (r_1, h_1) and we removed (r_8, h_1) for M_0, but there exists $(r_1, h_1) \in X_1$ for M_1, since $(r_1, h_1) \in X_1$ is an UBP found from the residents' point of view. It is explained similarly for $(r_7, h_2) \in X_1$. Since (r_1, h_1) dominates (r_4, h_1) from the hospitals' point of view, the algorithm removes (r_1, h_1) and (r_7, h_2) to obtain a matching M_2. The algorithm repeats until the fourth iteration, where $X_3 = \{\emptyset\}$, and it returns a perfect matching M_3.

Table 2. An execution of the algorithm for HRT in Table 1

Iter.	Input	UBPs	Remove	Output
1	M_0	$X_0 = \{(r_1, h_1), (r_2, h_4), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$	$\{(r_2, h_4), (r_6, h_2), (r_8, h_1)\}$	$M_1 = \{(r_1, \varnothing), (r_2, h_4), (r_3, h_1), (r_4, \varnothing), (r_5, h_3), (r_6, h_2), (r_7, h_3), (r_8, h_1)\}$
2	M_1	$X_1 = \{(r_1, h_1), (r_4, h_1), (r_7, h_2)\}$	$\{(r_1, h_1), (r_7, h_2)\}$	$M_2 = \{(r_1, h_1), (r_2, h_4), (r_3, h_1), (r_4, \varnothing), (r_5, h_3), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$
3	M_2	$X_2 = \{(r_4, h_4)\}$	$\{(r_4, h_4)\}$	$M_3 = \{(r_1, h_1), (r_2, h_4), (r_3, h_1), (r_4, h_4), (r_5, h_3), (r_6, h_2), (r_7, h_2), (r_8, h_1)\}$
4	M_3	$X_3 = \{\emptyset\}$		

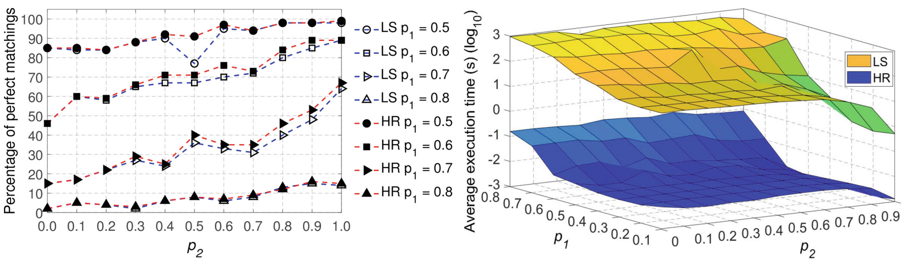
4 Experiments

In this section, we evaluate the performance of our heuristic repair algorithm, namely HR, for HRT. To do this, we applied the SMTI generator [6] to generate HRT instances with parameters (n, m, p_1, p_2), where n is the number of residents, m is the number of hospitals, p_1 is the probability of incompleteness, and p_2 is the probability of ties. Without loss of generality, we assume that in each generated instance, the preference lists of residents and hospitals consist of acceptance pairs. Otherwise, we run a preprocessing procedure to remove unacceptance pairs in HRT instances. We implemented all experiments by Matlab 2019a on a personal computer with a Core i7-8550U CPU 1.8GHz and 16 GB memory.

4.1 Comparison with Local Search

In this section, we present an experiment to compare the execution time and solution quality found by HR with those found by Local Search (LS) [4]. We set the probability $p = 0.03$ and the maximum number of iterations to 500 in both HR and LS algorithms.

Experiment 1. We chose $n = 100$, $m = 10$, $p_1 \in [0.1, 0.8]$ with step 0.1, and $p_2 \in [0.0, 1.0]$ with step 0.1. For each combination of parameters (n, m, p_1, p_2), we randomly generated 100 HRT instances, in which the capacity c_j of each hospital $h_j \in \mathcal{H}$ is generated randomly and $c_j \in [1, q]$, where q is the total number of residents ranked by hospital $h_j \in \mathcal{H}$. Then, we ran HR, LS and averaged results. Figure 1(a) shows the percentage of perfect matchings found by HR and LS. When p_1 varies from 0.1 to 0.4, both HR and LS always find 100% of perfect matchings (therefore, they are not depicted in Fig. 1(a)), while p_1 varies from 0.5 to 0.8, the percentage of perfect matchings found by HR is slightly higher than that found by LS. Figure 1(b) shows the average execution time of HR and LS. The experimental results show that HR runs about 100 times faster than LS for any p_1 and p_2. On average, the execution time of HR increases from about 0.008(s) to 0.02(s), while that of LS increases from about 0.5(s) to 43.5(s) for any value of p_2. In contrast, when p_2 varies from 0.0 to 1.0, the execution time of both HR and LS decreases slightly for any value of p_1. This can be explained as follows. Although LS considers only UBPs, the number of such UBPs is very large, i.e. the number of neighbor matchings is very large because a neighbor is generated by removing a blocking pair in the set of UBPs. This increases significantly the execution time of LS. However, HR finds the set of UBPs and removes many blocking pairs in the set of UBPs to generate a new matching for the next iteration without evaluating the cost of matchings as in LS and therefore, HR runs much faster than LS.



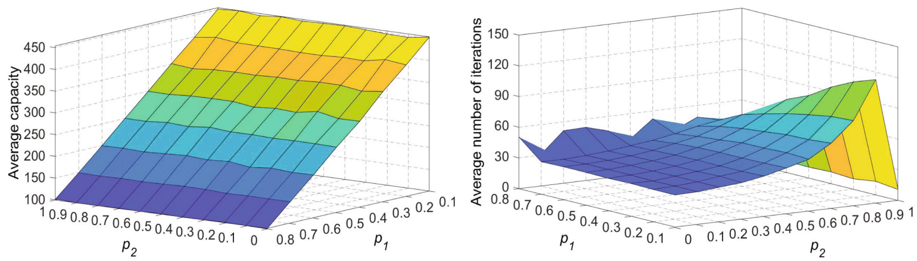
(a) Percentage of perfect matchings (b) Average execution time

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

4.2 Experiments for HRT of Large Sizes

In this section, we present experimental results for HRT instances of large sizes to consider the behavior of our algorithm. We set $p = 0.03$ and $max\_iters = 1000$ in HR.

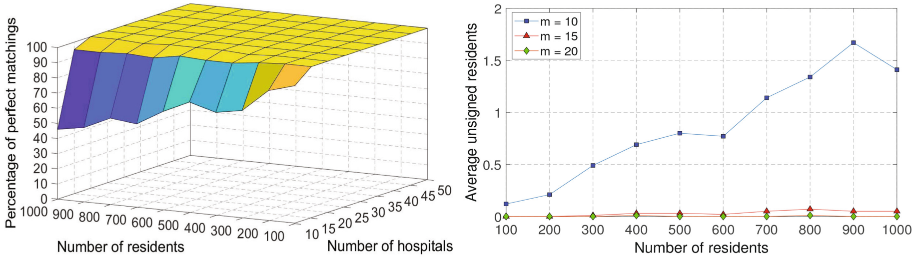
Experiment 2. We chose $n = 1000$, $m = 50$, $p_1 \in [0.1, 0.8]$ with step 0.1, and $p_2 \in [0.0, 1.0]$ with step 0.1. For each combination of parameters (n, m, p_1, p_2), we randomly generated 100 HRT instances, in which c_j of each hospital $h_j \in \mathcal{H}$ is generated randomly and $c_j \in [1, q]$, where q is the total number of residents ranked by hospital $h_j \in \mathcal{H}$. Our experimental results show that when $p_1 = 0.8$, HR finds 98% of perfect matchings for $p_2 \in \{0.0, 0.2, 0.3\}$ and 99% of perfect matchings for $p_2 \in \{0.4, 0.8\}$. For the remaining values of p_1 and p_2, HR finds 100% of perfect matchings. Figure 2(a) shows the average capacity in generated instances. For each $p_1 \in [0.1, 0.8]$, the average capacity of hospitals is about $0.5n(1-p_1)$ residents (i.e. from 450 residents to 100 residents). When p_2 increases from 0.0 to 1.0, the average capacity of hospitals remains unchanged. When $p_1 = 0.8$, meaning that h_j has the smallest capacity c_j, and therefore some instances may have no perfect matchings and HR cannot find perfect matchings for these instances. Figure 2(b) shows the average number of iterations used by HR. When p_1 increases from 0.1 to 0.8, the number of iterations used by HR slightly decreases. When p_2 increases from 0.0 to 0.9, the number of iterations used by HR increases. However, when $p_2 = 1.0$, the number of iterations used by HR decreases rapidly because the probability of ties is 100%, meaning that the ranks of hospitals in residents' preference lists are the same. Therefore, HR only considers the first accepted hospital instead of all hospitals in order to find an UBP from the resident's point of view. We can see that although the generated instances have large sizes, HR used a small number of iterations, about 40 to 100, to find perfect matchings.



(a) Average capacity (b) Average number of iterations

Fig. 2. Average capacity of instances and average number of iterations used by HR for $n = 1000$ and $m = 50$

Experiment 3. In this experiment, we chose $n \in [100, 1000]$ with step 100, $m \in [10, 50]$ with step 5, $p_1 = 0.5$, and $p_2 = 0.5$. For each combination of parameters (n, m, p_1, p_2), we randomly generated 100 HRT instances, in which the capacity of each hospital is chosen as in Experiment 2. Figure 3(a) shows the percentage of perfect matchings found by HR. We see that when $m \in [20, 50]$, HR always finds 100% of perfect matchings. When $m = 10$ and n increases from 100 to 1000, HR finds about from 85% down to 47% of perfect matchings, respectively, and the number of unassigned residents in stable matchings is about from 1 to 2 unassigned residents as shown in Fig. 3(b). When $m = 15$, HR finds about 98% of perfect matchings for all values of $n \in [100, 1000]$.

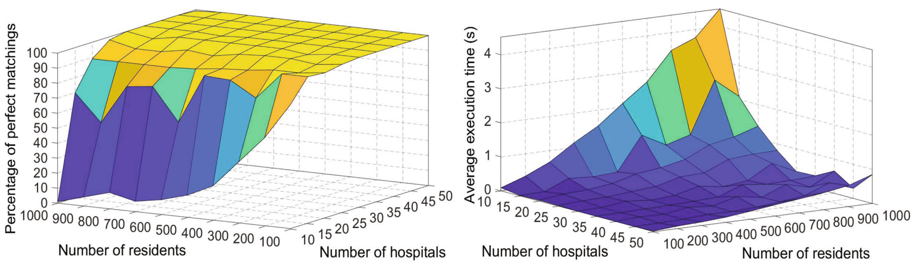


(a) Percentage of perfect matchings (b) Average unassigned residents

Fig. 3. Percentage of perfect matchings and average unassigned residents for $n \in [100, 1000]$ and $m \in [10, 50]$

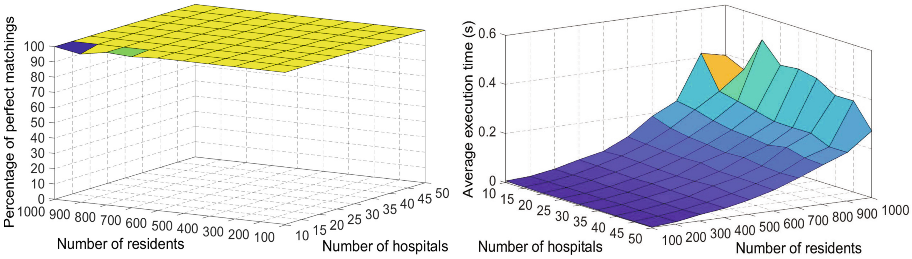
Experiment 4. In this last experiment, we evaluated the effect of capacities of hospitals on perfect matchings found by HR. To do this, we chose the values of n, m, p_1 and p_2 as in Experiment 3. We changed the capacity of each hospital as follows.

First, we considered a popular case, where $c_j = n/m$, meaning that the total capacity of hospitals is equal to the number of residents. The experimental results, depicted in Fig. 4(a), show that HR finds 90% of perfect matchings for $n \in [100, 1000]$ and $m \in [20, 50]$. When $m = 10$, HR finds about from 85% (at $n = 100$) down to 1% (at $n = 1000$) of perfect matchings. Figure 4(b) shows the average execution time found by HR. When m increases from 20 to 50 and n increases from 100 to 1000, the execution time found by HR increases about from 0.01(s) to 1.5(s). However, when $m = 10$ and n increases from 100 to 1000, the execution time found by HR increases about from 0.02(s) to 4.5(s), since the percentage of perfect matchings found by HR decreases, meaning that HR used many iterations to find perfect matchings for generated instances.



(a) Percentage of perfect matchings (b) Average execution time

Fig. 4. Percentage of perfect matchings and average execution time found by HR, where $c_j = n/m$



(a) Percentage of perfect matchings (b) Average execution time

Fig. 5. Percentage of perfect matchings and average execution time found by HR, where $c_j = [0.2q, 0.6q]$

Second, we randomly generated $c_j \in [0.2q, 0.6q]$, where q is the total number of residents ranked by hospital $h_j \in \mathcal{H}$. This means that each hospital ranks about 50% of residents (since $p_1 = 0.5$), but selects only about from 10% to 30% of the total of ranked residents. Figure 5(a) shows that when $(n, m) = (700, 10)$, HR finds 99% of perfect matchings, and when $(n, m) = (900, 10)$, HR finds 97% of perfect matchings. For the remaining values of n and m, HR finds 100% of perfect matchings. In this case, the percentage of perfect matchings found by HR is higher than that when $c_j = n/m$, meaning that the capacity for each hospital strongly affects the solution quality of HRT. Figure 5(b) shows the average execution time found by HR. When n increases from 100 to 1000, the average execution time of HR increases only about from 0.01(s) to 0.4(s). We see that when $n = 1000$ and $m \in [10, 50]$, the execution time of HR is very small, about 0.4(s), meaning that HR is efficient for solving HRT instances of large sizes.

5 Conclusions

In this paper, we proposed a heuristic repair algorithm to solve HRT. The algorithm starts to search a solution of the problem from a random matching. At each iteration, the algorithm finds a set of undominated blocking pairs from the residents' point of view for the matching. Then, the algorithm removes the best undominated blocking pair for each hospital such that it does not only remove many blocking pairs from the residents' of view as possible but also removes as many blocking pairs as possible from the hospitals' point of view. The algorithm repeats until it finds a perfect matching or reaches a maximum number of iterations. Experiments showed that our algorithm is efficient in terms of execution time and solution quality for HRT of large sizes. In the future, we plan to extend this approach to find strongly stable matchings or super-stable matchings for HRT [8,9].

References

1. Canadian resident matching service (CaRMS). http://www.carms.ca/
2. 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 (June 2014)
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 (August 2010)
5. Gelain, M., Pini, M.S., Rossi, F., Venable, K.B., Walsh, T.: Local search approaches in stable matching problems. Algorithms **6**(4), 591–617 (2013)
6. 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 (July 2002)
7. Irving, R.W.: Matching medical students to pairs of hospitals: A new variation on a well-known theme. In: Proceedings of ESA 1998: the 6th Annual European Symposium, Venice, Italy, pp. 381–392 (August 1998)
8. 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 (July 2000)
9. 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
10. Iwama, K., Miyazaki, S., Morita, Y., Manlove, D.: Stable marriage with incomplete lists and ties. In: Proceedings of International Colloquium on Automata, Languages, and Programming, Prague, Czech Republic, pp. 443–452 (July 1999)
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: Proceedings of the International Conference on Operations Research, pp. 263–269. Erasmus University Rotterdam (September 2013)
13. Manlove, D.: Algorithmics of Matching Under Preferences, vol. 2. World Scientific (2013)
14. Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theoret. Comput. Sci. **276**(1–2), 261–279 (2002)
15. Manlove, D.F., McBride, I., Trimble, J.: "Almost-stable" matchings in the hospitals / residents problem with couples. Constraints **22**(1), 50–72 (2017)
16. 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, UK (April 2015)
17. Munera, D., Diaz, D., Abreu, S., Rossi, F., Saraswat, V., Codognet, P.: Solving hard stable matching problems via local search and cooperative parallelization. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, pp. 1212–1218 (January 2015)

18. Roth, A.E.: The evolution of the labor market for medical interns and residents: A case study in game theory. J. Polit. Econ. **92**(6), 991–1016 (1984)
19. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River (2009)