Computational Study of Non-linear Great Deluge for University Course Timetabling

Joe Henry Obit and Dario Landa-Silva

Abstract. 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 method, the water level decreases steadily in a linear fashion. In this paper, we conduct a computational study of a modified version of the great deluge algorithm in which the decay rate of the water level is non-linear. For this study, we apply the non-linear great deluge algorithm to difficult instances of the university course timetabling problem. The results presented here show that this algorithm performs very well compared to other methods proposed in the literature for this problem. More importantly, this paper aims to better understant the role of the non-linear decay rate on the behaviour of the non-linear great deluge approach.

1 Introduction

The *great deluge* algorithm is a meta-heuristic approach proposed by Dueck [12] and is inspired by the behaviour that could arise when someone seeks higher ground to avoid the rising *water level* during constant rain. For a maximisation problem, the algorithm seeks to find the highest point on a certain surface with hills, valleys and plateaus (search space). Then, it starts to rain constantly and the algorithm walks around (explores the neighbourhood) but never makes a step into the increasing water level. As it continues raining, the algorithm can explore higher and lower ground

Joe Henry Obit
ASAP Research Group, School of Computer Science,
University of Nottingham, United Kingdom
e-mail: jzh@cs.nott.ac.uk

Dario Landa-Silva
ASAP Research Group, School of Computer Science,
University of Nottingham, United Kingdom
e-mail: dario.landasilva@nottingham.ac.uk

V. Sgurev et al. (Eds.): Intelligent Systems: From Theory to Practice, SCI 299, pp. 309–328.
springerlink.com

(improving and non-improving positions) but is continually pushed to a high point (hopefully close to the optimum) until eventually it cannot escape the rising water level and it stops. The initial water level is set to a value below the fitness of the initial solution and then is increased in a linear fashion as the search progresses. Note that for a minimisation problem, the water level starts on a value above the fitness of the initial solution and decreases constantly. In this case, the algorithm seeks to find the lower point by exploring the surface and maintaining its head below the descreasing water level. One can see that great deluge is similar to simulated annealing (SA) [1] but while SA accepts non-improving solutions based on probability, great deluge does this in a more deterministic manner by controlling the water level. The original great deluge algorithm was applied to course timetabling problems by Burke, Bykov, Newall and Petrovic [6]. They observed good performance of great deluge on all the problem instances tackled.

In our previous work [15] we presented a simple but effective modification of the conventional great deluge algorithm. In that variant, the water level decreases in a non-linear fashion and it also rises from time to time in order to improve the explorative ability of the algorithm. In the present paper, our aim is to conduct a computational study of the *non-linear great deluge* (NLGD) algorithm in order to investigate the key mechanisms that make this algorithm very effective. For this study, we use a number of well-known and difficult instances of the university course timetabling problem. This problem is NP complete [11, 13] and real-world instances are very difficult mainly due to the associated constraints. The present study uses the 11 instances of the course timetabling problem proposed by Socha, Knowles and Samples [18] and the 20 instances of the 1st International Timetabling Competition. All these instances consist of a set of events that need to be assigned into timeslots and rooms ensuring the satisfaction of a number of constraints (e.g. events should not be timetabled at certain times). These instances have been 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.

The rest of the paper is organised as follows. Section 2 describes the non-linear great deluge algorithm. Section 3 describes the university course timetabling problem considered in this paper and the instances used in our experiments. Important algorithm implementation details are given in Section 4. Experiments and results are presented and discussed in Section 5 focusing on the overall performance of NLGD and the effect that the non-linear decay rate has on the overall performance of the algorithm. Conclusions and future work are the subject of Section 6.

2 The Non-linear Great Deluge Algorithm

Consider a problem in which the goal is to find the solution that minimises a given objective function. The distinctive feature of the conventional great deluge algorithm is that when the candidate solution S^* is worse than the current solution S, then S^* replaces S depending on the current water level B. The water level is initially set

according to the quality of the initial solution, that is, $B > f(S^0)$ where $f(S^0)$ denotes the objective function value of the initial solution S^0. The decay rate, i.e. the speed at which B decreases, is determined by a linear function in the conventional great deluge algorithm:

$$B = B - \Delta B \text{ where } \Delta B \in \Re^+ \tag{1}$$

The non-linear great deluge algorithm uses a non-linear decay rate for decreasing the water level. The decay rate is given by the following expression:

$$B = B \times (\exp^{-\delta(rnd[min,max])}) + \beta \tag{2}$$

The various parameters in Eq. (2) control the speed and the shape of the water level decay rate. Parameter β represents the minimum expected value corresponding to the optimal solution. In this paper, we set $\beta = 0$ because we want the water level to reach that value by the end of the search. This is because we know that an optimal value of zero is possible for the problem instances tackled in this paper (see Section 3). If for a given minimisation problem we knew that the minimum objective value that can be achieved is lets say 100, then we would set β around that value. If there is no previous knowledge on the minimum objective value expected, then we suggest to tune β through preliminary experimentation for the problem in hand. The role of the parameters δ, *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. By changing the value of these three parameters, the water level goes down faster or slower.

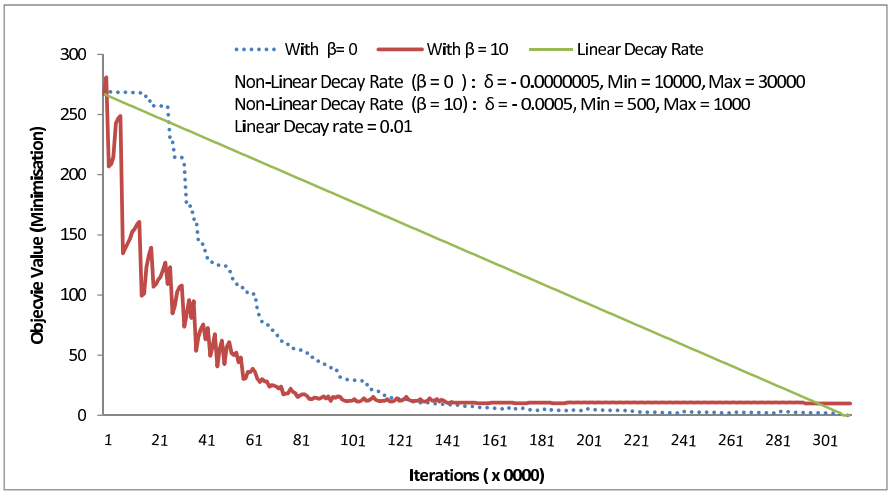

Fig. 1 Comparison between linear (Eq. 1) and non-linear (Eq. 2) decay rates and illustration of the effect of parameters β, δ, min and max on the shape of the non-linear decay rate

Figure 1 illustrates the difference between the linear and non-linear decay rates. The graph also illustrates the effect of parameters β, δ, *min* and *max* on the non-linear decay rate. The straight line in Figure 1 corresponds to the linear decay rate (with $\Delta B = 0.01$) originally proposed by Dueck [12]. 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 1 also illustrates the non-linear decay rate with different values for β, δ, *min* and *max*.

Algorithm 1: Non-linear Great Deluge (NLGD) 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 + rnd[B_min, B_max]
    else
      if ( f(S_best) < f_low ) then
        B = B + rnd[B_min, B_max]
      else
        B = B + 2
      end if
    end if
  else
    if f(S_best<=20) and Small then
      B = B × (exp^{−δ(rnd[min,max])}) + β (apply small instances parameters)
    else
      B = B × (exp^{−δ(rnd[min,max])}) + β
    end if
  end if
end while
```

Algorithm 1 corresponds to the Non-linear Great Deluge (NLGD) method and the use of the non-linear decay rate is shown in the last **else**. 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 $[B_{min}, B_{max}]$. Full details of this strategy to control the non-linear decay rate are shown in Algorithm 1 and discussed in detail in [15].

3 The University Course Timetabling Problem

3.1 Benchmark Instances

Educational timetabling refers to the allocation, subject to constraints on resources, of a set of timeslots and possibly rooms to events such as exams, lectures, lab sessions, etc. [20]. In general, educational timetabling problems can be classified into three types: school timetabling, course timetabling and examination timetabling [17]. Although these three timetabling problems share basic characteristics, significant differences among them still exist. In this paper, we are concerned with the *university course timetabling* problem which refers to the process of allocating, subject to constraints, a set of limited timeslots and rooms to events (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 university course timetabling problem have been proposed in the literature. Next, we refer to the formulation by Socha, Knowles and Samples [18].

More formally, the university course timetabling problem consists of:

- 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 limited capacity. Each student attends a number of events (subset of E). The problem is to assign the n events to the k timeslots and m rooms satisfying all hard constraints and minimising the violation of soft constraints.

There are four hard constraints in this problem:

- h1: A student cannot attend two events simultaneously, i.e. events with students in common must be timetabled in different timeslots.
- h2: Only one event can be assigned per timeslot in each room.

- h3: The room capacity must be equal to or greater than the number of students attending the event in each timeslot.
- h4: The room assigned to an event must satisfy the features required by the event.

There are three soft constraints in this problem:

- s1: Students should not have only one event timetabled on a day.
- s2: Students should not have to attend more that two consecutive events on a day.
- s3: Students should not have to attend an event in the last timeslot of a day.

We use two sets of benchmark instances for this problem. One is a set of 11 instances proposed by Socha, Knowles and Sampels [18].[1] The second set are the 20 instances used during the 1st International Timetabling Competition.[2] Details of these instances are given in Table 1 and Table 2.

Table 1 There are 11 instances (5 small, 5 medium and 1 large) in the set by Socha, Knowles and Sampels [18]. The last four rows give some indication about the structure of the instances.

	Small	Medium	Large
Number of events n	100	400	400
Number of rooms m	5	10	10
Number of room features $\|F\|$	5	5	10
Number of students $\|S\|$	80	200	400
Maximum events per student	20	20	20
Maximum students per event	20	50	100
Approximation features per room	3	3	5
Percent feature use	70	80	90

3.2 The Objective Function

The objective is to find a feasible timetable that also minimises the violation of soft constraints. The problem can be formalised as follows. Let X be the set of all possible solutions, $\bigotimes = \{h1, h2, h3, h4\}$ the set of hard constraints, $\bigoplus = \{s1, s2, s3\}$ the set of soft constraints and $\tilde{X} \subseteq X$ the set of all feasible solutions satisfying the hard constraints in $\bigotimes$. For each solution $x \in \tilde{X}$, $f(x)$ is the cost function measuring the violation of soft constraints in $\bigoplus$. The aim then is to find an optimal solution $x^* \in \tilde{X}$ such that $f(x^*) \leq f(x), \forall x \in \tilde{X}$. The cost function $f(x)$ is given by:

$$f(x) = \sum_{s \in S} (f_1(x,s) + f_2(x,s) + f_3(x,s)) \tag{3}$$

- $f_1(x,s)$: number of times a student s in timetable x has to attend a single event on a day (violation of s1). For example, $f_1(x,s) = 1$ if student s has only 1 event in a day and if student s has 2 days with only one event then $f_1(x,s) = 2$.

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

[2] These instances can be found at: http://www.idsia.ch/Files/ttcomp2002/

Table 2 There are 20 instances in the set for the 1st International Timetabling Competition. The last three columns give some indication about the structure of the instances.

Instance	No. events n	No. students $\|S\|$	No. rooms m	Rooms/event	Events/student	Students/event
com01	400	200	10	1.96	17.75	8.88
com02	400	200	10	1.92	17.23	8.62
com03	400	200	10	3.42	17.70	8.85
com04	400	300	10	2.45	17.43	13.07
com05	350	300	10	1.78	17.78	15.24
com06	350	300	10	3.59	17.77	15.23
com07	350	350	10	2.87	17.48	17.48
com08	400	250	10	2.93	17.58	10.99
com09	440	220	11	2.58	17.36	8.68
com10	400	200	10	3.49	17.78	8.89
com11	400	220	10	2.06	17.41	9.58
com12	400	200	10	1.96	17.57	8.79
com13	400	250	10	2.43	17.69	11.05
com14	350	350	10	3.08	17.42	17.42
com15	350	300	10	2.19	17.58	15.07
com16	440	220	11	3.17	17.75	8.88
com17	350	300	10	1.11	17.67	15.15
com18	400	200	10	1.75	17.56	8.78
com19	400	300	10	3.94	17.71	13.28
com20	350	300	10	3.43	17.49	14.99

- $f_2(x,s)$: number of times a student s in timetable x has to attend more than two consecutive events (violation of s2). Every extra consecutive event receives 1 penalty point. For example $f_2(x,s) = 1$ if a student s has three consecutive events and $f_2(x,s) = 2$ if the student s has four consecutive events, and so on.
- $f_3(x,s)$: number of times a student s in timetable x has to attend an event in the last timeslot of the day (violation of s3).

4 Algorithm Implementation Details

4.1 Neighbourhood Structures

We employ three neighbourhood moves in the overall approach 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. We use only these three simple neighbourhood moves (and not more sophisticated ones) to better assess the effectiveness of the non-linear decay rate in the NLGD algorithm.

4.2 Heuristic to Construct Feasible Timetables

To construct feasible timetables, we took the heuristic proposed by Chiarandini, Birattari, Socha and Rossi-Doria [10] 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 large problem instances. The resulting initialisation heuristic works as follows.

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, the maximum matching algorithm for bipartite graph (see [10] for details) is used 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 [14] using only move M1. The tabu list contains events that were assigned less than tl iterations before calculated as $tl = rnd(10) + \alpha \times n_c$, where $rnd(10)$ is a random number from a uniform distribution U[0,10], n_c is the number of events involved in hard constraint violations in the current timetable, and $\alpha = 0.6$. This step terminates if after 500 iterations no move has produced a better (closer to feasibility) solution.

In Steps 2 and 3 above, our initialisation heuristic uses simple local search and tabu search to achieve feasibility. The local search (Step 2) attempts to improve the solution but it also works as a disturbing operator, hence the reason for the maximum of 10 trials before switching to tabu search (Step 3). Note that in the tabu search, M1 selects only events that violate hard constraints. Then, Steps 2 and 3 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 Tables 3 and 4. The exception is the large instance L1 from Table 1 which is the most difficult and it takes much longer time (a minimum of 300 seconds) to find a feasible timetable. The density matrix for this instance indicates a large number of conflicting events (with students in common).

Table 3 Time range (in seconds) taken to construct an initial feasible timetable, for 10 runs of the initialisation heuristic on the instances by Socha, Knowles and Sampels [18] (see Table 1). Sx are small instances, Mx are medium instances and L1 is the large instance.

	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

Table 4 Time range (in seconds) taken to construct an initial feasible timetable, for 10 runs of the initialisation heuristic on the instances of the 1st International Timetabling Competition (see Table 2).

	Minimum Time (s)	**Maximum Time (s)**
com01	1.93	5.492
com02	1.36	2.644
com03	1.34	2.22
com04	4.464	28.98
com05	2.112	11.028
com06	1.33	3.272
com07	2.644	42.402
com08	1.82	11.086
com09	1.496	8.088
com10	4.644	29.045
com11	3.14	13.75
com12	3.016	12.632
com13	2.26	6.976
com14	5.816	50.675
com15	1.564	8.956
com16	1.092	3.884
com17	2.136	13.048
com18	1.292	2.948
com19	3.228	20.753
com20	1.804	0.085

5 Results with the NLGD Algorithm

5.1 Experimental Setting

We conducted several experiments using the two sets of benchmark instances described in Section 3. It is known that for each of those instances there is at least one assignment with an evaluation function value equal to zero, i.e. a feasible timetable satisfying all soft constraints too. For each type of instance (in terms of size) in Table 1, a fixed computation time (*timeLimit* in Algorithm 1) in seconds was set 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 NLGD algorithm, i.e. starting from an already feasible solution. However, for every instance in Table 2, the *timeLimit* was set to 2500 seconds but including finding the initial feasible timetable. The reason for this is that the time taken by our initialisation heuristic (see subsection 4.2) on the instances of Table 2 is negible, but considerable for the large instance of Table 1. For each problem instance we executed the NLGD algorithm 10 times after generating an initial timetable.

The value of the parameters in Eq. (2) were determined by experimentation. We assigned δ the values of 5×10^{-10}, 5×10^{-8} and 5×10^{-9} for small, medium and large instances of Table 1 respectively. As said before, $\beta = 0$ for all problem instances. The values of *min* and *max* were set as follows: for medium and large problems we used $min = 100000$ and $max = 300000$ while for small problems we used $min = 10000$ and $max = 20000$. However, we should note that the parameter values given above for the small instance only apply when the penalty cost reach around 20. That is, the NLGD uses the same parameter values as for the medium instances and changes to the small instance parameter values the cost function reaches the value of 20. The interval $[B_{min}, B_{max}]$ (see Algorithm 1) was set as follows. For small instances it was [2,5] and for large instances it was [1,3]. For medium 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]. Otherwise, we assume that the best solution so far seems to be stuck in local optima ($f(S_{best}) > f_{low}$) so we make $B = B + 2$ as shown in Algorithm 1.

5.2 The Computational Study

First, we evaluate how beneficial it is to have a non-linear decay rate and floating water level in the modified great deluge algorithm. In the first set of experiments, we compared the NLGD with other algorithms reported in the literature for the instances shown in Table 1. Results are reported in Table 5 where we can see the results obtained by the NLGD and by the original great deluge 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 decay rate and floating water level performs better than or similar to other algorithms that have been reported in the literature. We also want to assess if

Table 5 Comparison of results obtained by the non-linear great deluge (NLGD) against the best known results from the literature for the 11 instances of Table 1.

Instance	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 [18]
CFHH is the Choice Function Hyper-heuristic in [7]
VNS-T is the Hybrid of VNS with Tabu Search in [2]
HEA is the Hybrid Evolutionary Algorithm in [4]

the proposed modification to the water level decay rate produces better results than using the traditional linear and steady decay rate.

Table 5 shows that our algorithm outperforms some of the previous results and it is also competitive on the other instances. For the small problems, NLGD is able to solve instance S5 to optimality. For most of the medium problems, NLGD has shown significant improvement over other algorithms. However, for instance M5 the NLGD method is not able to improve the solution found by HEA. Still, NLGD is very competitive obtaining a solution quality of just around 8% worse than the best value for M5. Table 5 also shows that the NLGD algorithm obtained results that are much better than those produced with the conventional great deluge.

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 because the improvements achieved (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.

In the second set of experiments, we compared the NLGD with other algorithms reported in the literature for the instances shown in Table 2. These results are reported in Table 6 where we can see the best results obtained by different algorithms from the competition plus the results obtained by NLGD, best results are in bold. The table gives us an idea about the variability on the performance of different algorithm proposed in the competition. Results from Table 6 show that even though the NLGD did not obtain the best results, it is still very competitive particularly against the algorithms ranked fifth to ninth in the competition.

Table 6 Comparison of results obtained by the non-linear great deluge (NLGD) against the best 9 ranked algorithms for the 20 instances of Table 2. Details of the competition algorithms are available at: http://www.idsia.ch/Files/ttcomp2002/results.htm.

Instances	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	NLGD
com01	**45**	61	85	63	132	148	178	211	257	153
com02	**25**	39	42	46	92	101	103	128	112	118
com03	**65**	77	84	96	170	162	156	213	226	120
com04	**115**	160	119	166	265	350	399	408	441	358
com05	102	161	**77**	203	257	412	336	312	299	398
com06	13	42	**6**	92	133	246	246	169	209	129
com07	44	52	**12**	118	177	228	225	281	99	99
com08	**29**	54	32	66	134	125	210	214	194	111
com09	**17**	50	184	51	139	126	154	164	175	119
com10	**61**	72	90	81	148	147	153	222	308	153
com11	44	53	73	65	**35**	144	169	196	273	149
com12	107	110	**79**	119	290	182	219	282	242	229
com13	**78**	109	91	160	251	192	248	315	364	240
com14	52	93	**36**	197	230	316	267	345	156	282
com15	**24**	62	27	114	140	209	235	185	95	172
com16	**22**	34	300	38	114	121	132	185	171	91
com17	86	114	**79**	212	186	327	313	409	148	356
com18	**31**	38	39	40	87	98	107	153	117	190
com19	**44**	128	86	185	256	325	309	334	414	228
com20	**7**	26	**0**	17	94	185	185	149	113	72

In more detail, Figures 2-7 summarise the performance of NLGD compared to other allgorithms. In these graphs, the x-axis represents the instance type while the y-axis represents the penalty cost. Figure 3 shows the strong performance of NLGD on the medium and large instances. Figures 4-7 show details of the results achieved by NLGD when compared to the algorithms from the competition.

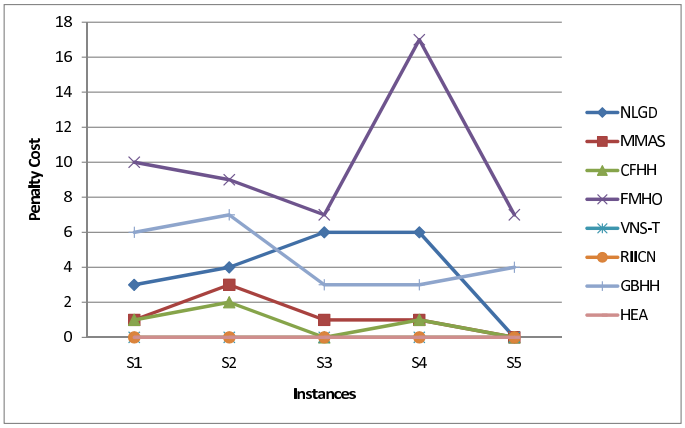

Fig. 2 Detailed comparison of non-linear great deluge against other algorithms for small instances from Table 1

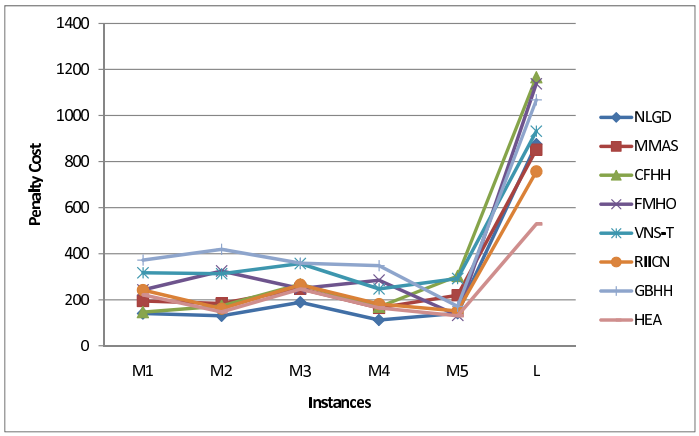

Fig. 3 Detailed comparison of non-linear great deluge against other algorithms for medium and large instances from Table 1

5.3 Effect of the Non-linear Decay Rate

Here we present more results to illustrate the positive effect that the non-linear decay rate has on the performance of the NLGD algorithm. Figures 8-10 show the performance of linear great deluge (GD) across iterations for three problem instances while Figures 11-13 do the same but for the non-linear version of the algorithm (NLGD). 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 so far which should be

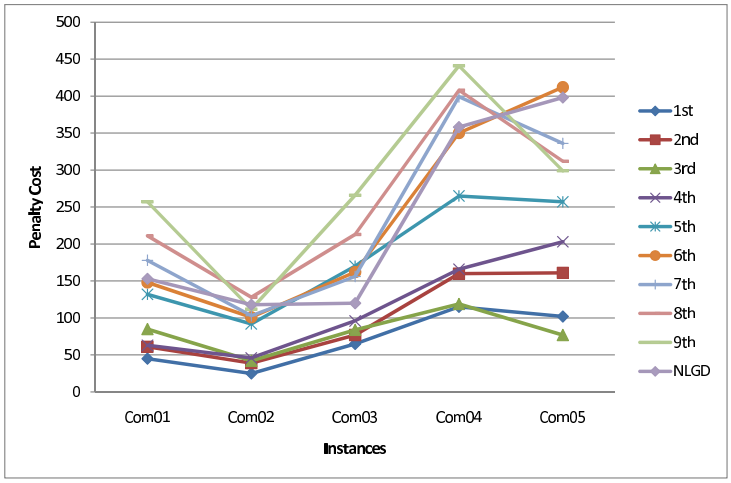

Fig. 4 Detailed comparison of non-linear great deluge against other algorithms for com01-com05 instances from Table 2

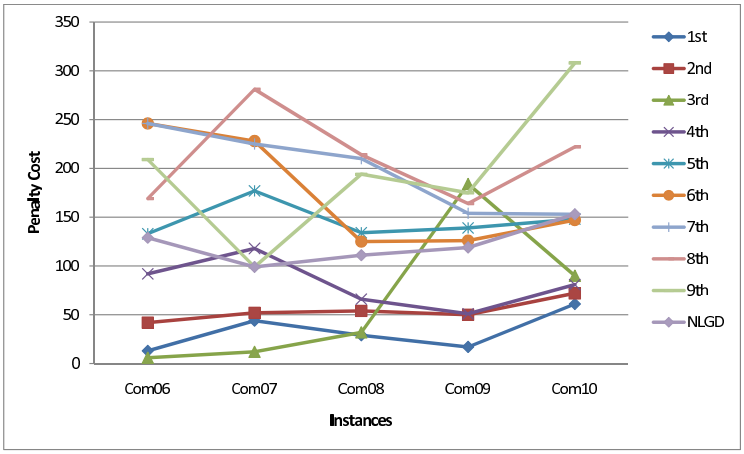

Fig. 5 Detailed comparison of non-linear great deluge against other algorithms for com06-com10 instances from Table 2

minimised. The water level in the GD decreases at the same rate in every iteration while in the NLGD the water level decreases exponentially according to Eq. (2).

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 instance while for the small and large instances this decay rate seems to be less effective in driving the search for

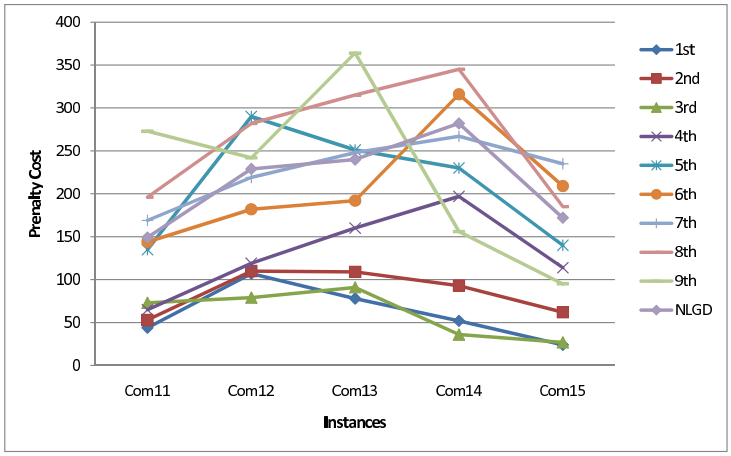

Fig. 6 Detailed comparison of non-linear great deluge against other algorithms for com11-com15 instances from Table 2

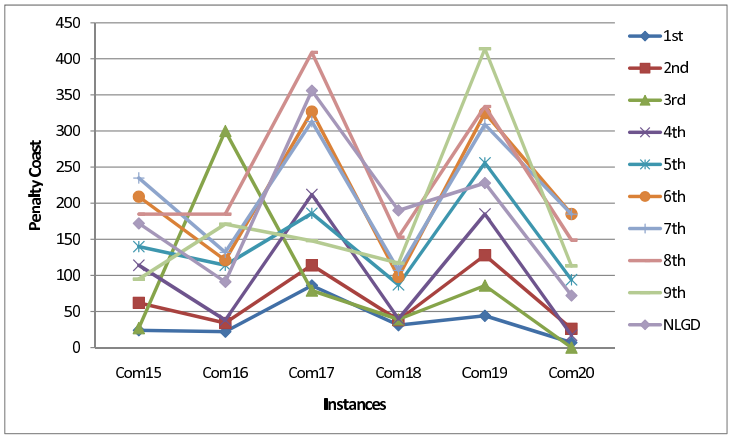

Fig. 7 Detailed comparison of non-linear great deluge against other algorithms for com16-com20 instances from Table 2

the best solution. Figure 8 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 9) and large (Figure 10) 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

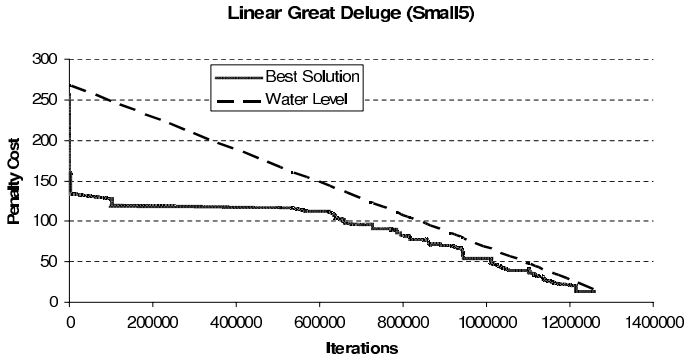

Fig. 8 Sample of search progress behaviour of GD on small instance

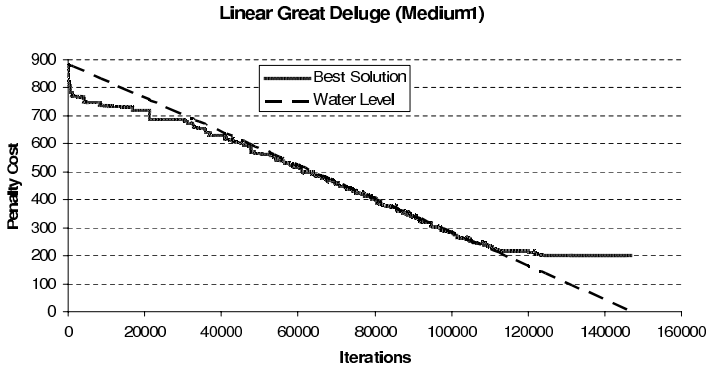

Fig. 9 Sample of search progress behaviour of GD on medium instance

search stagnates. That is, when the water level and the best solution so far '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 so far. This is precisely the shortcoming that we tackle with the 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 of Eq.(2) as explained in Section 2. We can see in Figures 11-13 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 so far becomes slower then the decay rate also slows in reaction to this. Moreover,

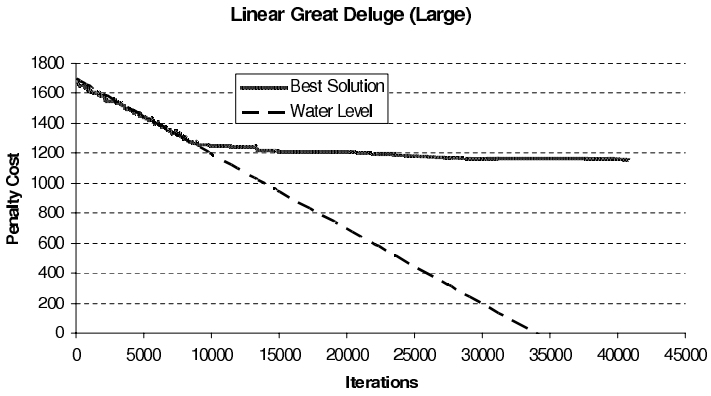

Fig. 10 Sample of search progress behaviour of GD on large instance

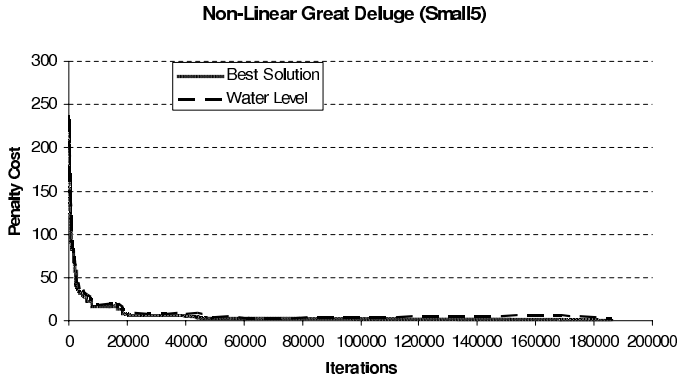

Fig. 11 Sample of search progress behaviour of NLGD on small instance

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 2. This 'floating' feature of the water level explains the small increases on the best solution penalty observed in the graphs of Figures 11-13. 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.

The above observations help us to summarise the key differences between the linear (GD) and non-linear (NLGD) 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

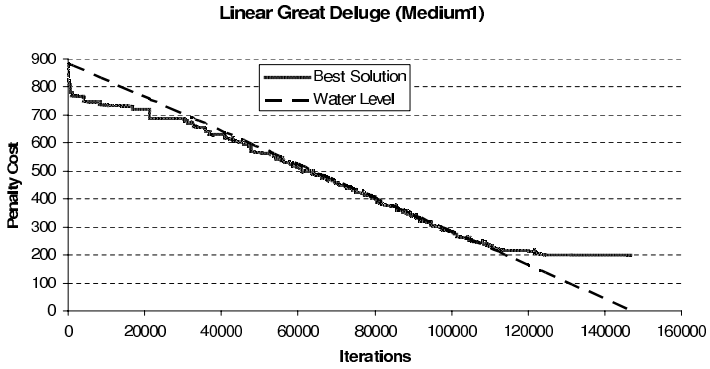

Fig. 12 Sample of search progress behaviour of NLGD on medium instance

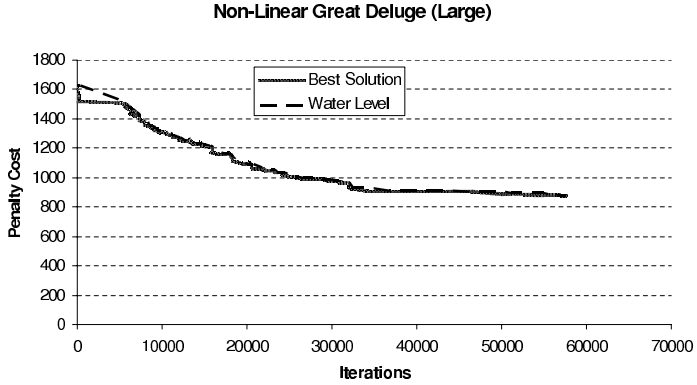

Fig. 13 Sample of search progress behaviour of NLGD on large instance

Non-Linear Great Deluge

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

6 Conclusions

This paper presented a computational study of the *non-linear great deluge* (NLGD) algorithm [15] which is an extension of the conventional great deluge method [12]. The NLGD approach incorporates a non-linear decay rate and floating water level. We applied this modified algorithm to well known benchmark instances of the university course timetabling problem: the 11 instances proposed by Socha, Knowles and Samples [18] and the 20 instances from the 1st International Timetabling Competition. The NLGD algorithm performs very well in both sets of instances and this

study showed that the non-linear decay rate and floating water level are key components for the robust performance on this algorithm. In future work, we intend to investigate mechanisms to automatically adapt the non-linear decay rate to the size of the problem instance being tackled. 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. Aarts, E., Korts, J.: Simulated Annealing and Boltzman Machines. Wiley, Chichester (1998)
2. Abdullah, S., Burke, E.K., McCollum, B.: An Investigation of Variable Neighbourhood Search for University Course Timetabling. In: Proceedings of MISTA 2005: The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, pp. 413–427 (2005)
3. Abdullah, S., Burke, E.K., McCollum, B.: A Hybrid Evolutionary Approach to the University Course Timetabling Problem. In: Proceedings of CEC 2007: The 2007 IEEE Congress on Evolutionary Computation, pp. 1764–1768 (2007)
4. Abdullah, S., Burke, E.K., McCollum, B.: Using a Randomised Iterative Improvement Algorithm with Composite Neighborhood Structures for University Course Timetabling. In: Metaheuristics - Progress in Complex Systems Optimization, pp. 153–172. Springer, Heidelberg (2007)
5. Asmuni, H., Burke, E.K., Garibaldi, J.: Fuzzy Multiple Heuristic Ordering for Course Timetabling. In: Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI 2005), pp. 302–309 (2005)
6. Burke, E.K., Bykov, Y., Newall, J., Petrovic, S.: A Time-predefined Approach to Course Timetabling. Yugoslav Journal of Operations Research (YUJOR) 13(2), 139–151 (2003)
7. Burke, E.K., Kendall, G., Soubeiga, E.: A Tabu-search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics 9, 451–470 (2003)
8. Burke, E.K., Eckersley, A., McCollum, B., Petrovic, S., Qu, R.: Hybrid Variable Neighbourhood Approaches to University Exam Timetabling. Technical Report NOTTCS-TR-2006-2, University of Nottingham, School of Computer Science (2006)
9. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A Graph Based Hyperheuristic for Educational Timetabling Problems. European Journal of Operational Research 176, 177–192 (2007)
10. Chiarandini, M., Birattari, M., Socha, K., Rossi-Doria, O.: An Effective Hybrid Algorithm for University Course Timetabling. Journal of Scheduling 9(5), 403–432 (2006)
11. Cooper, T., Kingston, H.: The Complexity of Timetable Construction Problems. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 283–295. Springer, Heidelberg (1996)
12. Dueck, G.: New Optimization Heuristic: The Great Deluge Algorithm and the Record-to-record Travel. Journal of Computational Physics 104, 86–92 (1993)
13. Even, S., Itai, A., Shamir, A.: On the Complexity of Timetabling and Multicommodity Flow Problems. SIAM Journal of Computation 5, 691–703 (1976)
14. Glover, F., Taillard, E., De Werra, D.: A User's Guide to Tabu Search. Annals of Operations Research 41, 3–28 (1993)
15. Landa-Silva, D., Obit, J.-H.: Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. In: Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), pp. 8.11–8.18. IEEE Press, Los Alamitos (2008)

16. Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., Stuetzle, T.: A Comparion of the Performance of Different Metaheuristics on the Timetabling Problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 333–352. Springer, Heidelberg (2003)
17. Schaerf, A.: A Survey of Automated Timetabling. Artificial Intelligence Review 13(2), 87–127 (1999)
18. Socha, K., Knowles, J., Sampels, M.: A Max-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)
19. Socha, K., Sampels, M., Manfrin, M.: Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) EvoIASP 2003, EvoWorkshops 2003, EvoSTIM 2003, EvoROB/EvoRobot 2003, EvoCOP 2003, EvoBIO 2003, and EvoMUSART 2003. LNCS, vol. 2611, pp. 334–345. Springer, Heidelberg (2003)
20. Wren, V.: Scheduling, Timetabling and Rostering A Specail Relationship? In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 46–75. Springer, Heidelberg (1996)