B-02

Finding Witnesses for Stability in the Hospitals/Residents Problem

Minseon Lee†　　Shuichi Miyazaki‡　　Kazuo Iwama†

Abstract. An instance I of the Hospitals/Residents problem (HR) comprises a set of residents (graduating medical students) and a set of hospitals, where each hospital has a given capacity. The residents have preferences for the hospitals, as do hospitals for residents. A solution of I is a stable matching, which is an assignment of residents to hospitals that respects the capacity constraints and preference lists in a precise way. In this paper, we consider the problem of deciding, given residents' preference lists and a matching, whether there are hospital's preference lists that make a given matching stable. We show that the problem is NP-complete, if we restrict the number of kinds of hospital's preference lists. We also present one exact algorithm and four greedy algorithms. We implement these algorithms and present a computational study using random instances.

1　Introduction

In 1962 David Gale and Lloyd Shapley published their paper 'College admissions and the stability of marriage' [1], which was the first to formally define the Stable Marriage problem (SM), and they provide its solution. An instance of SM involves a set of men and a set of women. Each person ranks all members of the opposite sex in strict order of his/her preference. We seek to find a stable matching between the men and the women. For a matching between men and women, a pair of a man and a woman is called a blocking pair if both prefer each other to their current partners. A matching with no blocking pair is called stable. Gale and Shapley showed that every instance admits at least one stable matching, and proposed a polynomial time algorithm to find one, which is known as the Gale-Shapley algorithm.

In the same paper [1], Gale and Shapley introduced a many-to-one generalization of the Stable Marriage problem called the Hospitals/Residents problem (HR), referred to by the authors as the College Admissions problem. An instance of HR has a set of residents (i.e. graduating medical students) and a set of hospitals. Each resident ranks in order of preference a subset of the hospitals. Each hospital has an integral capacity, and ranks in order of preference those residents who ranked it. The capacity of a hospital is its number of available posts that it requires to fill. We seek to match each resident to an acceptable hospital, in such a way that a hospital's capacity is never exceeded. Moreover the matching must be stable. A pair of a resident r and a hospital h is said to block a matching M for an instance of HR, and is called a blocking pair, when all the following conditions are satisfied: (i) h and r find each other acceptable; (ii) either r is unmatched, or prefers h to his assigned hospital; (iii) either h is under-subscribed, or h prefers r to its worst resident assigned. A matching is stable if it admits no blocking pair. Similarly to SM, Gale and Shapley described a linear-time algorithm for finding a stable matching, given an instance of HR [3].

To automate the process of assigning residents to hospitals, a number of countries use a centralized matching scheme that has at its heart an algorithm for HR and its variants [6]. For example, the US has the one of the largest automated schemes, namely the National Resident Matching Program (NRMP) [9]. The NRMP has been in operation since 1952 and allocates around 31,000 residents to hospital posts every year. Counterparts of the NRMP elsewhere are the Canadian Resident Matching Service (CaRMS) [7] and the Scottish Foundation Allocation Scheme (SFAS) [10]. And in Japan, the Japan Residency Matching Program (JRMP) has been in operation since 2003 [8]. Similar matching schemes are also used in educational and vocational contexts.

The matching intermediary such as NRMP, SFAS, and JRMP provides a mechanism for matching residents to hospitals according to the preferences expressed by both parties. The entire matching process is conducted by the matching intermediary under tight security. In general, there are three steps in a matching: Firstly, residents and

† Graduate School of Informatics, Kyoto University

‡ Academic Center for Computing and Media Studies, Kyoto University

hospitals submit their rank order lists directly to the matching intermediary. Each resident submits, in the resident's order of reference, a list of the hospitals where he/she has interviewed or will interview. Each hospital also submits, in its order of preference, a list of those residents who have had its interview. Secondly, the matching intermediary compares those rank order lists against each other, by using a computerized matching algorithm program, and obtains a stable matching. Lastly, the matching intermediary informs the residents and the hospitals of the matching results.

However, there is a potential for the residents to raise a question whether the matching obtained by the matching intermediary is truly stable. Because the hospitals' preference lists are not accessible to the public, the residents cannot check how the hospitals have ranked their applicants. So the residents may doubt whether the hospital's preference lists have been intentionally fixed. For example, when a resident is matched to the hospital listed lowest on his/her list, he/she is not satisfied with the hospital assigned to him/her, but he/she is more likely to find witnesses for stability in the matching. We can solve this problem by finding hospitals' preference lists that make a given matching stable. For SM, Kobayashi and Matsui have previously presented a similar issue and studied it in the context of strategic issue [4]: Given men's complete preference lists and a matching, they consider the problem of finding women's complete preference lists that make a given matching man-optimal stable. However, no research for HR has been considered before.

In this paper, we consider the problem of deciding, given residents' preference lists and a matching, whether there are hospitals' preference lists that make a given matching stable. It is not hard to see that such preference lists always exist, i.e., those in which assigned residents are located at the top of each resident's preference list. In this paper, we consider a restricted variant of the problem where there are only k preference lists and each hospital's preference list is identical to one of them. This reflects the scenario where residents take examinations of k different subjects. According to the results, k preference lists are constructed (possibly, by breaking ties), and each hospital adopts one of them. We call this problem k-SHPL (k-Stable Hospital Preference List problem). We show that 1-SHPL is solvable in polynomial time and for a constant $k \geq 2$, k-SHPL is NP-complete. We also present computational results for 2-SHPL. We provide an exponential-time exact algorithm and four greedy algorithms. These greedy algorithms do not necessarily output a correct answer. We implement these algorithms and compare their computational time and accuracy using random instances.

This paper is organized as follows. In Section 2, we define this problem and present some notations. In Section 3, we give complexity results for k-SHPL. In Section 4, we present one exact algorithm and four greedy algorithms, and present a computational study in Section 5.

2 Preliminaries

Residents' preferences	Hospitals' capacities and preferences
$r_1 : h_2\ h_3\ h_1$	$h_1 : (3) : r_2\ r_3\ r_5\ r_1$
$r_2 : h_2\ h_1$	$h_2 : (3) : r_3\ r_5\ r_1\ r_2\ r_4$
$r_3 : h_3\ h_2\ h_1$	$h_3 : (2) : r_6\ r_4\ r_1\ r_3\ r_5$
$r_4 : h_2\ h_3$	
$r_5 : h_2\ h_1\ h_3$	
$r_6 : h_3$	

Figure 1: HR instance I_1.

Residents' preferences	Matching	Hospitals' capacities
$r_1 : h_2\ h_3\ h_1$	(r_1, h_2)	$h_1 : (3)$
$r_2 : h_2\ h_1$	(r_2, h_1)	$h_2 : (3)$
$r_3 : h_3\ h_2\ h_1$	(r_3, h_1)	$h_3 : (2)$
$r_4 : h_2\ h_3$	(r_5, h_2)	
$r_5 : h_2\ h_1\ h_3$	(r_6, h_3)	
$r_6 : h_3$		

Figure 2: SHPL instance I_2.

We first give a formal definition of HR. An instance of HR is as follows: (i) a set of residents $R = \{r_1, r_2, \ldots, r_n\}$; (ii) a set of hospitals $H = \{h_1, h_2, \ldots, h_m\}$; (iii) a preference list of each $r \in R$, in which r ranks a subset of H in a strict order; (iv) a preference list of each $h \in H$, in which h ranks its applicants, those residents who find that particular hospital acceptable, in a strict order; (v) capacities $c_j (1 \leq j \leq m)$, indicating the number of posts that h_j has. An example of HR instance is shown in Figure 1 (the hospital capacities are indicated in brackets).

We say that a resident r finds a hospital h *acceptable* if r's preference list contains h, and h finds r acceptable if

h's preference list contains r. In instance I_1, r_2 finds both h_1 and h_2 acceptable, and h_1 finds r_2, r_3, r_5 and r_1 acceptable.

A *matching* M is a set of (resident, hospital) pairs $(r,h) \in R \times H$. In a matching M, $(r,h) \in M$ only if r and h find each other acceptable, and no resident is assigned to more than one hospital and no hospital is over-subscribed. If $(r,h) \in M$, we say that r is assigned to h, and h is assigned to r. For any $q \in R \bigcup H$, we denote by $M(q)$ the set of assignees of q in M. When there is no obscurity we use $M(r)$ to denote the single hospital assigned to r in M.

A pair $(r,h) \in R \times H$ is said to *block* a matching M for an instance of HR, and is called a *blocking pair*, if all the following conditions are satisfied: (i) h and r find each other acceptable; (ii) either r is unmatched, or prefers h to $M(r)$; (iii) either h is under-subscribed, or h prefers r to at least one of $M(h)$. A matching is *stable* if it admits no blocking pair.

Let us formally define the Stable Hospital's Preference List problem (SHPL) introduced in Section 1. An instance of SHPL consists of the set R of residents, the set H of hospitals, each resident's preference list, each hospital's capacity, and a matching M between R and H. It asks whether there exits hospital's preference lists that make M stable. k-SHPL is a variant of SHPL in which there are at most k kinds of hospitals' preference lists in a feasible solution.

Problem: Stable Hospital's Preference List problem (SHPL).
Instance: residents, hospitals, each resident's preference list, capacities, and a matching M.
Question: Are there any hospitals' preference lists that make M stable?

Problem: k-Stable Hospital's Preference List problem (k-SHPL).
Instance: An instance of SHPL.
Question: Are there any $\leq k$ kinds of hospitals' preference lists that make M stable?

3 Complexity Results

3.1 Encoding a hospital's preference list in complete bipartite digraph

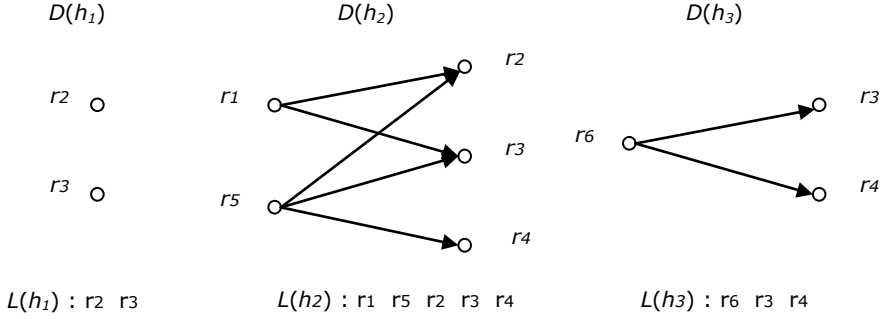


Figure 3: $D(h)$ of SHPL instance I_2

In this section, we show that k-SHPL is equivalent to the problem of partitioning given digraphs into k subsets so that, in each subset, a digraph constructed by taking the union of digraphs in the subset does not contain a directed cycle. To see this, we construct a digraph $D(h)$ corresponding to each hospital h. $D(h)$ is a bipartite graph $D(h) = (L, R, E)$ where vertices of $D(h)$ correspond to residents and L and R are bipartition of the vertices. L includes all the residents in $M(h)$, and R includes all the residents r such that (i) r and h are acceptable to each other and (ii) r is unassigned in M or r prefers h to $M(r)$. Finally, E includes all the arcs from R to L, i.e., $D(h)$ is a complete bipartite graph. Figure 3 illustrates three digraphs corresponding to the hospitals in the instance I_2 given in Figure 2.

We can interpret $D(h)$ in the following way: Let $r_1 \in L$ and $r_2 \in R$. By construction, we know that r_1 is assigned to h but r_2 is not, while r_2 desires to be assigned to h. Therefore, if h prefers r_2 to r_1, then (r_2, h) becomes a

blocking pair for M. The arc $(r_1, r_2) \in E$ stands for this situation, that is, for the matching M to be stable, r_1 must precede r_2 in h's preference list. Hence, if a preference list of h is a linear extension of the vertices of $D(h)$, then h does not create a blocking pair for M. Consider c hospitals $h_1, h_2, \ldots, h_c$ and their corresponding digraphs $D(h_1), D(h_2), \ldots, D(h_c)$. Construct the digraph $D(h_1, h_2, \ldots, h_c)$ by taking the union of edges in $D(h_1), D(h_2), \ldots, D(h_c)$. If $D(h_1, h_2, \ldots, h_c)$ does not contain a cycle, then we can have a linear list following all the edges in $D(h_1, h_2, \ldots, h_c)$. If hospitals $h_1, h_2, \ldots, h_c$ have this list, then there does not exist a blocking pair. On the other hand, if $D(h_1, h_2, \ldots, h_c)$ contains a cycle, there is no linear extension of the vertices of $D(h_1, h_2, \ldots, h_c)$ and hence there is no identical list of all the hospitals $h_1, h_2, \ldots, h_c$ without creating a blocking pair. From the above observation, we can conclude the equivalence we mentioned at the beginning of this section, which will be used in developing algorithms and proving the NP-completeness.

3.2 1-SHPL is Solvable in polynomial time

1-SHPL can be solved in polynomial time. A polynomial time algorithm is as follows: Given an instance of SHPL, we construct a digraph $D(h_j)$ corresponding to each hospital $h_j (1 \le j \le m)$. Since $k = 1$, there is no necessity to partition the digraphs into any subsets. We build one digraph $D(h_1, h_2, \ldots, h_m)$ by taking the union of edges in $D(h_1), D(h_2), \ldots, D(h_m)$. Then we check whether the digraph $D(h_1, h_2, \ldots, h_m)$ contains a cycle. This algorithm can be implemented to run in $O(m^2 n)$ time overall.

Theorem 1. 1-SHPL is solvable in $O(m^2 n)$ time.

3.3 2-SHPL is NP-complete

Theorem 2. 2-SHPL is NP-complete.

Proof (Sketch). Notice that 2-SHPL is in the class NP. Given $L(h)$, we can check whether there is any blocking pair. To show the NP-hardness, we give a reduction from Acyclic 2-Coloring problem known as NP-complete [5]. Given an instance of Acyclic 2-Coloring problem, we construct an instance of 2-SHPL, namely, the same number of residents and hospitals, each resident's preference list, each hospital's capacity, and a matching. We conclude that there are 2 kinds of hospital's preference lists if and only if the graph is acyclic 2-colorable. □

Problem: Acyclic 2-Coloring problem.
Instance: Directed graph $G = (V, A)$.
Question: Can we color the nodes of G with 2 colors such that no monochromatic cycle occurs?

3.4 *k*-SHPL for $k \ge 3$ is NP-complete

Theorem 3. For a positive constant k, k-SHPL is NP-complete.

Proof (Sketch). Notice that k-SHPL is in the class NP. Given $L(h)$, we can check whether there is any blocking pair. To show the NP-hardness, we give a reduction from k-Coloring problem known as NP-complete [2]. Given an instance of k-Coloring problem, we construct an instance of k-SHPL, namely, the same number of residents and hospitals, each resident's preference list, each hospital's capacity, and a matching. We conclude that there are k kinds of hospital's preference lists if and only if the graph is k-colorable. □

Problem: 3-Coloring problem.
Instance: Graph $G = (V, E)$.
Question: Can we color the nodes of G with k colors such that the endpoints of every edge are colored differently?

4 Greedy Algorithms for 2-SHPL

In this section, we describe one exact algorithm and four greedy algorithms for solving 2-SHPL. The exact algorithm is used to determine the accuracy of the others. We present GreedyAlg1 and GreedyAlg2, and then upgrade these algorithms to GreedyAlg3 and GreedyAlg4 respectively.

4.1 Exact Algorithm

A formal description of the exact algorithm is given by ExactAlg.

ExactAlg (I) : O(2^m)	
1.	construct a complete bipartite digraph $D(h)$ for each h
2.	for (each partition of $D(h)$ s into two groups) do
3.	if (neither of the 2 groups has any cycle) then return YES
4.	else goto 2
5.	return NO

4.2 Greedy Algorithms

A formal description of GreedyAlg1 is as follows.

GreedyAlg1 (I) : O(m)	
1.	construct a complete bipartite digraph $D(h)$ for each h
2.	sort $D(h)$ randomly
3.	for (the first $D(h)$ to the last $D(h)$) do
4.	add $D(h)$ to group A and merge
5.	if (group A contains a cycle) then
6.	delete $D(h)$ from group A
7.	add $D(h)$ to group B and merge
8.	if (group B contains a cycle) then return NO
9.	return YES

A formal description of GreedyAlg2 is as follows. The key difference between GreedyAlg2 and GreedyAlg1 is that GreedyAlg2 repeats the main process 10 times while GreedyAlg1 repeats only once.

GreedyAlg2 (I) : O(m)	
1.	construct a complete bipartite digraph $D(h)$ for each h
2.	for (10 times) do
3.	sort $D(h)$ randomly
4.	for (the first $D(h)$ to the last $D(h)$) do
5.	add $D(h)$ to group A and merge
6.	if (group A contains a cycle) then
7.	delete $D(h)$ from group A
8.	add $D(h)$ to group B and merge
9.	if (group B contains a cycle) then goto 2
10.	return YES
11.	return NO

We improve GreedyAlg1 by sorting $D(h)$ in descending order of its outdegree. A formal description of GreedyAlg3 is as follows.

GreedyAlg3 (I) : O(m)	
1.	construct a complete bipartite digraph $D(h)$ for each h
2.	sort $D(h)$ in descending order of its outdegree
3.	for (the first $D(h)$ to the last $D(h)$) do
4.	add $D(h)$ to group A and merge
5.	if (group A contains a cycle) then
6.	delete $D(h)$ from group A
7.	add $D(h)$ to group B and merge
8.	if (group B contains a cycle) then return NO
9.	return YES

We improve GreedyAlg2 by moving $D(h)$, which have caused a cycle in both of the two groups, to the head of the list. A formal description of GreedyAlg4 is as follows.

GreedyAlg4 (I) : O(m)	
1.	construct a complete bipartite digraph $D(h)$ for each h
2.	sort $D(h)$ randomly
3.	for (10 times) do
4.	for (the first $D(h)$ to the last $D(h)$)) do
5.	add $D(h)$ to group A and merge
6.	if (group A contains a cycle) then
7.	delete the $D(h)$ from group A
8.	add the $D(h)$ to group B and merge
9.	if (group B contains a cycle) then
10.	update the sorting by moving $D(h)$ to the head and goto 4
11.	return YES
12.	return NO

Notice that the above greedy algorithms have one-sided error, i.e., if the true answer is "NO" then the greedy algorithms always output "NO", but if the true answer is "YES" they may output the wrong answer. In other words, while a "YES" answer is guaranteed to be accurate, a "NO" answer is uncertain.

5 Computational Experiments

All algorithms have been coded in Java using Eclipse 2013. All the experiments were run on personal computer with Pentium processor with 2.0 GHz clock speed, equipped with Windows 7. Below, we first provide some details on the random instances. Then we subsequently discuss the computational results.

5.1 Instance Generation

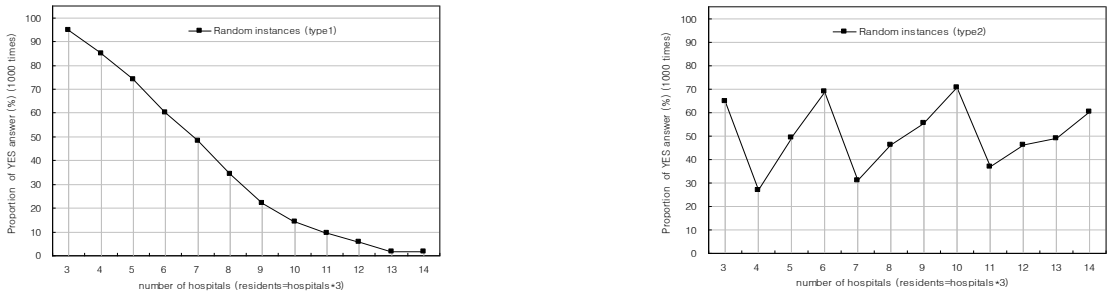
We show how to generate instances of SHPL. An instance of SHPL is given by each resident's preference list, each hospital's capacity, and a matching. In order to generate instances, we first need to decide which data structures we use for all these elements and how we set each element of instance to randomized value. For the convenience of implementation, we assume that all instances generated in this section consist of n hospitals and $3n$ residents.

Firstly, we use $n \times m$ ranking matrix P for residents' preference lists. The n rows and m columns of the matrix P represent n residents and m hospitals respectively. The element of the i^{th} row and j^{th} column, denoted by $P[i, j]$,

indicates the position of hospital h_j in resident r_i's preference list. As each resident's preference list is incomplete list, i.e., a resident is allowed to accept only a subset of the hospitals, we set $P[i, j]$ to a special null symbol if residents r_i and hospital h_j are unacceptable.

Next, to present a matching, we use an array A of length n, where $A[i]$ is the hospital assigned to resident r_i. We set $A[i]$ to a null symbol when we need to indicate that resident r_i is not assigned to any hospital. Similarly, capacities of hospitals can be presented by an array C of length m, where $C[j]$ indicates the capacity of hospital h_j.

We create an instance of SHPL as follows: We first initialize to $P[i, j] = j$. Then, to make each resident's preference list $P[i]$ ordered differently than others, we shuffle randomly each resident's preference list $P[i]$, an array of m hospitals. And we pick a random number s between 2 and $m+1$ (For simplicity, we assume that index of array starts from 1 instead of 0) and set elements $P[i, j](j \geq s)$ to null symbol, so that each resident's preference list $P[i]$ forms an incomplete list of size $s-1$. Also, to make a matching A, we select randomly a hospital j on each resident's preference list $P[i]$, and set $A[i]$ to the hospital j. Finally, we set $H[j]$ to the number of those residents who are assigned to hospital h_j. If an instance made in this way has a hospital h in which $|M(h)| = 0$, we discard the instance and make another one all over again.



(a) Random instances (type1) (b) Random instances (type2)

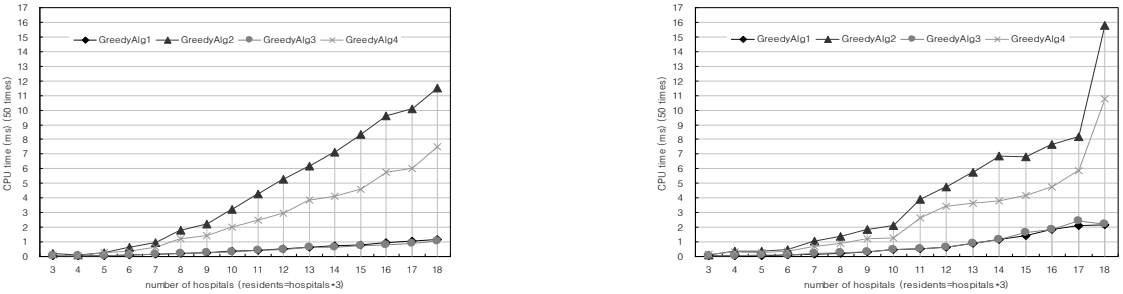
Figure 4: Proportion of YES answer

Figure 4(a) shows the proportion of YES instances in randomly generated 1000 instances in the above manner, according to the number of hospitals. As shown in Figure 4(a), the proportion of YES instances decreases gradually as the number of hospitals ascends stepwise. To generate harder instances, we consider such an instance generator that outputs YES instances and NO instances in the proportion of 50 to 50. The key difference between the random instances (type1) and the random instances (type2) is how to decide the size of each resident's preference lists $L(r)$. In the random instances (type1), we decide randomly the size of each resident's preference list, while in the random instances (type2), we decide by the following formula:

$$L(r) = \left\lceil \frac{\log m + 3\log 5 - 4\log 3}{\log 5 - \log 3} \right\rceil, \ (m = 3,4,5,\ldots) \qquad (1)$$

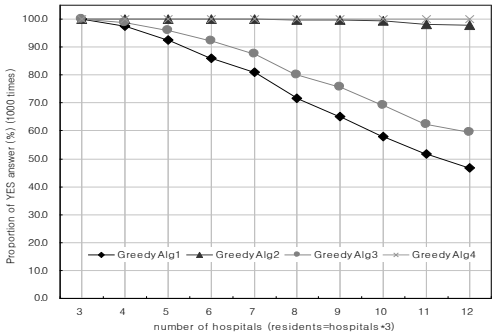
This formula is used to adjust YES instances and NO instances in the proportion of 50:50 as close as possible. We have obtained this formula by observing the relation of $L(r)$'s size and the proportion of YES answer. Figure 4(b) shows the proportion of YES instances of the random instances (type2).

5.2 Computational Results

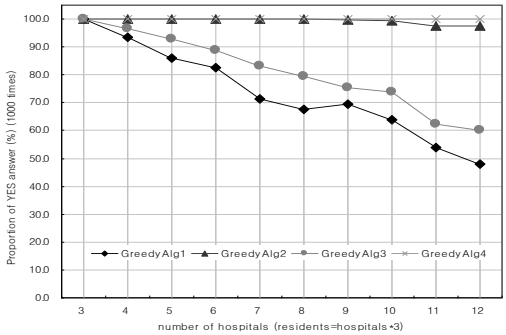


(a) Average CPU time of random instances (type1) (b) Average CPU time of random instances (type2)

Figure 5: The average CPU time of every greedy algorithm



(c) Proportion of correct answer of random instances (type1)



(d) Proportion of correct answer of random instances (type2)

Figure 6: The Proportion of correct answer of every greedy algorithm

We compare the four greedy algorithms based on their CPU time and the proportion of correct answer, on random instances. Figure 5 shows the average CPU time of every greedy algorithm, according to the number of hospitals. As shown in Figure 5, the average CPU time of GreedyAlg2 increases steepest among the four. It is natural that GreedyAlg2 spends more CPU time than GreedyAlg1 and GreedyAlg3, because GreedyAlg2 repeats the main process 10 times while GreedyAlg1 and GreedyAlg3 repeat only one. However, GreedyAlg4 is faster than GreedyAlg3 despite the same number of times. This difference probably arises because GreedyAlg4 has better accuracy than GreedyAlg3. Next, we verify the accuracy of each greedy algorithm. As mentioned earlier, every greedy algorithm has one-sided error, i.e., they may output a wrong answer "NO" when the true answer is "YES". Hence, our purpose of this experiment is to investigate the proportion that each greedy algorithm outputs the correct answer when it is given YES instances. To do this, we generate instances by our generator and check the answer by ExactAlg. We used instances which ExacAlg outputs YES. Figure 6 shows the proportion of correct answer of each greedy algorithm. As shown in Figure 6, The GreedyAlg4 shows the best accuracy, which is very close 100% even in the largest instances. The result from this study allows us to conclude that GreedyAlg4 is the most efficient algorithm of four greedy algorithms when all the things above are taken into consideration.

[References]

[1] D. Gale and L.S. Shapley. College admissions and the stability of marriage. American Mathematical Monthly, 69:9–15, 1962.

[2] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co, 1979.

[3] D. Gusfield and Robert W. Irving. The Stable Marriage Problem: Structure and Algorithms. MIT Press, 1989.

[4] H. Kobayashi and T. Matsui. Successful manipulation in stable marriage model with complete preference lists. IEICE TRANS. INF. & SYST., Vol.E92-D, No.2, pages 116-119, February 2009.

[5] F.T. Nobibony, C.A.J. Hurkensz, R. Leus and F.C.R. Spieksma. Exact algorithms for coloring graphs while avoiding monochromatic cycles. Proc. AAIM 2010, LNCS 6124, pages 229-242, 2010.

[6] A.E. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. Journal of Political Economy, 92(6):991–1016, 1984.

[7] Canadian Resident Matching Service. Web document available at http://www.carms.ca/matching/algorith.htm.

[8] Japan Residency Matching Program. Web document available at http://www.jrmp.jp/aboutmatching.htm.

[9] National Resident Matching Program. Web document available at http://www.nrmp.org/about_nrmp/how.html.

[10] Scottish Foundation Allocation Scheme. Web document available at http://www.nes.scot.nhs.uk/sfas.