Random Partial Neighborhood Search for University Course Timetabling Problem

Yuichi Nagata[1] and Isao Ono[2]

[1] Education Academy of Computational Life Sciences,
Tokyo Institute of Technology, Japan
[2] Institute of Technology and Science,
The University of Tokushima, Japan
nagata@is.tokushima-u.ac.jp, isao@dis.titech.ac.jp

Abstract. We propose an tabu search algorithm using an candidate list stratety with random sampling for the university course timetabling problem, where the neighborhood size can be adjusted by a parameter *ratio*. With this framework, we can control the trade-off between exploration and exploitation by adjusting the neighborhood size. Experimental results show that the proposed algorithm outperforms state-of-the-art algorithms when the neighborhood size is set properly.

1 Introduction

To solve optimization problems that are computationally intractable, heuristic (approximation) algorithms have been widely used for finding nearly optimal solutions in a reasonable computation time. In particular, neighborhood search is a wide class of heuristic algorithms, where the current solution is iteratively moved to a solution in the neighborhood at each iteration.

Crucial issues in the design of an effective neighborhood search algorithm are the choices of the neighborhood structure and search strategy. The neighborhood is defined as a set of solutions that are obtained typically by performing prearranged local modifications on the current solution. The choice of the neighborhood structure is very important because it directly affects the fitness landscape. The choice of the search strategy is also important because it controls the trade-off between exploration and exploitation of the search. A lot of search strategies have been proposed in the literature, ranging from simple hill-climbing, simulated annealing, tabu search, guided local search [7] to more sophisticated ones, where the trade-off between exploration and exploitation can be controlled by their specific parameters (e.g. *temperature* for SA, *tabu tenure* for TS, and λ value for GLS).

Most of the neighborhood search algorithms use a fixed neighborhood structure (sometimes a composite of several neighborhoods) throughout the search while controlling the trade-off between exploration and exploitation by the search strategy. Contrary to these algorithms, candidate list strategies [3] consider only a subset of a predefined full neighborhood at each iteration in order to reduce

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 782–791, 2014.

the computational effort. The simplest way of introducing candidate list strategy is to define a neighborhood as a randomly selected part of a predefined full neighborhood. Apart from the effect of reducing the computational effort, this candidate list strategy is useful for diversifying the search. With this strategy, we can control the trade-off between exploration and exploitation by adjusting the ratio of the size of the partial neighborhood to that of the full neighborhood; the smaller the neighborhood size is, the more diverse is the search. This is because the quality of a solution accepted in the partial neighborhood becomes worse with decreasing the neighborhood size (if a solution must be accepted at each iteration).

In this paper, we incorporate the candidate list strategy with random sampling into a tabu search framework to construct an effective neighborhood search algorithm for the university course timetabling problem (UCTP), and in this paper we call this algorithm random partial neighborhood search (RPNS). In addition, we analyze the effect of changing the neighborhood size on the performance. Experimental results on the well-studied benchmark set of Socha *et al.* [6] show that the RPNS algorithm outperforms the state-of-the-art algorithms [1][4][5][2] when the neighborhood size is set properly.

The remainder of this paper is organized as follows. The definition of the UCTP is described in Section 2. The framework of RPNS is presented in Section 3. The computational analysis of the RPNS algorithm and the performance comparison with state-of-the-art algorithms are presented in Section 4. Conclusions are presented in Section 5.

2 The University Course Timetabling Problem

Several formulations of the UCTP have been proposed in the literature, and we select the one proposed by Socha *et al.* [6] because the corresponding benchmark set, which is available at `http://iridia.ulb.ac.be/~msampels/tt.data/`, have been intensively tackled by many algorithms.

Let $E = \{e_1, e_2, \ldots, e_N\}$ denote a set of N events, $T = \{t_1, t_2, \ldots, t_K\}$ a set of K timeslots ($K = 45$, 5 days of 9 hours), and $R = \{r_1, r_2, \ldots, r_L\}$ a set of L rooms, $S = \{s_1, s_2, \ldots, s_M\}$ a set of M students. For each event, the students who attend this event and a set of the rooms that meet requirements for this event [1] are known.

A timetable is represented as an assignment of the events to the timeslots and rooms. A timetable is said to be *feasible* if it satisfies the following hard constraints (H1-H3).

H1: No student attends more than one event in the same timeslot.
H2: Each event must be assigned to a room that meets the requirements for this event.
H3: Only one event can be assigned to each room at any timeslot.

[1] In the original benchmark data set, this information is not explicitly given, but it is easily obtained from the given data.

The objective is to find a feasible timetable that minimizes the total violations of the soft constraints (S1-S3) described below.

S1: A student has an event scheduled in the last timeslot of a day.
S2: A student has more than two consecutive events [2].
S3: A student has only one event on a day.

More formally, the total violations of the soft constraints for a timetable x is defined as follows:

$$f(x) = f_1(x) + f_2(x) + f_3(x),$$

where $f_i(x)$ $(i = 1, 2, 3)$ is the number of the occurrence of constraint violations in terms of soft constraint Si.

3 Solution Method

As used in some of the previous works, we employ a two-stage approach where a feasible timetable is constructed in the first stage (not main focus of this paper) and then the total violation of the soft constraints is minimized in the second stage (main focus of this paper) while maintaining the feasibility. We first present the RPNS algorithm that is used in the second stage and then describe the outline of the first stage.

3.1 Random Partial Neighborhood Search

Neighborhoods. We first define the two neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$, which are widely used in neighborhood searches for various timetabling problems.

$\mathcal{N}_1(x)$**:** A set of the feasible timetables that are obtained from a timetable x by moving an event e to another timeslot-room pair (t, r) for all possible (e, r, t) combinations.

$\mathcal{N}_2(x)$**:** A set of the feasible timetables that are obtained from a timetable x by exchanging the timeslots of two events (e_1, e_2) for all possible (e_1, e_2) combinations, where a change of the room is allowed unless the room is not occupied by other events.

The neighborhood $\mathcal{N}_2$ is slightly different from the standard swap neighborhood because the standard swap move is to exchange the assignment of timeslot-room pairs between two events (or exchange is allowed only if two events are assigned to the same room). This modification is reasonable because the $\mathcal{N}_2$ neighborhood is more flexible in the assignment of rooms and it scarcely increases the computational effort to evaluate all possible moves in the neighborhood.

To vary the neighborhood size, we introduce two neighborhoods $\mathcal{N}_1(x, ratio)$ and $\mathcal{N}_2(x, ratio)$ as partial neighborhoods of $\mathcal{N}_1(x)$ and $\mathcal{N}_2(x)$, respectively, where $ratio$ $(0 \leq ratio \leq 1)$ is a parameter that specifies the ratio of the size of the partial neighborhood to that of the full neighborhood. These neighborhoods are defined as follows.

[2] Two events in different days are not regarded as consecutive. If the number of consecutive events is s (≥ 3), the number of violations caused by these events is $s - 2$.

$\mathcal{N}_1(x, ratio)$**:** A set of the feasible timetables that are obtained from a timetable x by moving an event e to another timeslot-room pair (t, r) for $e \in I_1$ and all possible (r, t) combinations, where I_1 is defined by randomly selecting $\lceil ratio \times N \rceil$ elements of E.

$\mathcal{N}_2(x, ratio)$**:** A set of the feasible timetables that are obtained from a timetable x by exchanging the timeslots of two events (e_1, e_2) for $e_1 \in I_2$ and $e_2 \in E$ $(e_1 < e_2)$ (*i.e.*, at least one of the two events is selected from I_2) where I_2 is defined by randomly selecting $\lceil \frac{1}{2} \left\{ 2N - 1 - \sqrt{4N(N-1)(1-ratio)+1} \right\} \rceil$ $(= N_2)$ element of E. The change of room is allowed for both events e_1 and e_2 unless the room is not occupied by other events.

Note that N_2 in the definition of $\mathcal{N}_2(x, ratio)$ neighborhood is determined such that the size of $\mathcal{N}_2(x, ratio)$ neighborhood divided by the size of $\mathcal{N}_2(x)$ neighborhood is equal to $ratio$. In fact, N_2 is obtained by solving the following equation: $N_2 N - \frac{N_2(N_2+1)}{2} = ratio \times \frac{N(N-1)}{2}$.

Algorithm. The idea of using the random partial neighborhood is incorporated into tabu search (TS) to construct a RPNS algorithm, which is presented in Algorithm 1. Before starting iterations, the current timetable x and the current best timetable x_{best} are initialized with an input feasible timetable (line 1). At each iteration, the best non-tabu solution x', which will become the next current timetable, is selected from the union of the two partial neighborhoods $\mathcal{N}_1(x, ratio)$ and $\mathcal{N}_2(x, ratio)$ where the selection of a tabu solution is forbidden (line 3). Tabu solutions are defined as follows. If an event e is moved using $\mathcal{N}_1$ neighborhood (or two events e_1 and e_2 are swapped using $\mathcal{N}_2$ neighborhood), event e (or two events e_1 and e_2) is regarded as "tabu event" during the subsequent T iterations, where T is a parameter called tabu tenure. At each iteration, a timetable obtained by moving an tabu event using $\mathcal{N}_1$ neighborhood or by swapping two tabu events using $\mathcal{N}_2$ neighborhood is regarded as a tabu solution. In addition, the aspiration criterion is considered where a solution that improves the current best solution x_{best} is regarded as a non-tabu solution. After selecting the best non-tabu solution x', the current solution x and current best solution x_{best} (if necessary) are updated by x' (line4). Iterations are repeated until the total number of iterations reaches a given maximum number of iterations $iterMax$ (lines 2 and 5), and the current best timetable x_{best} is returned (line 7).

We should note that it is also possible to define tabu solutions based on the attribute of event-timeslot pairs (e.g. an event is forbidden from moving back to timeslot t during the subsequent T iterations after this event is moved from timeslot t in a previous iteration) rather than the attribute of only events. However, we decided to employ our definition because we confirmed that the performance of RPNS with the tabu definition based on only events was slightly better than that with tabu definition based on event-timeslot pairs and the former one is simpler.

Algorithm 1. TABU-SEARCH(x_{input})

1: Set $x := x_{input}$, $x_{best} := x_{input}$ and $iter := 0$;
2: **while** ($iter \leq iterMax$) **do**
3: Select a best non-tabu solution $x' \in \mathcal{N}_1(x, ratio) \cup \mathcal{N}_2(x, ratio)$ (the aspiration criterion is considered);
4: Update $x := x'$ and $x_{best} := x'$ (if x' is better than x_{best});
5: Set $iter := iter + 1$;
6: **end while**
7: **return** x_{best};

3.2 Construction of an Initial Feasible Timetable

To construct an initial feasible timetable in the first stage, we use an algorithm similar to tabu search (TS) where a current solution is represented as a partial timetable during the course of the search. Here, a partial timetable refers to an incomplete timetable in which one or more events are not scheduled. A partial timetable is regarded as feasible if the scheduled events satisfy all hard constraints. We evaluate the quality of partial timetables by the number of the unscheduled events (small number is better). The unscheduled events are stored in a list called ejection list (EL).

The procedure is started by initializing both the current timetable x and the current best timetable x_{best} with an empty timetable and EL with all events in a random order. At each iteration, an event e_{in} is popped from EL, and an attempt is made to schedule the selected event into the current (partial) timetable x without violating any hard constraints. Let $\mathcal{N}_{in\_out}(e_{in}, x)$ be a set of the feasible (partial) timetables that are obtained from x by assigning e_{in} to a timeslot-room pair and ejecting the conflicting events caused by the insertion of e_{in} (no event is ejected if no hard constraint violation occurs) in all possible ways. Note that e_{in} must be assigned to a room that satisfies the requirement for this event (hard constraint H2), while the violation of the hard constraints H1 and H3 caused by the insertion of e_{in} can be resolved by ejecting the conflicting events in the same timeslot. The next current timetable x' is selected from $\mathcal{N}_{in\_out}(e_{in}, x)$ so that the number of the ejecting events is minimized. In practice, the idea of tabu search is incorporated to diversity the search; the selection of a tabu solution is forbidden where a timetable obtained by assigning e_{in} to a timeslot t is regarded as a tabu solution during the subsequent $T(= 100)$ iterations after e_{in} is assigned to a timeslot t in the previous iteration. In addition, the aspiration criterion is considered, where a solution that improves the current best solution x_{best} is always regarded as a non-tabu solution. After the selection of the next current timetable, push the ejected events into EL if one or more events are ejected, and the current solution x and current best solution x_{best} (if necessary) are updated by x'. Iterations are repeated until all events are scheduled, and the obtained feasible timetable is returned.

4 Computational Experiments

4.1 Experimental Settings

We investigated the performance of the RPNS algorithm on the benchmark set of Socha *et al.* [6] because this benchmark set has been intensively tackled by many algorithms. This benchmark set consists of 11 instances, which are divided into three categories (5 small instances, 5 medium instances, and one large instance) according to the number of courses, rooms, and students. The numbers of (courses, rooms, students) are (100, 5, 80) for the small instances, (400, 10, 200) for the medium instances, and (400, 10, 400) for the large instance. We do not show results for the small instances because some of the previous approaches in the literature as well as RPNS can find feasible timetables with a penalty cost of zero, which makes these instances useless for comparison purposes.

The RPNS algorithm was implemented in C++ and was executed in a virtual machine environment (*i.e.*, each job is executed on a single core, but multiple jobs may be executed in the same node) on a cluster with Intel Xeon 2.93 GHz nodes. We performed the RPNS algorithm with various combinations of tabu tenure ($T = 0$, 5, 10, 20, 30, 50, 70, 100, 120, 150, and, 200) and neighborhood size ($ratio = 0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.5$, and 1) in order to investigate their effects and relation on the performance. We set $iterMax = \lceil \frac{100{,}000}{ratio} \rceil$ in order to makes the total number of evaluations of solutions in a single run (and therefore the computation time) the same regardless of the difference in the neighborhood size. For each of all configurations, we performed the RPNS algorithm 10 times on each instance.

4.2 Results

Figure 1 shows the results of the RPNS algorithm for all possible pairs of *ratio* and T; each curve, which corresponds to a value of *ratio*, is a plot of the average penalty values over 10 runs against the different values of T.

The RPNS algorithm with the full neighborhood ($ratio = 1$) finds high quality timetables that improves the lowest penalty timetables reported in the literature (except for instance `large`) if T is set to the best value for each instance. However, the quality is fairy sensitive to the value of T, meaning that the trade-off between exploitation and exploration must be controlled appropriately to achieve a high performance.

We can see that when T is set to the best value for each value of *ratio*, the use of partial neighborhoods ($ratio < 1$) improves the result of the full neighborhood ($ratio = 1$) unless the neighborhood size is too small. This is attributed to the trade-off between the positive effect of the increase in the number of iterations and the negative effect of the decrease in the possibility of finding a better solution in the neighborhood at each iteration. An important observation is that the best value of T decreases as the value of *ratio* is decreased. The reason for this is that a reduction of the neighborhood size has an effect to diversify the search. An interesting observation is that the best performance over all possible pairs of *ratio*

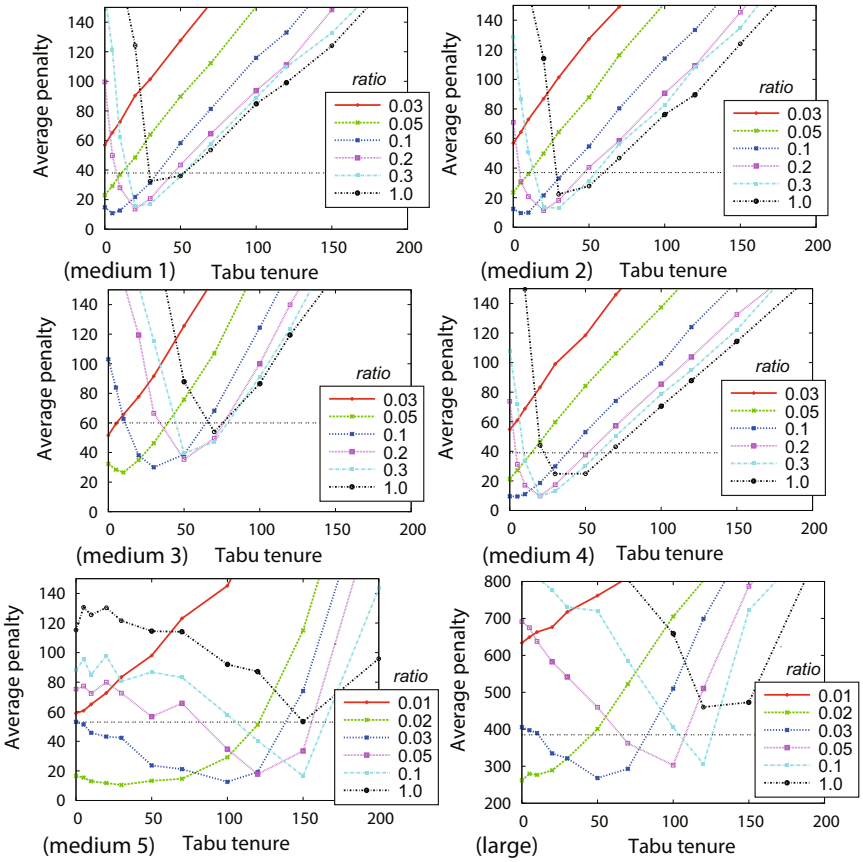

Fig. 1. Results of RPNS ($iterMax = \lceil \frac{100{,}000}{ratio} \rceil$), where horizontal dotted lines represent the best penalty values found in the literature

and T seems to be obtained when the neighborhood size is optimized under the setting of $T = 0$. For example, in the result on `medium 1`, the best performance is obtained by setting $T = 5$ and $ratio = 0.1$, but a better (or similar) result will be obtained by setting $T = 0$ and $ratio$ between 0.05 and 0.1. This is a little bit surprising because the strength of tabu search is completely spoiled in the setting of $T = 0$, meaning that a mechanism of diversifying the search depends solely on the reduction of the neighborhood size. From a practical standpoint, this is a nice property because the best performance is attained by adjusting only one parameter $ratio$ instead of adjusting both parameters T and $ratio$.

4.3 Analysis

In RPNS, the reduction of the neighborhood size plays an important role in diversifying the search, and we analyze this effect in more detail. Graph (a) of Figure 2 shows the number of students attending to each of the events for

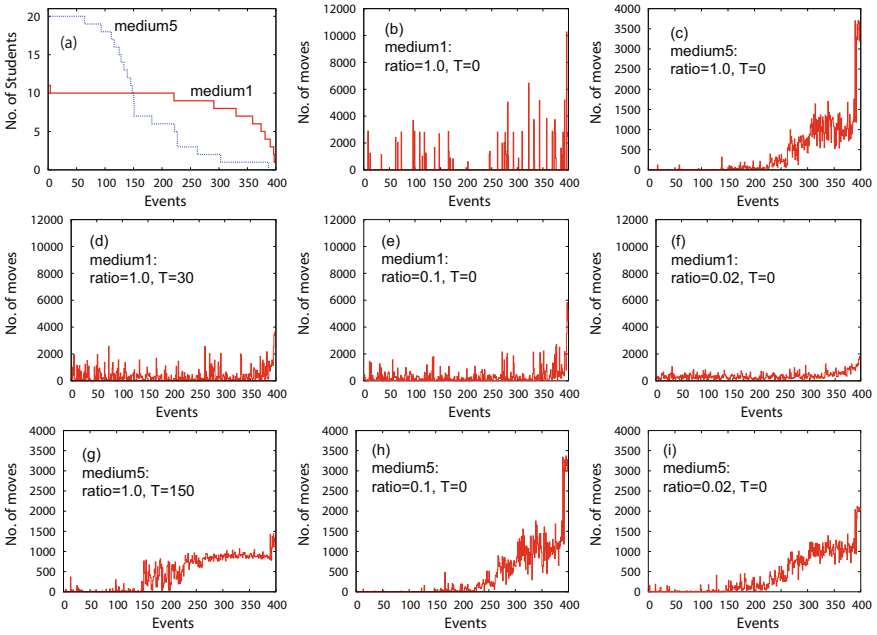

Fig. 2. (a) The number of students attending to each of the events, and (b–i) the cumulative number of moves for each of the events in a single run of RPNS

`medium 1` and `medium 5`, where the events are re-indexed in descending order of the number of the students. We can see that the number of students varies widely among the events in `medium 5`, while the difference in the number of students is relatively small in `medium 1`. The other graphs (b)–(i) show the cumulative number of moves for each of the events (e.g. if events e_1 and e_2 are exchanged, both counts of e_1 and e_2 are incremented) in a single run of the RPNS algorithm for several settings of *ratio* and T on the two instances, where the events are re-indexed in the same order as in the graph (a). Note that the cumulative number of moves presented in Figure 2 is the result when the number of iterations is 100,000.

Graphs (b) and (c) show the results of a setting ($ratio = 1,\ T = 0$) on the two instances. We can see that in `medium 5` the cumulative number of moves tends to increase as the number of students decreases (graph (c)). This is a natural consequence because the impact of an event on the penalty cost increases as the number of students increases. A similar situation does not occur in `medium 1` because the difference in the number of students between the events is small, but *cycling* occurred frequently during the search and therefore the cumulative number of moves are concentrated in a part of the events (graph (b)).

Graphs (d)–(f) shows the results of three settings ($ratio = 1,\ T = 30$), ($ratio = 0.1,\ T = 0$), and ($ratio = 0.02,\ T = 0$) on `medium 1`, where $T = 30$ is the best value (when $ratio = 1$) and $ratio = 0.1$ is the best value

Table 1. Comparisons with the state-of-the-art algorithms

	GDTS (5 runs) best	NGDHH (10 runs) best	ENGDHH (20 runs) best	HHSA (10 runs) best	RPNS (10 runs) ave.	RPNS (10 runs) best
`medium 1`	78	71	38	99	14.8	7
`medium 2`	92	82	37	73	12.4	10
`medium 3`	135	112	60	130	32.3	24
`medium 4`	75	55	39	105	9.6	5
`medium 5`	68	103	55	53	16.6	6
`large`	556	777	638	385	262.0	205

(when $T = 0$) for this instance. Graphs (g)–(i) shows the results of three settings ($ratio = 1,\ T = 150$), ($ratio = 0.1,\ T = 0$), and ($ratio = 0.02,\ T = 0$) on `medium 5`, where $T = 150$ is the best value (when $ratio = 1$) and $ratio = 0.02$ is the best value (when $T = 0$) for this instance.

Compared to the result of the full neighborhood ($ratio = 1$) with $T = 0$ (graphs (b) and (c)), the distribution of the cumulative number of moves is spread by setting $T = 30$ (`medium 1`) and $T = 150$ (`medium 5`) through the effect of tabu search. More importantly, we can see that a similar effect is obtained by decreasing the neighborhood size even if the value of T is set to zero, and this effect becomes more prominent as the value of *ratio* is decreased. These results show that the degree of diversification of the search can be controlled by the neighborhood size.

4.4 Comparisons with Other Algorithms

Finally, we compare the performance of RPNS ($iterMax = \lceil \frac{100{,}000}{ratio} \rceil$) with those of leading algorithms that have shown competitive performance on the benchmark set of Socha *et al.* Table 1 shows the comparison results. The compared algorithms, which are selected from about thirty algorithms found in the literature are Great Deluge with Tabu Search (GDTS) [1], Non-linear Great Deluge Hyper Heuristic (NGDHH) [4], Extended version of Non-linear Great Deluge Hyper Heuristic (ENGDHH) [5], and Hybrid Harmony Search Algorithm (HHSA) [2]. For RPNS, we present the best and average results of 10 runs obtained with the setting of $T = 0$ and the best value of *ratio* for each instance (see Figure 1) where the values of *ratio* are 10 (`medium 1`), 10 (`medium 2`), 5 (`medium 3`), 10 (`medium 4`), 2 (`medium 5`), and 2 (`large`), respectively. For each of the compared algorithms, the best results obtained by multiple runs with various parameter settings (if experiments were conducted with various settings) are presented. Note that our purpose here is not to allege the superiority of RPNS over the compared algorithms because the value of *ratio* was adjusted for each instance in RPNS, but we can see that the quality of the timetables obtained by RPNS is far ahead of others.

The average computation times for a single run of RPNS with the best values of *ratio* (and $T = 0$) were 323 seconds (`medium 1`), 282 seconds (`medium 2`), 451 seconds (`medium 3`), 315 seconds (`medium 4`), 826 seconds (`medium 5`), and 418

seconds (`large`), respectively. For the compared algorithms, the average computation times for a single run for each of the instances were approximately 12 hours (GDTS), 3–5 hours (NGDHH), 2.5–5 hours (ENGDHH), and 6 hours (HHSA), respectively. We can see that the computational effort of RPNS is reasonable even allowing for the differences in the computer speed and implementation.

5 Conclusion and Future Work

We have proposed an tabu search algorithm using an candidate list stratety with random sampling (random partial neighborhood search, RPNS) for the the university course timetabling problem, where the neighborhood size can be adjusted by a parameter *ratio*. Experimental results show that the degree of diversification of the search can be controlled by the neighborhood size, and the RPNS algorithm attains a very good performance when the neighborhood size is set properly especially when the tabu tenure is set to zero. In fact the quality of the timetables obtained by the RPNS algorithm with the best value of *ratio* is far ahead of those of the state-of-the-art algorithms. However, the quality is fairy sensitive to the value of *ratio*. At the current moment, it is difficult to estimate the best value of *ratio* for a given instance before or during the search, and a possible direction for future research is to develop a self-adapting mechanism for the value of *ratio*.

References

1. Abdullah, S., Shaker, K., McCollum, B., McMullan, P.: Construction of course timetables based on great deluge and tabu search. In: Proceedings of MIC 2009: VIII Metaheuristic International Conference, pp. 13–16 (2009)
2. Al-Betar, M.A., Khader, A.T., Zaman, M.: University course timetabling using a hybrid harmony search metaheuristic algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 42(5), 664–681 (2012)
3. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
4. Obit, J., Landa-Silva, D., Ouelhadj, D., Sevaux, M.: Non-linear great deluge with learning mechanism for solving the course timetabling problem. In: 8th Metaheuristics International Conference, MIC 2009 (2009)
5. Obit, J.H., Landa-Silva, D., Sevaux, M., Ouelhadj, D.: Non-linear great deluge with reinforcement learning for university course timetabling. In: Metaheuristics–Intelligent Decision Making. Series Operations Research/Computer Science Interfaces. Springer (2011)
6. Socha, K., Knowles, J.D., Sampels, M.: A $\mathcal{MAX} - \mathcal{MIN}$ ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002)
7. Voudouris, C., Tsang, E.P., Alsheddy, A.: Guided local search. Springer (2010)