A Two-phase Heuristic Approach for Solving Trainee Rotation Assignment Problem at a Local School of Nursing

Ziran Zheng · Xiaoju Gong · Xiyu Liu

Abstract Students in nursing school need to receive practical training in various departments in hospitals within a certain period of time before graduate. With the constraints imposed by hospital wards and training program from school, solving this rotation assignment problem by hand is a very complicated task even for small scale instances. Such problems also commonly occur in many educational settings where an organization may wish to schedule trainees to a training programs of different rotations. In this work, a two-phase heuristic algorithm is proposed for solving a trainee rotation assignment problem appears in a local school of nursing and its training hospital. At the first phase, the model is reduced to a simplified assignment problem and solved using a random search procedure. At the second phase, a problem-specific operator is designed and employed with a hill climber to further improve solutions. We benchmark our algorithm with instances generated based on the real-life rules. Results show that the proposed algorithm yields high-quality solutions in less computation time when compared with integer linear programming formulation using Cplex.

Keywords Trainee rotation assignment · Personnel scheduling · Heuristic

The work is supported by the Project of Shandong Province Higher Educational Science and Technology Program(Grant No. J14LN10), National Natural Science Foundation of China (Grant No. 61472231) and National Natural Science Foundation of China (Grant No. 61502283)

Ziran Zheng
School of Management Science and Engineering, Shandong Normal University
E-mail: zzr_nature@126.com

Xiaoju Gong
Shandong Provincial Hospital Affiliated to Shandong University

Xiyu Liu
School of Management Science and Engineering, Shandong Normal University

1 Introduction

Since the job requirements become more diverse, graduated students of medical college are usually faced with a gap between their major of study and growing demand of skill variety. In particular, at school of nursing, education programs usually involve teaching only general expertise and thus cannot cover all specialized skills which would be needed in the practical working situation. Therefore, it is necessary for them to gain sufficient supervised practical experiences in various departments in hospitals before they begin their careers. For most cases in our local district, all of the nursing students upon their graduate will receive a complete training program through all departments at their internship hospitals within the last year of study to meet this demand. In this proposal, a class of such rotation assignment problem, appears in school of nursing in Shandong University, is addressed.

The problem being considered involves several constraints which are imposed by school training program, the internship hospital and the students themselves. The first category of constraints is on the basis of program rules. First, students should complete the whole training program which consists of a full set of different rotations in the corresponding hospital wards. These must be done by all trainees within a same range of period. To finish each rotation, a trainee must perform it for a number of periods and there is a lower and upper bound for the duration. For instance, students should complete training program in all five different wards(rotations) and in each ward they must work at least three weeks and at most five weeks. Second, for each student, once a training rotation starts at a certain period, it must be performed in the following consecutive periods until it is finished. We call this rule the *consecutive constraint*, which is a major feature of this type of problem. Another constraint comes from the training hospital where the number of available positions for interns has certain demand. For a specific ward, as too many intern students will influence the normal efficiency and the quality of training is also hard to guarantee, the number of available positions is usually fixed to a small value which is confirmed before scheduling. The last constraint comes from students. Since at the last year of study they often have requests not to want to work on specific time periods, the schedule should be made with less conflicts with such preferences. In this study, the constraints imposed by school and hospital are treated as hard constraints while those that represent period preferences of students are treated as soft ones. If the hard constraints are not violated, we say the solution is feasible and otherwise infeasible.

The problem can be mathematically formulated as an integer linear programming model. Let n, r and m be the number of whole periods, rotations and trainees respectively. The model is described as follows.

Sets and indices
$\{R_k\}$: set of rotations and let $R_k, k = 1, ..., r$ denote a type of rotation;
$\{T_j\}$: set of trainees and let $T_j, j = 1, ..., m$ denote a single trainee;

I: set of whole periods where $I = \{1, 2, ..., n\}$.

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{r} p_{ij}x_{ijk} \tag{1}$$

Subject to:

$$\sum_{k=1}^{r} x_{ijk} \leq 1, \forall i = 1, ..., n \text{ and } \forall j = 1, ..., m, \tag{2}$$

$$\sum_{j=1}^{m} x_{ijk} = 1, \forall i = 1, ..., n \text{ and } \forall k = 1, ..., r, \tag{3}$$

$$\sum_{i=1}^{n} x_{ijk} \geq l, \forall k = 1, ..., r \text{ and } \forall j = 1, ..., m, \tag{4}$$

$$\sum_{i=1}^{n} x_{ijk} \leq u, \forall k = 1, ..., r \text{ and } \forall j = 1, ..., m, \tag{5}$$

$$y_{1jk} = x_{1jk}, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r \tag{6}$$

$$y_{ijk} \geq x_{ijk} - x_{(i-1)jk}, \forall i = 2, ..., n, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r, \tag{7}$$

$$\sum_{i=1}^{n} y_{ijk} = 1, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r, \tag{8}$$

$$y_{ijk}, x_{ijk} \in \{0, 1\}, \forall i = 1, ..., n, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r. \tag{9}$$

There are two sets of binary decision variables. The first set defines rotation type in each period:

$$x_{ijk} = \begin{cases} 1, & \text{in period } i \text{ trainee } j \text{ perform rotation } k \\ 0, & \text{otherwise} \end{cases}$$

Another set defines the start of a rotation:

$$y_{ijk} = \begin{cases} 1, & \text{trainee } j \text{ perform rotation } k \text{ starts at period } i \\ 0, & \text{otherwise} \end{cases}$$

In this model, parameter p_{ij} defines the soft constraint for a trainee i at a period j. When the trainee i does not want to work at period j, p_{ij} is 1 and otherwise it equals to 0. Throughout this paper, we call it undesired period for a trainee. Constraint (2) makes sure that each trainee cannot perform more than one rotation within each period. Constraints (3) implies that in each period every rotation must be performed exactly by one trainee. This constraint set reflects the practical setting in our local school that every ward has exactly one internship position in a period for all trainees and during the

Periods	Trainee 1	Trainee 2	Trainee 3
1	R_1	R_2	
2	R_1	R_2	
3		R_1	R_2
4		R_1	R_2
5	R_2	R_1	
6	R_2		R_1
7	R_2		R_1

Fig. 1: A feasible solution for a rotation assignment problem instance. Here $n = 7, r = 2, m = 3, l = 2$ and $u = 3$. The cells with bold border represent the undesired periods of trainees, which are: $p_{21} = 1$, $p_{51} = 1$, $p_{62} = 1$ and $p_{33} = 1$. Different colors are used to help distinguish two types of rotations.

whole period this position should be filled. Constraint (4) and (5) describe that the number of periods for a trainee performing a rotation should be between a minimum and a maximum number. The parameters l and u represent these two values respectively. Constraints (6) to (8) state that each rotation starts only once in the whole time period so that it is performed in consecutive periods. The decision variables are defined as binary variables in constraint (9). As described above, all constraints in the model are treated as hard constraints and only parameters representing undesired periods in the objective function describe the soft ones. The model tries to minimize the cost of feasible solutions by reducing soft violated constraints as much as possible. Note that although this model is borrowed from [2], there are adaptations to fit the scenarios appears in our local school and hospital. First, in their model, for a specific rotation, there is a subset of trainees which perform the task while in this study, each rotation should be performed by all trainees, since in our school all students are at the same level of skills upon graduate. Second, in constraint (4), l and u are fixed for all trainees and rotations in this study. Thus the model in this work can be seen as a special case of the model provided in [2].

Figure 1 presents an example of such a problem instance and its solution. Suppose that the whole time range has seven time periods and we want to assign two rotations to three trainees. At each time period, the internship hospital has one available position for each rotation and in each row every rotation must be performed by one trainee. The l and u in this example are 2 and 3 respectively. In this solution the consecutive constraints are all satisfied. This solution is clearly feasible and three soft constraints are violated.

Constructing such a rotation assignment schedule is a very complicated task by hand even for small instances. It is also worth noting that such a scheduling problem arises in many various scenarios, particularly in education

institutions where students need a series of practical training programs before their graduation.

In this paper, a two-phase heuristic algorithm is proposed for solving this problem. Section 2 presents a brief overview of relevant literature. Detailed heuristic approach is described in Section 3. In Section 4, computational results for random generated instances based on real-life rules are presented and discussed. The paper ends up with conclusions and future work in Section 5.

2 Related work

Although there exist some related papers for rotation assignment problem, the models are very problem-dependent and there is no generalized and suitable model definition for all situations among literature. For a similar case, the resident scheduling problem tries to assign resident physicians to rotations in hospitals. In [8], Ozkarahan developes a decision model which attempts to schedule residents based on balance of requirements of the residency program and workload fatigue. Javeri [7] develops a schedule tool using integer programming for medical resident education. Franz and Miller [5] propose a solution approach for assigning 24 medical residents to rotations in a year using linear programming with a rounding procedure. In their model there are 12 rotations and the preference parameters reflect the priority choices and the objective is to maximize the fitness value. Other constraints involve specific rules based on their problems being solved. Smalley and Keskinocak [9] present two integer programming models with the goals of creating feasible rotation assignments over a one-year period and constructing schedules for resident physicians to night and weekend shifts. In their work, official rules of duty hours for residents are considered. In [11], a genetic algorithm with a newly designed mutation operator is proposed for solving resident physician scheduling problem for their own model and dataset.

In [6], Guo et al provide complexity results for a basic and general residency scheduling problem. They prove that the problem is NP-complete and present polynomial special case. Beliën and Demeulemeester [2] solve trainees scheduling problem using a branch-and-price approach with several critical speed-up techniques. Compared with most brand-and-price algorithms that decompose on staff members, the decomposition scheme they use is for tasks. In [3], they compare two decomposition techniques to solve the same staff scheduling problem. For another rotation assignment problem, [1] is about job rotation scheduling appears in manufacturing industry where ergonomic and competence criteria are taken into account. In a similar work [10], a heuristic solution method is developed to solve a rotation scheduling problem in order to reduce the likelihood of low back injury.

Regarding the solution approach for solving staff scheduling problem, although a large number of methods are proposed [4], little attention is given to develop approaches for solving the model in this paper. To the best of our

Periods	Trainee 1	Trainee 2	Trainee 3
1	R_1	R_2	
2	R_1	R_2	
3		R_1	R_2
4		R_1	R_2
5	R_2	R_1	
6	R_2		R_1
7	R_2		R_1

(a) Schedule in normal form

Periods	Trainee 1	Trainee 2	Trainee 3
1	R_1	R_2	
2			
3		R_1	R_2
4			
5	R_2		
6			R_1
7			

(b) Schedule in block form

Fig. 2: Two equivalent solutions in two forms

knowledge, there is no work about heuristic algorithm for solving such rotation assignment problems.

3 Solution approach

3.1 Overview of the algorithm

Our algorithm uses two phases in order to produce feasible, high-quality schedules. The idea is that because rotations must be performed in consecutive periods, these periods can be seen as a block of this type of rotation. Throughout this paper, such groups of rotations of same type are called *rotation blocks*. Figure 2 presents a solution the same as in Figure 1 together with its corresponding view based on rotation blocks. In the following paper, a schedule is also referred to as a solution.

In the first phase, we optimize the schedule based on rotation blocks. Since the solution space is much larger than solutions only through this phase alone, in the next a problem-specific operator is employed in order to further explore solution space to search better schedules. These two phases are executed repeatedly until the running time limit is reached, and the algorithm will return the best solution found during the process. The overall algorithm is presented by the pseudo-code in Figure 3 and details of these two phases are described in following sections.

3.2 First phase

3.2.1 Periods Partition

In the first phase, in order to generate a feasible solution based on rotation blocks, rows(periods) of a solution are firstly grouped into blocks to reduce the whole periods into a new set. Note that each rotation block has to be performed by every trainee exactly once and in each period the blocks of the same type will not overlap due to the constraints (3). Given these reasons, in

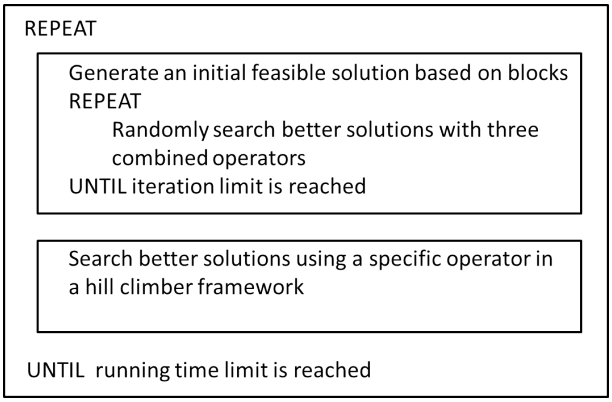


Fig. 3: Overview of the proposed algorithm

the whole time period, the number of blocks of each rotation must be equal to the number of trainees. So the whole periods are partitioned into m(number of trainees) groups. Let $i', i' = 1, ..., m$ be the new index of periods in blocks. To keep the solution feasible, in each block of period, rotations of all types must be performed by distinct trainees. To produce a solution in blocks, this partition should be fixed first.

Before we further describe how to partition the periods, conditions of feasibility to a problem instance are discussed. It is easy to observe that, for an instance, if there exists a feasible solution, then the value of n(total number of period) must be within the range of $[m \times l, m \times u]$. If n is larger than $m \times u$, there is not enough trainee to perform the rotation in additional periods since each rotation have been performed with the maximum number of periods. On the other hand, if n is smaller than $m \times l$, there is not enough period to complete the rotations for one or more trainees since at least one of rotations must cover less periods than l. Besides, another condition is that the number of rotations is always no larger than that of trainees. For a problem instance, if these conditions are satisfied, we call it a *feasible instance*. It turns out that for such an instance there must exist a feasible solution which can be obtained using a simple procedure, which is used to generate an initial solution at the beginning of first phase. In this paper, it is assumed that instances to be solved are always feasible since in real-life scenario, this can be guaranteed by the rule maker of the training program in school.

Assume we have an instance with period set $I = \{1, 2, ..., n\}$. We partition this set into a new form $\tilde{I} = \{a_1, a_2, ..., a_m\}$, where $a_{i'}, i' = 1, ..., m$ are subsets of I such that $\cup_{i'=1}^{m} a_{i'} = I$, $a_{i'} \cap a_{j'} = \emptyset, \forall i \neq j$, and $|a_{i'}| \in [l, u], \forall a_{i'} \in \tilde{I}$. While partitioning, the order and position of periods should keep unchanged. For example, when $I = \{1, 2, 3, 4, 5, 6, 7\}$, two possible partitions are $\tilde{I} = \{\{1, 2\}, \{3, 4\}, \{5, 6, 7\}\}$ and $\tilde{I} = \{\{1, 2\}, \{3, 4, 5\}, \{6, 7\}\}$, where $l = 2, u = 3$ and $m = 3$. Although the partition will influence quality of the final solution,

we cannot employ any fitness function to evaluate it at this stage and it is generated randomly.

3.2.2 Problem Reduction

After the partition configuration is fixed, the original problem is transformed into a simplified version, where the number of periods are reduced from n to m and the cells of schedule are also grouped together. The new problem can be modeled as follows and we call it reduced problem and the model introduced in the first section is referred to as original problem.

$$\text{Minimize} \quad \sum_{i'=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{r}\tilde{p}_{i'j}\tilde{x}_{i'jk} \tag{10}$$

Subject to:

$$\sum_{k=1}^{r}\tilde{x}_{i'jk} \leq 1, \forall i' = 1, ..., m \text{ and } \forall j = 1, ..., m, \tag{11}$$

$$\sum_{j=1}^{m}\tilde{x}_{i'jk} = 1, \forall i' = 1, ..., m \text{ and } \forall k = 1, ..., r, \tag{12}$$

$$\sum_{i'=1}^{m}\tilde{x}_{i'jk} = 1, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r, \tag{13}$$

$$x_{i'jk} \in \{0, 1\}, \forall i' = 1, ..., m, \forall j = 1, ..., m \text{ and } \forall k = 1, ..., r. \tag{14}$$

The decision variables have only one set defined as:

$$\tilde{x}_{i'jk} = \begin{cases} 1, & \text{in period } i' \text{ trainee } j \text{ perform rotation block of activity } k \\ 0, & \text{otherwise} \end{cases}$$

This model also tries to minimize the violated soft constraints. The parameter $\tilde{p}_{i'j}$ represents the new preference cost of non-available period i' for trainee j. It can be computed by letting $\tilde{p}_{i'j} = \sum_{i \in a_{i'}} p_{ij}$. Like the original model, constraints (11) make sure each trainee can perform at most one rotation. Constraints (12) ensure that in each period a rotation is performed by exact one trainee. Constraints (13) guarantee that throughout the whole period every rotation is performed by each trainee exactly once. Figure 4 presents an example of a feasible solution based on block forms and its corresponding original solution. It is clear that a feasible solution to reduced problem is also feasible to the original problem.

To deal with this new model, we use a random iterative search algorithm instead of the exact method. The reason is that the method in the next phase only searches the local optima in a deterministic way and we want to obtain different solutions from the first phase to diversify the search.

Periods	Trainee 1	Trainee 2	Trainee 3
1	R_1	R_2	
2		R_1	R_2
3	R_2		R_1

(a)A feasible solution of the reduced model

Periods	Trainee 1	Trainee 2	Trainee 3
1	R_1	R_2	
2	R_1	R_2	
3		R_1	R_2
4		R_1	R_2
5	R_2		R_1
6	R_2		R_1
7	R_2		R_1

(b)The corresponding solution of the original model

Fig. 4: In (a), the periods are partitioned to the form $\tilde{I} = \{\{1,2\},\{3,4\},\{5,6,7\}\}$, and the new preference costs are: $\tilde{p}_{31} = 2, \tilde{p}_{12} = 1, \tilde{p}_{22} = 1, \tilde{p}_{23} = 1$ and $\tilde{p}_{33} = 1$.

Algorithm 1: Overview of the first phase

```
   input  : A feasible instance
   output: A feasible solution
 1 Produce a feasible solution X;
 2 while iteration limit is not reached do
 3 |   X' = X;
 4 |   Generate b in [0,1] randomly;
 5 |   if b > 0.5 then
 6 |   |   Row_swap to X';
 7 |   else
 8 |   |   Column_swap to X';
 9 |   end
10 |   foreach violated preferences P_ij do
11 |   |   Partial_row_swap to X';
12 |   end
13 |   if X' is better than X then
14 |   |   X = X';
15 |   end
16 end
17 return X;
```

3.2.3 Overview of First Phase

In Algorithm 1, the first phase of the proposed algorithm is presented in detail. As shown, a feasible solution is firstly produced using a procedure presented in Algorithm 2.

After an initial solution is generated, better solutions are searched iteratively using three operators described as follows.

- *Row_swap*: two rows of a solution are randomly selected and swapped;
- *Column_swap*: two columns of a solution are randomly selected and swapped;
- *Partial_row_swap*: for two rows, if there exists a subset of columns(trainees) where they perform two equal subsets of rotations, then the rotations in such columns between these two rows are swapped. For instance, in Figure

Algorithm 2: Procedure of generating feasible solution

```
   input  : A feasible instance
   output: An initial feasible solution
1  k = 0;
2  for i' = 1 to m do
3  |   for j = 0 to m − 1 do
4  |   |   Assign R_{j+1} to T_{(k+x)(mod m)+1} at P_{i'};
5  |   end
6  |   k = k + 1;
7  end
```

Periods`	Trainee 1	Trainee 2	Trainee 3	Trainee 4	Trainee 5
1					
2	R_1	R_2		R_3	
3		R_1	R_2		R_3
4					

(a)Solution before the operation

Periods`	Trainee 1	Trainee 2	Trainee 3	Trainee 4	Trainee 5
1					
2		R_1	R_2	R_3	
3	R_1	R_2			R_3
4					

(b)Solution after the operation

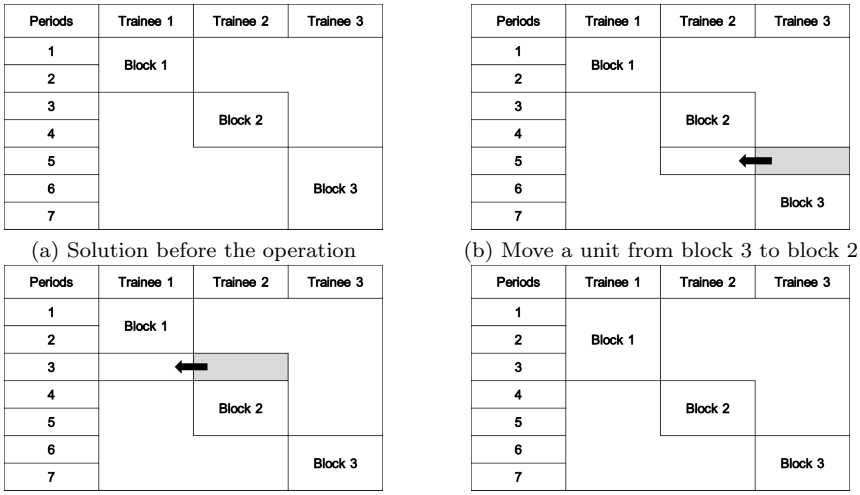
Fig. 5: In (a), the cost is 2 and in (b) the cost is reduced to 1 after *Part_swap_operator* is performed to T_1,T_2 and T_3 between P_2 and P_3.

5, trainees T_1,T_2 and T_3 performs R_1 and R_2 both in periods P_2 and P_3, and thus their rotations can be swapped between these two periods.

During one iteration of the search, the *row_swap* and *column_swap* are selected randomly with equal probability and perform to the current solution. Then all violated rotations are checked and use *part_swap_operator* to see if this violation can be eliminated. Note that the number of violated rotation is $O(m)$ and the possible swaps for each one is $O(m^2)$. All of these operators will not influence the hard constraints of a solution. Figure 5 is also an example of using the *part_swap_operator* to eliminate a violated assignment. When stopped, the best solution obtained enters the next phase. As for the stop criterion of this phase, we generate a number in $[10, 220]$ randomly and use this value to be the iteration limit of the first phase. Note that this parameter is fixed for all instances. It is obvious that the best solution of reduced problem could not be reached in this manner. If we fix it, however, same solutions could be obtained among different runs of first phase. Given that the second phase is not stochastic, this phenomenon could lead to stagnation in local optima. To avoid this, we use this stochastic mechanism to diversify the search for the second phase.

3.3 Second phase

Since the neighourhood in first phase is only based on rotations blocks, solution space of the original model is apparently larger than solutions of the



(a) Solution before the operation

(b) Move a unit from block 3 to block 2

(c) Move a unit from block 2 to block 1

(d) Solution after the operation

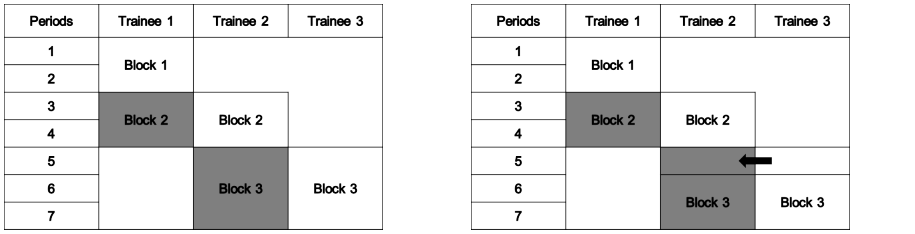
Fig. 6: *One_step_move* operator description. Here the three rotation blocks are of the same type.

reduced problem. In this phase, searching of a neighbourhood defined by a *move_and_repair* procedure is incorporated to further explore solution space. Notice that the periods are not viewed in blocks any more in this phase while consecutive rotations can still be seen as blocks. In this section we first describe the operator, then the overview of second phase is presented.

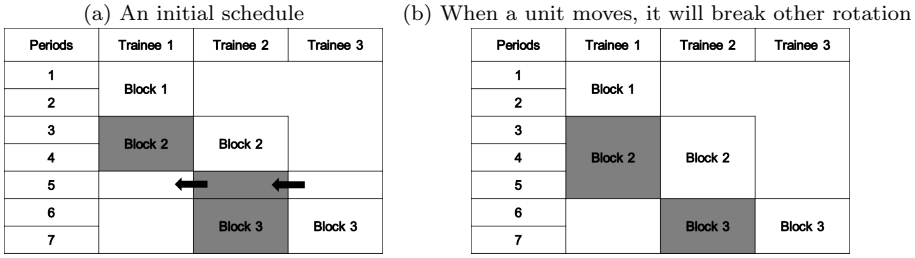
3.3.1 Operator: move_and_repair

The *move_and_repair* procedure consists of two steps. At the first step, an operator which aims to move rotations between blocks of rotations is defined. We call it *one_step_move* and it is always performed to blocks of the same type. Intuitively, this operator mimics the act of adjustment by a human scheduler to change the structure of a schedule when trying to find better ones. Figure 6 illustrates an example of this operator. In order to express the concept clearly, the blocks in the figure are of the same rotation type while others are omitted. In the following description, a single cell is also referred to as a unit.

As shown in the figure, a top unit of block 3 can be moved to the bottom of block 1 and the feasibility is still maintained if rotation blocks of other types are ignored temporarily. During this process, lengths of blocks between the two blocks will not change but the position will shift vertically. Implementation steps we use to carry out this operator are shown in (b) and (c), where a unit of assignment is moved to the bottom of the adjacent upper block iteratively until the top block is reached. It is worth noting that in our approach, we only consider the direction of move from bottom up.



(a) An initial schedule (b) When a unit moves, it will break other rotation



(a) Repair the replaced rotation (b) Solution after move and repair

Fig. 7: A case of broken rotation caused by the *one_step_move* operator and the repair process. Here we use deep color to denote R_j and white color to denote R_i. We find that the rotation in deep color is broken when the white color rotation moves. In (c), we use a identical move to repair the broken rotation.

To perform this operation, a pair of blocks of the same rotation type should be firstly located. For a rotation type $R_k, k \in \{1, ..., r\}$, we use $B_k(i), i = 1, ..., m$ to denote the ith(ordered from top to bottom) block of this rotation. Since each block stretches over a number of original periods, we call this number the *length* of a block and let $|B_k(i)|$ denote this value. Suppose there is a pair of blocks $B_k(p)$ and $B_k(q)$ such that $p < q$, if $|B_k(p)| + 1 \leq u$ and $|B_k(q)| - 1 \geq l$, this move can be performed and we call them an *available pair* and let $B_k(p, q)$ denote this pair. To find such a pair is straightforward. Indeed, we can loop through all blocks of a certain type of rotation and check all the pairs, which is $O(m^2)$, to see if it is an available pair. When this pair is found, a move act is performed $(q - p)$ times, as shown in (b) and (c) in Figure 6, to complete this operator.

An issue of this operator is that if blocks of other rotations are taken into account, this shift can influence assignments of other blocks. Since $(q - p)$ units of blocks will be moved, the positions taken up by blocks of other types of rotations will be replaced during this process. Consequently, at that period, the rotation being replaced is broken, and thus the schedule becomes infeasible. In Figure 7, (a) and (b) provide an example of this case between two blocks R_i and R_j, where the same move as in Figure 6 (b) is performed to R_i. When such situation occurs, the second step of *move_and_repair* is employed which aims to repair this broken point.

In order to fix this, we can perform a similar move to pairs of blocks of R_j which contains the broken period. As shown in Figure 7 (c), we find *Block* 2

and *Block* 3 of rotation R_j(deep color) are also an available pair, so we can perform another *one_step_move* to them, which can still cause another broken rotation. We continue using this procedure iteratively until no such broken rotation exists so that the solution becomes feasible again. To implement this process, we can maintain a set V to record the indices of broken rotations and check it when we repair a rotation. When the set becomes empty, which means no violation exists, the *move_and_repair* operator is successfully performed.

If a feasible solution can be obtained, the *move_and_repair* procedure returns $True$. In some cases, however, there are no available pair to fix the broken rotation, then the initial *one_step_move* operator is discarded and the whole procedure will yield $false$. Since we only consider the direction from bottom up, cycling never happens.

3.3.2 Algorithms of Second Phase

Algorithm 3 presents overview of the second phase of our algorithm. Note that the input solution is the one generated from the first phase. We use *move_and_repair* for a hill climber. We check for all rotations, and for each one we loop through all available pairs to try the operator. If a better solution is found, we start over the whole procedure from this better solution. If either the *move_and_repair* does not success, or the solution does not become better, the changed solution will be recovered and we continue to try the next pair.

Algorithm 3: Overview of second phase

```
   input  : A solution X generated by the first phase
   output: An optimized solution
 1 foreach R_i of X do
 2 |   foreach feasible pairs of block B_i(p,q) of X do
 3 |   |   X' = X;
 4 |   |   if move_and_repair to B_i(p,q) of X' return True then
 5 |   |   |   if X' is better than X then
 6 |   |   |   |   X = X';
 7 |   |   |   |   goto step 1;
 8 |   |   |   end
 9 |   |   end
10 |   end
11 end
12 return X';
```

Algorithm 4 describes the procedure of *move_and_repair*. It maintains a set V and uses *one_step_move* to the available pair. Then, all broken rotation indices are extracted and moved into V. While V is not empty we use a *repair* procedure with the solution together with V as inputs. Algorithm 5 presents this procedure. In step 2 of this procedure, the algorithm tries to find an available pair involving the broken periods by looping through all pairs of the broken rotation. Note that there could be more than one available pairs that

can repair the broken rotation and we let the process stop when the first one is found.

Algorithm 4: Procedure of *move_and_repair*

```
   input  : Solution X and B_i(p, q)
   output: True or False
1  Create an empty set V;
2  One_step_move to B_i(p, q) of X;
3  Search all broken rotations, add their indices into V;
4  while V is not empty do
5  |  if repair to X is not True then
6  |  |  return False;
7  |  end
8  end
9  return True
```

Algorithm 5: Procedure of *repair*

```
   input  : Solution X and set V
   output: True of False
1  foreach i in V do
2  |  if a feasible pair B_i(p, q) which can repair i can be found then
3  |  |  One_step_move to B_i(p, q) of X;
4  |  |  Search all broken rotations, add their indices into V;
5  |  |  return True;
6  |  else
7  |  |  return False;
8  |  end
9  end
```

We have to mention that there exist other cases that procedure *repair* fails to fix. For instance, when a rotation has two or more consecutive broken periods, more than one *one_step_move* to disrupted rotations are required. To tackle this, we make the procedure return *False* to stop when this case appears. Empirically, the probability of this case is quite low.

4 Computational results

4.1 Problem instances

To test our algorithm, we solved randomly generated instances based on rules of training program and the history data of five years from school of nursing in Shandong University. A trainee in the model described in this paper might represent a group of students since in practical scenarios several trainees are

grouped together as a unit to complete the program. In this case, the undesired periods of a unit naturally consist of those of each person. Without loss of generality, we still use trainee instead of trainee unit throughout the paper. Training program in the school has two types according to the number of trainees. Each type also has two subtypes according to number of rotations. Table 1 presents the specific values of these parameters for each type. For the number of whole periods, since the real-life data of this parameter can vary among years, we select several typical values and all the instances are feasible. For an instance, the number of non-available period of each trainee is often fixed(about 20% of the whole number of periods), yet the specific indices could vary among years. Given this characteristic, instances are generated with randomness for this factor. Indices of undesired periods are generated uniformly in the range of the whole period. We generate three problem instances per factor setting, and thus the total number of the instances is $2 \times 2 \times 5 \times 3 = 60$. A single period in the original model represents a week in real life, and a rotation is usually performed for about one month based on the rule. For this reason, l and u are set to 3 and 5 for all instances.

Table 1: Parameter values based on real-life rules for experiments

Type I		Type II	
Trainees	Rotations	Trainees	Rotations
8	5, 6	12	9, 10
Total periods(undesired periods)		Total periods(undesired periods)	
26(5),29(6),32(7),35(7),38(7)		40(8),44(9),48(9),52(10),57(11)	

4.2 Experimental protocol

Although in [2], a branch-and-price algorithm is proposed to solve the general version of this problem, since the source code is not public and the instances are different either, we compare our algorithm with Cplex using the integer linear programming formulation(ILP) provided in Section 1. All instances are solved by Cplex 12.4.0 and the proposed algorithm is implemented in C++ using Visual Studio 2008 and run on Inter(R) Pentium(R) G3220 @3.00GHz with 4.00GB RAM under the Windows 7 OS. For Cplex, we set the time limit to 3600 and 7200 seconds for two types of instances respectively. For the proposed algorithm, we set 18 seconds for both types of instances. Since our algorithm is random in nature, we run 20 times for each instance. There are no other parameters to be tuned in our algorithm.

4.3 Results and comparison with ILP formulation

Table 2 presents the results of our algorithm and the comparison with ILP formulation. The first 4 columns identify the instances according to the parameter values with No.01 $\sim$ 60, where No.01 $\sim$ 30 and No.31 $\sim$ 60 belong to type I and type II, respectively. Next two columns show the best solution and lower bound found within the given computation time using ILP formulation. The remaining columns provide statistics of minimum, maximum, average and standard deviation values of the results achieved by our approach.

We can see that, for instance of type I(No.01 $\sim$ 30), only one instance is solved to optimality with ILP formulation. Moreover, being given 200 times less computation time, our algorithm can yield equal or better solutions for 23 out of 30 instances, which are indicated in bold, demonstrating the efficacy of our algorithm. For those which are worse from our algorithm, best values are also very close to those from Cplex. For instances of type II(No.31 $\sim$ 60), no feasible solution can be produced within the time limit for all instances and only lower bounds are provided. Our algorithm can produce feasible solutions for all cases. Regarding the stability, standard deviations are more than 1.00 for only 3 instances out of 60. We conclude that our algorithm can reach high quality solutions and is stable for these instances.

4.4 Investigation of the second phase

As indicated in previous section, the second phase of our algorithm employs a *move_and_repair* operator to further explore the solution space. Contribution of this procedure is investigated through additional experiments in this section. We compare the two-phase algorithm with the one without second phase. The experiment protocol is the same as above except that in the first phase, we fix its iteration limit to 200 and the whole algorithm only contains calling this phase iteratively. Experiments are carried out on a type I instance No.11, since similar results are observed on other instances.

Figure 8 shows how the best objective value(averaged over 20 independent runs) evolves with the iterations. We observe that both algorithms converge very quickly at the beginning of the process, but the one only with the first phase cannot improve solutions afterward. For the original algorithm, it could obtain better solutions continuously as the search progresses. This experiment provides an empirical justification of the contribution of the second phase in our approach.

5 Conclusions and future research

In this paper, a class of trainee rotation assignment problem appears in local school of nursing is addressed. A two-phase heuristic approach is proposed to tackle this problem. In the first phase, due to the consecutive constraints,

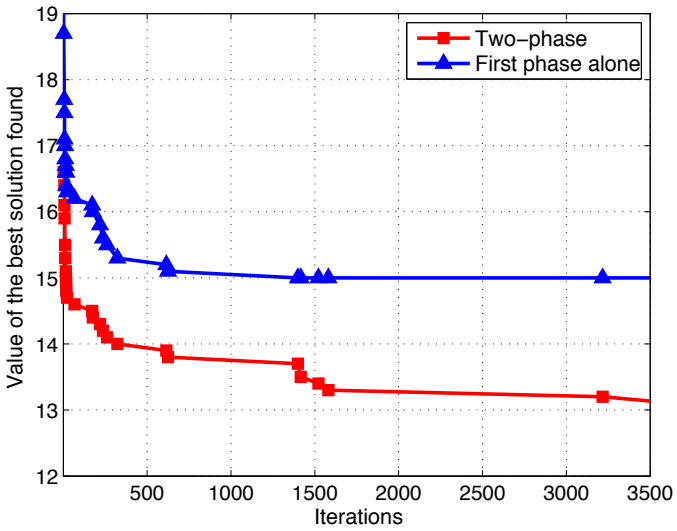


Fig. 8: Significance of the second phase

rotations in a row are grouped into blocks and a simpler assignment problem is solved by a random search algorithm. In the second phase, we use a hill climber to search a neighbourhood defined by a problem-specific operator which uses a *move_and_repair* procedure to complement the first phase. We test our algorithm with instances generated on the basis of rules in real-life training program in the local school. Compared with Cplex in integer linear programming formulation, our heuristic approach can obtain high-quality solutions with much less computation time.

Our future research intentions are twofold. Firstly, besides hill climber, we want to investigate more usage strategies of our *move_and_repair* operator. For instance, in this proposal the operator being used only move one unit between a pair of blocks. This is because the gap between l and u being used in our local setting are very small. When it gets larger, intuitively, more units should be involved in a single procedure. Secondly, since our model and benchmark instances all come from real-life cases, it is interesting to extend our algorithm for solving more generalized trainee rotation scheduling problem as well as other discrete optimization problems.

References

1. Asensiocuesta, S., Diegomas, J.A., Canósdarós, L., Andrésromano, C.: A genetic algorithm for the design of job rotation schedules considering ergonomic and competence criteria. International Journal of Advanced Manufacturing Technology **60**(9-12), 1161–1174 (2012)

2. Beliën, J., Demeulemeester, E.: Scheduling trainees at a hospital department using a branch-and-price approach. European Journal of Operational Research **175**(1), 258–278 (2006)
3. Beliën, J., Demeulemeester, E.: On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. Annals of Operations Research **155**(1), 143–166 (2007)
4. Bergh, J.V.D., Beliën, J., Bruecker, P.D., Demeulemeester, E., Boeck, L.D.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367–385 (2013)
5. Franz, L.S., Miller, J.L.: Scheduling medical residents to rotations: Solving the large-scale multiperiod staff assignment problem. Operations Research **41**(2), 269–279 (1993)
6. Guo, J., Morrison, D.R., Jacobson, S.H., Jokela, J.A.: Complexity results for the basic residency scheduling problem. Journal of Scheduling **17**(3), 211–223 (2014)
7. Javeri, M.A.: Rotation planning and scheduling for medical resident education. Master's thesis, Purdue University,Department of Industrial Engineering (2011)
8. Ozkarahan, I.: A scheduling model for hospital residents. Journal of Medical Systems **18**(5), 251–265 (1994)
9. Smalley, H.K., Keskinocak, P.: Automated medical resident rotation and shift scheduling to ensure quality resident education and patient care. Health Care Management Science pp. 1–23 (2014)
10. Tharmmaphornphilas, W., Norman, B.A.: A methodology to create robust job rotation schedules. Annals of Operations Research **155**(1), 339–360 (2007)
11. Wang, C., Sun, L., Jin, M., Fu, C., Liu, L., Chan, C., Kao, C.: A genetic algorithm for resident physician scheduling problem. In: Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007, pp. 2203–2210 (2007)

Table 2: Results and comparison with Cplex using ILP formulation

Instance				ILP formulation		Proposed algorithm			
No.	Trainees	Rotations	Periods	best	lower bound	min	max	avg	std
1	8	5	26	8	3.67	8	8	8.00	0.00
2				7	7	**7**	7	7	0.00
3				7	5.67	**7**	8	7.75	0.43
4	8	5	29	5	1.13	**5**	7	5.95	0.67
5				8	4	9	11	10.15	0.57
6				11	7	12	13	12.65	0.48
7	8	5	32	10	6	11	13	12.50	0.59
8				12	3.42	**12**	14	13.10	0.54
9				9	2.14	11	12	11.70	0.46
10	8	5	35	12	4	**12**	13	12.80	0.40
11				14	4	**13**	14	13.20	0.40
12				12	5	**12**	14	13.45	0.59
13	8	5	38	16	1.82	**14**	15	14.25	0.43
14				20	1	**11**	13	12.20	0.60
15				20	3.57	**15**	17	16.00	0.32
16	8	6	26	13	9.15	14	15	14.10	0.30
17				16	6	**14**	15	14.80	0.40
18				16	4	**13**	14	13.30	0.46
19	8	6	29	17	10.22	**16**	17	16.50	0.50
20				18	10.80	**15**	16	15.95	0.22
21				16	11	**16**	17	16.75	0.43
22	8	6	32	22	12	**20**	21	20.60	0.49
23				22	14	23	24	23.80	0.40
24				28	16.18	**24**	26	24.95	0.38
25	8	6	35	26	10	**22**	24	22.95	0.59
26				25	15	**23**	24	23.70	0.46
27				22	12	**20**	22	20.95	0.59
28	8	6	38	27	12	**23**	24	23.85	0.36
29				32	11	**21**	22	21.95	0.22
30				34	11	**23**	24	23.15	0.36
31	12	9	40	–	16	33	35	34.10	0.62
32				–	14	31	33	32.55	0.59
33				–	15	32	34	33.10	0.70
34	12	9	44	–	11	36	40	38.60	1.07
35				–	23	43	46	45.00	0.89
36				–	19	42	43	42.85	0.36
37	12	9	48	–	11	38	41	39.10	0.62
38				–	13	39	41	40.35	0.57
39				–	11	39	43	41.30	1.14
40	12	9	52	–	11	46	49	47.95	0.86
41				–	8	45	48	46.50	0.67
42				–	8	43	46	44.65	0.73
43	12	9	57	–	18	57	60	58.95	0.74
44				–	20	59	61	60.50	0.59
45				–	15	56	58	56.85	0.48
46	12	10	40	–	27	47	49	48.50	0.59
47				–	35	48	49	48.65	0.48
48				–	27	46	48	47.50	0.59
49	12	10	44	–	37	53	55	53.95	0.59
50				–	37	56	59	58.25	0.83
51				–	35	55	56	55.65	0.48
52	12	10	48	–	36	54	57	55.40	0.80
53				–	36	59	60	59.60	0.49
54				–	29	55	59	56.95	1.02
55	12	10	52	–	38	65	67	65.65	0.65
56				–	33	64	67	65.45	0.80
57				–	35	61	64	62.50	0.67
58	12	10	57	–	45	79	80	79.80	0.40
59				–	43	74	76	75.30	0.64
60				–	46	75	77	76.00	0.45