# Great Deluge with Non-linear Decay Rate for Solving Course Timetabling Problems

Dario Landa-Silva and Joe Henry Obit

*Abstract*— **Course timetabling is the process of allocating, subject to constraints, limited rooms and timeslots for a set of courses to take place. Usually, in addition to constructing a feasible timetable (all constraints satisfied), there are desirable goals like minimising the number of undesirable allocations (e.g. courses timetabled in the last timeslot of the day). The construction of course timetables is regarded as a complex problem common to a wide range of educational institutions. The great deluge algorithm explores neighbouring solutions which are accepted if they are better than the best solution so far or if the detriment in quality is no larger than the current water level. In the original great deluge, the water level decreases steadily in a linear fashion. In this paper, we propose a modified version of the great deluge algorithm in which the decay rate of the water level is non-linear. The proposed method produces new best results in 4 of the 11 course timetabling problem instances used in our experiments.**

*Index Terms*— **course timetabling, great deluge algorithm, local search, meta-heuristics.**

## I. Introduction

Constructing good quality timetables is a difficult task due to the combinatorial and highly constrained nature of most timetabling problems [1]. In this paper, we are interested in the course timetabling problem in which a set of events must be assigned to timeslots and rooms ensuring the satisfaction of a number of constraints (e.g. events should not be timetabled at certain times). In particular, we tackle the set of 11 test instances of the course timetabling problem proposed by Socha, Knowles and Samples [2]. A number of heuristic approaches have been proposed in the literature to tackle those instances, which are representative of real-world course timetabling problems. Those instances have proven to be very challenging for most of the methods proposed in the literature. In this problem the quality of a solution is measured by the overall penalty due to the violation of soft constraints and the aim is to minimise such penalty. We present a simple yet effective modification of the great deluge algorithm proposed by Dueck [3]. In the original great deluge method, the water level is set to a value higher than the expected penalty of the best solution at the start of the search. Then, the water level is decreased in a linear fashion during the search until it reaches a value of zero. During the search, the algorithm explores solutions in the neighborhood of the best solution. A new solution with a lower penalty is accepted straight away replacing the best solution. A new solution with a higher penalty is accepted only if this worse penalty is not higher than the current water level.

The modification to the conventional great deluge method proposed in this paper is on the decay rate of the water level. We propose a non-linear great deluge algorithm in which the water level decay rate is controled by an exponenetial function. The proposed algorithm is capable of producing new best results for 4 of the 11 instances while still finding competitive solutions to the other 7 instances. The rest of this paper is organised as follows. Section II describes the course timetabling problem considered in this paper. Section III gives an account of heuristic algorithms proposed previously to tackle this problem and the best results obtained so far. The non-linear great deluge algorithm proposed in this paper is described in Section IV. Experiments and results are presented and discussed in Sections V and VI. While Section V focuses on the overall performance of the proposed method, Section VI studies in more detail the effect that the non-linear decay rate has on the overall performance of the algorithm. Final remarks and future work are the subject of Section VII.

## II. The Course Timetabling Problem

We tackle the university course timetabling problem (see [4] for description of different types of timetabling problems) which refers to the process of allocating, subject to constraints, a set of limited timeslots and rooms to courses, in such a way as to satisfy as nearly as possible a set of desirable objectives. In this problem, constraints can be distinguished into hard constraints and soft constraints. Hard constraints must be satisfied, i.e. a timetable is feasible only if no hard constraint is violated. Soft constrains might be violated but the number of violations has to be minimised in order to increase the quality of the timetable.

Several formulations of the course timetabling prob-

Dario Landa Silva is with the School of Computer Science, University of Nottingham, Wollaton Road, Nottingham, NG8 1BB, United Kingdom, e-mail: jds@cs.nott.ac.uk

Joe Henry Obit is with the School of Computer Science, University of Nottingham, Wollaton Road, Nottingham, NG8 1BB, United Kingdom, e-mail: jzh@cs.nott.ac.uk

978-1-4244-1739-1/08/$25.00 © 2008 IEEE

lem have been proposed in the literature. We adopt the one by Socha, Knowles and Samples [2] and the corresponding benchmark data sets in order to test the algorithm proposed in this paper.

More formally defined, the course timetabling problem tackled in this paper 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
- a set $S$ of students

Each room has a limited capacity and each student attends a number of events which is a subset of $E$. The problem is to assign the $n$ events to the $k$ 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 sets proposed by Socha, Knowles and Samples [2] are split according to their size into 5 small, 5 medium and 1 large, i.e. 11 instances in total. 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 instances, $n = 400$, $m = 10$, $|S| = 400$, $|F| = 10$. For all instances, $k = 45$ (9 hours in each of 5 days)[1].

### A. Hard Constraints

There are four hard constraints in this problem:

- A student cannot attend two events simultaneously, i.e. events with students in common must be timetabled in different timeslots.
- Only one event is allowed to be assigned per timeslot in each room.
- The room capacity must be equal to or greater than the number of students attending the event in each timeslot.
- The room assigned to an event must satisfy the features required by the event.

### B. Soft Constraints

There are three soft constraints in this problem:

- Students should not have only one event timetabled on a day.
- Students should not have to attend more that two consecutive events on a day.
- Students should not have to attend an event in the last timeslot of a day.

### C. Data Structures

The data for each problem instance includes the size and features for each room, the number of students attending each event and information about conflicting events (those with students in common). The information from each problem instance is stored into five matrices to be used by the heuristic algorithm described here. These matrices are named: *StudentEvent*, *EventFeatures*, *RoomFeatures*, *SuitableRoom* and *EventConflict*.

The *StudentEvent* matrix of size $|S| \times n$ has a value of 1 in cell $(i, j)$ if student $i \in S$ should attend event $j \in E$, 0 otherwise. The *EventFeatures* matrix of size $n \times |F|$ has a value of 1 in cell $(i, j)$ if event $i \in E$ requires room feature $j \in F$, 0 otherwise. The *RoomFeatures* matrix of size $m \times |F|$ has a value of 1 in cell $(i, j)$ if room $i \in R$ has feature $j \in F$, 0 otherwise. The *SuitableRoom* matrix of size $n \times m$ is used to quickly identify all rooms that are suitable (in terms of size and features) for each event, a value of 1 in cell $(i, j)$ indicates that room $j \in R$ has the capacity and features required for event $i \in E$. The *EventConflict* matrix of size $n \times n$ has a value of 1 in cell $(i, j)$ if events $i, j \in E$ have students in common. The *EventConflict* matrix helps to quickly identify events that can potentially be assigned to the same timeslot.

## III. Previous Work

In this section we give an account of previous work by other researchers on this course timetabling problem. Socha, Knowles and Samples [2] presented a *MAX-MIN* ant system in which a construction graph is followed by the artificial ants in order to assign timeslots to events. Then, rooms are assigned to pairs event-timeslot using the matching algorithm to produce a full timetable which is then further improved by local search. Socha, Knowles and Samples found that the artificial ants were indeed capable of learning to construct good timetables. Later, Socha, Samples and Manfrin [5] compared the *MAX-MIN* ant system to an ant colony system but the former algorithm had a better overall performance. The main difference between these two algorithms is on the strategy to update the pheromone.

Burke, Kendall and Soubeiga [6] tackled this problem using a hyper-heuristic in which a strategy based on a choice function and a tabu list guides the iterative application of a set of simple local search heuristics. They used the same six local search heuristics proposed earlier by Socha, Knowles and Samples [2]. The choice function assigns fitness to each heuristic according to their success during the search. The tabu list prevents some heuristics to be used sometimes during the search. That hyper-heuristic method produced better results than the *MAX-MIN* ant system of Socha, Knowles and Samples on 4 instances (2 small and 2 medium).

Rossi-Doria et al. [7] compared the performance of several meta-heuristics on this course timetabling problem. The methods compared were: evolutionary algorithm, ant colony optimisation, iterated local search, simulated annealing and tabu search. No best solutions were reported in that work as the intention was to assess the strengths and weaknesses of each algorithm when solving this timetabling problem. As a

[1]The problem instances described above can be found at: http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html

result of their work, Rossi-Doria et al. suggested that a hybrid algorithm would be a promising research direction. They also suggested that at least two phases were needed in such algorithm, one to construct feasible timetables and another one to minimise the violation of soft constraints.

Asmuni, Burke and Garibaldi [8] implemented fuzzy multiple heuristic ordering for this problem. The key feature in that method is to use fuzzy logic to establish the ordering of events prior to be timetabled. Three graph colouring heuristics were selected as the fuzzy system input variables. The event with the highest crisp value is selected to be scheduled first, this process continues until all events are scheduled into feasible timeslots and rooms. They compared their fuzzy ordering approach, using three ordering heuristics, against five versions of single orderings, each using a single graph colouring heuristics. At the time of publication, the fuzzy multiple heuristic ordering method gave reasonable good results but it found a new best solution for only one problem instance of medium size.

Abdullah, Burke and McCollum [9] proposed several versions of variable neighbourhood search (VNS) for this problem. One basic version (VNS-basic), one modification using an exponential Monte Carlo acceptance criterion (VNS-MC) and a hybridisation with a tabu list (VNS-Tabu). The three VNS variants used eleven neighbourhood structures in increasing order of their size. In VNS-MC, non-improving solutions are accepted with some probability. In VNS-Tabu, the tabu list was used to penalise the neighborhood structures not performing well or not leading to promising solutions after certain number of iterations. The version with the best overal performance was VNS-Tabu which produced best known results for 3 of the small instances.

Later, Abdullah, Burke and McCollum [10] applied a randomised iterative improvement approach using a composite of eleven neighbourhood structures to tackle this problem. In each iteration of that algorithm, each of the eleven neighbourhood structures is explored for the current solution and the best of the eleven candidate solutions is pre-selected. Then, the Monte Carlo criterion is used to decide whether to replace the current best solution with the pre-selected candidate solution. This algorithm found overall best results for 2 of the medium problem instances.

Burke, McCollum, Meisels, Petrovic and Qu [11] applied 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. The key feature of this approach is to find good orderings of constructive heuristics to schedule events. An ordering is used to schedule several events but the ordering might change several times during the construction of one timetable. Although this hyper-heuristic method gave reasonable good results across the 11 problem instances, not new best solutions were produced. Note that Burke et al. [6], [11] proposed hyper-heuristics as more general methods but not with the intention to outperform algorithms tailored for course timetabling problems.

TABLE I
BEST RESULTS WITH PREVIOUS ALGORITHMS FOR THE COURSE TIMETABLING PROBLEM INSTANCES BY SOCHA, KNOWLES AND SAMPLES [2].

| | MMAS | CFHH | FMHO | VNS-T | RIICN | GBHH | HEA |
|---|---|---|---|---|---|---|---|
| S1 | 1 | 1 | 10 | 0 | 0 | 6 | 0 |
| S2 | 3 | 2 | 9 | 0 | 0 | 7 | 0 |
| S3 | 1 | 0 | 7 | 0 | 0 | 3 | 0 |
| S4 | 1 | 1 | 17 | 0 | 0 | 3 | 0 |
| S5 | 0 | 0 | 7 | 0 | 0 | 4 | 0 |
| M1 | 195 | 146 | 243 | 317 | 242 | 372 | 221 |
| M2 | 184 | 173 | 325 | 313 | 161 | 419 | 147 |
| M3 | 248 | 267 | 249 | 357 | 265 | 359 | 246 |
| M4 | 164.5 | 169 | 285 | 247 | 181 | 348 | 165 |
| M5 | 219.5 | 303 | 132 | 292 | 151 | 171 | 130 |
| L1 | 851.5 | 1166 | 1138 | 932 | 757 | 1068 | 529 |

MMAS is the MAX-MIN Ant System in [2]
CFHH is the Choice Function Hyper-heuristic in [6]
FMHO is the Fuzzy Multiple Heuristic Ordering in [8]
VNS-T is the Hybrid of VNS with Tabu Search in [9]
RIICN is the Randomised Iterative Improvement with Composite Neighbourhoods in [10]
GBHH is the Graph-based Hyper-heuristic in [11]
HEA is the Hybrid Evolutionary Algorithm in [12]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L1 represents the large problem instance

Recently, Abdullah, Burke and McCollum [12] presented another algorithm for this course timetabling problem. This is a hybrid approach combining a mutation operator with their previous randomised iterative improvement procedure [10]. This hybrid evolutionary algorithm produced new best results for 4 problem instances, 3 medium and the large one. The time taken by their hybrid algorithm to produce those results was around 10 hours of computation time on a personal computer.

Table I shows the results reported in the literature for the heuristic approaches surveyed in this section. 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 [2]), none of these heuristic methods has found the ideal timetable for the medium and large instances. Hence, these data sets are still very challenging for heuristic search methods. Next, we describe our proposed non-linear great deluge approach to tackle this course timetabling problem.

## IV. Non-linear Great Deluge Approach

The great deluge algorithm was originally proposed by Dueck [3] and the basic idea is very similar to simulated annealing [13]. A new candidate solution is accepted if it is better or equal than the current solution.

A candidate solution worse than the current solution will only be accepted if the penalty of the candidate solution is less than or equal to a pre-defined limit called *water level*. The great deluge algorithm was applied to course timetabling by Burke, Bykov, Newall and Petrovic [14] using the 2002 international timetabling competition dataset. The problem instances in this dataset are an extended version of the instances proposed by Socha, Knowles and Samples [2] which are used in the present paper. Burke, Bykov, Newall, and Petrovic observed good performance of great deluge on all the 20 problem instances in the 2002 competition dataset. Overall, their experimental results showed superiority of great deluge when compared to simulated annealing. Given the success of great deluge on course timetabling problems reported by Burke, Bykov, Newall and Petrovic [14], our aim here is to investigate how the performance of this simple but effective method can be further improved. In this paper, we test the proposed modified great deluge method on the instances by Socha, Knowles and Samples [2]. In the future we also intend to test our method on the 20 instances of the 2002 timetabling competition and other course timetabling benchmarks that are available in the literature.

### *A. Neighbourhood Structures*

We employ three neighbourhood moves in the overall algorithm from initialisation to improvement of solutions. Move M1 selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random. Move M2 selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained. Move M3 identifies an event that violates soft constraints and then it moves that event to another pair timeslot-room selected at random and also ensuring feasibility is maintained. Note that the three neighbourhood moves are based on random search but always seeking the satisfaction of hard constraints. Also note that the difference between moves M1 and M3 is whether the violation of soft constraints is taken into account or not when selecting the event to re-schedule. The three neighbourhood moves used here might seem too simple but this is because we want to better assess the effectiveness of the non-linear decay rate in the proposed algorithm for guiding the local search.

### *B. Heuristic to Construct Feasible Timetables*

To construct feasible timetables, we took the heuristic proposed by Chiarandini, Birattari, Socha and Rossi-Doria [15] and added the highest degree heuristic (a well-known graph colouring heuristic) to Step 1 as described next. This modification was necessary in our approach because otherwise we were unable to generate feasible solutions for the large problem instance. The resulting initialisation heuristic works as follows.

TABLE II
TIME RANGE (IN SECONDS) TAKEN TO CONSTRUCT AN INITIAL FEASIBLE TIMETABLE FOR 10 RUNS OF THE INITIALISATION HEURISTIC.

| | **Minimum Time (s)** | **Maximum Time (s)** |
|---|---|---|
| S1 | 0.07800 | 0.12500 |
| S2 | 0 .0790 | 0.10900 |
| S3 | 0.06800 | 0.11000 |
| S4 | 0.04700 | 0.11000 |
| S5 | 0.07800 | 0.11000 |
| M1 | 7.54600 | 9.3130 |
| M2 | 9.65600 | 10.9370 |
| M3 | 13.4370 | 21.7020 |
| M4 | 6.89100 | 7.76600 |
| M5 | 16.6700 | 143.560 |
| L1 | 300 | 3000 |

S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L1 represents the large problem instance

**Step 1 - Highest Degree Heuristic**. In each iteration, the unassigned event with the highest number of conflicts (other events with students in common) is assigned to a timeslot selected at random. Once all events have been assigned to a timeslot, we use the maximum matching algorithm for bipartite graph (see Chiarandini, Birattari, Socha and Rossi-Doria [15]) to assign each event to a room. At the end of this step, there is no guarantee for the timetable to be feasible.

**Step 2 - Local Search**. We use neighbourhood moves M1 and M2 to improve the timetable generated in Step 1. A move is only accepted if it improves the satisfaction of hard constraints (this is because the moves seek to achieve feasibility). This step terminates if after 10 iterations no move has produced a better (closer to feasibility) solution.

**Step 3 - Tabu Search**. We apply tabu search using only move M1. The tabu list contains events that were assigned less than $tl$ iterations before calculated as $tl = ran(10) + \delta \times n_c$, where $ran(10)$ is a random number between 0 and 10, $n_c$ is the number of events involved in hard constraint violations in the current timetable, and $\delta = 0.6$. This step terminates if after 500 iterations no move has produced a better (closer to feasibility) solution.

Steps 2 and 3 above are executed iteratively until a feasible solution is found. This three-step initialisation heuristic is capable of finding feasible timetables for most problem instances in reasonable computation times as shown in Table II. The exception is the large instance which is the most difficult and it takes much longer time (a minimum of 300 seconds) to find a feasible timetable. The reason is that the density matrix for this dataset indicates a large number of conflicting events (with students in common).

### *C. Non-linear and Floating Water Level Decay Rate*

The distinctive feature of the great deluge algorithm is that when the new candidate solution $S^*$ is worse than the current solution $S$, then $S^*$ replaces $S$ de-

pending on the current water level $B$. At the start of the algorithm, the water level is usually set according to the quality of the initial solution. The decay rate, i.e. the speed at which $B$ decreases, is determined by a linear function in the original great deluge algorithm [3] ($B = B - \Delta B$ where $\Delta B$ is a constant).

The modification proposed in this paper is to use a non-linear decay rate for the water level given by the following expression:

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

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. In this paper we set $\beta = 0$ because we want the water level to reach the value of zero (i.e. touching the $y$ axis) by the end of the search. This is because, as we mentioned at the end of Section III, we know that a penalty on zero is possible in the problem instances tackled in this paper. If for a given problem we knew that the minimum penalty that can be achieved is lets say 100, then we would set $\beta$ around that value. If there is no previous knowledge on the minimum penalty expected, then we suggest to tune $\beta$ through preliminary experimentation for the problem in hand.

The role of the parameters $min$ and $max$ (more specifically the expression $\exp^{-\delta(rnd[min,max])}$) is to control the speed of the decay rate and hence the speed of the search process. The higher the values of $min$ and $max$, the faster the water level goes down, and in consequence, the search quickly achieves improvement but it also gets stuck in local optima very early.

In this paper, the value of the parameters in Eq. (1) were determined by experimentation. We assigned $\delta$ the values of $5 \times 10^{-10}$, $5 \times 10^{-8}$ and $5 \times 10^{-9}$ for small, medium and large instances respectively. As said before, the value of $\beta$ for all problem instances is $\beta = 0$. The values of $min$ and $max$ in Eq. (1) are set according to the size of the problem instance. For medium and large problems we used $min = 100000$ and $max = 300000$. For small problems we used $min = 10000$ and $max = 20000$. The use of the non-linear decay rate is shown in the last **else** of Algorithm 1 below.

In addition to using a non-linear decay rate for the water level $B$, we also allow $B$ to go up when its value is about to converge with the penalty cost of the candidate solution $S^*$. This occurs when $range < 1$ in Algorithm 1. We increase the water level $B$ by a random number within the interval $[B_{min}, B_{max}]$. For small problem instances the interval used was [2,5]. For the large problem instance the interval used was [1,3]. For medium problem instances, we first check if the penalty of the best solution so far $f(S_{best})$ is lower than a parameter $f_{low}$. If this is the case, then we use [1,4] as the interval $[B_{min}, B_{max}]$. Otherwise, we assume that the best solution so far seems to be stuck in local optima ($f(S_{best}) > f_{low}$) so we make $B = B + 2$. Full details of this strategy to control the water level decay rate in the modified great deluge are shown in Algorithm 1 below.

**Algorithm 1**: Non-linear Great Deluge Algorithm

```
Construct initial feasible solution S
Set best solution so far S_best ← S
Set timeLimit according to problem size
Set initial water level B ← f(S)
while elapsedTime ≤ timeLimit do
  Select move at random from M1,M2,M3
  Define the neighbourhood N(S) of S
  Select candidate solution S* ∈ N(S) at random
  if ( f(S*) ≤ f(S) or f(S*) ≤ B ) then
    S ← S* {accept new solution}
    S_best ← S {update best solution}
  end if
  range = B − f(S*)
  if (range < 1) then
    if (Large or Small Problem) then
      B = B + rand[B_min, B_max]
    else
      if ( f(S_best) < f_low ) then
        B = B + rand[B_min, B_max]
      else
        B = B + 2
      end if
    end if
  else
    B = B × (exp^{−δ(rnd[min,max])}) + β
  end if
end while
```

## V. Experiments and Results

We evaluate how beneficial it is to modify the water level decay rate from linear to non-linear and floating in the great deluge algorithm. For each type of dataset (in terms of size) a fixed computation time ($timeLimit$) in seconds was used as the stopping condition: 3600 for small problems, 4700 for medium problems and 6700 for the large problem. This fixed computation time is only for the improvement phase, i.e. the non-linear great deluge starting from a feasible solution. For each problem instance we executed the non-linear and the original (linear) great deluge algorithms for 10 times.

Table III shows the results obtained by the non-linear and by the original great deluge algorithms alongside other results reported in the literature. The table also shows the penalty of the initial solution provided to the great deluge approaches. The best results are shown in bold for each dataset. The main goal of this comparison is to assess whether great deluge with non-linear and floating water level performs better than or similar to other algorithms that have

TABLE III

COMPARISON OF RESULTS OBTAINED BY THE NON-LINEAR GREAT DELUGE (NLGD) PROPOSED IN THIS PAPER AGAINST THE BEST KNOWN RESULTS FROM THE LITERATURE FOR THE 11 COURSE TIMETABLING PROBLEM INSTANCES.

| | Init. Sol. | GD | NLGD | Best Known |
|---|---|---|---|---|
| S1 | 198 | 17 | 3 | **0** (VNS-T) |
| S2 | 265 | 15 | 4 | **0** (VNS-T) |
| S3 | 214 | 24 | 6 | **0** (CFHH) |
| S4 | 196 | 21 | 6 | **0** (VNS-T) |
| S5 | 233 | 5 | 0 | **0** (MMAS) |
| M1 | 858 | 201 | **140** | 146 (CFHH) |
| M2 | 891 | 190 | **130** | 147 (HEA) |
| M3 | 806 | 229 | **189** | 246 (HEA) |
| M4 | 846 | 154 | **112** | 164.5 (MMAS) |
| M5 | 765 | 222 | 141 | **130** (HEA) |
| L1 | 1615 | 1066 | 876 | **529** (HEA) |

MMAS is the MAX-MIN Ant System in [2]
CFHH is the Choice Function Hyper-heuristic in [6]
VNS-T is the Hybrid of VNS with Tabu Search in [9]
HEA is the Hybrid Evolutionary Algorithm in [12]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5
L1 represents the large problem instance

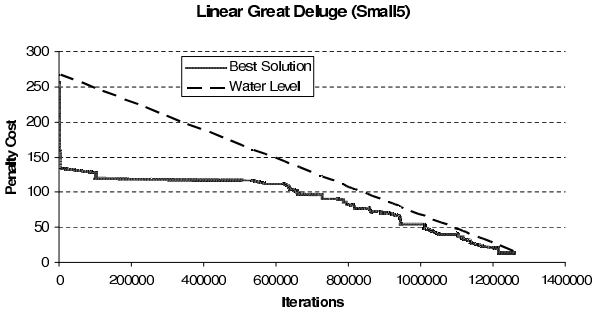
been reported in the literature. We also want to assess if the proposed modification to the water level decay rate produces better results than using the traditional linear and steady decay rate. The table shows that our algorithm outperforms some of the previous results and it is also competitive on the rest of the datasets.

First, we can see in Table III that the modified great deluge obtained results that are much better than those produced with the conventional great deluge. Also, the table shows that our algorithm outperforms some of the previous best known results and it is also competitive on the rest of the problem instances. The proposed non-linear great deluge seems particularly effective on the medium problem instances producing new best results in 4 of those problems.

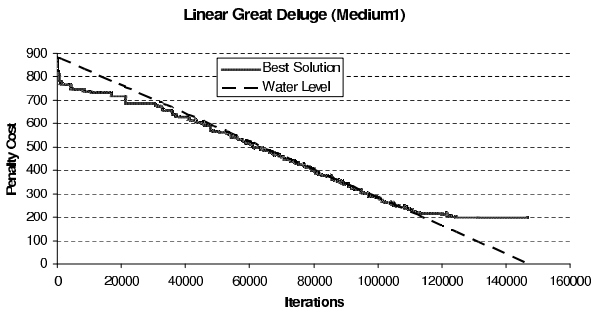
It must be said that adequate parameter tuning was required in our experiments, but the algorithm can definitely produce better results compared to the best results already published. But more importantly, the proposed algorithm can do that in short computation time, usually less than 700 seconds. We can also observe that in the small instances the algorithm is able to find solutions with low penalty cost but it cannot outperform those results reported previously. We need to further investigate this but we believe this is due to the ineffectiveness of the neighbourhood search for small instances, particularly when the penalty cost is too low. We plan to design a more effective strategy for exploring the neighbourhood of solutions and be sure to reach unexplored areas of the search space. We believe that the proposed non-linear great deluge algorithm has considerable potential to succeed in other timetabling and similar problems. This is because the improvements achieved in this paper (4 new best results in the medium instances) are mainly due to the strategy used to control the water level decay rate. Remember that the neighbourhood moves and local search strategy implemented here are quite simple and general, that is, the local search is not dependant on the problem domain.

## VI. EFFECT OF THE NON-LINEAR DECAY RATE

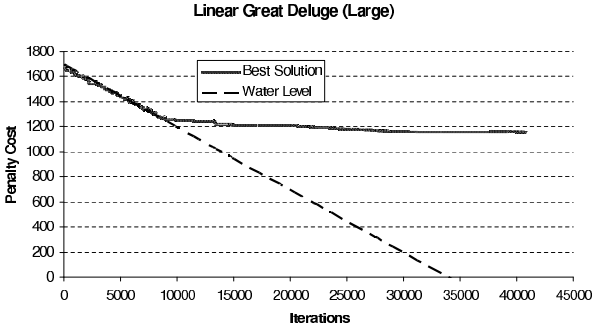
In this section we present more results to illustrate the positive effect that the non-linear decay rate has on the performance of the great deluge algorithm. Figure 1 shows the performance of linear great deluge across iterations for three problem instances while Figure 2 does the same but for the non-linear version of the algorithm. Each graph in these figures shows the search progress for one sample run of the corresponding algorithm. The dotted line corresponds to the water level and the solid line corresponds to the penalty of the best solution which should be minimised. Figure 1 shows that the water level in the original great deluge decreases at the same rate in every iteration while in the modified great deluge proposed in this paper the water level decreases exponentially according to Eq. (1).




(a)




(b)




(c)

Fig. 1. Search Progress in Linear Great Deluge

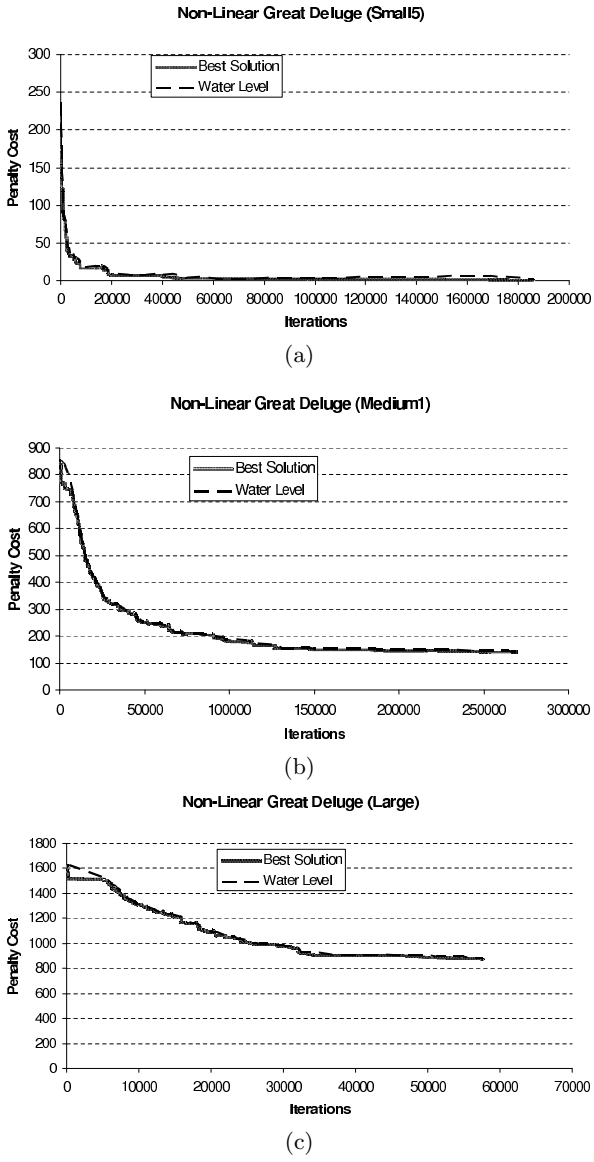



Fig. 2. Search Progress in Non-linear Great Deluge

The first interesting observation is that the relation between the water level and the best solution varies for different instance sizes. The rigid and pre-determined linear decay rate appears to suit better the medium problem instance while for the small and large instances this decay rate seems to be less effective in driving the search for the best solution. Figure 1(a) shows that in the small instance the water level is too high with respect to the best solution and this provokes that the best solution is not 'pushed down' for the first 60000 or so iterations, i.e. improvements to the best solution are rather slow. However, for the medium (Figure 1(b)) and large (Figure 1(a)) instances the water level and the best solution are very close from the start of the search so the best solution is 'pushed down' as the water level decreases. We can also see that in the medium and large instances there is a point after which the water level continues decreasing but the best solution does not improve further, i.e. the search stagnates. That is, when the water level and the best solution 'converge', the search becomes greedy and improvements are more difficult to achieve while the water level continues decreasing. This occurs around iteration 110000 in the medium instance and around iteration 8000 in the large instance. We argue that the simple linear water level decay rate in the original great deluge algorithm does not adapt easily to the quality of the best solution. This is precisely the shortcoming that we tackle in this paper and hence our proposal for a non-linear great deluge algorithm.

Then, in the non-linear version of the algorithm, the decay rate is adjusted at every iteration and the size of the problem instance being solved is taken into account when setting the parameters $\beta$, $\delta$, $min$ and $max$ as explained in Section IV. We can see in Figure 2 that this modification helps the algorithm to perform a more effective search regardless of the instance size. We can see that in the three sample runs of the non-linear great deluge algorithm, if drastic improvements are found then the water level also decreases more drastically. But when the improvement to the best solution becomes slower then the decay rate also slows in reaction to this. Moreover, to avoid (as much as possible) the convergence of the water level and the best solution, the water level is increased from time to time as explained in Section IV. This 'floating' feature of the water level explains the small increases on the best solution penalty observed in the graphs of Figure 2. As in many heuristics based on local search, the rationale for increasing the water level is to accept slightly worse solutions to explore different areas of the search space in the hope of finding better solutions.

These observations help us to summarise the key differences between the linear (original) and non-linear (modified) great deluge variants:

**Linear Great Deluge**

1. The decay rate is pre-determined and fixed
2. Mainly, the search is driven by the water level
3. When the best solution and water level converge the algorithm becomes greedy

**Non-Linear Great Deluge**

1. The decay rate changes every iteration based on (1)
2. Mainly, the water level is driven by the search
3. This algorithm never becomes greedy

## VII. Final Remarks

This paper focused on extending the conventional great deluge algorithm proposed by Dueck [3] to a version with a non-linear and floating water level decay rate. We applied this modified algorithm to 11 instances of the course timetabling problem proposed by Socha, Knowles and Samples [2]. Based on the experimental results, we showed that the non-linear great deluge outperformed previous results reported in the literature in 4 instances while still competitive in the other 7 instances. The proposed approach found new best solutions in 4 of the 5 medium problem instances. Unfortunately, it seems that this method is not very effective on the small instances. We speculate that

this is because the size of the neighbourhood in those instances is not that large to allow the non-linear and floating decay rate to take its time to diversify and intensify the search repeatedly. Another potential explanation is that the neighbourhood structures might no be effective anymore once the penalty cost reaches a very low value. Further investigation is needed on what type of moves should be implemented when the search is stuck in that stage on small problems.

In the future we intend to test the proposed non-linear great deluge approach on other instances of course timetabling problems available in the literature and other related timetabling problems. We also intend to investigate mechanisms to automatically adapt the non-linear decay rate to the size of the problem being tackled. Another future research direction is to use the GRASP meta-heuristic [16] to construct initial solutions for course timetabling. Our algorithm is able to find feasible solutions but it takes long time to do that for the large instance (see Table II). Also, we want to investigate a population-based version of the non-linear great deluge algorithm taking into consideration the diversity among a set of timetables.

## References


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

[2] K. Socha, J. Knowles, and 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*. Springer, 2002, pp. 1–13.

[3] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *Journal of Computional Physics*, vol. 104, pp. 86–92, 1993.

[4] A. Schaerf, "A survey of automated timetabling," *Artificial Intelligence Review*, vol. 13(2), pp. 87–127, 1999.

[5] K. Socha, M. Sampels, and 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, 2003, pp. 334–345.

[6] E. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, pp. 451–470, 2003.

[7] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Sttzle, "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, 2003, pp. 330–352.

[8] H. Asmuni, E. Burke, and J. Garibaldi, "Fuzzy multiple heuristic ordering for course timetabling," in *Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI 2005)*, 2005, pp. 302–309.

[9] S. Abdullah, E. Burke, and B. McCollum, "An investigation of variable neighbourhood search for university course timetabling," in *Proceedings of MISTA 2005: The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, 2005, pp. 413–427.

[10] ——, *Metaheuristics - Progress in Complex Systems Optimization*. Springer, 2007, ch. Using a Randomised Iterative Improvement Algorithm with Composite Neighborhood Structures for University Course Timetabling, pp. 153–172.

[11] E. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177–192, 2007.

[12] S. Abdullah, E. Burke, and B. McCollum, "A hybrid evolutionary approach to the university course timetabling problem," in *Proceedings of CEC 2007: The 2007 IEEE Congress on Evolutionary Computation*, 2007, pp. 1764–1768.

[13] E. Aarts and J. E. Korts, *Simulated annealing and boltzman machines*. Wiley, 1998.

[14] E. Burke, Y. Bykov, J. Newall, and S. Petrovic, "A time-predefined approach to course timetabling," *Yugoslav Journal of Operations Research*, vol. 13(2), pp. 139–151, 2003.

[15] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria, "An effective hybrid algorithm for university course timetabling," *Journal of Scheduling*, vol. 9(5), pp. 403–432, 2006.

[16] S. Casey and J. Thompson, "Grasping the examination scheduling problem," in *Selected Papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), LNCS 2740*. Springer, 2003, pp. 231–243.