A min-conflicts algorithm for maximum stable matchings of the hospitals/residents problem with ties

Nguyen Thi Uyen
Dept. of Information Technology
Vinh University
Vinh City, Vietnam
uyennt@vinhuni.edu.vn

Nguyen Long Giang
Institute of Information Technology
Vietnam Academy of Science and Technology
Hanoi, Vietnam
nlgiang@ioit.ac.vn

Truong-Thang Nguyen
Institute of Information Technology
Vietnam Academy of Science and Technology
Hanoi, Vietnam
ntthang@ioit.ac.vn

Hoang Huu Viet(✉)
Dept.of Information Technology
Vinh University
Vinh City, Vietnam
viethh@vinhuni.edu.vn

***Abstract*—This paper presents a min-conflicts algorithm to find a maximum weakly stable matching for the Hospitals/Residents problem with Ties (MAX-HRT). We represent the problem in terms of a constraint satisfaction problem and apply a local search approach to solve the problem. Our key idea is to find a set of undominated blocking pairs from residents' point of view and then remove the best one to not only reject all the blocking pairs formed by the residents but also to reject as many as possible the blocking pairs formed by hospitals from the hospitals' point of view. Experiments show that our algorithm is efficient for solving MAX-HRT of large sizes.**

***Index Terms*—Heuristics. Hospitals/Residents. Min-Conflicts. Local Search. Undominated Blocking Pairs.**

I. Introduction

The Hospitals/Residents problem (HR) was first introduced by Gale and Shapley in 1962 under the name "*College Admissions Problem*" [1]. Recently, HR has been attracting much attention from the research community due to its important role in a wide range of real-life applications such as the National Resident Matching Program (NRMP) in the US [2], the Scottish Pre-registration house officer Allocations (SPA) matching scheme [3], or the Canadian Resident Matching Service (CaRMS) in Canada [4].

An instance I of HR [5], [6] consists of a set $R = \{r_1, r_2, \cdots, r_n\}$ of *residents* and a set $H = \{h_1, h_2, \cdots, h_m\}$ of *hospitals* in which each resident $r_i \in R$ ranks in strict order a subset of H in its *preference list*, each hospital $h_j \in H$ ranks in strict order applicants in its preference list, and each hospital $h_j \in H$ has a *capacity* $c_j \in \mathbb{Z}^+$ indicating the maximum number of residents that can be assigned to it. A pair $(r_i, h_j) \in R \times H$ is said to find *acceptable* each other if h_j appears in r_i's preference list, and vice versa. A *matching* M is a set of pairs $(r_i, h_j) \in R \times H$ such that (*i*) r_i and h_j find each other acceptable, (*ii*) r_i is assigned to at most one hospital in M, and (*iii*) h_j is assigned to at most c_j residents in M. We denote $(r_i, h_j) \in M$ as $M(r_i) = h_j$ and $M(h_j)$ as the set of residents assigned to h_j. A hospital $h_j \in H$ is *under-subscribed*, *full*, or *over-subscribed* if $|M(h_j)| < c_j$, $|M(h_j)| = c_j$, or $|M(h_j)| > c_j$, respectively. A pair $(r_i, h_j) \in R \times H$ is a *blocking pair* for M if (*i*) r_i and h_j find each other acceptable, (*ii*) r_i either is *unassigned* or *prefers* h_j to $M(r_i)$, and (*iii*) h_j either is *under-subscribed* or *prefers* r_i to the worst resident in $M(h_j)$. A matching M is said to be *stable* if it has no blocking pairs. Given an instance I of HR, the problem is to find a stable matching in I.

From the aspect of practical applications, HR is unrealistic because both hospitals and residents are required to rank applicants in strict order of preference. Therefore, several extensions of HR have been proposed [6], [7], [8]. The most popular extension of HR is the Hospitals/Residents problem with Ties (HRT) [5], [6], in which the preference lists of both residents and hospitals are allowed to contain ties. Accordingly, three definitions of matching are given, consisting of *weak stability*, *strong stability*, and *super-stability*. Given an instance I of HRT, it is known that weakly stable matchings may have different sizes and therefore, in order to every resident can be assigned to a hospital, it is natural to find a matching which is not only stable but also with maximum size, i.e. the problem of finding a weakly stable matching with maximum size. This problem is known as MAX-HRT and shown to be NP-hard, even if each $h_j \in H$ has $c_j = 1$ [9], [10]. Manlove et al. [11] proved that the size of the largest stable matching is at most twice the size of the smallest for any HRT instance. Kwanashie et al. [12] proposed an integer programming approach to solve HRT. Munera et al. [13] converted HRT to the stable matching with ties and incomplete lists problem (SMTI) and applied the adaptive search [14] to solve MAX-HRT.

In this paper, we present a min-conflicts algorithm, called MCA, to solve MAX-HRT. Given an instance I of HRT, MCA starts to search a solution of I from a random matching. At each iteration, MCA finds a set of undominated blocking pairs (UBP(s)) for the matching from the residents' point of view, selects a resident who is most preferred by hospitals in UBPs. However, to avoid getting stuck in a local minimum, MCA chooses a random resident in UBPs in a small probability. Then, it removes the pair of (resident, hospital) and repeats for the matching. The experiments show that our algorithm is efficient in terms of execution time and solution quality for HRT of large sizes.

The rest of this paper is organized as follows. Section 2 presents definitions, Section 3 presents our MCA algorithm, Section 4 discusses the experimental results, and Section 5 concludes our work.

II. Definitions

Given an instance I and a matching M of HRT, the main definitions for HRT is as follows [5], [6], [15].

Definition 1 (blocking pair): A pair $(r_i, h_j) \in R \times H$ is a *blocking pair* for M if (*i*) r_i and h_j find each other acceptable, (*ii*) r_i either is unassigned or *strictly* prefers h_j to $M(r_i)$, and (*iii*) h_j either is under-subscribed or *strictly* prefers r_i to the worst resident in $M(h_j)$.

It should be noted that the definition of blocking pair in HRT is different from that in HR at the property "*strictly prefer*" of r_i and h_j instead of "*prefer*".

Definition 2 (weakly stable matching): A matching M is weakly stable if it has no blocking pairs, otherwise, it is said to be *unstable*.

Definition 3 (matching size): The size of a weakly stable matching M is the number of residents assigned to hospitals in M. If the size of M is n, then M is called a *perfect matching*.

For a simple implementation of our algorithm, we assume that a matching M can include residents $r_i \in R$ such that r_i is unsigned to hospitals or $M(r_i) = \emptyset$.

Definition 4 (dominated blocking pair): A blocking pair (r_i, h_j) dominates a blocking pair (r_i, h_k) from the residents' point of view if r_i prefers h_j to h_k.

Definition 5 (undominated blocking pair): A blocking pair (r_i, h_j) is called an undominated blocking pair if there exists no other blocking pair that dominates it from the residents' point of view.

The concept of the undominated blocking pair was given in [15] and then it was applied to solve efficiently SMTI [15], [16]. This is because removing an undominated blocking pair results in removing all blocking pairs that are dominated by the undominated blocking pair.

Table I shows an instance of HRT of 8 residents and 5 hospitals. In residents' preference lists, for example, the notation r_2: h_1 (h_4 h_5) h_3 means r_2 strictly prefers h_1 to h_4 and h_5, which are equally preferred. In the hospitals' preference lists, for example, h_2: (3) means $c_2 = 3$. The

TABLE I
Preference lists of eight residents and five hospitals

Resident's preference list	Hospital's preference list
r_1: h_1 h_3 h_2	h_1: (2): r_3 (r_7 r_5 r_2) r_4 r_6 r_1
r_2: h_1 (h_5 h_4) h_3	h_2: (3): r_5 r_6 (r_3 r_4) r_1
r_3: h_1 h_5 h_2	h_3: (1): (r_5 r_2) r_6 r_1 r_7
r_4: h_1 (h_2 h_4)	h_4: (1): r_8 r_2 r_4 r_7
r_5: h_3 h_1 h_2	h_5: (1): r_3 (r_7 r_6 r_8) h_2
r_6: (h_3 h_2) h_1 h_5	
r_7: h_3 h_4 h_5 h_1	
r_8: h_5 h_4	

matching M = {(1,0), (2,5), (3,1), (4,4), (5,0), (6,1), (7,3), (8,0)} is unstable matching because it contains 10 blocking pairs {(1,2), (2,1), (4,1), (5,1), (5,2), (5,3), (6,2), (6,3), (8,4), (8,5)}. The matching M = {(1,0), (2,1), (3,1), (4,2), (5,3), (6,2), (7,5), (8,4)} is weakly stable because it has no any blocking pairs, where resident r_1 is *unassigned* and hospital h_2 is *under-subscribed*. However, the matching M = {(1,2), (2,1), (3,1), (4,2), (5,3), (6,2), (7,5), (8,4)} is perfect since its size is 8. The blocking pair $(5, 3)$ dominates the blocking pair $(5, 1)$ from the residents' point of view and the blocking pair $(5, 1)$ is undominated since there exists no blocking pairs dominating it from the residents' point of view.

III. Algorithm

In this section, we propose a min-conflicts algorithm, called MCA, to find a maximum weakly stable matching for HRT. We represent HRT as a constraint satisfaction problem (CSP), in which residents are variables, hospitals ranked in each resident's preference list form the domain of each variable and constraints are conditions of the blocking pair definition. Accordingly, a stable matching is an assignment of hospitals to residents such that it does not violate constraints. Our key idea is that we apply a local search algorithm for the CSP [17], in which at each search step we select a hospital to assign for a resident that results in the minimum number of blocking pairs.

MCA is shown in Algorithm 1. Initially, the algorithm generates a random matching M and assigns the best matching M_{best} to M. At each iteration, the algorithm calls the function depicted in Algorithm 2 to find the cost, $f(M)$, of M and the set of UBPs, $X = \{(r_i, h_j) \in R \times H\}$, for M. If X is empty, i.e. M is stable, the algorithm checks if $f(M_{best})$ is larger than $f(M)$, then it assigns M_{best} to M. If M is not perfect, i.e. $f(M) > 0$, the algorithm restarts a new matching M and continues the next iteration. Then, the algorithm checks if a small probability of p is accepted, it chooses a random resident $r_j \in X$. Otherwise, it chooses the resident, $r_j \in X$, such that it is most preferred by hospitals $h_k \in X$, i.e. the rank of r_j in h_k's rank list, denoted by $r(h_k, r_j)$, is smallest. After taking a resident, r_j, the algorithm removes the blocking pair $(r_j, X(r_j))$ to obtain a new matching, where $X(r_j)$ is the hospital making an UBP with r_j. 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

Algorithm 1: Min-Conflicts Algorithm

```
Input: - An HRT instance $I$ of size $n \times m$.
       - A small probability $p$.
       - The maximum iterations $max\_iters$.
Output: A matching $M$.
1.  function Main($I$)
2.      $M$ := a randomly generated matching;
3.      $M_{best} := M$;
4.      $f_{best} := n$;
5.      $iter := 0$;
6.      while ($iter \leq max\_iters$) do
7.          $iter := iter + 1$;
8.          $[f, X]$ := Find_Cost_And_UBPs($M$);
9.          if ($X = \emptyset$) then
10.             if ($f_{best} > f$) then
11.                 $M_{best} := M$;
12.                 $f_{best} := f$;
13.             end
14.             if ($f > 0$) then
15.                 $M$ := a randomly generated matching;
16.                 continue;
17.             else
18.                 break;
19.             end
20.         end
21.         if (a small probability of $p$) then
22.             $r_j$ := a random resident $r_i \in X$;
23.         else
24.             $r_j := \text{argmin}(r(h_k, r_i)), \forall (r_i, h_k) \in X$;
25.         end
26.         $M$ := removing bloking pair $(r_j, X(r_j))$;
27.     end
28.     return $M_{best}$;
29. end function
```

either a maximum stable matching or an unstable matching. It should be noted that removing a blocking pair (r_i, h_j) for M results in a matching M' in which h_j is assigned to r_i and if $|M(h_j)| = c_j$ then the worst resident in $M(h_j)$ becomes unassigned, and the other pairs in M are unchanged.

The function to find both the cost $f(M)$ and a set of UBPs for M is shown in Algorithm 2. We define the cost $f(M) = \#nbp(M) + \#nur(M)$, where $\#nbp(M)$ is the number of UBPs for M and $\#nur(M)$ is the number of unassigned residents in M which is not in any UBPs. The function runs as follows. For each $r_i \in R$, the function sorts the r_i's rank list in ascending order. Then, it considers each hospital h_j in r_i's rank list, denoted by $r(r_i, h_j)$, such that r_i strictly prefers h_j to $M(r_i)$ i.e. $r(r_i, h_j) < r(r, M(r_i))$. If (r_i, h_j) is a blocking pair, then (r_i, h_j) is an UBP and the function increases $\#nbp$ and adds the pair (r_i, h_j) to a set of UBPs, X. Otherwise, if every h_j does not form a blocking pair with r_i and r_i is unassigned to any hospitals, the function increases $\#nur$ and performs iteratively for the next resident.

Algorithm 2: Find the cost and UBPs of a matching

```
Input: A matching $M$.
Output: The cost, $f(M)$, and the set of UBPs, $X$.
1.  function Find_Cost_And_UBPs($M$)
2.      $X := \emptyset$;
3.      $\#nur := 0$;
4.      $\#nbp := 0$;
5.      for (each $r_i \in R$) do
6.          $ubp := false$;
7.          sort $r_i$'s rank list in ascending order;
8.          for (each $h_j$ that $r(r_i, h_j) < r(r_i, M(r_i))$) do
9.              if (($r_i, h_j$) is a blocking pair) then
10.                 $X := X \cup (r_i, h_j)$;
11.                 $\#nbp := \#nbp + 1$;
12.                 $ubp := true$;
13.                 break;
14.             end
15.         end
16.         if (($ubp = false$) and ($r_i$ is unassigned)) then
17.             $\#nur := \#nur + 1$;
18.         end
19.     end
20.     $f := \#nbp + \#nur$;
21.     return $(f, X)$;
22. end function
```

IV. Experiments

In this section, we present the experiments to evaluate the efficiency of our MCA. To do so, we adapted the SMTI generator given by Gent et al. [18] to generate HRT instances with four 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. The capacity of each hospital is generated randomly such that it is smaller than or equal to the number of residents in its preference lists. It should be noted that, without loss of generality, our HRT generator created preference lists such that if a hospital does not appear in a resident's preference list, then the resident does not appear in the hospital's preference list. All the experiments were implemented by Matlab software on a personal computer with a Core i7-8550U CPU 1.8GHz and 16 GB memory.

A. *Comparison with* LTIU

This section presents experimental results to compare the execution time and solution quality of MCA with those of LTIU [15]. To run experiments, we randomly generated HRT instances of parameters (n, m, p_1, p_2) by letting $n = 50$, $m = 10$, p_1 vary in $[0.1, 0.8]$ with step 0.1 and p_2 vary in $[0.0, 1.0]$ with step 0.1. For each (n, m, p_1, p_2), we ran 50 HRT instances and averaged the results. The probability and the maximum number of iterations in both MCA and LTIU were 0.03 and 2000, respectively.

Figure 1 shows the average execution time of MCA and LTIU for finding maximum stable matchings of the instances.

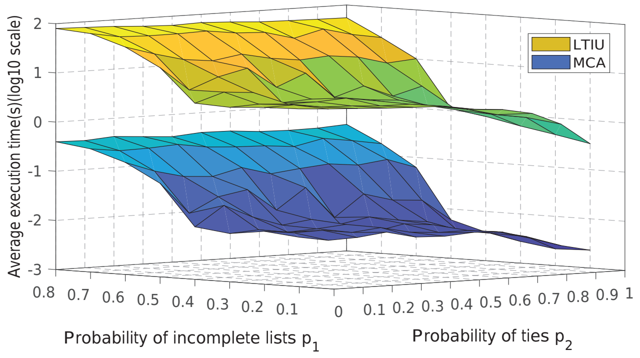

Fig. 1. MCA vs. LTIU - execution time

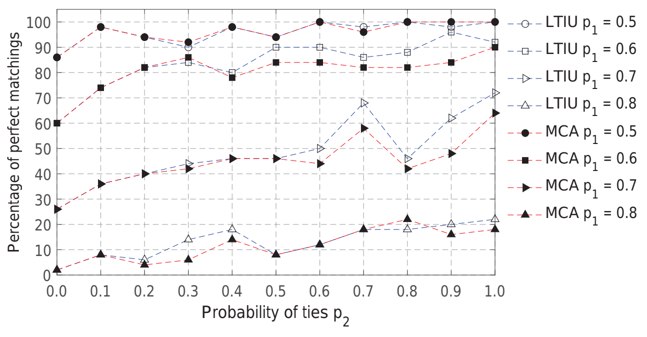

Fig. 2. MCA vs. LTIU - the percentage of perfect matchings

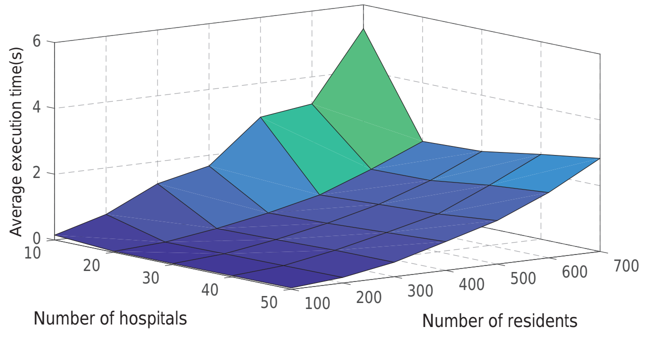

Fig. 3. The average execution time

The average execution time of MCA is much smaller than that of LTIU (thus, we use a log10 scale on Y-axis). When p_1 varies from 0.1 to 0.8, the execution time of both MCA and LTIU increases significantly for any value of p_2. On average, the execution time of MCA increases from about 0.007(s) to 0.37(s), while that of LTIU increases from about 2.55(s) to 65.80(s) for any value of p_2. In contrast, when p_2 varies from 0.0 to 1.0, the execution time of both MCA and LTIU decreases slightly for any value of p_1. The experimental results also show that MCA runs about 190 times faster than LTIU for any p_1 and p_2. This can be explained as follows. LTIU is a local search algorithm and therefore, at each iteration it has to find a set of neighbor matchings of the current matching, compute the cost of all the neighbors and move the current matching to the best one in the set of neighbors. Although LTIU considers only UBPs, the number of such UBPs is very large, i.e. the number of neighbors 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 LTIU. However, MCA finds the set of UBPs and removes only one in the set of UBPs based on the min-conflicts heuristic to generate a new matching for the next iteration and therefore, MCA runs much faster than LTIU.

Figure 2 shows the percentage of perfect matchings found by MCA and LTIU algorithms (when p_1 varies from 0.1 to 0.5, both MCA and LTIU always find 100% of perfect matchings). When p_1 varies from 0.6 to 0.8, the percentage of perfect matchings found by LTIU is slightly higher than that found by MCA. This is because at each iteration, LTIU moves the current matching to the best one in terms of the cost function in the set of neighbors, while MCA does not do so such that it decreases its execution time. The experimental results also show that when p_1 is large (i.e. $p_1 = 0.7$ and $p_1 = 0.8$), both MCA and LTIU find only a small percentage of perfect matchings since the preference lists of residents and hospitals are sparse.

B. *Experiments for* HRT *of large sizes*

This section presents experimental results for instances of HRT of large sizes to consider the behavior of MCA. To perform experiments, we randomly generated HRT instances of parameters (n, m, p_1, p_2), by letting n vary from 100 to 700 with step 100, m vary from 10 to 50 with step 10, $p_1 = 0.5, p_2 = 0.5$ and the capacity of hospitals be generated randomly. For each (n, m, p_1, p_2), we ran 50 HRT instances and averaged the results. The probability and the maximum number of iterations in MCA were 0.03 and 2000, respectively.

Figure 3 shows the average execution time found by MCA. When $m = 10$ the execution time of MCA increases from about 0.15(s) to 5.25(s) for n varying from 100 to 700, respectively. For the other values of m, the execution time increases from about 0.03(s) to 2.25(s). Fig. 4 shows the percentage of perfect matchings. MCA finds 100% of stable matchings, and furthermore, it finds 100% of perfect matchings for every n and $m \neq 10$. However, when $m = 10$ MCA finds only 86%, 74%, 64%, 66%, 58%, 62% and 44% of perfect matchings for $n = 100, 200, 300, 400, 500, 600$ and 700, respectively. This means that the percentage of perfect matchings decreases when $m = 10$ but n increases. When MCA cannot finds 100% of perfect matchings, meaning that there exist some matchings which are stable but not perfect and therefore, it has to reach the maximum number of iterations. This leads to that MCA increases the execution time when $m = 10$, however, even so, the experimental results show that MCA is efficient for HRT instances of large sizes.

Figure 5 shows the average number of iterations found by MCA. As we mentioned above, when $m = 10$, MCA has to reach the maximum number of iterations to find perfect

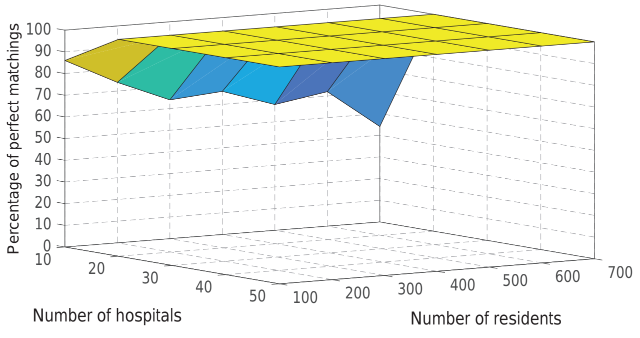

Fig. 4. The percentage of perfect matchings, where c_j is generated randomly.

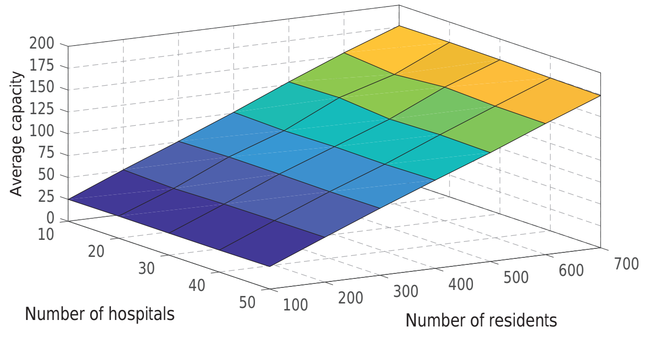

Fig. 6. The average capacity

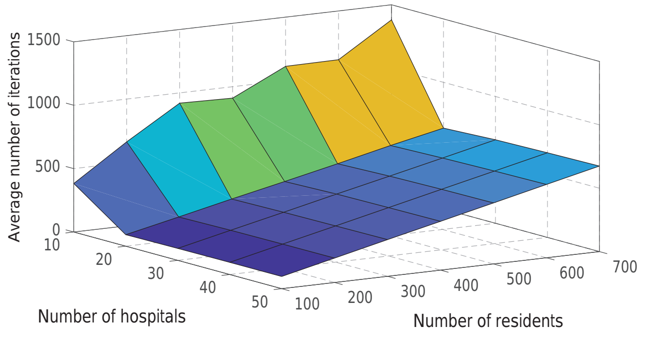

Fig. 5. The average number of iterations

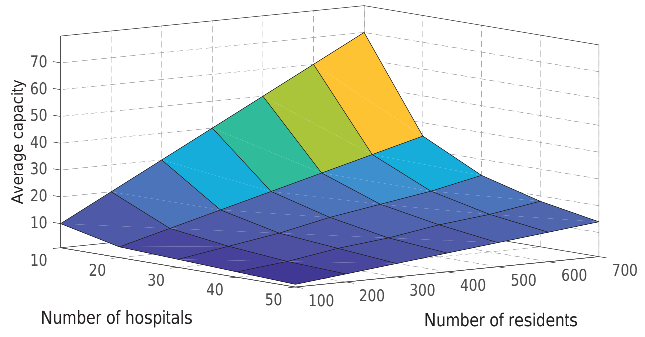

Fig. 7. The capacity of hospitals: $c_j = n/m$

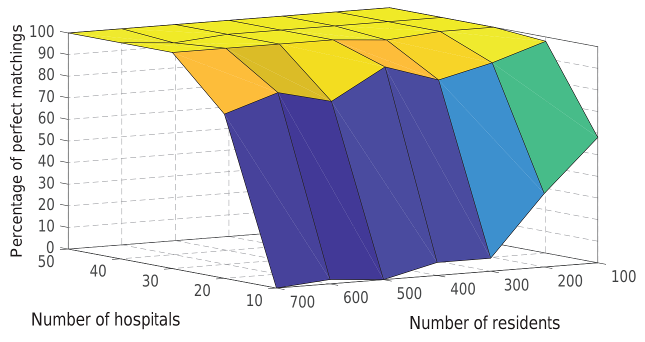

Fig. 8. The percentage of perfect matchings, where $c_j = n/m$.

matchings and therefore, the average number of iterations is bigger than that compared to the other values of m. However, even if $n = 700$, the maximum number of iterations cannot exceed 1500. This means that MCA is efficient in terms of execution time.

Next, we consider the capacity of hospitals randomly generated in the instances. Fig. 6 shows the average capacity of hospitals. It can be seen that the average capacity of hospitals for any value of n is about $n/4$, i.e. 25, 50, 75, 100, 125, 150, 175 for $n = 100, 200, 300, 400, 500, 600, 700$, respectively. This can be explained as follows. In the method to generate HRT instances [18], the number of hospitals in the residents' preference lists as well as the number of residents in the hospitals' preference lists depends on the probability of incompleteness p_1. In our experiments, we let $p_1 = 0.5$, meaning that there exist 50% (i.e. $n/2$) of residents appearing in each hospital's preference list. Moreover, the capacity of hospitals is generated random uniform of $n/2$ residents appearing in each hospital's preference list. Therefore, when many HRT instances generated for experiments, the average capacity of hospitals converges to $n/4$ and therefore, the total capacity of all hospitals is about $m \times n/4$.

Finally, we consider a popular case, where the capacity of h_j is the average of n/m, i.e. $c_j = n/m$. We generated 50 HRT instances for each (n, m, p_1, p_2) as the above experiment and averaged the results. Fig. 7 shows the capacity of hospitals. It can be seen that the capacity of each hospital is smaller 2.5 times than the average capacities generated randomly shown in Fig. 6. Fig. 8 shows the percentage of perfect matchings found by this experiment. MCA cannot find 100% of perfect matchings for $m = 10$ or $m = 20$. Again, this experiment shows that MCA is difficult to find 100% of perfect matchings when m is small but n is large.

V. Conclusions

In this paper, we proposed a min-conflicts algorithm, named MCA, based on a local search approach to solve MAX-HRT. Starting from a randomly generated matching, MCA finds a resident in the set of undominated blocking pairs for the matching such that he is most preferred by hospitals. Then, it removes the pair formed by the resident and the hospital and repeats for the matching until it is stable. Experiments showed that MCA 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 variants of HR such as the Hospitals/Residents problem with Couples or the Hospitals/Residents problem with Free Pairs.

References

[1] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 9, no. 1, pp. 9–15, 1962.

[2] A. E. Roth, "The evolution of the labor market for medical interns and residents: A case study in game theory," *Journal of Political Economy*, vol. 92, no. 6, pp. 991–1016, 1984.

[3] R. W. Irving, "Matching medical students to pairs of hospitals: A new variation on a well-known theme," in *in Proceedings of ESA 1998: the 6th Annual European Symposium*, Venice, Italy, Aug. 1998, pp. 381–392.

[4] "Canadian resident matching service (CaRMS)," http://www.carms.ca/.

[5] R. W. Irving, D. F. Manlove, and S. Scott, "The hospitals/residents problem with ties," in *in Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, Bergen, Norway, Jul. 2000, pp. 259–271.

[6] D. F. Manlove, *The Hospitals/Residents Problem. In: Kao MY. (eds) Encyclopedia of Algorithms*. Boston, MA: Springer, 2008.

[7] P. Biró, D. F. Manlove, and I. McBride, "The hospitals/residents problem with couples: Complexity and integer programming models," in *in Proceeding of SEA 2014: 13th International Symposium on Experimental Algorithms*, Copenhagen, Denmark, Jun. 2014, pp. 10–21.

[8] D. F. Manlove, I. McBride, and J. Trimble, ""almost-stable" matchings in the hospitals / residents problem with couples," *Constraints*, vol. 22, no. 1, pp. 50–72, 2017.

[9] K. Iwama, S. Miyazaki, Y. Morita, and D. Manlove, "Stable marriage with incomplete lists and ties," in *in Proceedings of ICALP 1999: the 26th International Colloquium on Automata, Languages, and Programming*. Springer, 1999, pp. 443–452.

[10] R. W.Irving, D. F.Manlove, and S. Scott, "The stable marriage problem with master preference lists," *Discrete Applied Mathematics*, vol. 156, no. 15, pp. 2959–2977, 2008.

[11] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita, "Hardvariants of stable marriage," *Theoretical Computer Science*, vol. 276, no. 1, pp. 261–279, 2002.

[12] A. Kwanashie and D. F. Manlove, "An integer programming approach to the hospitals/residents problem with ties," in *Proceedings of the International Conference on Operations Research*, Erasmus University Rotterdam, Sep. 2013, pp. 263–269.

[13] D. Munera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and P. Codognet, "A local search algorithm for smti and its extension to hrt problems," in *Proceedings of the 3rd International Workshop on Matching Under Preferences*, University of Glasgow, UK, Apr. 2015, pp. 66–77.

[14] P. Codognet and D. Diaz, "Yet another local search method for constraint solving," in *Proceedings of the International Symposium on Stochastic Algorithms*, Berlin, Germany, Dec. 2001, pp. 73–90.

[15] M. Gelain, M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh, "Local search for stable marriage problems with ties and incomplete lists," in *Proceedings of 11th Pacific Rim International Conference on Artificial Intelligence*, Daegu, Korea, Aug. 2010, pp. 64–75.

[16] D. Munera, D. Diaz, S. Abreu, F. Rossi, V. Saraswat, and P. Codognet, "Solving hard stable matching problems via local search and cooperative parallelization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, Texas, Jan. 2015, pp. 1212–1218.

[17] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[18] I. P. Gent and P. Prosser, "An empirical study of the stable marriage problem with ties and incomplete lists," in *in Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, France, Jul. 2002, pp. 141–145.