Electronic Preprint for Journal of Information Processing Vol.23 No.2

Recommended Paper

Finding Witnesses for Stability in the Hospitals/Residents Problem

Minseon Lee[1,a] Shuichi Miyazaki[2,b] Kazuo Iwama[1,c]

Received: March 2, 2014, Accepted: November 10, 2014

Abstract: The Hospitals/Residents problem is a many-to-one generalization of the well-known Stable Marriage problem. Its instance consists of a set of residents, a set of hospitals, each resident's preference list, each hospital's preference list, and each hospital's capacity (i.e., the number of available positions). It asks to find a stable matching between residents and hospitals. 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. We call this problem Stable Hospital's Preference List problem (SHPL). It is easy to see that there always exists a solution if we allow arbitrary preference lists of hospitals. Considering more suitable situations, we pose a restricted version, called k-SHPL, in which there are only k kinds of preference lists of hospitals. We show that 1-SHPL is solvable in polynomial time, while k-SHPL is NP-complete for any k such that $2 \leq k \leq n^{1-\epsilon}$, where n is the number of residents and ϵ is any positive constant. We also present four heuristics algorithms (first-fit algorithms) for 2-SHPL. We implement these algorithms and present a computational study using random instances.

Keywords: the Hospitals/Residents problem, hospital's preference list, NP-complete, first-fit algorithm

1. Introduction

The *Stable Marriage problem* (*SM*) is a classical assignment problem introduced by Gale and Shapley in 1962 [1]. Its instance consists of a set of men, a set of women, and each person's preference list that orders all the members of the opposite gender according to his/her preference. A *matching* is a one-to-one correspondence between men and women. For a matching M, a (man, woman)-pair is said to be a *blocking pair* if each prefers the other to his/her actual partner in M, and a matching with no blocking pair is a *stable matching*. It is known that every instance has at least one stable matching, and one of them can be found by a polynomial time algorithm, called the Gale-Shapley algorithm or Deferred Acceptance algorithm.

The *Hospitals/Residents problem* (*HR*) is a many-to-one generalization of the stable marriage problem [1]. In HR, men and women correspond to residents and hospitals, and each hospital has a *capacity* or a *quota*, which is an upper bound on the number of available positions it provides. (A formal definition of HR will be given in Section 2.) A straightforward extension of the Gale-Shapley algorithm can find a stable matching also for this generalization [1], [3].

There are many applications of HR in real world assignment systems, such as assigning primary school students to secondary schools and university students to supervisors. Among others, one of the most famous applications is to assign graduating medical students (residents) to hospitals [6], known as the National Resident Matching Program (NRMP) in the US [9], the Canadian Resident Matching Service (CaRMS) in Canada [7], the Scottish Foundation Allocation Scheme (SFAS) in Scotland [10], and the Japan Residency Matching Program (JRMP) in Japan [8].

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 hospitals submit their rank order lists directly to the matching intermediary. Each resident submits, in the resident's order of preference, 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 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 informed 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 residents may doubt whether a coalition of hospitals and the matching intermediary has manip-

[1] Graduate School of Informatics, Kyoto University, Kyoto 606–8501, Japan
[2] Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606–8501, Japan
[a] yiminseon@ai.soc.i.kyoto-u.ac.jp
[b] shuichi@media.kyoto-u.ac.jp Supported by JSPS KAKENHI Grant Number 24500013.
[c] iwama@kuis.kyoto-u.ac.jp Supported by JSPS KAKENHI Grant Number 25240002.

A preliminary version of this paper was presented at the IPSJ Kansai-Branch Convention 2013 held in September 2013. This paper was recommended to be submitted to Journal of Information Processing (JIP) by the Chairman of IPSJ Kansai-Branch.

ulated the result in their favor. For example, when residents are matched to hospitals listed relatively lower rank in their lists, residents are not satisfied with the result and they may collectively try to find witnesses for instability in the matching to claim the invalidness of the result. We can partially solve this problem by checking whether there are hospitals' preference lists that make a given matching stable.

Our contributions: In this paper, we consider the problem of deciding, given residents' preference lists and a matching, whether there are hospitals' preference lists that make the given matching stable. It is not hard to see that such preference lists always exist if we allow arbitrary preference lists of hospitals, i.e., those in which each hospital includes only assigned residents in its preference list. Also, even if we require that each hospital's preference list be complete, namely, it includes all the residents, there still always exists a solution, that is, those in which assigned residents are located at the top of each hospital's preference list. In this paper, we consider more restricted variant where there are only k preference lists, and each hospital's preference list is identical to one of them. We call this problem k-SHPL (k-Stable Hospital's Preference List problem). 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 show that 1-SHPL is solvable in polynomial time, while k-SHPL is NP-complete for any k such that $2 \le k \le n^{1-\epsilon}$, where n is the number of residents and ϵ is any positive constant. We also present computational results for 2-SHPL. We provide four first-fit algorithms which do not necessarily output a correct answer. We implement these algorithms and compare their computation time and accuracy using random instances.

Related work: For SM, Kobayashi and Matsui [4] have previously presented a similar issue and studied it in the context of strategic issue: 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.

Organization of the paper: This paper is organized as follows. In Section 2, we define the problem and present some notations. In Section 3, we give complexity results for k-SHPL. In Section 4, we present four first-fit algorithms. Finally, in Section 5, we present a computational study.

2. Preliminaries

2.1 Hospitals/Residents Problem and Stable Hospital's Preference List Problem

We first give a formal definition of HR. An instance of HR is as follows: (i) a set of residents $R = \{r_1, r_2, \dots, r_n\}$; (ii) a set of hospitals $H = \{h_1, h_2, \dots, h_m\}$; (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 a subset of R in a strict order; (v) capacities c_j ($1 \le j \le m$), indicating the number of posts that h_j has. An example of HR instance is shown in **Fig. 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.

Residents' preferences	Hospitals' capacities and preferences
$r_1 : h_2 \ \ h_3 \ \ h_1$	$h_1 : (2) : r_2 \ \ r_3 \ \ r_5 \ \ r_1$
$r_2 : h_2 \ \ h_1$	$h_2 : (2) : r_3 \ \ r_5 \ \ r_1 \ \ r_2 \ \ r_4$
$r_3 : h_3 \ \ h_2 \ \ h_1$	$h_3 : (1) : r_6 \ \ r_4 \ \ r_1 \ \ r_3 \ \ r_5$
$r_4 : h_2 \ \ h_3$	
$r_5 : h_2 \ \ h_1 \ \ h_3$	
$r_6 : h_3$	

Fig. 1 HR instance I_1.

Residents' preferences	Matching	Hospitals' capacities
$r_1 : h_2 \ \ h_3 \ \ h_1$	(r_1, h_2)	$h_1 : (2)$
$r_2 : h_2 \ \ h_1$	(r_2, h_1)	$h_2 : (2)$
$r_3 : h_3 \ \ h_2 \ \ h_1$	(r_3, h_1)	$h_3 : (1)$
$r_4 : h_2 \ \ h_3$	(r_5, h_2)	
$r_5 : h_2 \ \ h_1 \ \ h_3$	(r_6, h_3)	
$r_6 : h_3$		

Fig. 2 SHPL instance I_2.

A *matching* M is a set of (resident, hospital) pairs $(r, h) \in R \times H$ such that $(r, h) \in M$ only if r and h find each other acceptable, no resident is assigned to more than one hospital, and no hospital h_j is assigned more than c_j residents. If $(r, h) \in M$, we say that r is *assigned to* h, and h is *assigned* r. For any $q \in R \cup H$, 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 the resident r in M. A hospital h_j is *under-subscribed* if $|M(h_j)| < c_j$ and is *full* if $|M(h_j)| = c_j$.

A pair $(r, h) \in R \times H$ is said to *block* a matching M, or is called a *blocking pair* for M, 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 exist hospitals' preference lists that make M stable. **Figure 2** gives an example instance of SHPL.

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' complete 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: k-Stable Hospital's Preference List problem (k-SHPL).
Instance: An instance of SHPL.
Question: Are there any hospitals' complete preference lists that make M stable, where there are at most k kinds of preference lists of hospitals?

We give some observation on SHPL. Suppose that a hospital h is under-subscribed. If there is a resident r who is unassigned to any hospital but includes h in her list, or prefers h to $M(r)$, then

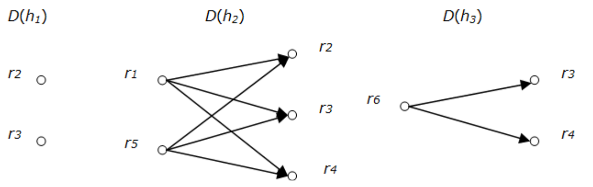


Fig. 3 $D(h)$ of SHPL instance I_2.

(r, h) is a blocking pair no matter what preference list h may have (because h wishes to hire one more resident without depending on the preference order). If there is no such resident r, then h is never included in a blocking pair no matter what preference list h may have (because no resident has incentive to go to h). Therefore, we may deal with such under-subscribed hospitals first, and may consider only full hospitals in the main process. Hence, without loss of generality, we assume in the following that all the hospitals are full under the given matching M.

2.2 Acyclic Partitioning Digraphs

We show that k-SHPL can be reduced to the following problem *k-Acyclic Partitioning Digraphs* (*k-APD*). This property is convenient and is used several times for developing algorithms and showing hardness of k-SHPL.

Problem: k-Acyclic Partitioning Digraphs (k-APD).

Instance: A set of digraphs (where two digraphs may share common vertices).

Question: Is there a partition of given digraphs into k subsets such that, in each subset, the digraph constructed by taking the union of digraphs in the subset does not contain a directed cycle?

In the reduction, we construct a digraph $D(h)$ corresponding to each hospital h. $D(h)$ is a bipartite graph $D(h) = (V^L, V^R, E)$ where each vertex of $D(h)$ corresponds to a resident, and V^L and V^R are bipartition of the vertices. V^L includes all the residents in $M(h)$, and V^R includes all the residents r such that (i) h is acceptable to r and (ii) r is unassigned in M or prefers h to $M(r)$. Finally, E includes all the arcs from V^L to V^R, i.e., $D(h)$ is a complete bipartite graph (if we ignore the direction of edges). **Figure 3** illustrates three digraphs corresponding to the hospitals in the instance I_2 given in Fig. 2.

We can interpret $D(h)$ in the following way: Let $r_1 \in V^L$ and $r_2 \in V^R$. By construction of $D(h)$, 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, h does not create a blocking pair for M if and only if the order of residents in a preference list of h is consistent with arcs of $D(h)$.

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 arcs in $D(h_1), D(h_2), \ldots, D(h_c)$. If $D(h_1, h_2, \ldots, h_c)$ does not contain a directed cycle, then we can obtain a linear list being consistent with all the arcs in $D(h_1, h_2, \ldots, h_c)$. If all hospitals $h_1, h_2, \ldots, h_c$ have this list in common, then these hospitals do not create 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 any identical list for $h_1, h_2, \ldots, h_c$ creates at least one blocking pair. The correctness of the reduction is validated from these observations.

3. Complexity Results

3.1 1-SHPL is Solvable in Polynomial Time

1-SHPL can be solved in polynomial time by the following algorithm: Given an instance of 1-SHPL, we construct a digraph $D(h_j)$ corresponding to each hospital h_j $(1 \le j \le m)$ and solve 1-APD. 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 arcs in $D(h_1), D(h_2), \ldots, D(h_m)$ and then check whether the digraph $D(h_1, h_2, \ldots, h_m)$ contains a cycle. We output "Yes" if and only if $D(h_1, h_2, \ldots, h_m)$ is acyclic. This algorithm can be implemented to run in $O(mn^2)$ time overall.

Theorem 1. *1-SHPL is solvable in $O(mn^2)$ time.*

3.2 2-SHPL is NP-complete

Theorem 2. *2-SHPL is NP-complete even if each hospital's capacity is one.*

Proof. It is easy to see that 2-SHPL is in NP: Given hospitals' preference lists, one can check whether it is a blocking pair for each pair of a resident and a hospital. This can be done in polynomial time. We show a polynomial time reduction from Acyclic 2-Coloring, which is already known to be NP-complete [5], to 2-SHPL.

Problem: Acyclic 2-Coloring.

Instance: Directed graph $G = (V, A)$.

Question: Can we color the nodes of G with 2 colors such that no monochromatic directed cycle occurs?

We reduce Acyclic 2-Coloring to 2-SHPL. Let $G = (V, A)$ be a directed graph with $V = \{v_1, v_2, \ldots, v_n\}$, which is an arbitrary instance of Acyclic 2-Coloring. We construct an instance $I(G)$ of 2-SHPL, namely, (i) the sets of residents and hospitals, (ii) a matching, (iii) each hospital's capacity, and (iv) each resident's preference list.

First of all, we introduce the n residents $r_1, r_2, \ldots, r_n$ and the same number of hospitals $h_1, h_2, \ldots, h_n$. A resident r_i and a hospital h_i are associated with a vertex v_i of G $(1 \le i \le n)$. Next, we construct a matching $M = \{(r_1, h_1), (r_2, h_2), \ldots, (r_n, h_n)\}$. Then we set each hospital's capacity c_i to one. Note that every hospital is full under M.

We construct each resident's preference list using 2-APD defined in Section 2.2. For each i, we first construct a bipartite digraph $D(h_i) = (V_i^L, V_i^R, A_i)$ corresponding to the hospital h_i, where each vertex of $D(h_i)$ corresponds to a resident. Recall that the hospital h_i of $I(G)$ is associated with vertex v_i of G. Let $v_{i_1}, v_{i_2}, \ldots, v_{i_s}$ be the direct successor of v_i in G, i.e., there is an arc (v_i, v_{i_j}) in A for each $1 \le j \le s$. Then let $V_i^L = \{r_i\}$, $V_i^R = \{r_{i_1}, r_{i_2}, \ldots, r_{i_s}\}$, and A_i include arcs from r_i to each vertex in V_i^R. We finally construct residents' preference lists from these digraphs $D(h_i)$ using the reduction described in Section 2.2 in the reverse direction: r_i's preference list includes a hospital h_j if and only if $D(h_j)$ contains vertex r_i. (Note that, by construction, $D(h_i)$ contains vertex r_i in V_i^L, and if $D(h_j)$ contains vertex r_i for $j \ne i$, then r_i appears in V_j^R.) In r_i's preference list, h_i must be the bottom (i.e., the lowest rank) of the list, but the order of other hospitals is arbitrary. It is not hard to see that this is a reverse reduction of the one in Section 2.2 and hence if we regard the set of digraphs $D(h_1), D(h_2), \ldots, D(h_n)$ as an instance $I'(G)$ of 2-APD, then $I(G)$ is an yes-instance of 2-SHPL if and only if $I'(G)$ is an yes-instance of 2-APD. Therefore, to complete the proof, we show that $I'(G)$ is an yes-instance of 2-APD if and only if G is an yes-instance of Acyclic 2-Coloring.

We first show the "if"-part. Since G is an yes-instance, there is a 2-coloring of the vertices of G that yields no monochromatic directed cycle. Let the used colors be integers 1 and 2. We construct a solution of $I'(G)$, i.e., a partition of digraphs to two subsets, in the following way: If v_i is colored with color c ($c \in \{1, 2\}$), then we put $D(h_i)$ into the subset S_c. We show that neither S_1 nor S_2 induces a cycle. For, suppose that there is a cycle in S_c, and let the cycle be $r_1, r_2, \ldots, r_\ell, r_1$ without loss of generality. Then, there is an arc (r_i, r_{i+1}) in $D(h_i)$ for each $1 \le i \le \ell - 1$ and an arc (r_ℓ, r_1) in $D(h_\ell)$. By construction of $I'(G)$, arcs (v_i, v_{i+1}) for $1 \le i \le \ell - 1$ and (v_ℓ, v_1) exist in G. Also, $D(h_i)$ for each $1 \le i \le \ell$ is in S_c because there is an arc emanating from r_i in S_c. Hence, all v_i for $1 \le i \le \ell$ are colored with color c. This means that there is a monochromatic cycle $v_1, v_2, \ldots, v_\ell, v_1$ in G, a contradiction.

The "only if"-part is almost the same as the "if"-part. Suppose that there is a partition of digraphs $D(h_1), D(h_2), \ldots, D(h_n)$ into two subsets S_1 and S_2 so that there is no induced directed cycle in each subset. We will construct a solution of G as follows: If $D(h_i)$ is included in S_c ($c \in \{1, 2\}$), then we color the vertex v_i with color c. We show that this coloring creates no monochromatic cycle. For, suppose that there is a monochromatic directed cycle $v_1, v_2, \ldots, v_\ell, v_1$. Since these vertices are colored with the same color c, $D(h_i)$ for each $1 \le i \le \ell$ is classified into the subset S_c. Since $D(h_i)$ for each $1 \le i \le \ell - 1$ contains the arc (r_i, r_{i+1}) and $D(h_\ell)$ contains the arc (r_ℓ, r_1), S_c induces a cycle $r_1, r_2, \ldots, r_\ell, r_1$, a contradiction. □

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

Theorem 3. *For any k such that $3 \le k \le n^{1-\epsilon}$, where ϵ is any positive constant, k-SHPL is NP-complete even if every hospital's capacity is one.*

Proof. It is easy to see that k-SHPL is in NP: Given hospitals' preference lists, one can check whether it is a blocking pair for each pair of a resident and a hospital. This can be done in polynomial time. We give a polynomial time reduction from k-Coloring to k-SHPL. It is known that k-Coloring is NP-complete for any constant $k \ge 3$ [2]. Also, it is not hard to see that NP-completeness holds for k being as large as $n^{1-\epsilon}$ for any positive constant ϵ.

Problem: k-Coloring.

Instance: Graph $G = (V, E)$.

Question: Can we color the vertices of G with k colors such that the endpoints of every edge are colored differently?

Let $G = (V, E)$ be an undirected graph with $V = \{v_1, v_2, \ldots, v_n\}$, which is an arbitrary instance of k-Coloring. We construct an instance $I(G)$ of k-SHPL, namely, (i) the sets of residents and hospitals, (ii) a matching, (iii) each hospital's capacity, and (iv) each resident's preference list.

First of all, we introduce the following n residents $r_1, r_2, \ldots, r_n$ and the same number of hospitals $h_1, h_2, \ldots, h_n$. A resident r_i and a hospital h_i are associated with a vertex v_i ($1 \le i \le n$) of G. Next, we construct a matching $M = \{(r_1, h_1), (r_2, h_2), \ldots, (r_n, h_n)\}$. Then we set each hospital's capacity c_i to one. Note that every hospital is full under M.

To construct each resident's preference list, we again use k-APD. For each i, we construct a bipartite digraph $D(h_i) = (V_i^L, V_i^R, A_i)$ corresponding to the hospital h_i, where each vertex of $D(h_i)$ corresponds to a resident. Let $v_{i_1}, v_{i_2}, \ldots, v_{i_s}$ be the neighbors of v_i in G. Then let $V_i^L = \{r_i\}$, $V_i^R = \{r_{i_1}, r_{i_2}, \ldots, r_{i_s}\}$, and A_i include arcs from r_i to each vertex in V_i^R. We finally construct residents' preference lists from these digraphs $D(h_i)$ in exactly the same manner as we have done in the proof of Theorem 2, and we obtain an instance $I(G)$ of k-SHPL. Let $I'(G)$ be the instance of k-APD, which is the set of digraphs $D(h_1), D(h_2), \ldots, D(h_n)$. By the same argument as in the proof of Theorem 2, it suffices to show that $I'(G)$ is an yes-instance of k-APD if and only if G is an yes-instance of k-Coloring.

We first show the "if"-part. Since G is an yes-instance, there is a proper k-coloring of vertices of G. Let the used colors be integers $1, 2, \ldots, k$. Then we construct a solution of $I'(G)$ using this proper coloring: If v_i is colored with color c ($c \in \{1, 2, \ldots, k\}$), then we put $D(h_i)$ into the subset S_c. We show that none of $S_1, S_2, \ldots, S_k$ induces a directed cycle. For, suppose that there is a cycle in S_c, and let the cycle be $r_1, r_2, \ldots, r_\ell, r_1$ without loss of generality. Since $D(h_1)$ contains arc (r_1, r_2), there is edge (v_1, v_2) in G. Since there is an arc emanating from r_1 in S_c, $D(h_1)$ is in S_c. For the same reason, $D(h_2)$ is also in S_c. Hence both v_1 and v_2 are colored with color c. This is a contradiction because edge (v_1, v_2) breaks the condition for proper coloring.

We next show the "only if"-part. Suppose that there is a partition of digraphs $D(h_1), D(h_2), \ldots, D(h_n)$ into k subsets $S_1, S_2, \ldots, S_k$ so that there is no induced directed cycle in each subset. We will construct a k-coloring of G as follows: If $D(h_i)$ is included in S_c ($c \in \{1, 2, \ldots, k\}$), then we color vertex v_i with color c. We show that this is a proper coloring. For, suppose not and there is an edge (v_i, v_j) in G such that v_i and v_j are colored with the same color c. Then $D(h_i)$ and $D(h_j)$ are classified into the subset S_c and there is an arc (r_i, r_j) in $D(h_i)$ and (r_j, r_i) in $D(h_j)$ by construction. This implies that S_c induces a length-two cycle r_i, r_j, r_i, a contradiction. □

4. First-Fit Algorithms for 2-SHPL

In this section, we describe four first-fit algorithms for solving 2-SHPL. Considering the observation in Section 2.2, 2-SHPL is the problem of partitioning digraphs $D(h)$ for hospitals h into two subsets A and B so that digraphs classified into the same subset do not induce a directed cycle. The concept of first-fit type algorithms is as follows: We determine a processing order of digraphs first. According to this order, we process digraphs one by one. When processing the current digraph $D(h)$, we first try to put it into the subset A. If it is successful, i.e., it creates no directed cycle in A, we do so. Otherwise, we try to put $D(h)$ to B. If it is successful, we do so. Otherwise, putting $D(h)$ to either of A and B creates a cycle, so we give up here. The performance of first-fit depends on the order of processing digraphs determined at the beginning of the algorithm. We propose two algorithms in this respect. FirstFit1 determines this order randomly.

```
FirstFit1
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 $A$
5     if ( group $A$ contains a cycle ) then
6        delete $D(h)$ from group $A$ and add it to group $B$
7        if ( group $B$ contains a cycle ) then
            return NO and halt
8  return YES
```

Note that first-fit algorithms are popular for the bin packing problem [2]. It is well-known that the first-fit for bin packing problem generally performs well when we process items in the decreasing order of their sizes; intuitively, small items are flexible and we can use them to fill small gaps at the later stages of the algorithm. Therefore, it is natural to employ this strategy. Our second algorithm FirstFit2 determines the order of $D(h)$ in descending order of its number of edges.

```
FirstFit2
1  construct a complete bipartite digraph $D(h)$ for each $h$
2  sort $D(h)$ in descending order of its number of edges
3  for ( the first $D(h)$ to the last $D(h)$ ) do
4     add $D(h)$ to group $A$
5     if ( group $A$ contains a cycle ) then
6        delete $D(h)$ from group $A$ and add it to group $B$
7        if ( group $B$ contains a cycle ) then
            return NO and halt
8  return YES
```

Notice that the above first-fit algorithms have one-sided error, i.e., if the true answer is NO, then the first-fit 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. To reduce the error rate of such one-sided error randomized algorithms, it is common to repeat the same algorithm many times and answer YES if at least one execution outputs YES, and answer NO otherwise.

To see the effect of such repetitions, we propose FirstFit3, which repeats the main process of FirstFit1 t times and outputs YES if and only if at least one execution is successful.

```
FirstFit3
1  construct a complete bipartite digraph $D(h)$ for each $h$
2  for ( $t$ 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 $A$
6        if ( group $A$ contains a cycle ) then
7           delete $D(h)$ from group $A$ and add it to group $B$
8           if ( group $B$ contains a cycle ) then goto 3
9     return YES and halt
10 return NO
```

We also propose another reordering method; we update the list by moving $D(h)$, which has caused a cycle in both of the two groups, to the head of the list in the next run. This strategy is based on the idea that such $D(h)$ is critical and inflexible for our purpose, which suggests we should process it in an earlier stage of the algorithm. We call this algorithm FirstFit4.

```
FirstFit4
1  construct a complete bipartite digraph $D(h)$ for each $h$
2  sort $D(h)$ randomly
3  for ( $t$ times ) do
4     for ( the first $D(h)$ to the last $D(h)$ ) do
5        add $D(h)$ to group $A$
6        if ( group $A$ contains a cycle ) then
7           delete $D(h)$ from group $A$ and add it to group $B$
8           if ( group $B$ contains a cycle) then
9              update the sorting order by moving $D(h)$
               to the head and goto 4
10    return YES and halt
11 return NO
```

To evaluate the accuracy of the first-fit algorithms, we need a correct answer for randomly generated instances. The following exact algorithm ExactAlg, which performs exhaustive search, is used to determine the correct answer of instances.

```
ExactAlg
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 and halt
4  return NO
```

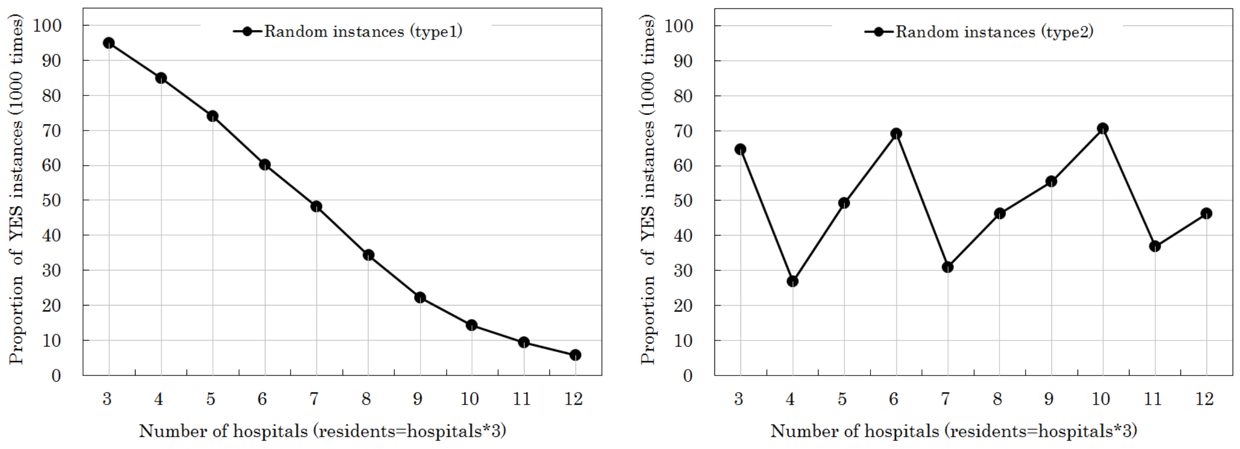
5. Computational Experiments

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 performance of first-fit algorithms.

5.1 Instance Generation

We show how to generate instances of SHPL. Recall that 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 use instances such that the number of residents is exactly three times that of hospitals.

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 ith row and jth 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 an 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



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

Fig. 4 Proportion of YES instances.

hospital h_j is unacceptable to resident r_i.

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 ordered differently than others, we shuffle randomly each row of $P[i, j]$. We then pick a random number s between 1 and m uniformly and set elements $P[i, j]$ ($j \geq s + 1$) to the null symbol, so that each resident's preference list forms an incomplete list of size s. Also, to make a matching A, we select a hospital j uniformly at random such that $P[i, j]$ is not null on each resident's preference list, and set $A[i]$ to the hospital j. Finally, we set $C[j]$ 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.

Figure 4 (a) shows the proportion of yes-instances in randomly generated 1,000 instances in the above manner, according to the number of hospitals. As shown in Fig. 4 (a), the proportion of yes-instances decreases gradually as the number of hospitals ascends stepwise. To generate harder instances, we want to seek for such an instance generator that outputs yes-instances and NO instances in the proportion of 50 to 50. For this purpose, we modify the above simple generator (type1) to obtain the following modified generator (type 2). The key difference between the two types of generators is how to decide the size of each resident's preference lists $L(r)$. In type1-generator, we decide the size of each resident's preference list uniformly at random, while in type2-generator, we decide it by the following formula, which is obtained by trial and error:

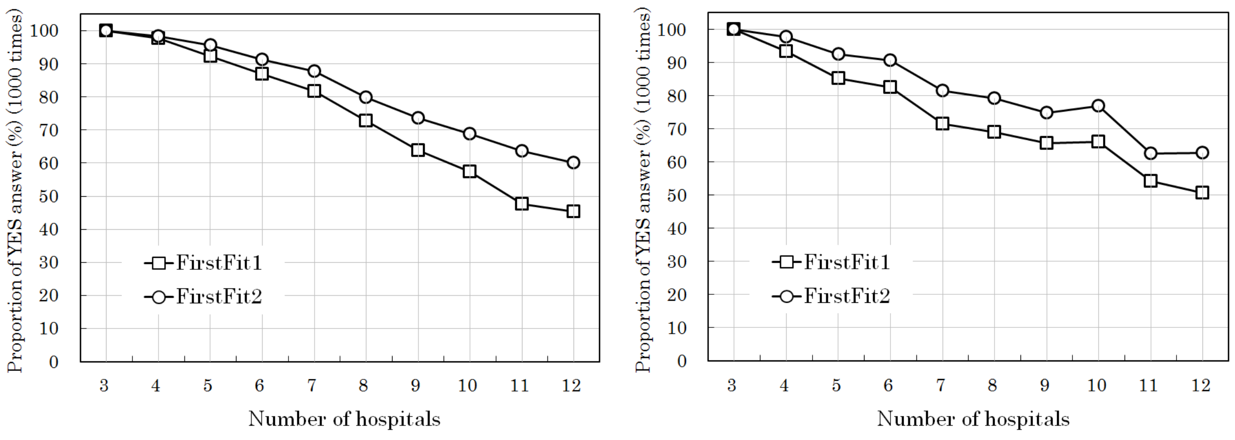
$$L(r) = \left[\frac{\log m + 3 \log 5 - 4 \log 3}{\log 5 - \log 3} \right] \quad \text{for} \quad m = 3, 4, 5, \ldots \quad (1)$$

Figure 4 (b) shows the proportion of yes-instances of those generated by type2-generator. We can see that the proportion is not perfectly 50:50, but is better than type1-generator.

5.2 Computational Results

We conducted two experiments. The first one compares the performance of FirstFit1 and FirstFit2 to see the effect of the initial processing order of digraphs. As mentioned earlier, every first-fit 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 first-fit 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 only instances which ExactAlg outputs YES. **Figure 5** shows the accuracy of FirstFit1 and FirstFit2 (i.e., the rate these algorithms output YES) according to the number of hospitals. The number of hospitals varies from 3 to 12, and the number of residents is three times that of hospitals as mentioned before. For each size, we generated 1,000 instances in the way described in Section 5.1, and applied FirstFit1 and FirstFit2 to them. As shown in Fig. 5, FirstFit2 has an accuracy higher than that of FirstFit1. This means that, similarly to the case of the bin packing problem, processing larger digraphs earlier would be effective.

Next, we verify the number of iterations that each of FirstFit3 and FirstFit4 outputs a correct answer. It is natural that as the number of iterations increases, the accuracy of the algorithm grows higher. Our purpose of this experiment is to compare the effectiveness of reordering the processing order of digraphs; we investigate the average number of iterations needed for each of FirstFit3 and FirstFit4 to output YES when they are given yes-instances. **Figure 6** shows the result according to the number of hospitals. The number of hospitals varies from 3 to 24, and the number of residents is three times that of hospitals as in the previous experiment. This time, however, we used only type2-generator since the rate of yes-instances generated by type1-generator decreases as the number of hospitals grows (see Fig. 4). Furthermore, when the size of the instance grows, the computation time of ExactAlg (for checking its correct answer) grows exponentially, and it was impossible to generate 1,000 different yes-instances within reasonable time. Therefore, for each size, we generated 10 yes-instances and applied FirstFit3 and FirstFit4 to them 100 times, i.e., we conducted 1,000 trials for each size.



(a) Proportion of correct answer of random instances (type1) (b) Proportion of correct answer of random instances (type2)

Fig. 5 The Proportion of correct answer of FirstFit1 and FirstFit2.

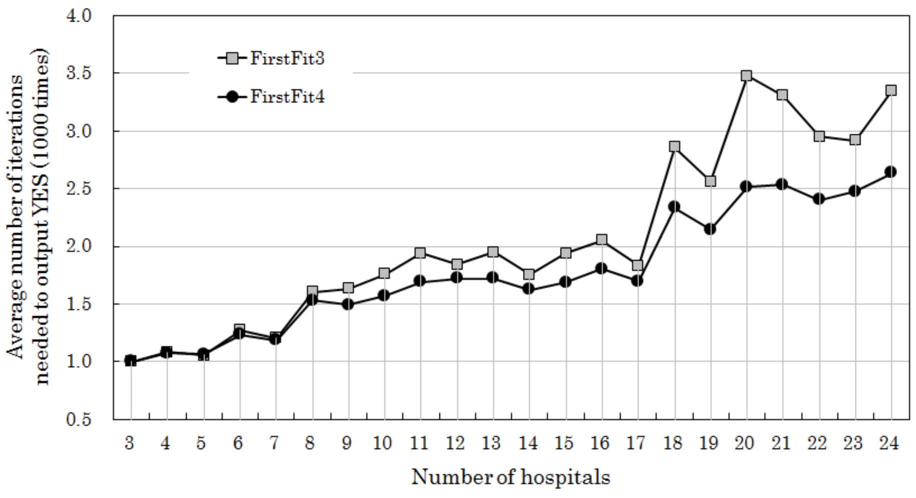
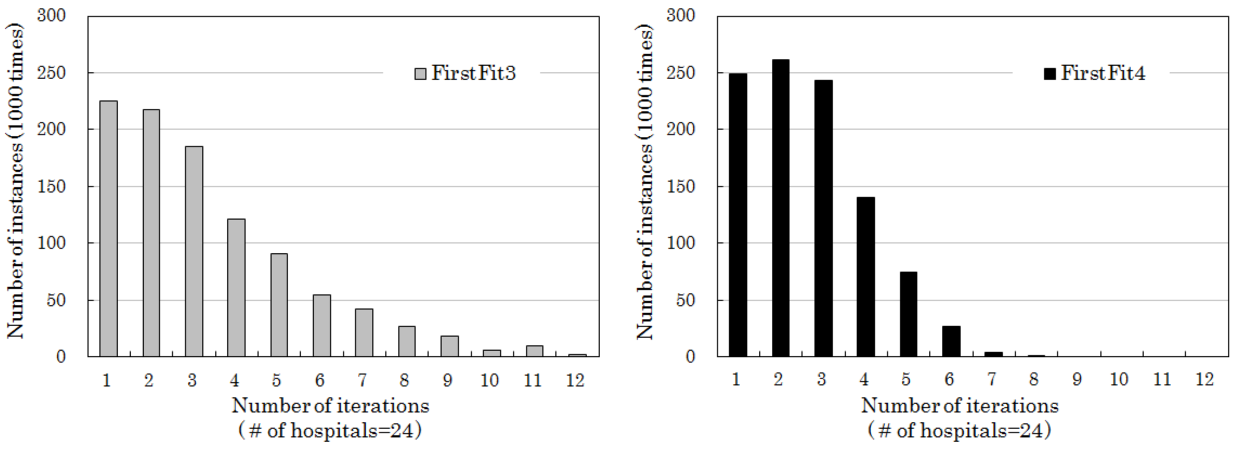


Fig. 6 The average number of iterations needed to output YES of random instances (type2).



(a) FirstFit3 (b) FirstFit4

Fig. 7 The distribution of the number of iterations.

Also, to see the distribution of the number of iterations of this experiment, we take the case where the number of hospitals is 24 in Fig. 6, and give histograms in **Fig. 7**. The result shows that FirstFit4 finds a correct answer in a smaller average number of iterations than FirstFit3.

6. Conclusion

In this paper, we studied the problem of deciding, given residents' preference lists and a matching, whether there are hospitals' preference lists that make a given matching stable. We were motivated to consider a restricted variant of the problem k-SHPL where there are only k preference lists and each hospital's preference list is identical to one of them. We proved that 1-SHPL is solvable in polynomial time and for a constant $k \geq 2$, k-SHPL is NP-complete even if every hospital's capacity is one. We also presented computational results for 2-SHPL. We provided first-fit algorithms and implemented them. By computational experiments, we compared the efficiency of first-fit algorithms in terms of the ways of ordering digraphs, i.e., the initial ordering and the reordering in repetitions. As a result, we concluded that a descending order is more efficient than a random order in the initial ordering, and moving the critical digraph to the top is more efficient than random reordering when we repeat the first-fit algorithm.

References

[1] Gale, D. and Shapley, L.S.: College admissions and the stability of marriage, *American Mathematical Monthly*, Vol.69, pp.9–15 (1962).
[2] Garey, M.R. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Co (1979).
[3] Gusfield, D. and Irving, R.W.: *The Stable Marriage Problem: Structure and Algorithms*, MIT Press (1989).
[4] Kobayashi, H. and Matsui, T.: Successful manipulation in stable marriage model with complete preference lists, *IEICE Trans. Inf. & Syst.*, Vol.E92-D, No.2, pp.116–119 (2009).
[5] Nobibony, F.T., Hurkensz, C.A.J., Leus, R. and Spieksma, F.C.R.: Exact algorithms for coloring graphs while avoiding monochromatic cycles, *Proc. AAIM 2010*, LNCS 6124, pp.229–242 (2010).
[6] Roth, A.E.: 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).
[7] Canadian Resident Matching Service: The Match Algorithm, available from ⟨http://www.carms.ca/matching/algorith.htm⟩ (accessed 2013-05-30).
[8] Japan Residency Matching Program: About Japan Residency Matching Program, available from ⟨http://www.jrmp.jp/aboutmatching.htm⟩ (accessed 2013-05-30).
[9] National Resident Matching Program: Match Algorithm available from ⟨http://www.nrmp.org/about_nrmp/how.html⟩ (accessed 2013-05-30).
[10] Scottish Foundation Allocation Scheme: Scottish Foundation Allocation Scheme, available from ⟨http://www.nes.scot.nhs.uk/sfas⟩ (accessed 2013-05-30).

Editor's Recommendation

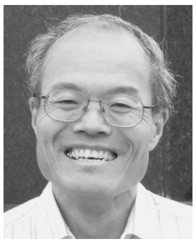
IPSJ Kansai-Branch chose the recommended papers out of the papers presented at IPSJ Kansai-Branch Convention. Session chairpersons and Committee members of the convention nominated four candidates from the 83 papers presented in the Convention 2013, and each of the candidates was reviewed by two referees. After careful discussion among the committee members, we finally selected two papers to be recommended.

This paper proposes an algorithm for solving the hospitals/residents problem, and we found significant novelty of the algorithm in the research area. Therefore we have decided to recommend that this paper would be submitted to Journal of Information Processing.

(Toru Fujiwara, Chairman of IPSJ Kansai-Branch)

Minseon Lee is currently a Ph.D. student of Kyoto University. She received her B.E. degree from Inha University in 2004, and M.E. degree from Kyoto University in 2013. Her research interests include algorithms, complexity theory and public-key cryptography. She is a member of IPSJ.

Shuichi Miyazaki is an associate professor at Academic Center for Computing and Media Studies, Kyoto University. He received his B.E., M.E., and Ph.D. degrees from Kyushu University in 1993, 1995 and 1998, respectively. His research interests include design and analysis of algorithms, and complexity theory. He is a member of IPSJ, IEICE, and EATCS.



Kazuo Iwama is a professor of Graduate School of Informatics, Kyoto University. He received his B.E., M.E., and Ph.D. degrees from Kyoto University in 1978, 1980, and 1985, respectively. His research interests include algorithms, complexity theory, and quantum computation. He is Editor-in-Chief of EATCS Bulletin and also a member of Academia Europaea.