# Non-Linear Great Deluge with Reinforcement Learning for University Course Timetabling

Joe Henry Obit[1], Dario Landa-Silva[1], Marc Sevaux[2] and Djamila Ouelhadj[1]

[1] ASAP Research Group, School of Computer Science, University of Nottingham, UK
jzh@cs.nott.ac.uk, dario.landasilva@nottingham.ac.uk, dxs@cs.nott.ac.uk
[2] Centre de Recherche - BP 92116, University of South-Brittany, France
marc.sevaux@univ-ubs.fr

**Abstract.** This paper describes a non-linear great deluge hyper-heuristic incorporating a reinforcement learning mechanism for the selection of low-level heuristics and a non-linear great deluge acceptance criterion. The proposed hyper-heuristic deals with complete solutions, i.e. it is a solution improvement approach not a constructive one. Two types of reinforcement learning are investigated: learning with static memory length and learning with dynamic memory length. The performance of the proposed algorithm is assessed using eleven test instances of the university course timetabling problem. The experimental results show that the non-linear great deluge hyper-heuristic performs better when using static memory than when using dynamic memory. Furthermore, the algorithm with static memory produced new best results for five of the test instances while the algorithm with dynamic memory produced four best results compared to the best known results from the literature.

## 1 Introduction

The university course timetabling problem has been tackled using a wide range of exact methods, heuristics and meta-heuristics. In recent years, the term *hyper-heuristic* has emerged for referring to methods that use (meta-) heuristics to choose (meta-) heuristics [CE8]. Then, a hyper-heuristic is a process which, given a particular problem instance and a number of *low-level heuristics*, manages the selection and acceptance of the low-level heuristic to apply at any given time, until a stopping condition is met. Low-level heuristics are simple local search operators or domain dependent heuristics. Typically, a hyper-heuristic is meant to search in the space of heuristics instead of searching in the solution space directly. One of the main challenges in designing a hyper-heuristic method is to manage the low-level heuristics with minimum parameter tuning.

Early research work on hyper-heuristics focused on the development of advanced strategies for choosing the heuristics to be applied at different points of the search. For example, Soubeiga [CE25] used a choice function that incorporates principles from reinforcement learning. That choice function rewards or penalises the low-level heuristics according to their success in finding a better solution. Another mechanism based on tabu search was proposed by Burke et

al. [CE9] in which a tabu list is used to prevent (for a number of iterations) the acceptance of low-level heuristics with poor performance. Ross et al. [CE21] used a learning classifier system to learn which heuristics were more useful than others when tackling bin packing problems. Other hyper-heuristic approaches include the GA-based hyper-heuristic by Cowling et al. [CE14], the case-based hyper-heuristic approach by Burke et al. [CE11] and the ant-based hyper-heuristic by Burke et al. [CE12]. Also, researchers have proposed different acceptance criteria to drive the selection of low-level heuristics within a hyper-heuristic framework. For example, Soubeiga [CE25] used a simulated annealing acceptance criterion, Ayob and Kendall [CE5] used a Monte Carlo acceptance criterion while Kendall and Mohamad [CE16] used the great deluge acceptance criterion.

We propose an approach that uses Reinforcement Learning and a Non-Linear Great Deluge (NLGD) acceptance criterion in order to choose which low-level heuristic to apply to solve university course timetabling problem instances. Section 2 describes the course timetabling problem tackled in this work. Section 3 reviews previous meta-heuristic and hyper-heuristic methods used to tackle this problem. Section 4 presents the non-linear great deluge hyper-heuristic method proposed in this paper while Section 5 describes and discusses our experimental results. Finally, conclusions and future research are the subject of Section 6.

## 2 The University Course Timetabling Problem

The university course timetabling problem can be defined as a process of allocating, subject to predefined constraints, a set of limited timeslots and rooms to courses, while satisfying as nearly as possible a set of desirable objectives. In the timetabling problem, constraints can be divided into two categories: hard and soft constraints. A timetable is said to be feasible (usable) if no hard constraints are violated. However, soft constraints may be violated and the objective is to minimise their violation in order to increase the quality of the timetable. The course timetabling problem is very complex (as discussed by Cooper and Kingston [CE13]) and common to a wide range of educational institutions. The manual process of preparing the timetable is tedious, time consuming and yet not guaranteed to produce a timetable free of conflicts.

Several formulations of the course timetabling problem exist in the literature. We adopt the one by Socha et al. [CE23] and the corresponding benchmark data sets to test the proposed algorithm. More formally, the course timetabling problem tackled here consists of the following: $n$ events $E = \{e_1, e_2, \ldots, e_n\}$, $k$ timeslots $T = \{t_1, t_2, \ldots, t_k\}$, $m$ rooms $R = \{r_1, r_2, \ldots, r_m\}$ in which events can take place, a set $F$ of room features satisfied by rooms and required by events, and a set $S$ of students. Each room has a limited capacity and each student attends a number of events. The problem is to assign $n$ events to $k$ timeslots and $m$ rooms in such a way that all hard constraints are satisfied and the violation of soft constraints is minimised. The benchmark data set proposed by Socha et al. [CE23] involves 11 instances which are split according to their size into 5 small, 5 medium and 1 large. For the small instances, $n = 100$, $m = 5$, $|S| = 80$,

$|F| = 5$. For the medium instances, $n = 400$, $m = 10$, $|S| = 200$, $|F| = 5$. For the large instance, $n = 400$, $m = 10$, $|S| = 400$, $|F| = 10$. For all instances, $k = 45$ (9 hours in each of 5 days).

There are **4 hard constraints**: 1) a student cannot attend two events simultaneously (events with students in common must be timetabled in different timeslots); 2) only one event can be assigned per timeslot in each room; 3) the room capacity must be equal to or greater than the number of students attending the event in each timeslot; 4) the room assigned to the event must satisfy the features required by the event. There are **3 soft constraints**: 1) students should not have exactly one event timetabled on a day; 2) students should not attend more than two consecutive events on a day; 3) students should not attend an event in the last timeslot of the day.

## 3 Summary of Related Work

Various heuristics have been proposed to tackle the course timetabling problem described above. Socha et al. first proposed a *MAX-MIN* ant system [CE23] and then later an ant colony system [CE24] in which artificial ants follow a construction graph to build a timetable. Rossi-Doria et al. [CE22] compared the performance of several meta-heuristics to solve this problem. The methods compared: evolutionary algorithm, ant colony optimisation, iterated local search, simulated annealing, and tabu search. No best results were reported by Rossi-Doria et al. as the intention was to assess the strength and weaknesses of each algorithm. Asmuni et al. [CE4] implemented a fuzzy multiple heuristic ordering in which fuzzy logic was used to establish the ordering of events prior to be timetabled. Abdullah et al. [CE1] proposed versions of variable neighbourhood search while Abdullah et al. [CE2] applied a randomised iterative improvement approach using a composite of eleven neighbourhood structures in exploring the current solution. Later, Abdullah et al. [CE3] presented a hybrid approach combining a mutation operator with their previous randomised iterative improvement procedure. Recently, a non-linear great deluge algorithm (NLGD) was proposed by Landa-Silva and Obit [CE17]. That method produced new best results in 4 of 11 problem instances. Finally, McMullan [CE19] proposed an extended great deluge algorithm (EGD), which allows re-heating similar to simulated annealing, and found new best results for the 5 medium instances.

Hyper-heuristics have also been applied to solve this timetabling problem. Burke et al. [CE9] applied a choice function hyper-heuristic which also uses a tabu list to guide the iterative application of a set of simple local search heuristics. Rattadilok et al. [CD20] proposed a distributed choice function hyper-heuristic and implemented two designs based on a parallel architecture: hierarchical and agent-based. Burke et al. [CE10] proposed a graph-based hyper-heuristic in which a tabu search procedure is used to change the permutations of six graph colouring heuristics before applying them to construct a timetable. Bai et al. [CE6] developed a simulated annealing hyper-heuristic which selects low-level heuristics based on a stochastic ranking mechanism.

## 4 The Non-linear Great Deluge Hyper-heuristic

In this paper, we use our non-linear great deluge algorithm (NLGD) [CE17] as an acceptance criterion and incorporate reinforcement learning to select the low-level heuristics to apply at each step of the search process. That is, while in a NLGD meta-heuristic *candidate solutions* are accepted or not based on the great deluge criterion, in the proposed Non-Linear Great Deluge Hyper-heuristic (NLGDHH) it is *candidate low-level heuristics* which are accepted or not, i.e. the method operates in the heuristic search space.

Figure 1 illustrates the proposed hyper-heuristic in which the low-level heuristics are local search operators which explore the solution space while the reinforcement learning and the NLGD acceptance criterion explore the heuristic space. We use the non-linear great deluge criterion because of its simplicity and less dependent nature upon parameter tuning compared to simulated annealing [CE7,CE17]. The low-level heuristics implemented in this work are listed below. These heuristics are based on random search but always ensuring the satisfaction of hard constraints.

**H1**: selects 1 event at random and assigns it to a feasible pair timeslot-room also selected at random.

**H2**: selects 2 events at random and swaps their timeslot-room while ensuring feasibility.

**H3**: selects 3 events at random and exchanges timeslot-room at random while ensuring feasibility.

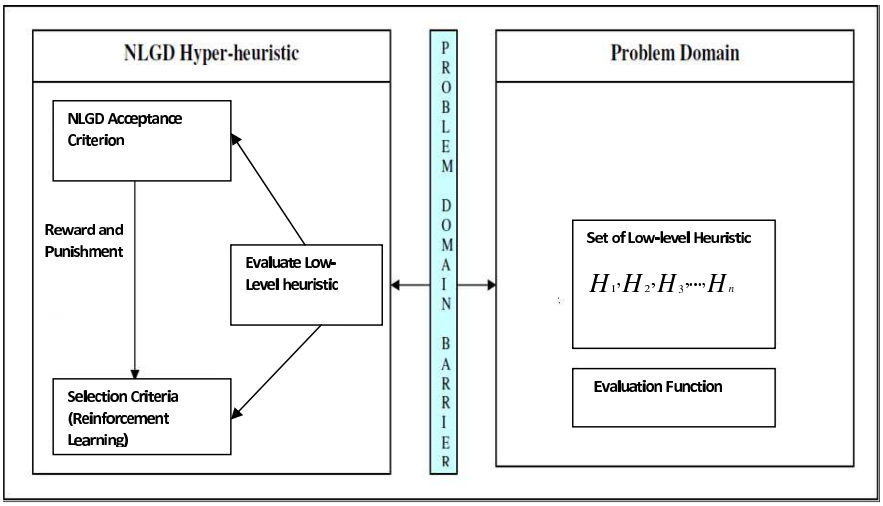


Fig. 1: Non-Linear Great Deluge Hyper-heuristic Approach

### 4.1 Non-linear Great Deluge (NLGD) Acceptance Criterion

The NLGD acceptance criterion refers to accepting improving and non-improving low-level heuristics depending on the performance of the heuristic and the current water level $B$. Improving heuristics are always accepted while non-improving ones are accepted only if the detriment in quality is less than or equal to $B$. The initial water level is usually set to the quality of the initial solution and then decreased by a non-linear function proposed in our previous work [CE17]:

$$B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta \tag{1}$$

The various parameters in Eq. (1) control the speed and the shape of the water level decay rate. Parameter $\beta$ influences the shape of the decay rate and it represents the minimum expected penalty corresponding to the best solution. The role of parameters $min$ and $max$ is to control the speed of the decay rate. However, the search could get stuck and to avoid this, it is necessary sometimes to relax the water level. When the water level is about to converge to the current penalty cost, the algorithm then allows the water level to go up.

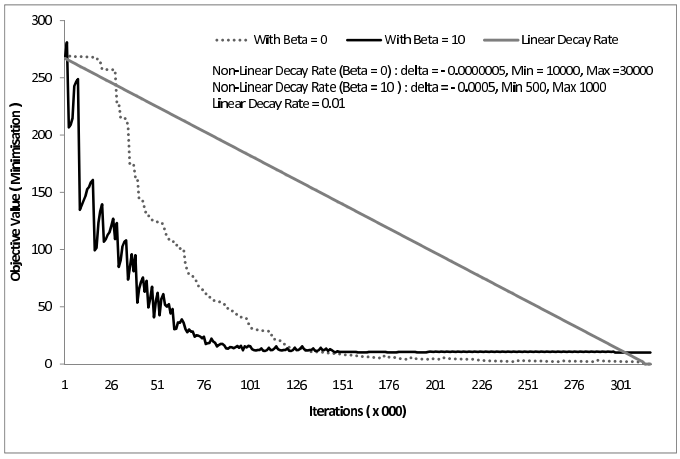


Fig. 2: Comparison Between Linear (straight line) and Non-linear (curves) Decay Rates and Illustration of the Effect of Parameters $\beta, \delta, min$ and $max$ on the Shape of the Non-linear Decay Rate.

Figure 2 illustrates the difference between the linear and non-linear decay rates. The graph also illustrates the effect of parameters $\beta$, $\delta$, $min$ and $max$ on the non-linear decay rate. The straight line in Figure 2 corresponds to the linear decay rate originally proposed by Dueck [CE15]. In this case, a non-improving candidate solution $S^*$ is accepted only if its objective value $f(S^*)$ is below the water level $B$. When $f(S^*)$ and $B$ converge the algorithm becomes greedy and

it is more difficult for the search to escape from local optima. Figure 2 also illustrates the non-linear decay rate with different values for $\beta$, $\delta$, $min$ and $max$.

We set $\delta = 5 \times 10^{-7}$ and $\beta = 0$ for all datasets in this paper. The reason for setting $\beta = 0$ is that we want $B$ to reach the value of zero by the end of the search. If for a given problem, the minimum penalty that should be achieved is 100, then $\beta$ should be set around that value. If there is no previous knowledge on the minimum penalty expected (best expected fitness), then we suggest to tune $\beta$ through preliminary experimentation for the problem in hand. The values of $min$ and $max$ in Eq. (1) are set according to the current penalty cost. When the penalty cost is more than 20, $min = 80000$ and $max = 90000$. When the penalty cost goes below 20, $min = 20000$ and $max = 30000$. When the $range < 1$ (range is the difference between the water level $B$ and the current penalty), $B$ is increased by a random number within the interval $[B_{min}, B_{max}]$, we call this mechanism *floating B*. For small and medium problem instances the interval used is $[0.85, 1.5]$ while for the large problem instance the interval used is $[1, 5]$.

### 4.2 Learning Mechanism

A reinforcement learning strategy (adapted from Bai et al. [CE6]) is used to guide the selection of low-level heuristics during the search. Initially, all low-level heuristics have the same probability to be selected. Then, we tune the priorities of the low-level heuristics as the search progresses so that the algorithm tries to *learn* which low-level heuristic to use for better exploring the solution space. In this paper, we investigate two types of reinforcement learning (RL): *RL with static memory length* and *RL with dynamic memory length* as described below.

**RL with Static Memory Length:** In each iteration, a low-level heuristic $i$ is selected with probability $p_i$ given by Eq. (2) where $n$ is the number of heuristics and $w_i$ is the weight assigned to each heuristic.

$$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{2}$$

Initially, every weight is set to $w_i = 0.01$. At each iteration, the algorithm starts to reward or punish the heuristics according to their performance. When the chosen heuristic improves the current solution, a reward of 1 point is given to the heuristic. If the heuristic does not improve the solution, the punishment is to award no points. This amount of reward/punishment never changes. However, the algorithm updates the set of weights $w_i$ in every learning period ($lp$) given by $lp = max(K/500, n)$, where $K$ is the total number of feasible moves explored.

We use the following counters to track the performance of each low-level heuristic: $Ctotal_i$, is the number of times that low-level heuristic $i$ is called; $Cnew_i$ is the number of times that low-level heuristic $i$ generates solutions with different fitness value; and $Caccept_i$ is the number of times that low-level heuristic $i$ meets the non-linear great deluge acceptance criterion. Each heuristic weight $w_i$ is updated at every learning period $lp$ and normalised by the ratio

$Caccept_i/Ctotal_i$ when $range > 1$ and by $Cnew_i/Ctotal_i$ when $range < 1$. At every learning period $lp$ and if $range < 1$, the water level increases to $B = B + rand[B_{min}, B_{max}]$. We call this mechanism *surge B*. We set $B_{min}$ equal to 1 and $B_{max}$ equal to 4 regardless to the size of the dataset. Note that the water level can increase due to the *floating B* (continuous) mechanism or the *surge B* (every $lp$ feasible moves) mechanism.

**RL with Dynamic Memory Length:** In each iteration, a low-level heuristic $i$ is selected with probability $p_i$ given by Eq. (3) where $n$ is the number of heuristics, $w_i$ is the weight assigned to each heuristic and $w_{min} = min\{0, w_i\}$.

$$p_i = \frac{w_i + w_{min}}{\sum_{i=1}^{n} w_i + w_{min}} \tag{3}$$

Initially, every weight is set to $w_i = 0.01$ as before, however, each $w_i$ is updated every time the algorithm performs a feasible move. When the selected heuristic improves the current solution, the heuristic is rewarded, otherwise the heuristic is punished. The value $\Re_{ij}$ of reward/punishment applied to heuristic $i$ at iteration $j$ is as given below where $r = 1$, $\Im = 0.1$ and $\Delta$ is the difference between the best solution (lowest penalty) so far and the current solution (current penalty).

$$\Re_{ij} = \begin{cases} r & \text{if } \Delta < 0 \\ -r & \text{if } \Delta > 0 \\ \Im & \text{if } \Delta = 0 \text{ and new solution} \\ -\Im & \text{if } \Delta = 0 \text{ and no new solution} \\ 0 & \text{if not elected} \end{cases}$$

Then, at each iteration $h$, each weight $w_i$ is calculated using Eq.( 4) where $\sigma$ gives the length of the dynamic memory.

$$w_{ih} = \sum_{j=k}^{h} \sigma^j \Re_{ij} \tag{4}$$

In every learning period $lp$, the algorithm updates $\sigma$ with a random value in $(0.5, 1.0]$. Here, we also set $lp = max(K/500, n)$ as before. At every learning period $lp$ and if $range < 1$, the water level increases to $B = B + rand[B_{min}, B_{max}]$. We set $B_{min}$ equal to 1 and $B_{max}$ equal to 4 regardless to the size of the dataset.

## 5 Computational Experiments and Results

To evaluate the performance of the proposed algorithm, we conducted a range of experiments using the standard course timetabling benchmark instances proposed by Socha et al. [CE23]. For each problem instance we run the algorithm 20 times. The stopping condition is a maximum computation time $t_{max}$ or achieving a penalty value of zero, whatever was sooner. For small instances, we set

$t_{max} = 0.75$ hours as the algorithm takes less than 2500 seconds (42 minutes). For medium instances, we set $t_{max} = 2.5$ hours. For the large instance, we set $t_{max} = 5$ hours. Our previous NLGD meta-heuristic [CE17] was not able to improve results even after extending the execution time. However, the approach proposed here is now able to find better solutions thanks to the learning mechanism that selects low-level heuristics accurately to further improve the solution quality. In the rest of this paper, NLGDHH-SM and NLGDHH-DM refer to the algorithm proposed here when using static memory length or dynamic memory length respectively.

We conducted several experiments to evaluate the performance of the two algorithm variants. The first set of experiments compared the performance of NLGDHH-SM and NLGDHH-DM to the great deluge (GD) meta-heuristics. The second set of experiments compared the performance of NLGDHH-SM and NLGDHH-DM to other hyper-heuristics reported in the literature. The third set of experiments investigates the performance of NLGDHH-SM when using different learning period length. Finally, the performance of NLGDHH-SM and NLGDHH-DM are compared to the best known results reported in the literature for the subject problem.

### 5.1 Illustration of the Weights Adaptation

Before presenting our experimental results in detail, we further illustrate the weight adaptation mechanism. As explained above, the weight $w_i$ for each of the low-level heuristics is set to 0.01 at the start of the search. Then, these weights are updated depending on the success or failure of the low-level heuristics to improve the current solution. In order to appreciate how this works, Figures 3 and 4 show the weight values for a particular run of the NLGDHH-SM algorithm on each of the test instances. The initial weights have the same value for all the low-level heuristics but as the search progress, we can see that these weights are adapted for each instance. For example, Figure 3 shows that for small instances, the probability of low-level heuristic H3 being selected is reduced quickly down to zero. However, Figure 4 shows that in the case of three medium instances and the large one, this probability remains above zero and fluctuating for most of the search. We can also see in these Figures that the weights for H1 and H2 are tuned for each test instance and there is no clearly defined common pattern.

### 5.2 Static vs. Dynamic Memory

We first compare NLGDHH-SM to NLGDHH-DM with the objective of examining the effect of the RL mechanism when using Static Memory (SM) or Dynamic Memory (DM). Figure 5 shows the best results obtained by the algorithm with each type of memory. We can see that both learning mechanisms are able to produce optimal solutions for all small instances for at least one out of 20 runs. For medium instances, both mechanisms perform well and the results obtained with the dynamic memory are competitive with those obtained with the static memory, particularly for the M1 instance (for which NLGDHH-SM obtained a

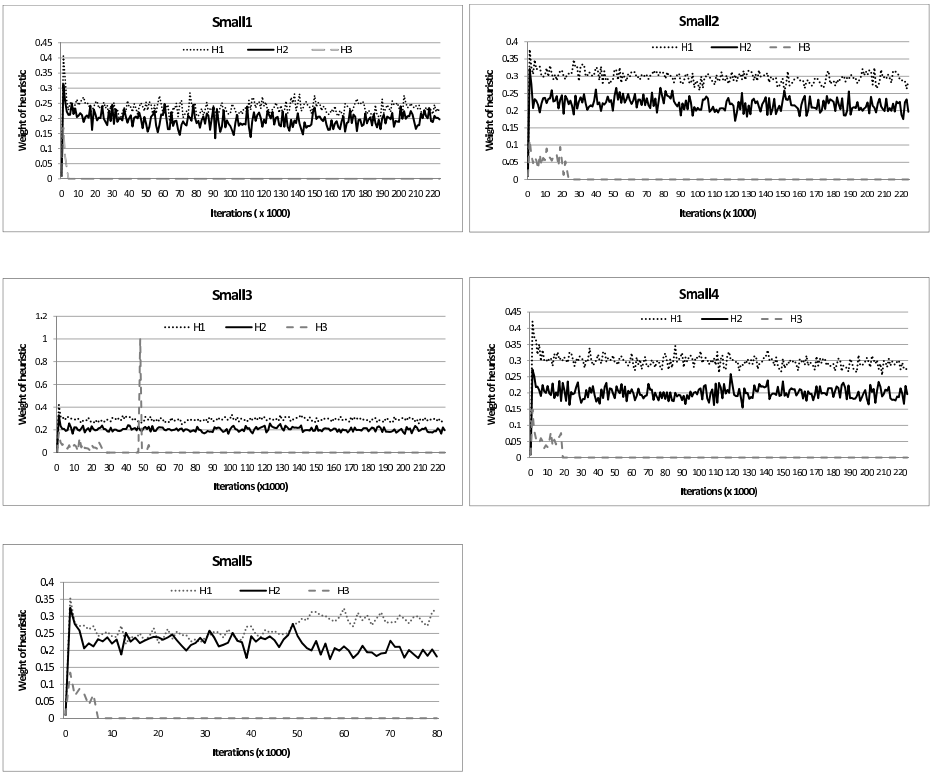


Fig. 3: Adaptation of Weights ($w_i$) During a Run of NLGDHH-SM on Small Instances.

value of 51 while NLGDHH-DM obtained a value of 54). The exact values are reported in Table 1. For instances M2, M3, M4, M5 and L, the results show that NLGDHH-SM obtained better solution quality compared to NLGDHH-DM.

In addition to reporting the best results obtained from the 20 runs, we also report in Figure 6, the average results over the 20 runs for each of the approaches. We can see that although both algorithms reach optimal solutions for all small instances, NLGDHH-SM does this more often compared to NLGDHH-DM. The overall results obtained by NLGDHH-SM are better than those achieved by NLGDHH-DM. It was shown above that the best results obtained by both algorithms on the M1 instance are pretty close. However, on average, the results obtained by NLGDHH-DM seem less consistent than the results achieved by NLGDHH-SM.

We now have a closer look at the performance of each algorithm on instances S1, M1 and L. Figures 7-9 show the results obtained by each algorithm on these instances over all 20 runs. We can see in Figure 7 and Figure 8 that the algorithm with static memory shows a more consistent performance compared

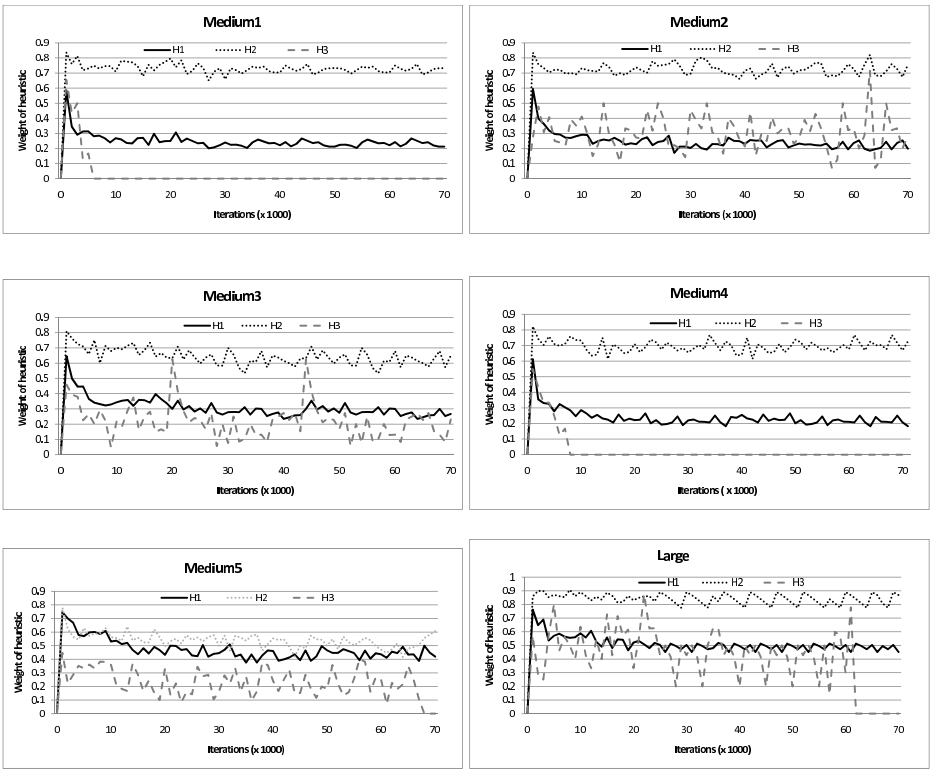


Fig. 4: Adaptation of Weights ($w_i$) During a Run of NLGDHH-SM on Medium and Large Instances.

to the algorithm with dynamic memory. For example, for the small instance S1, NLGDHH-SM found a solution with penalty zero in 15 runs while NLGDHH-DM did it only for 9 of the 20 runs. On the medium instance M1, the algorithm with static memory found better results in almost all the 20 runs and with less variability compared to the results obtained by the algorithm with dynamic memory. However, for the large instance, Figure 9 shows that the algorithm with dynamic memory shows a more consistent performance over the 20 runs although the results obtained with the static memory are still better overall.

### 5.3 Comparison to Previous Great Deluge

The second set of experiments compared the proposed NLGDHH (with static and with dynamic memory length) to previous great deluge meta-heuristics in order to assess the performance of the non-linear acceptance criterion and the RL mechanism. Table 1 shows the results obtained by the non-linear great deluge hyper-heuristic with static (NLGDHH-SM) and with dynamic (NLGDHH-DM) memory, the extended great deluge (EGD) [CE19], the non-linear great deluge

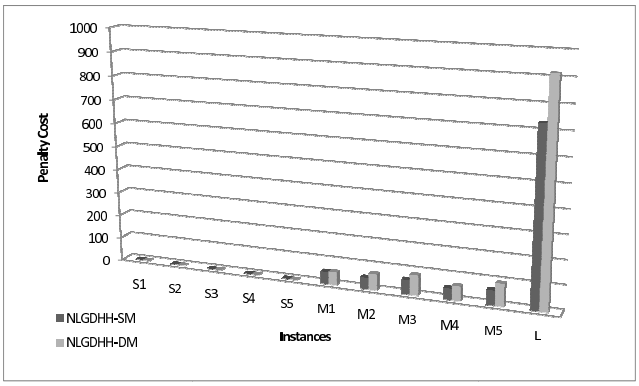


Fig. 5: Best Results Obtained by NLGDHH-SM and NLGDHH-SM

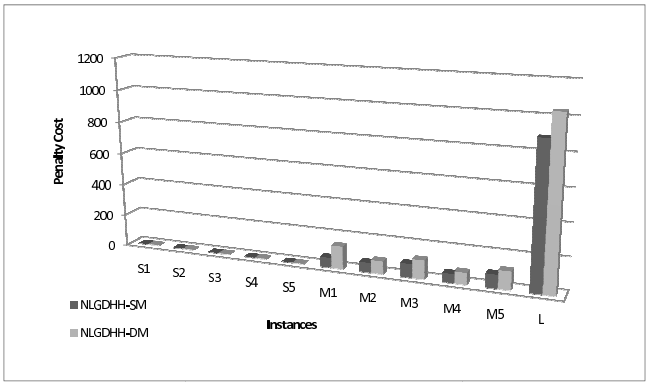


Fig. 6: Average Results Obtained by NLGDHH-SM and NLGDHH-SM

(NLGD) [CE17], the evolutionary non-linear great deluge (ENLGD) [CE18], and the conventional great deluge (GD). We can see in Table 1 that NLGDHH-SM mostly outperforms NLGDHH-DM in terms of the number of best solutions found across all instances. Both variants of the proposed method obtained equal or better results than the other approaches, except for instance L where EGD found better solutions. However, NLGDHH-SM produced better solutions for 10 out of the 11 instances. In fact, NLGDHH-SM improved the solutions by 36.25% for M1, 54.29% for M2, 56.83% for M3, 46.59% for M4 and 30.68% for M5. The average improvements are 40.72%, 49.49%, 48.24%, 49.54% and 29.70% for M1, M2, M3, M4 and M5 respectively. For the large instance, the best result obtained by EGD is 0.13% better and in average 6.10% better than the best

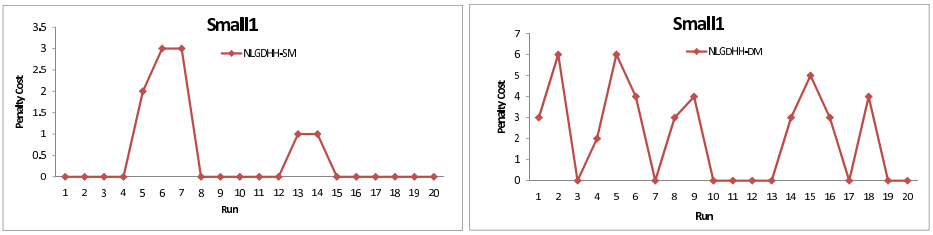


Fig. 7: Results on 20 Runs of NLGDHH-SM and NLGDHH-DM on Instance S1.

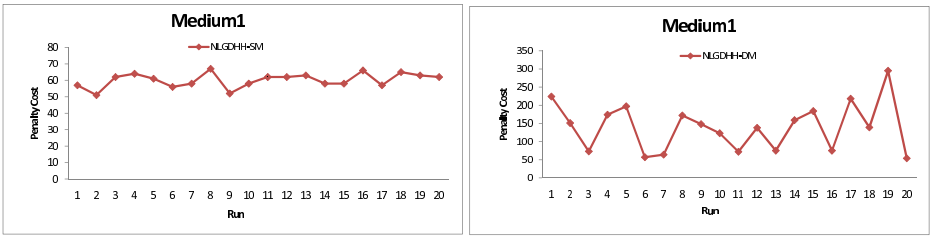


Fig. 8: Results on 20 Runs of NLGDHH-SM and NLGDHH-DM on Instance M1.

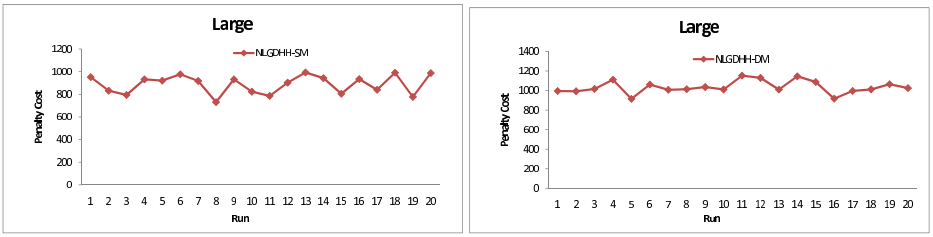


Fig. 9: Results on 20 Runs of NLGDHH-SM and NLGDHH-DM on Instance Large.

result by NLGDHH-SM. The overall performance of both NLGDHH-SM and NLGDHH-DM is quite good according to these results.

Table 1: Comparison of the Proposed Great Deluge Based Hyper-heuristic and Other Great Deluge Methods from the Literature.

| Instance | NLGDHH-SM | | NLGDHH-DM | | EGD | | NLGD | ENLGD | GD |
|---|---|---|---|---|---|---|---|---|---|
| | Best | Avg | Best | Avg | Best | Avg | Best | Best | Best |
| S1 | 0 | 0.5 | 0 | 2.5 | 0 | 0.8 | 3 | 0 | 17 |
| S2 | 0 | 0.65 | 0 | 1.9 | 0 | 2 | 4 | 1 | 15 |
| S3 | 0 | 0.20 | 0 | 2.05 | 0 | 1.3 | 6 | 0 | 24 |
| S4 | 0 | 1.5 | 0 | 2.85 | 0 | 1 | 6 | 0 | 21 |
| S5 | 0 | 0 | 0 | 0.85 | 0 | 0.2 | 0 | 0 | 5 |
| M1 | 51 | 60.1 | 54 | 139 | 80 | 101.4 | 140 | 126 | 201 |
| M2 | 48 | 59.05 | 67 | 78.2 | 105 | 116.9 | 130 | 123 | 190 |
| M3 | 60 | 83.9 | 84 | 115.45 | 139 | 162.1 | 189 | 185 | 229 |
| M4 | 47 | 54.9 | 60 | 72.05 | 88 | 108.8 | 112 | 116 | 154 |
| M5 | 61 | 84.15 | 93 | 112.8 | 88 | 119.7 | 141 | 129 | 222 |
| L | 731 | 888.65 | 917 | 1035.25 | 730 | 834.1 | 876 | 821 | 1066 |

### 5.4 Comparison to Other Hyper-heuristics

We now compare the proposed NLGDHH to other hyper-heuristics reported in the literature. Table 2 shows the results obtained by the following approaches: NLGDHH-SM, NLGDHH-DM, choice function hyper-heuristic (CFHH) [CE9], case-based hyper-heuristic (CBHH) [CE10], simulated annealing hyper-heuristic (SAHH) [CE6] and distributed-choice function hyper-heuristic (DCFHH) [CD20]. The results show that the proposed method finds equal or better solutions for 5 out of the 11 instances. For all small instances, both NLGDHH-SM and NLGDHH-DM are able to find the optimal solutions. For all medium instances, the NLGDHH variants achieve a significant improvement over the other hyper-heuristics. The NLGDHH approaches are also quite competitive in the large instance when compared to the results obtained by SAHH.

### 5.5 Experiments With Different Memory Lengths

Since NLGDHH-SM produced better results, we conducted experiments with different learning period length ($lp$). We ran experiments with $lp = 250$, $lp = 500$, $lp = 1000$, $lp = 2500$, $lp = 5000$ and $lp = 10000$. The best and average results are presented in Table 3.

We can see that for different values of $lp$, the proposed methods perform different. All static memory (SM) variants are able to find the optimal solution

Table 2: Comparison of the Proposed Great Deluge Based Hyper-heuristic and Other Hyper-heuristics from the Literature.

| Instance | NLGDHH-SD | NLGDHH-DM | CFHH | CBHH | SAHH | (DCFHH) |
|---|---|---|---|---|---|---|
| S1 | 0 | 0 | 1 | 6 | 0 | 1 |
| S2 | 0 | 0 | 2 | 7 | 0 | 3 |
| S3 | 0 | 0 | 0 | 3 | 1 | 1 |
| S4 | 0 | 0 | 1 | 3 | 1 | 1 |
| S5 | 0 | 0 | 0 | 4 | 0 | 0 |
| M1 | **51** | 54 | 146 | 372 | 102 | 182 |
| M2 | **48** | 67 | 173 | 419 | 114 | 164 |
| M3 | **72** | 84 | 267 | 359 | 125 | 250 |
| M4 | **47** | 60 | 169 | 348 | 106 | 168 |
| M5 | **61** | 93 | 303 | 171 | 106 | 222 |
| L1 | 731 | 915 | 1166 | 1068 | **653** | - |

Table 3: Comparison of the NLGDHH-SM with Different $lp$ Values

| Instance | $lp = 250$ | | $lp = 500$ | | $lp = 1000$ | | $lp = 2500$ | | $lp = 5000$ | | $lp = 10000$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg |
| S1 | 0 | 0.7 | 0 | 0.5 | 0 | 0.5 | 0 | 0.3 | 0 | 0.35 | 0 | 0.35 |
| S2 | 0 | 0.95 | 0 | 0.9 | 0 | 0.65 | 0 | 0.4 | 0 | 0.2 | 0 | 0.35 |
| S3 | 0 | 0.35 | 0 | 0.4 | 0 | 0.20 | 0 | 0.2 | 0 | 0.3 | 0 | 0.40 |
| S4 | 0 | 1 | 0 | 0.85 | 0 | 1.5 | 0 | 0.8 | 0 | 0.5 | 0 | 0.55 |
| S5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| M1 | 54 | 61.6 | 53 | 56.9 | 51 | 60.1 | **38** | 53 | 42 | 51.35 | 44 | 52.15 |
| M2 | 51 | 61.6 | 52 | 63.35 | 48 | 59.05 | **37** | 50.3 | 44 | 51.4 | 44 | 52.75 |
| M3 | 70 | 101.2 | 62 | 78.4 | 60 | 83.9 | 61 | 75.45 | **60** | 79.5 | 61 | 79.65 |
| M4 | 40 | 56.45 | 53 | 61.25 | 47 | 54.9 | 41 | 49.35 | **39** | 47.2 | 43 | 49.1 |
| M5 | 68 | 87.8 | 62 | 77.15 | 61 | 84.15 | 61 | 76.95 | **55** | 79.05 | 62 | 78.45 |
| L | 818 | 937.4 | 755 | 939.85 | 731 | 888.65 | **638** | 829.05 | 713 | 875.1 | 831 | 918.75 |

for small instances. For medium and large instances $lp = 2500$ and $lp = 5000$ give better results. For the large instance $lp = 2500$ gives better results than all other values of $lp$. NLGDHH-SM performed worst with $lp = 250$. The overall performance for different $lp$ values is shown in Figures 10 and 11. From these experiments, we can conclude that longer length of learning period produces better quality solutions than $lp$ with shorter values.

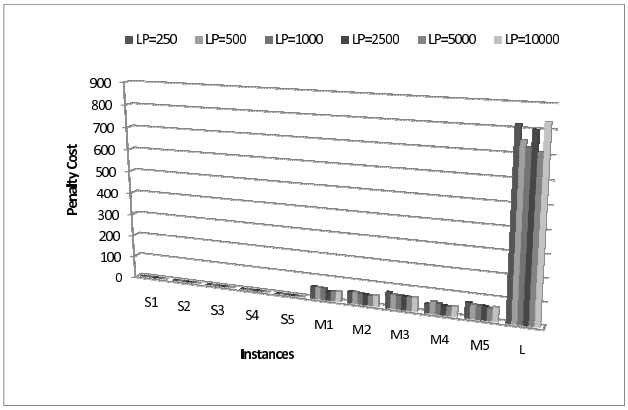


Fig. 10: Best Results Obtained by NLGDHH-SM with Different $lp$ Values

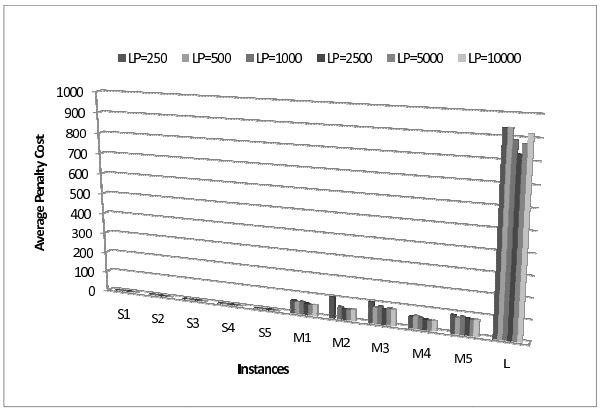


Fig. 11: Average Results Obtained by NLGDHH-SM with Different $lp$ Values

### 5.6 Comparison to Best Known Results

Finally, we compare the results obtained by the NLGDHH to the best results reported in the literature for the subject problem. The first five columns in

Table 4 show the results obtained by NLGDHH, while the fifth column shows the best known results and the corresponding approaches. It should be noted that although a timetable with zero penalty exists for each problem instance (the data sets were generated starting from such a timetable [CE23]), to the best of our knowledge no heuristic method has found the ideal timetable for the medium and large instances. Hence, these data sets are still very challenging for heuristic search methods. For all small instances, both approaches NLGDHH-SM and NLGDHH-DM produced optimal solutions. For medium instances, NLGDHH-SM improved the best solutions of M1, M2, M3, M4 and M5 while NLGDHH-DM improved the best solution of M1, M2, M3, and M4. For the large instance, neither NLGDHH-DM nor NLGDHH-DM improved the best solution reported but they are very competitive.

Table 4: Comparison of the Proposed Great Deluge Based Hyper-heuristic to the Best Results Reported in the Literature for the Course Timetabling Problem of Socha et al. [CE23].

| Instance | NLGDHH-SM LP=1000 | NLGDHH-SM LP=2500 | NLGDHH-SM LP=5000 | NLGDHH-DM | Best Known |
|---|---|---|---|---|---|
| S1 | **0** | **0** | **0** | **0** | **0** (VNS-T) |
| S2 | **0** | **0** | **0** | **0** | **0** (VNS-T) |
| S3 | **0** | **0** | **0** | **0** | **0** (CFHH) |
| S4 | **0** | **0** | **0** | **0** | **0** (VNS-T) |
| S5 | **0** | **0** | **0** | **0** | **0** (MMAS) |
| M1 | 51 | **38** | 42 | 54 | 80 (EGD) |
| M2 | 48 | **37** | 44 | 67 | 105 (EGD) |
| M3 | 60 | 61 | **60** | 84 | 139 (EGD) |
| M4 | 47 | 41 | **39** | 60 | 88 (EGD) |
| M5 | 61 | 61 | **55** | 93 | 88 (EGD) |
| L1 | 731 | 638 | 713 | 915 | **529**(HEA) |

NLGDHH-SM is the Non-Linear Great Deluge Hyper-heuristic with fixed memory length
NLGDHH-DM is the Non-Linear Great Deluge Hyper-heuristic with dynamic memory length
MMAS is the MAX-MIN Ant System in [CE23]
CFHH is the Choice Function Hyper-heuristic in [CE9]
VNS-T is the Hybrid of VNS with Tabu Search in [CE1]
HEA is the Hybrid Evolutionary Algorithm in [CE2]
EGD is the Extended Great Deluge in [CE19]

## 6 Conclusions

We have developed a hyper-heuristic approach that uses a reinforcement learning (RL) mechanism and a non-linear great deluge (NLGD) acceptance criterion to manage the selection of low-level heuristics during the search process. The method focuses on trying to choose the most appropriate heuristic in each step

of the search and hence it follows the *hyper-heuristic* concept. We appplied the proposed method to well-known instances of the university course timetabling problem proposed by Socha et al. [CE23]. The experimental results showed that the proposed non-linear great-deluge hyper-heuristic (NLGDHH) was able to find new best solutions for 5 out of the 11 problem instances compared to results reported in the literature. However, for the large instance, the algorithm produced only competitive results. We believe that for very large search spaces, the learning mechanism becomes less effective. Our future work contemplates the decomposition of large problems into smaller ones where the proposed algorithm seems to be very effective. We also want to incorporate a larger number of low-level heuristics and perhaps some more specialised operators. Another issue that requires further investigation is the robustness of the learning mechanism with respect to the various algorithm parameters.

## References


[CE1] S. Abdullah, E.K. Burke, B. McCollum: An investigation of variable neighbourhood search for university course timetabling. In: Proceedings of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, pp. 413-427, NY, USA, 2005.

[CE2] S. Abdullahm, E.K. Burke, B. McCollum: Using a randomised iterative improvement algorithm with composite neighbourhood structures for university course timetabling. In: Metaheuristics - progress in complex systems optimization, Springer, pp. 153-172, 2007.

[CE3] S. Abdullah, E. Burke, B. McCollum: A hybrid evolutionary approach to the university course timetabling problem. In: Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007), pp. 1764-1768, 2007.

[CE4] H. Asmuni, E. Burke, J. Garibaldi: Fuzzy multiple heuristic ordering for course timetabling. In: Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI 2005), pp. 302-309, 2005.

[CE5] M. Ayob, G. Kendall: An investigation of an adaptive scheduling approach for multi-head placement machines. In: Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2003), pp. 363-380, Nottingham, UK, 2003.

[CE6] R. Bai, E.K. Burke, G. Kendall, B. McCollum: Memory length in hyper-heuristics: An empirical study. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched2007), pp. 173-178, Hawaii, USA, 2007.

[CE7] E. Burke, Y. Bykov, J.P. Newall, S. Petrovic: A time-predefined approach to course timetabling. Yugoslav Journal of Operations Research (YUJOR), 13(2):139-151, 2003.

[CE8] E. Burke, K. Hart, G. Kendall, J. Newall, P. Ross, S. Schulenburg: Hyper-heuristics: an emerging direction in modern search technology. In: Handbook of Meta-heuristics, Fred Glover, Gary A. Kochenberger (eds.), pp. 457-474, Kluwer Academic Publishers, 2003.

[CE9] E. Burke, G. Kendall, E. Soubeiga: A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics, 9:451-470, 2003.

[CE10] E. Burke, B. McCollum, A. Meisels, S. Petrovic, Q. Rong: A graph based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176:177-192, 2007.

[CE11] E. Burke, S. Petrovic, R. Qu: Case based heuristic selection for timetabling problems. Journal of Scheduling, 9(2):115-132, 2006.

[CE12] E.K. Burke, G. Kendall, J.D. Landa-Silva, R. O'Brien, E. Soubeiga: An ant algorithm hyperheuristic for the project presentation scheduling problem. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Volume 3, pp. 2263-2270, Edinburgh, Scotland, 2005.

[CE13] T.B. Cooper, H. Kingston: The complexity of timetable construction problems. Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT 1995), LNCS, 1153, Springer, pp. 283-295, 1996.

[CE14] P. Cowling, G. Kendall, L. Han: An investigation of a hyper-heuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of the 2002 IEEE Congress on Evolutionary Computation (CEC 2002), pages 1185-1190, Honolulu, Hawaii, 2002.

[CE15] G. Dueck: New Optimization Heuristic: The Great Deluge Algorithm and the Record-to-record Travel. Journal of Computational Physics, 104:86–92, 1993.

[CE16] G. Kendall, M. Mohamad: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proceedings of the 2004 IEEE International Conference on Network (ICON2004), pp. 769-773, Singapore, 2004.

[CE17] D. Landa-Silva, J.H. Obit. Great deluge with nonlinear decay rate for solving course timetabling problems. In: Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), IEEE Press, pp. 8.11-8.18, 2008.

[CE18] D. Landa-Silva, J.H. Obit: Evolutionary nonlinear great deluge for university course timetabling. In: Proceedings of the 2009 International Conference on Hybrid Artificial Intelligence Systems (HAIS 2009), LNAI 5572, Springer, pp. 269-276, 2009.

[CE19] P. McMullan: An extended implementation of the great deluge algorithm for course timetabling. In: Proceedings of the 2007 International Conference in Computational Science (ICCS 2007), LNCS 4487, Springer-Verlag, pp. 538-545, 2007.

[CD20] P. Rattadilok, A. Gaw, R. Kwan: Distributed choice function hyper-heuristics for timetabling and scheduling. In: Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004), 2004.

[CE21] P. Ross, S. Schulenburg, J. Marin-Blazquez, H. Hart: Hyper-heuristics: learning to combine simple heuristic in bin-packing problems. In: Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO 2002), pp. 942-948, New York, USA, 2002.

[CE22] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, L. Mastrolilli, B. Paechter, L. Paquete, T. Stuetzle: A comparion of the performance of different metaheuristics on the timetabling problem. In: Selected Papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), LNCS 2740, Springer, pp. 330-352, 2003.

[CE23] K. Socha, J. Knowles, M. Samples: A max-min ant system for the university course timetabling problem. In: Ant Algorithms - Proceedings of the Third International Workshop (ANTS 2002), LNCS 2463, pp. 1-13, Springer, 2002.

[CE24] K. Socha, M. Sampels, M. Manfrin: Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Applications of Evolutionary Computing - Proceedings of the 2003 EvoWorkshops, LNCS 2611, Springer, pp. 334-345, 2003.

[CE25] E. Soubeiga: Development and application of hyper-heuristic to personnel scheduling. PhD thesis, School of Computer Science, University of Nottingham, UK, 2003.