Comparing Three Improved Variants of Simulated Annealing for Optimizing Dorm Room Assignments

Nguyen Thanh Trung and Duong Tuan Anh
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology, Vietnam
dtanh@cse.hcmut.edu.vn

***Abstract*—Assigning dorm rooms to 2500 students with complex and interdependent preferences is a difficult optimization problem. In this paper, we present an optimization method which is based on an improved variant of simulated annealing for the whole dorm room assignment problem. We also compare empirically the performance of three different improved variants of simulated annealing in solving this problem: Informed Simulated Annealing, Very Fast Simulated Re-Annealing and Simulated Annealing with Non-monotonic Reheating.**

Keywords - dorm room assignment; simulated annealing .

I. INTRODUCTION

The task of assigning students to dorm rooms, taking into account their preferences is a formidable task by hand and is not trivial to automate. For example, at the new dormitory of Ho Chi Minh City (HCMC) University of technology, every semester, appropriately 2500 students are assigned to 2500 beds in such a way that all student preferences and management policy constraints should be satisfied as much as possible. Consequently, dorm room assignment for a dormitory of a large university is a difficult combinatorial optimization problem.

Simulated annealing (SA) is one of the well-known meta-heuristics for solving complex constraint satisfaction optimization problems ([2]). Simulated annealing is a variant of local search algorithm which allows non-improving moves to be accepted in a controlled manner. It has been applied successfully in solving several real world applications.

There have been several approaches to improve the efficiency of simulated annealing. Some of them are listed as follows. Elmohamed et al. investigated simulated annealing with different cooling schedules ([1]). Osman proposed a variant of simulation annealing, called simulated annealing with non-monotonic reheating which divide the search in several phases and allows temperature to be increased to escape local minima at low temperatures ([6]). Poupaert et al proposed a new simulated annealing variant using acceptance probability as control parameter of the search ([7]). Ingber proposed a version of SA called Very Fast Simulated Re-Annealing which permits an annealing schedule for temperature decreasing exponentially ([3]). Miki et al proposed a simulated annealing variant with adaptive neighborhood ([5]). Li developed an enhanced variant of SA, called Informed Simulated Annealing which uses the information collected during the annealing search to direct the search ([4]). In summary, there are three main approaches to improve SA. One is determining the cooling schedule properly. SA with proper cooling schedule can provide an optimum solution quickly. The second approach is to choose a proper neighborhood. The third approach is using a learning mechanism to collect information during the annealing search and use the information to direct the search towards near optimal solution.

There has been one research work that applied SA to dorm room assignment problem (DRAP): the work done by E. Settanni ([8]) for the dormitory at the University of New Mexico. In the project, the author used only standard simulated annealing with some traditional cooling schedules for solving DRAP.

In this work, we propose an optimization method which is based on an improved variant of simulated annealing and investigate three different improved variants of SA to solve the DRAP. They are Simulated Annealing with Reheating, Very Fast Simulated Re-Annealing and Informed Simulated Annealing. We perform experiments of each of the algorithms on real datasets from the dormitory at HCMC University of Technology.

II. THE DORM ROOM ASSIGNMENT PROBLEM

The choices that describe the DRAP are a combination of student requests and management regulations. Students specify their preferences for halls, room types, music tastes, study habits, and specific requests for roommates on their application forms. The dorm room assignment system is responsible for assigning students to their beds each semester in such a way that all student preferences and management regulations should be satisfied as much as possible. The generation of a dorm room assignment requires assigning a given set of students to a set of beds so as to satisfy a given set of constraints. There are two kinds of constraints: hard constraints that must be satisfied, and soft constraints that should be satisfied to improve the solution.

A. The Set of Constraints

Hard constraints. At the dormitory of HCMC University of Technology, there are four constraints of this kind, mainly from management regulations.

HC1. *Gender restriction*. The dorm is divided into two quarters: one quarter for male students and the other for female students. So each student must be assigned to a room designated properly for his or her gender.

HC2. *Room capacity restriction*. Depending on room type, a room can accommodate only a limited number of occupants.

HC3. *Privileged students*. The students who are disabled soldiers, or children of war heroes are privileged students. The requests from these students are considered with high priority.

HC4. *Room change request*. Senior students who are living in the dorm may request to change to better rooms. The requests from these students are considered with high priority.

Soft constraints. There are two main kinds of soft constraints: room-student constraints and student-student constraints.

i) Room-student constraints take student and room attributes into accounts. For example, a room-student mismatch occurs when a student's hall does not match the hall of his or her choice.

ii) Student-student constraints are concerned only with the students in the room. Student-student constraint violations occur when the occupants of the room do not agree, for example, a student who prefers traditional music is in the same room as someone who objects to this kind of music.

*Room-student Constraints***.** When a student's preference about room is not satisfied, a constraint violation occurs. Some examples of the violations of this constraint group and their penalty computations are as follows:

- Unsatisfied preference about hall: w*ERR_HALL
- Unsatisfied preference about room type: w*ERR_ROOM
- Unsatisfied preference about bed (upper or lower): w*ERR_BED
- Unsatisfied preference about rent price: w*ERR_PRICE
- Unsatisfied preference about floor: w*ERR_FLOOR

  where *w* is the priority rate for senior students and students with early reservation. *w* is calculated as follows:

  senior*SENIORITY_WEIGHR + k

  (*senior* is the number of semesters residing in the dormitory and *k* is priority level for early reservation, in the range from 0 to 10).

- Unsatisfied room change request: This constraint violation occurs if a senior student that wants to change his or her room has been assigned to the same room. The penalty of this constraint violation is computed as follows:

  # of students * ERR_SAME_ROOM

Room-student (RS) constraints are divided into two subgroups: subgroup RS1 that consists of constraints regarding to management regulations and subgroup RS2 that consists of constraints regarding to room-attribute preferences.

Student-student constraints. When a student preference about roommates is not satisfied, a constraint violation occurs. Some examples of the violations of this constraint group are as follows:

- Unsatisfied music preference: This constraint violation occurs if one student objects to the music that is preferred by another student. The penalty of this constraint violation is computed as follows:

  min(# students who like, # students who dislike) * ERR_MUSIC

- Unsatisfied study habit: This constraint violation occurs if a late studier mixes with non-late studiers in a room. The penalty of this constraint violation is computed as follows:

  min(# of late studiers, # of non-late studiers) * ERR_STUDY_LATE

- Unsatisfied preference about computer game: This constraint violation occurs if a student that dislikes playing computer game has to mix with a student that likes playing it. The penalty of this constraint violation is computed as follows:

  min(# of students who like, # of students who dislike) * ERR_PLAY_GAME.

- Unsatisfied friend preference: This constraint violation occurs if the friend one student registers to be his or her roommate is not assigned to share the same room. The penalty of this constraint violation is computed as follows:

  w*ERR_FRIEND*# of unsatisfied students.

- Unsatisfied preference about same department roommates: This constraint violation occurs if one student's request about roommates from the same department is not satisfied. The penalty of this constraint violation is computed as follows:

  w* # of students of different departments * ERR_DIFF_DEPART

- Unsatisfied preference about same hometown roommates: This constraint violation occurs if one student's request about roommates from the same native province is not satisfied. The penalty of this constraint violation is computed as follows:

  w* # of students of different home towns * ERR_DIFF_HOMETOWN

Student-student (SS) constraints are divided into two subgroups: subgroup SS1 which consists of constraints regarding to personal preferences and subgroup SS2 which consists of constraints regarding to room-mate preferences.

B. *The Cost Function*

The DARP consists in minimizing the objective function which iterates over all the students and sums up their "unhappiness". The objective function f is computed as follows:

$$f(e) = \sum_{s=1}^{m} \Big(\sum_{j \in RS1} \mathrm{RSerr}_j(e,s) + \sum_{j \in RS2} \mathrm{RSerr}_j(e,s) \Big) +$$
$$\sum_{r=1}^{n} \sum_{s=1}^{t} \Big(\sum_{j \in SS1} \mathrm{SSerr}_j(e,s,r) + \sum_{j \in SS2} \mathrm{SSerr}_j(e,s,r) \Big)$$

where e is the current state of dorm room assignments, m is the number of students, n is the number of rooms, t is the number of students in a room, *RSerr* is the penalty from room-student constraints, *SSerr* is the penalty from student-student constraints, *RS1* and *RS2* are the two subgroups of room-student constraints, SS_1 and SS_2 are the two subgroups of student-student constraints.

Since the objective function is the sum of errors in a state of dorm room assignments, it is also referred as a *cost function* which can be used to measure how distant a solution is from the optimal solution.

III. SIMULATED ANNEALING WITH NON-MONOTONIC REHEATING

In geometric cooling scheme, the temperature is only ever reduced. Once the temperature reaches a very low level, the system will not be able to accept cost increases that are sufficiently large. However, these large increases may be required in order for the SA to proceed to the global optimum. Thus, no matter how slow the cooling rate, from that point on, the algorithm will always deliver a local minimum. The only solution to this problem is to detect that a local minimum has been reached and to *reheat* the system, thus allowing it to escape from local minimum.

There are various reheating schemes that have been proposed. Here, we use the reheating scheme called *non-monotonic reheating*. The basic idea of non-monotonic reheating is that whenever the occurrence of a local minima is detected, this scheme attempts to escape from it by resetting the temperature to twice of the temperature at which the best solution was attained.

This scheme (abbreviated by SA_REHEAT) starts from an initial temperature, which is decremented after each Markov chain according to a standard temperature decrement rule. When, at an iteration k, a local minimum is detected (a cycle of search is made in which no transition is accepted), the temperature at the iteration k+1, T_{k+1}, is updated according to the following rule: $T_{k+1} = 2T_b$. Here, T_b is the temperature corresponding to the best solution obtained. The algorithm stops when a certain number of reheats have been performed.

IV. VERY FAST SIMULATED RE-ANNEALING

The standard simulated annealing needs a lot of time to converge to the optimal solution. It is well-known that the standard SA requires a cooling schedule in which the temperature T must satisfy the following equation:

$$T_k = T_0/\ln(1+k) \qquad (1)$$

where T_k is the temperature at the k-th iteration and T_0 is the initial temperature.

Equation (1) characterizes the simplest cooling scheme for SA which is also called Boltzmann annealing. Since k must become very large for T_k approaching to zero, the equation can explain why this cooling scheme is always slow. Ingber ([3]) proposed a variant of SA which is called Very Fast Simulated Re-Annealing (VFSA). In VFSA algorithm, the neighborhood function is the product of all the neighborhood in every parameter dimension. If the optimization problem consists of D parameters, it will have D dimensions. Ingber advocates a cooling schedule for VFSA in which the temperature T decreases exponentially in annealing time. That means for the dimension i, the cooling schedule must satisfy the equation:

$$T_i(k) = T_{0i}\exp(-b_i k^{1/D}) \qquad (2)$$

where D is the number of parameter-dimensions in the problem, i is the parameter-dimension under consideration and b_i is a constant.

From the equation (2), we can see that the cooling schedule in VFSA can be faster than the Boltzmann annealing characterized by the equation (1).

V. INFORMED SIMULATED ANNEALING

Informed Simulated Annealing (ISA), proposed by Y. Li, is an improved variant of simulated annealing ([4]).

Any constraint satisfaction optimization problem (CSOP) consists of a number of variables and a number of associated values for each variable. A *variable-value pair* represents the assignment of a value to a variable. Suppose there are m variables and each variable has n values. Then v_{ij} is the variable-value pair consisting of the jth value in the domain of the ith variable (where $0 < i \le m$ and $0 < j \le n$). A solution of a CSOP is a set of variable-value pairs which satisfy all the hard constraints and as many soft constraints as possible.

A. *Utilities of Variable-Value Pairs*

The *utility* of a variable-value pair is an estimation of the probability that the variable-value pair appears in an optimal solution. A straightforward definition of utilities can be based on the *frequency* with which a variable-value pair appears in new current solutions, i.e. the solutions accepted after repairs. Let m be the number of repairs in which a variable-value pair v_{ij} is involved. It can be involved in two ways, either as the value being repaired or through the value being substituted. Now let n be the number of times that v_{ij} appears in the new current solution. Then the desired probability is n/m.

B. *Utilities in Practical Annealing Search*

A utility which is established in practical annealing search is *significant* if it can be used to improve on the best solution found by standard SA. Given an ordering of variable-value pairs for each variable, from the higher to the lower, the task is to find a *threshold* which divides the significant utilities for each variable from the non-significant ones.

The set of variable-value pairs distinguished as significant by a threshold serves to point to solution better than those found in standard SA. The basic strategy is to *bias* the search in favor of the set of significant pairs. This is achieved through the neighborhood operator, which *repairs* solutions. At a certain point in the search, the choice of value used to repair the assignment of a variable is biased towards those which appear in the significant pairs for that variable.

C. *Implementing Informed Simulated Annealing*

The procedure is that we select a specific threshold, represented by a positive integer *n*, which is to be used as a criterion for assessing the utilities of variable-value pairs. For each variable we take the top *n* utilities of the variable-value pairs for that variable to form a set of variable-value pairs, which is called the *bias set*.

In directed search there are two sets of variable-value pairs which are relevant to each variable, (1) those which belong to the bias set for that variable, and (2) the set of all possible pairs for that variable. Both are used in the repair process of ISA.

In the basic repair procedure, there are two basic steps: (1) randomly choose a variable and randomly select a value to replace its current assignment; (2) maintain feasibility by repairing those other variables whose current assignments violate constraints. In ISA step (1) will be modified. At a certain point in the schedule, *ISA will bias the selection of repair values to those which occur in pairs belonging to the bias set.*

To constrain the degree of bias, ISA uses two new probabilities: the first, referred to as P_1, fixes *the degree to which search is biased towards the bias set*; the value of P_1 is fixed from the beginning. The second, referred to as P_2, is used to decide the choice set on a given occasion, i.e., whether it is the bias set or the set of all possible values; this probability is randomly generated during repair. The procedure for applying these probabilities is as follows. At each repair the algorithm randomly chooses a variable, say V_i, and then generates a value for P_2; if it less than P_1, the new variable-value pair v_{ij} must be selected from the bias set, otherwise v_{ij} is selected from the set of all possible values of variable V_i.

One key issue in directing search concerns where the search becomes directed. In this work, we use the following implementation of ISA. The algorithm uses the whole of the basic annealing search to build up the utilities of the variable-value pairs. It then assesses these utilities and initiates further search using a new schedule. This search is directed towards the bias set.

VI. SOME OTHERS ISSUES

The neighborhood operator we use in this work is described as follows. From a current state of room assignment s_0, a student is selected randomly, then another student of the same gender is also selected randomly, and their beds are swapped. By that move, we obtain a neighbor state *s* for the current state s_0.

The approach we take to the DRAP at the dormitory of HCMC University of Technology consists of two stages:

1. Phase I: to obtain an initial state of dorm room assignment which satisfies all the hard constraints (HC1, HC2, HC3, and HC4). The algorithm used in the first phase is just a simple greedy algorithm.

2. Phase II: to improve the quality of the initial dorm room assignment state, taking the soft constraints into account. The method used in the second phase is optimization method which will seek to optimize the given objective function. We used each of the three improved variants of SA in this optimization phase.

VII. EXPERIMENTAL RESULTS AND DISCUSSIONS

We implemented the dorm room assignment system with Microsoft Visual C# and conducted most of the tests on a Pentium IV 3 GHz PC with 1 GB RAM. The real dataset is from the dormitory at the HCMC University of Technology. The data set consists of 2500 students and about 2500 beds. For each student, there are 4 hard constraints and 19 soft constraints gathered from student dorm registration forms. So the system has in total about 9000 hard constraints and 57000 soft constraints to be considered. The three SA improved variants we explored are ISA, SA_REHEAT and VFSA. As for ISA (and Standard SA), after some experiments with different parameter values, we used the following parameters: T_0 = 10000, T_f= 0.0001, *nrep* = 6 (*nrep* is the number of iterations at each temperature level) and *P1* = 0.75. As for SA_REHEAT, we used T_0 = 10000, T_f = 0.0001 and *maxReheats* = 5 (*maxReheats* is the maximum number of reheatings). For VFSA, the number of dimensions *D* is of value 2 since in the DRAP, we have 2 dimensions: students and beds.

Table 1 reports the change of the cost function value along with the number of iterations. These results allow some comparisons can be made between SA improved variants. It can be seen that among the three tested SA variants, SA_REHEAT has been robust and capable of bringing out the lowest cost solution in an acceptable run time. In other words, in terms of solution quality, SA_REHEAT is the best performer. On the contrary, it can be seen that among the three SA improved variants, ISA produces the lowest quality solution.

TABLE 1. A COMPARISON OF COST FUNCTION LEVELS ALONG WITH THE NUMBER OF ITERATIONS

	1000	2000	5000	10000
Initial cost value	7692992	7692992	7692992	7692992
Final cost value (ISA)	6401015	5828375	5165973	4873260
Final cost value (SA_REHEAT)	5310702	5002136	4826315	4770025
Final cost value (VFSA)	5768564	5258280	4913792	4850245
Final cost value (Standard SA)	6210856	5512860	4920234	4901568

A comparison of run times for different numbers of iterations through the tests on real dataset are given in Table 2. According to the experiments, the ISA is the best performer in terms of time efficiency. After 1000 iterations, ISA brings out a good quality solution with only about 7 minutes. In terms of run times, SA_REHEAT is the slowest of the three SA variants. Beside the two important criteria for evaluating an optimizing algorithm, time efficiency and solution quality, we also concern about another aspect: the ease of implementation. Among ISA, VFSA and SA_REHEAT, we found out that the SA_REHEAT is the easiest to implement and VFSA is the most complicated to implement.

TABLE 2. A COMPARISON OF RUN TIMES (IN SEC)

Number of iterations	1000	2000	5000	10000
Run time(ISA)	33	71	229	433
Run time(SA_REHEAT)	124	245	636	2512
Run time (VFSA)	103	215	581	1005
Run time (Standard SA)	103	213	592	1015

From the experiments, we can come to the following conclusion. For the DRAP, if less solution time is available, ISA is preferable. However it needs the full run in order to converge to good solutions. If the quality of solution is vital, then SA_REHEAT has been shown to be robust and capable of achieving solutions of highest quality in reasonable time. VFSA seems unsuitable to be applied to DRAP due to its unstable performance, i.e., sometimes it runs fast and sometimes it takes a long time to get an acceptable solution.

ACKNOWLEDGMENT

This research has been financially supported by Vietnam National University of Ho Chi Minh City.

REFERENCES

[1] M.A. S. Elmohamed, G. Fox, P. Coddington, "A Comparison of Annealing Techniques for Academic Course Scheduling", In: *Proc of 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT'97)*, Syracuse, NY, USA, Apr. 1997, (Eds. E. Burke and M. Carter), LNCS, Springer, 1998,146-166.

[2] F. Glover & G.A. Kochenberger, *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2003.

[3] L. Ingber, "Very Fast Simulated Re-annealing", Mathl. Comput. Modeling Vol 12, no. 8, 967-973, 1989.

[4] Y. Li, "Directed Simulated Annealing In Constraints Satisfaction And Optimization", Ph.D. Thesis, Imperial College of Science, Technology and Medicine, University of London, October 1997.

[5] M. Miki, T. Hiroyasu, K. Ono, "Simulated Annealing with Advanced Adaptive Neighborhood", 2002.

[6] I. H. Osman, "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem", *Annals of Operations Research*, 41:421-451, 1993.

[7] E. Poupaert, Y. Deville, "Simulated Annealing with estimated temperature", *AI Communications*, Vol 13, 2000, 19-26.

[8] E. Settanni, "Improving Dorm Room Assignments Using Simulated Annealing", Master thesis, Dept. of Computer Science, The University of New Mexico, Albuquerque, New Mexico, Dec. 2000.