An Evolutionary Non-Linear Great Deluge Approach for Solving Course Timetabling Problems

Joe Henry Obit[1], Djamila Ouelhadj[2], Dario Landa-Silva[3] and Rayner Alfred [4]

[1] Labuan School of Informatics Science, Universiti Malaysia Sabah,
87000 Labuan F.T, Malaysia

[2] Department of Mathematics, University of Portsmouth,
PO3 1HF, United Kingdom

[3] School of Computer Science, (ASAP), University of Nottingham,
Jubilee Campus Wollaton Road Nottingham, NG8 1BB, United Kingdom

[4] School of Engineering and Information Technology, Universiti Malaysia Sabah,
2073, 88999, Kota Kinabalu, Malaysia

Abstract

The aim of this paper is to extend our non-linear great deluge algorithm into an evolutionary approach by incorporating a population and a mutation operator to solve the university course timetabling problems. This approach might be seen as a variation of memetic algorithms. The popularity of evolutionary computation approaches has increased and become an important technique in solving complex combinatorial optimisation problems. The proposed approach is an extension of a non-linear great deluge algorithm in which evolutionary operators are incorporated. First, we generate a population of feasible solutions using a tailored process that incorporates heuristics for graph colouring and assignment problems. The initialisation process is capable of producing feasible solutions even for large and most constrained problem instances. Then, the population of feasible timetables is subject to a steady-state evolutionary process that combines mutation and stochastic local search. We conducted experiments to evaluate the performance of the proposed algorithm and in particular, the contribution of the evolutionary operators. The results showed the effectiveness of the hybridisation between non-linear great deluge and evolutionary operators in solving university course timetabling problems.

Keywords: *Evolutionary Algorithm, Non-linear Great Deluge* and *Course Timetabling*.

1. Introduction

The central aim of this paper is to hybridise the non-linear great deluge algorithm presented in our previous paper [18] with the evolutionary approach by incorporating a population and a mutation operator to solve the university course timetabling problem. This technique might be seen as a variation of memetic algorithms in particular as presented in [1, 12, 21, 22]. The popularity of evolutionary computation approaches has increased and become an important technique in solving complex combinatorial problems. They are powerful techniques and have been applied to many complex problems *e.g.* the travelling salesman problem [20,17,16], university exam timetabling [10], and university course timetabling problems [11, 22, 21].

Finding good quality solutions for timetabling problems is a very challenging task due to the combinatorial and highly constrained nature of these problems [13]. In recent years, several researchers have tackled the course timetabling problem, particularly the set of 11 instances of course timetabling problem proposed by Socha et al. [26]. Among the algorithms proposed there are: *MAX-MIN* ant system [26]; tabu search hyper-heuristic strategy [8]; evolutionary algorithm, ant colony optimisation, iterated local search, simulated annealing and tabu search [24]; fuzzy multiple heuristic ordering [6]; variable neighbourhood search [3]; iterative improvement with composite neighbourhoods [2]; a graph-based hyper-heuristic [9] and a hybrid evolutionary algorithm [1].

There are many versions of evolutionary algorithms that have been discussed in the literature, however, there is a common underlying idea that underpins the basic structure of these algorithms [14], such as, and most of the evolutionary algorithms are population-based meta-heuristics. These algorithms maintain a population of solutions and conduct the search process by simulating natural selection based on Darwin's theory of survival of the fittest. This means that only strong individual solutions will survive and participate in the selection for reproduction before being subject to the process of recombination and mutation. Sastry et al. [25]

explained various types of recombination and mutation operators. Recombination is an operator which combines two or more individuals from the mating pool in order to create one or more new candidate solutions, whereas mutation is usually designed to add more diverse solutions to increase the chances of exploring large areas of the search space [25]. Mutation is only applied to one candidate solution and produces one new solution. Even though crossover is one of the main components in genetic algorithms and other evolutionary algorithms, Moscato and Norman [20] and Radcliff and Surry [23] have argued whether crossover should be the main operator in Genetic Algorithms. It is not an unusual practice that some papers present different implementations of Evolutionary Algorithms in which local search are used as a replacement for crossover. For example, Ackley [5] proposed a genetic hill-climbing approach in which the crossover operator only plays a small role in the algorithm. In addition, according to Bäeck et al. [7] the Evolutionary strategies community has emphasised on mutation rather than crossover.

This paper proposes a two-stage hybrid meta-heuristic approach to tackle course timetabling problems. The first stage constructs feasible timetables while the second stage is an improvement process that also operates within the feasible region of the search space. The second stage is a combination of non-linear great deluge [18] with evolutionary operators to improve the quality of timetables.

The rest of this paper is organised as follow, in Section 2, the subject problem and test instances are described. Section 3 gives the description of the evolutionary non-linear great deluge approach proposed for solving the university course timetabling problems. Computational experiments and results are presented in Section 4 and the paper ends with a conclusion in Section 5.

2. University Course Timetabling

In general, university course timetabling is the process of allocating, subject to predefined constraints, a set of limited timeslots and rooms to courses, in such a way as to achieve as close as possible a set of desirable objectives. In timetabling problems, constraints are commonly divided into hard and soft constraints. A timetable is said to be feasible if no hard constraints are violated while soft constraint may be violated but we try to minimise such violation in order to increase the quality of the timetable. In this work, we tackle the course timetabling problem defined by Socha et al. [26] where there are: n events $E = \{ e_1, e_2, ..., e_n \}$, k timeslots $T = \{ t_1, t_2, ..., t_k \}$ and m rooms $R = \{ r_1, r_2, ... , r_m \}$ and a set S of students. Each room has a limited capacity and a set F of features that might be required by the events. Each student must attend a number of events within 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.

Hard Constraints. There are four in this problem:

- H1: a student cannot attend two events simultaneously.
- H2: only one event can be assigned per timeslot in each room.
- H3: the room capacity must not be exceeded at any time.
- H4: the room assigned to an event must have the features required by the event.

Soft Constraints. There are three:

- S1: students should not have exactly one event timetabled on a day.
- S2: students should not have to attend more than two consecutive events on a day.
- S3: students should not have to attend an event in the last timeslot of the day.

The benchmark data sets proposed by Socha et al. [26] are split according to their size into 5 small, 5 medium and 1 large, as shown below :

Category	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
Number of 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

Table 1 Parameter values for the course timetabling problem categories in the set by Socha, Knowles and Samples [26]. The last four rows give some indication about the structure of the instances.

For all instances, k = 45 (9 hours in each of 5 days). It should be noted that although a timetable with zero penalty exists for each of these problem instances (the data sets were generated starting from such a timetable [26]), so far

no heuristic method has found the ideal timetable for the medium and large instances. Hence, these data sets are still very challenging for most heuristic search algorithms.

2.1 Problem Formulation

The objective in this problem is to find a feasible solution that minimises the violation of soft constraints. The problem data sets described above (Socha et al. instances) can be formalised as follows. Let X is the set of all possible solutions, where each event has been assigned a pair timeslot-room. Let $A = \{h1, h2, h3, h4\}$ *be* the set of all hard constraints. Let $B = \{s1, s2, s3\}$ *be* the set of all soft constraints for which violation should be minimised. Let $\tilde{X} \subseteq X$ be the set of all feasible solutions that satisfy the hard constraints in A. The cost function $f(x)$ for both problem data sets can be represented by this formulation. Each solution $x \in \tilde{X}$ is associated with a cost function measuring the total violation of soft constraints in B. The main objective of this problem is to search for an optimal solution $x^* \in \tilde{X}$, in this case an optimal solution is, if $f(x^*) \leq f(x), \forall x \in X$. The cost function $f(x)$ measures the quality of the feasible solution $x \in X$ by measuring the violation of the total soft constraints given by:

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

- $f_1(x,s)$: number of times a student s in timetable x is assigned to the last timeslot of the day.
- $f_2(x,s)$: number of times a student s in timetable x is assigned more than two consecutive classes. Every extra consecutive class will add 1 penalty point, for example $f_2(x,s) = 1$ if a student s has three consecutive classes and $f_2(x,s) : = 2$ if the student s has four consecutive classes, and so on.
- $f_3(x,s)$: number of times a student s in timetable x is assigned a single class on a day. $f_3(x,s) = 1$ if student s has only 1 class in a day and if student s has two days with only one class $f_3(x,s) = 2$.

3. Evolutionary Non-Linear Great Deluge Approach

As discussed in the introduction, crossover operator can be replaced by local search. For example Ackley [5] used hill-climbing as an operator instead of crossover after arguing that crossover was not effective and played less dominant role. Gorges-Schleuter [15] used local search as an operator in evolutionary algorithms, and showed that it definitely improves the quality of the solutions.

In this work, we propose to extend the single solution non-linear great deluge approach to a population-based evolutionary approach by incorporating tournament selection, a mutation operator and a replacement strategy. The motivation behind the introduction of evolutionary operators into our great deluge algorithm comes from the interest for striking a good balance between diversification and intensification, which are the main strategic forces in meta-heuristic approaches. Therefore, a good search technique must balance these two forces in order to achieve robustness and effectiveness in the search as well as to help the search activity to find optimal or near optimal solutions. *Diversification* is the ability to reach not yet visited regions in the search space and it can be achieved by disturbing some of the solutions using special operators (in our case, we use mutation) when necessary. *Intensification* is about exploiting the current search space regions by using local search (non-linear great deluge in our case) to obtain better quality of solutions.

Figure 2 shows the components of the proposed evolutionary non-great deluge algorithm. It begins by generating an initial population of solutions of size P which becomes the pool of solutions. Then, a number of generations take place and in each of them the algorithm works as follows. First, tournament selection is used to choose 5 individuals at random from the pool of solutions and the one with the best fitness is selected (x^t). With probability less or equal to *0.5*, a mutation operator is applied to x^t while maintaining feasibility and obtaining solution x^m. The probability value was determined by experimentation (If we apply the mutation too high or too low, no much improvement can be found). Next, the non-linear great deluge algorithm is applied to x^m to obtain an improved solution x^i. Then, the worst solution in the pool of solutions, x^w (ties broken at random) is identified and if x^i is better than x^w then x^i replaces x^w in the pool of solutions. This evolutionary non-liner great deluge algorithm is then executed for a pre-determined computation time according to the size of the problem instance. Note that this is a steady-state evolutionary approach that uses non-linear great deluge for intensification and a mutation operator for diversification. The following subsections describe each of the algorithm components is more detail.

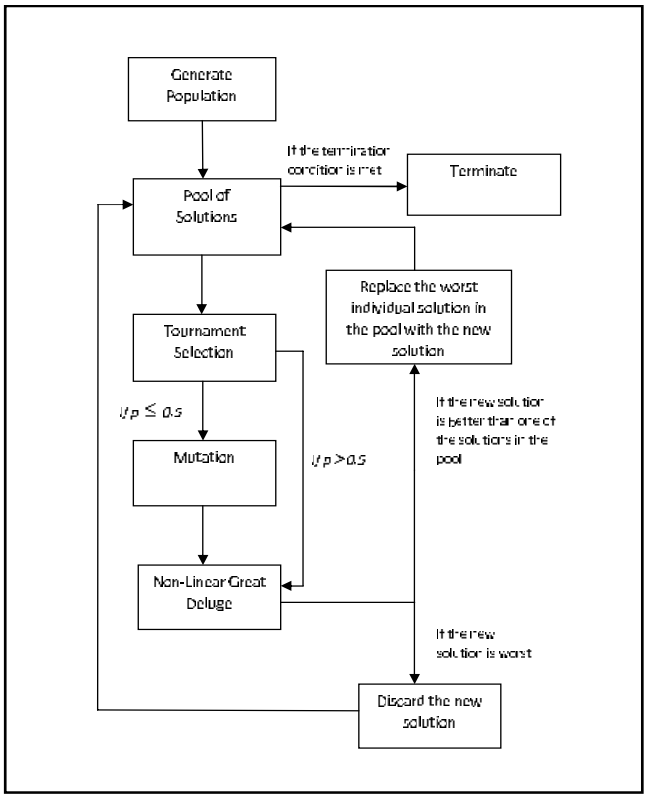

Figure 1:The Evolutionary Non-linear Great Deluge Algorithm.

3.1 Solution Representation

Each solution in the population uses a direct representation, consisting of a chromosome with information on what events or courses are assigned into a pair of timeslot-room. In addition, the chromosome is also used to keep information on forbidden assignments for a particular timeslot and room. Figure 2 illustrates the direct encoding of an individual solution used in the population. e_i is an event number i, $i \in \{1,...,n\}$ where n is the number of events that need to be scheduled in the available timeslot t, $t \in \{1, k\}$ where k is the number of available timeslots. For example event e_4 is assigned to timeslot 1 in room 1.

Timeslot 1		Timeslot 2			Timeslot k	
Room 1	e_4	Room 1	e_{14}		Room 1	e_{34}
Room 2	e_{45}	Room 2	e_{52}		Room 2	e_{18}
Room 3	e_{25}	Room 3	e_{23}	. . . . .	Room 3	e_{56}
Room 4	e_5	Room 4	e_{19}		Room 4	e_{90}
Room 5	e_{14}	Room 5	e_{100}		Room 5	e_{12}

Figure 2: Solution Representation (direct encoding) of a Timetable where events are assigned to pairs timeslot-room.

3.2 Initialisation of the Population

The initial population of solutions is generated using the heuristic described in Algorithm 1. Two well-known graph colouring heuristics are incorporated, Largest Degree (LD) and Saturation Degree (SD). First, the events in the pool of unscheduled events are sorted based on LD. After that, we choose the event with the highest LD and calculate its SD. In the first while loop, the initialisation heuristic attempts to place all events into timeslots while avoiding conflicts. In order to do that, the heuristic uses the SD criterion and a list of rescheduled events to temporarily insert the conflicting events. The heuristic tries to do this for a given $time_U$ but once that time has elapsed, all remaining unscheduled events are inserted into random timeslots. If by the end of the first while loop the solution is not yet feasible, at least the penalty due to hard constraint violations is already very low. In the second while loop, the heuristic uses simple local search and tabu search to achieve feasibility with two neighbourhood moves M1 and M2. M1 selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random. M2 selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained. The local search attempts to improve the solution but it also works as a perturbation operator. The tabu search uses move M2 only, which selects only an event that violates the hard constraints. The tabu search runs for a fixed number of iterations ts_{max}. In our experiments, this initialisation heuristic always finds a feasible solution for all the problem instances considered.

Algorithm 1: Initialisation Heuristic

```
Input: set of events in the poolOfUnscheduled events list E;
Sort the events in E by using Largest Degree (LD) heuristic;
while (poolOfunscheduled events list E is not empty) do
    Choose event e from E with the LD (tie break at random);
    Calculate SD for event e;
    if (e with SD = 0) then
        Select a timeslot t at random;
        From those events already scheduled in timeslot t (if any), move those that conflict
        with event e (if any) to the poolOfRescheduled events list;
        Place event e into timeslot t;
        for (each event e in the poolOfRescheduled events list with SD > 0) do
            Select a feasible timeslot t for event e at random;
            Recalculate SD for all events in the poolOfRescheduled events list;
        Move all events that remain in the poolOfRescheduled events list (those with SD =
        0) to the poolOfUnscheduled events list E;
    else
        Select a feasible timeslot t_t at random to place e;
    if (poolOfUnscheduled events list E is not empty and timeU has elapsed) then
        One by one, place events from the poolOfUnscheduled events list into any random
        selected timeslot without respecting the conflict between the events;
S = current solution;
loop = 0;
while (S is not feasible ) do
    if (loop < 10) then
        if ( coinflip()) then
            S* = M1(S); // apply M1 to S
        else
            S* = M2(S); // apply M2 to S
        if ( f(S*) ≤ f(s)) then
            S ← S* //accept new solution;
    else
        EHC = set of events that violate hard constraints;
        e = randomly selected member of EHC;
        S* = M2b(S, e); //Perform one iteration tabu search with move M2b using e;
        if ( f(S*) < f(S) then
            S ← S*; //accept new solution
        if (loop == ts_max + 10 ) then
            loop = 0;
    loop++;
Output: S feasible solution (timetable)
```

4.3 The Evolutionary Operator: Mutation

With a probability less or equal to 0.5 ($p \leq 0.5$), the mutation operator is applied to the solution selected from the tournament (x^t). The mutation operator selects at random one out of three types of neighbourhood moves in order to change the solution while maintaining feasibility. These moves are described below.

1. Move M1. Selects one event at random and assigns it to a feasible timeslot and room.
2. Move M2. Selects two events at random and swaps their timeslots and rooms while ensuring feasibility is maintained.
3. Move M3. Selects three events at random, then it exchanges the position of the events at random and ensuring feasibility is maintained.

5. Non-linear Great Deluge Algorithm

The non-linear great deluge algorithm is a modified great deluge algorithm which incorporates a non-linear decay rate. The motivation behind the use of a non-linear decay rate and floating water level is to enhance the feedback between the search activity and the water level. Early in the search the algorithm is able to reduce the penalty cost considerably and the gap between the water level and the penalty cost is usually very large. Therefore, the algorithm must prevent the cost function to go back near to the water level and for this reason it is important to reduce the gap between the water level and the penalty cost. Later in the search, it becomes more difficult to find the improvement moves. To manage this situation, we float the water level to prevent the algorithm becoming greedy. By floating the water level the algorithm tries to diversify the search by extending its search to a different region of the search space. Therefore, at the early stage of the search this algorithm performs more intensification and less diversification. However, when the search gets stuck in the local optima the algorithm begins to diversify the search by floating the water level (increasing the water level). The main weakness with the linear decay of the water level is that the water level decreases too quick in the later stages of the search. At the beginning, the algorithm seems to produce several successful moves. However when the search is in the middle or approaching the end of the search and the water level converges with the value of the current best solution, most of the neighbourhood solutions are rejected and this situation hinders the algorithm in diversifying the search. Therefore, the algorithm suffers on its own greediness by trapping itself in local optimum. In the conventional great deluge approach, there is no mechanism to help escaping local optima once
the water level and the best solution penalty cost converge. The non-linear great deluge algorithm is described in Algorithm 2.

5.1 Non-linear and Floating Water Level Decay

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 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, $B > f(S^\circ)$ where $f(S^\circ)$ denotes the objective function value of the initial solution S°. The decay, i.e. the speed at which B decreases, is determined

by a linear function in the conventional great deluge algorithm:

$$B = B - \Delta B \quad \text{where } \Delta B \in \Re^{+} \qquad (3.0)$$

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

$$B = B\, x\, (\exp^{-\delta([\min, \max])}) + \beta \qquad (3.1)$$

The various parameters in Eq. (3.1) 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 β = 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. If for a given minimisation problem we knew that the minimum objective value that can be achieved is let's 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 $\exp^{-\delta([\min, \max])}$) is to control the speed of the decay and hence the speed of the search process. A random *min* and *max* are drawn from the uniform distribution interval [min, max] and the *min* and *mix* are integer numbers. By changing the value of these three parameters, the water level goes down faster or slower. Therefore, the lower the values of min and max, the faster the water level goes down, and in consequence, the search quickly achieves an improvement but it also gets stuck in local optima very early. To escape from the local optima, the algorithm needs to increase the water level.

In this paper, the value of the parameters in Eq. (3.1) were determined by experimentation. We tested different combination of parameter values (- δ and *rnd* [*min*, *max*]) and observe the effect of each combination in order to find suitable parameters for given problem. Based on the preliminary experiments, we now then assigned, δ the values of $5x10^{-10}$, $5x10^{-8}$ and $5x10^{-9}$ for small, medium and large instances respectively. As said before, the value of β for all problem instances is β = 0. The values of *min* and *max* in Eq. (3.1) 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 parameter values for small instance is only apply when the penalty cost reach to 10 points. Therefore, it means that from the first iteration the non-linear great deluge algorithm uses the same parameters used for medium instances and changes the parameters when it reaches the penalty cost to 10 points. The use of the non-linear decay rate is shown in algorithm 2 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* $\leq$ 1 in Algorithm 2 (range is the difference between the water level and the penalty cost). We increase the water level *B* by a random number within the interval [$B_{\min}$, $B_{\max}$]. All the parameter values in [$B_{\min}$, $B_{\max}$] were identified by experimentation. 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. The concept of floating water level might be similar to reheating concept in simulated annealing, however in simulated annealing to reheat the temperature, it uses the geometric reheating method. In our method we increase the water level at random. In addition, acceptance in simulated annealing uses probability while great deluge does not employ probability. Full details of this strategy to control the water level decay rate in the modified great deluge are shown in Algorithm 2.

Algorithm 2: 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
    if f(S_best<−20) and Small then
      B = B × (exp^{−δ(rnd[min,max])}) + β
      (Apply small instances parameter)
    else
      B = B × (exp^{−δ(rnd[min,max])}) + β
    end if
  end if
end while
```

The behaviour of the proposed Algorithm 2 can be illustrated in Figure 3. From the outset, the water level is equal to the current penalty cost. When the search progress the current penalty cost is improving as shown by the blue line. The water level decreases quickly to prevent a huge gap between the water level and the current penalty cost. As shown in the figure, when the water level and current penalty cost is about the converge the algorithm then float the water level as shown by the up and down red line.

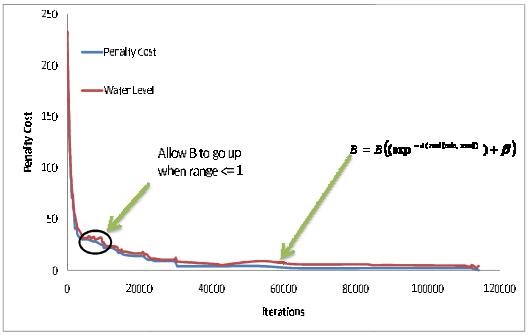

Fig. 3 Non-Linear Great Deluge Behaviour.

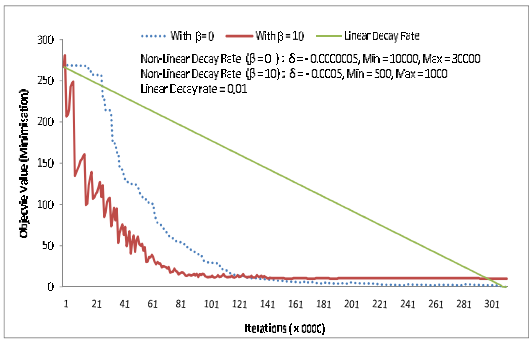

Fig. 4 Comparison between linear (Eq. 1) and non-linear (Eq. 2) decay rates and illustration of the effect of parameters β, δ, *min* and *max* on the shape of the non-linear decay rate.

6. Experiments and Results

In this paper we propose two different stopping conditions for the algorithm. Since non-linear great deluge plays the main role in the evolutionary non-linear great deluge algorithm, we want to investigate which are the adequate criteria for stopping the non-linear great deluge search before it goes to the next process which is update of the pool of solutions (see Figure 1). It should be clear that the non-linear great deluge search promotes intensification in the overall evolutionary method. The use of a population of solutions and the mutation operator promote diversification. Then, by setting the stopping condition for the non-linear great deluge search, we are effectively setting (in a simple manner) the balance between intensification and diversification in the overall evolutionary approach. The first strategy for this balance is to stop the non-linear great deluge after 8000 idle iterations or 30 seconds of computational time, whichever happens first. The second strategy is to stop the non-linear great deluge after three seconds of computational time. The first strategy gives more time to intensification while the second strategy attempts to promote diversification more by stopping intensification sooner. In general, the whole hybrid evolutionary process can be described as follows.

After generating the initial set of solutions, this population then becomes the pool of individual solutions (refer to Figure 1). After the tournament selection of a solution s, this solution is mutated or not as explained above according to the set probability. Then, the non-linear great deluge search takes place over the solution s. The non-linear great deluge search continues until the given stopping condition, one of the two strategies explained above, is satisfied. We implemented three variations of the proposed evolutionary algorithm in order to examine the performance of the algorithm when each of the two stopping conditions is used and also when the mutation operator is re-moved. The three algorithm variants are: Evolutionary Non-linear Great Deluge Without Mutation (ENLGD-M), Evolutionary Non-linear Great Deluge using stopping condition 1 (ENLGD-1) and Evolutionary Non-linear Great Deluge using stopping condition 2 (ENLGD-2). Both ENLGD-1 and ENLGD-2 have the mutation operator incorporated. The aim of examining these algorithm variants is to assess the robustness of the proposed evolutionary algorithm with different settings. By robustness we mean the reliability of the algorithm to produce high-quality of solutions under different settings. Table 2 shows the various parameter settings for the three algorithm variants examined here.

Table 2: Parameter Setting for the Three Variants of the Proposed Evolutionary Non-Linear Great Deluge Algorithm.

Parameter	ENLGD-M	ENLGD-1	ENLGD-2
Mutation	no mutation applied	0.5	0.5
Stopping condition	idle 8000 iterations	Idle 8000 iterations	every 3 seconds
	or 30 seconds	or 30 seconds	of computation time
Replacement	Steady state	Steady state	Steady state
Stopping time for	small (2600 seconds)	small (2600 seconds)	small (2600 seconds)
whole search	Medium (7200 seconds)	medium(7200 seconds)	medium (7200 seconds)
Process	large (10000 seconds)	large (10000 seconds)	large (10000 seconds)

We now evaluate the performance of the proposed evolutionary algorithm (in this experiments, we used the benchmark instances by Socha et, al. [26]). For each

problem size, a fixed computation time ($time_{max}$) in seconds was used as the stopping condition: 1000 for small problems, 7200 for medium problems and 10000 for the large problem. This fixed computation time is for the whole process including the construction of the initial population. We executed the proposed evolutionary algorithm 20 times for each problem instance.

Table 3 shows the experimental results for the three algorithm variants described above, *i.e.* ENLGD-M, ENLGD-1 and ENLGD-2. The Table shows the best and the average results obtained for each method. For each dataset, the best results are indicated in bold. As shown in Table 3, the evolutionary non-great deluge algorithms (ENLGD-1 and ENLGD-2) clearly outperform NLGD. The results also show that both ENLGD-2 and ENGLD-1 produce better results when compared to ENLGD-M. This means that the tailored mutation operator makes a significant impact to the good performance of ENLGD. Besides that, the results also show that ENLGD-2 outperforms ENLGD-1 and ENLGD-M. This means that balancing the intensification and diversification helps the ENLGD approach to better explore the search space rather than run the intensification for longer which makes the local search to converge earlier (as in the ENLGD-1 case). The intensification phase is mainly carried out by NLGD.

Table 3: Comparison of NLGD, ENLGD-M, ENLGD-1 and ENLGD-2 on the Socha et al. UCTTP Instances.

IN	NLGD		ENLGD-M		ENLGD-1		ENLGD-2	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg
S1	3	3.6	**0**	1.55	**0**	0.95	**0**	0.7
S2	4	4.85	**0**	2.2	**0**	1.45	**0**	0.3
S3	6	6.85	1	2.7	**0**	1.3	**0**	1.05
S4	6	6.85	**0**	1.7	**0**	1.35	**0**	1.25
S5	**0**	1.75	**0**	**0**	**0**	**0**	**0**	**0**
M1	140	160.75	144	176.65	125	140	**59**	84.8
M2	130	156	140	162	123	149.1	**51**	93.8
M3	189	212.1	182	204.8	178	199.3	**75**	121.05
M4	112	138.3	135	164.6	116	130.2	**48**	72.8
M5	141	192.6	123	173.15	129	168.6	**65**	110.2
L	876	974.3	970	1026	821	946.1	**703**	819.2

Further investigation was also carried out to inspect the overall performance of ENLGD algorithm. Figures 5, 6 and 7 the performance of the various versions of the algorithm together with NLGD. The x-axis corresponds to the instance type while the y-axis corresponds to the penalty cost. Figure 5 shows the strong performance of ENLGD-2 on medium and large instances, while also obtaining optimal solutions with the same quality as the other algorithms for small instances. In addition, Figure 6 and Figure 7 show details of the results achieved by the proposed algorithms. Both figures show that according to the average results, ENLGD-2 outperformed the other algorithms.

Overall, this experimental evidence shows that by combining some key evolutionary components with single-solution NLGD approach, we have been able to produce a hybrid evolutionary approach that is still quite simple but much more effective than the single-solution stochastic local search in generating best known solutions for a well-known set of difficult university course timetabling instances. It is also evident that the mutation operator makes a significant contribution to the good performance of ENLGD as the results obtained without this operator (ENLGD-M) are considerably worse in medium and large instances. The proposed algorithm seems particularly effective on small and medium problem instances.

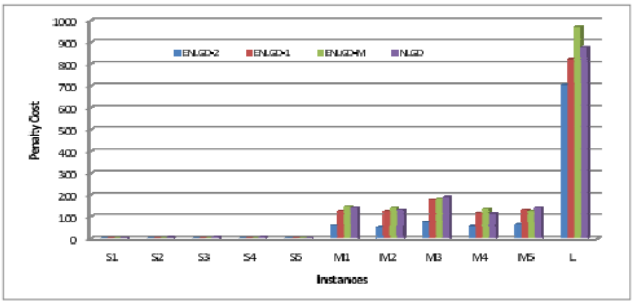

Fig. 5 Best Results Obtained by the Proposed Algorithm

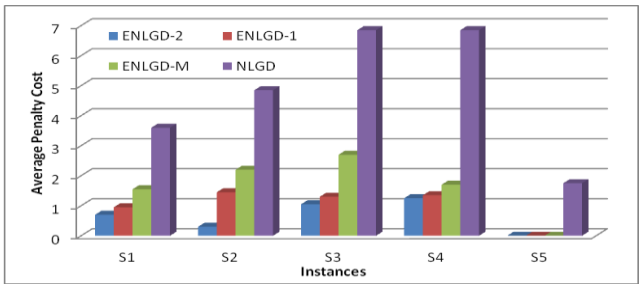

Fig. 6 Average Results Obtained by the Proposed Algorithm Variants on Small Instances.

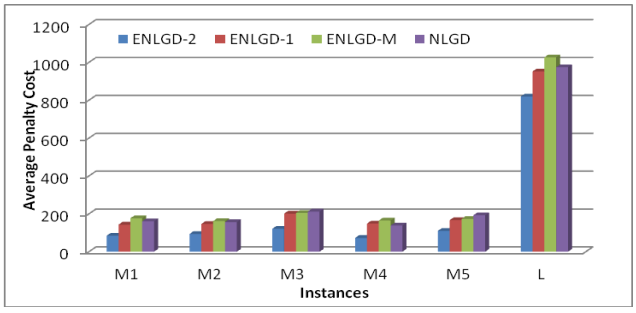

Fig. 7 Average Results Obtained by the Proposed Algorithm Variants on medium and large Instances.

Table 4: Comparison of results obtained by the Evolutionary Non-Linear Great Deluge (ENLGD) proposed in this chapter against the best known results from the literature for the 11 Socha et al. UCTTP instances.

	(ENLGD-2)	(NLGD)	RRLS	MMAS	GALS	RIICN	GBHH	CFHH	VNS-T	HEA	FMHO	EGD
S1	**0**	3	8	1	**0**	**0**	6	1	**0**	**0**	10	**0**
S2	**0**	4	11	3	**0**	**0**	7	2	**0**	**0**	9	**0**
S3	**0**	6	8	1	**0**	**0**	3	**0**	**0**	**0**	7	**0**
S4	**0**	6	7	1	**0**	**0**	3	1	**0**	**0**	17	**0**
S5	**0**	**0**	5	**0**	**0**	**0**	4	**0**	**0**	**0**	7	**0**
M1	**59**	140	199	195	175	242	372	146	317	221	243	80
M2	**51**	130	202.5	184	197	161	419	173	313	147	325	105
M3	**75**	189	77.5%Inf	248	216	265	359	267	357	246	249	139
M4	**48**	112	177.5	164.5	149	181	348	169	247	165	285	88
M5	**65**	141	100%Inf	219.5	190	151	171	303	292	130	132	88
L	703	876	100%Inf	851.5	912	-	1068	80%Inf	-	**529**	1138	730

ENLGD-2 is Evolutionary Non-Linear Great Deluge with stopping strategy 2.
NLGD is Non-Linear Great Deluge [18].
RRLS is the Local Search and Ant System in [27]
MMAS is the MAX-MIN Ant System in [26]
GALS is Genetic algorithm and local search by Abdullah and Turabieh [4].
RIICN is Randomised iterative improvement algorithm by Abdullah et al. [1].
GBHH is Graph-based Hyper-heuristic by Burke et al. [9].
CFHH is the Choice Function Hyper-heuristic in [8]
VSN-T is Variable neighbourhood search with tabu by Abdullah et al. [3].
HEA is Hybrid evolutionary approach by Abdullah et al. [2].
FMHO is fuzzy multiple heuristic ordering [6]
EGD is Extended Great Deluge [19]
S1-S5 represent small problem instances 1 to 5
M1-M5 represent medium problem instances 1 to 5

Table 4 compares the results obtained by the approach proposed with the state of the art approaches in the literature that have been used to solve the course timetabling problem. The term *x%Inf* in Table 4 illustrates a percentage of runs that were unable to achieve feasibility. The figures in bold indicate the best results. Results in the Table indicate that some of the algorithms were unable to produce feasible solutions. However, in contrast, our approach was able to achieve feasible solutions. It can be seen that the proposed hybrid evolutionary approach (ENLGD-2) matches the best known solution quality for all small problem instances. For medium instances, ENLGD-2 was able to achieve better quality solutions when compared against all other methods listed in Table 3. More interestingly ENLGD-2 is able to produce high quality solutions and outperformed the best known results obtained by other algorithms as reported in the literature. Only on the case of the large problem instance, we see that our algorithm does not match the best known result reported by Abdullah at al. [2]. However, our result is still comparable to other results reported in the literature. Overall, this experimental evidence shows that by combining some key evolutionary components and an effective stochastic local search procedure, we have been able to produce a hybrid evolutionary approach that is still quite simple but more effective than the single-solution stochastic local search in generating best know solutions for well-known set of difficult course timetabling problem instances. The proposed algorithm seems particular effective on small and medium problem instances.

6.1 Statistical Analysis

To compare the performance of the different methods proposed, we run some statistical analysis. Even though conclusions can usually be made based on the best and average results obtained by each algorithm, those conclusions and analysis might be premature. Therefore, ANOVA was used to determine whether there is a significant difference in performance among ENLGD-2, ENLGD-1, ENLGD-M and NLGD. Before we proceed to the analysis, it is essential to verify the compatibility of the models with the sample data. There are important hypotheses that need to be verified: normality, independency and homogeneity of the sample data. After running the descriptive analysis, we found that our sample data fulfils the hypothesis requirements. For that reason variance analysis (ANOVA) is considered suitable for the sample data hypothesis ensuring the validity of the experiment. ANOVA is one of the existing statistical models used to test significant differences between means and this tool is very useful to make comparison when dealing with three or more means.

The analysis showed that there are statistically significant differences among the proposed algorithms with the p-value very close to zero as shown in Figure 8.
The p-value stands for probability ranging from zero to one. Therefore, the p-value is used to measure the difference in population means and used as an evidence to reject or accept the null hypothesis. In our case the null hypothesis *H*0 is that there are no significant differences in performance between the algorithms. Therefore, if we reject *H*0 then we accept that there are significant differences in performance
among the algorithms. Tables 7, 8 and 9 clearly show that there are significant differences between the algorithms as described below:

- For small instances, the p-value are less than the confidence level at 0.05 for every pair of algorithms (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M), (ENLGD-2, NLGD), (ENLGD-1, ENLGD-M) and (ENLGD-M, NLGD).
- For medium instances there are significant differences in performance between (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M), (ENLGD-2, NLGD), (ENLGD-1, ENLGD-M) where the p-value are less than the confidence level at 0.05.
- However, there is no significant difference in performance between NLGD and ENLGD-M, where the Post-Hoc analysis shows that the p-value is 0.659 (greater than 0.05).

- Finally for the large instance, there are significant differences in performance between (ENLGD-2, ENLGD-1), (ENLGD-2, ENLGD-M) (ENLGD-2, NLGD) and (ENLGD-1, ENLGD-M) where the p-value for the respective pairs are less than 0.05 significance level. Interestingly, the Post-Hoc test shows that there is no significant difference in performance between (ENLGD-1, NLGD) and (NLGD-M, NLGD) where the p-value are 0.697 and 0.063 respectively, where both p-value is greater than significant level at 0.05.

The Post-Hoc analysis clearly showed that all four algorithms perform differently. However, at this stage we still do now know which algorithm is actually outperforming the others across the eleven instances. Thus, to evaluate this, we plot the mean of each algorithm with Least Significant Difference (LSD) intervals at 95% confidence level for the different algorithms as shown in Figures 8.

Figure 9, Figure 10 and Figure 11 present the means plot of each algorithm, for the specific instances. Figure 8 shows that there are three homogenous groups for small instances (ENGLD-1, ENLGD-2), (ENLGD-M) and (NLGD). The best algorithm is ENGLD-2 followed by ENLGD-1 and ENLGD-M, the worst algorithm is NLGD. In medium instances we also found three homogenous groups as shown in Figure 9 and they are (ENGLD-1), (ENLGD-2) and (ENLGD-M, NLGD). The algorithm that performs well in medium instances is ENGLD-2 followed by ENLGD-1 and two algorithms which perform slightly worst are ENLGD-M and NLGD. Finally, for the large instance, we found that there are three homogenous group (ENGLD-1, NLGD), (ENLGD-2) and ENLGD-M. In the large instance case, we found that ENLGD-2 outperforms the other algorithms and ENLGD-M is the worst. In conclusion, considering the overall performance, ENLGD-2 is the best algorithm followed by ENLG-1, NLGD and the worst algorithm is ENLGD-M (mutation operator removed). to 10. LSD is used to measure the significant differences between group means in ANOVA. From the mean plot, we see that ENLGD-2 outperforms the other algorithms followed by ENLGD-1, NLGD and the worst algorithm is ENLGD-M.

The statistical analysis presented in the paper suggest that each algorithm performs differently across all 11 Socha et al. Instances[26]. This analysis also shows that ENLGD-2 outperforms the three other algorithms across all instances. It is also evident that the mutation operator makes a significant contribution to the good performance of ENLGD-2 as the results obtained by ENLGD-M are considerably worse. Moreover, the strategy applied in ENLGD-2 to balance intensification and diversification proves to be a good strategy as it managed to further improve the solution quality compared to ENLGD-1. As a conclusion, the proposed evolutionary non-linear great deluge approach matches the best known solution quality for almost all small problem instances and improves the best known results for most all medium instances. For large instances, the evolutionary non-linear great deluge algorithm did not match the best known results published in the literature. However, the results are still competitive when compared to the results obtained by other algorithms reported in the literature.

Table 5: Average Penalty Cost of ENLGD-2 and ENLGD-1 Across the 11 Socha et al. Instances.

	ENLGD-2			ENLGD-1		
Run	Small	Medium	Large	Small	Medium	Large
1	0.8	95.6	703	0.20	159	821
2	0.4	85.8	927	1.4	165.4	940
3	0.4	95.4	835	1	167.8	963
4	0.4	93.6	968	1.2	163.6	879
5	0.4	108.6	895	1	165.2	954
6	0.4	99.8	730	1.2	162	952
7	0.2	81.2	782	8.8	146.4	938
8	0.4	91.6	711	1.2	148.2	976
9	0.8	110.4	777	1	147.4	1018
10	1	96.4	838	0.6	144.4	1020
11	0.4	96.6	808	1	171.6	968
12	1	98.4	944	1.6	178	904
13	0.8	91.2	870	1.2	158.8	958
14	1.2	96.4	807	0.4	159.2	876
15	0.4	83.6	849	1.8	165	876
16	1.2	90.6	713	1.6	156	970
17	0.4	117.8	852	1.2	169.6	918
18	0.6	102.2	795	0.6	172.8	1003
19	1.6	106	779	0.6	148.2	1031
20	0.8	89.4	801	0.6	175.2	1072

Table 6: Average Penalty Cost of ENLGD-M and NLGD Across the 11 Socha et al. Instances.

	ENLGD-M			**NLGD**		
Run	Small	Medium	Large	Small	Medium	Large
1	2	186.2	1023	3.8	142.4	966
2	2	176.6	1070	4.8	165	1070
3	1.4	191.6	998	6	165.6	876
4	2	177.6	1142	5.2	162.2	935
5	1.4	205.8	1114	5	165.2	971
6	1	189.8	984	4.6	166.8	942
7	1	184	923	5	165.4	895
8	1.8	179.6	970	5.2	156.8	976
9	2	166.4	1082	5.4	160.4	986
10	1.4	185	1023	5.4	172.8	1005
11	1.8	192.2	1023	3.8	185	966
12	2	159.2	1070	4	171.6	1070
13	2	178.8	998	4.2	177	935
14	2.2	156.4	1142	4.2	181	1024
15	1.6	167.6	984	4	172.4	942
16	2	166.6	923	5	188.4	958

17	1.6	168.6	970	4.2	179.6	978
18	0.8	168.8	1082	5.4	182.6	1005
19	1.4	156.6	1023	5.4	196	1078
20	1.2	166.8	982	5	183.8	907

ANOVA

		Sum of Squares	df	Mean Square	F	Sig.
Small	Between Groups	212.788	3	70.929	322.792	.000
	Within Groups	16.700	76	.220		
	Total	229.488	79			
Medium	Between Groups	82928.598	3	27642.866	212.846	.000
	Within Groups	9870.338	76	129.873		
	Total	92798.936	79			
Large	Between Groups	466403.500	3	155467.833	36.805	.000
	Within Groups	321033.700	76	4224.128		
	Total	787437.200	79			

Fig 8: ANOVA Results.

Table 7: Post Hoc Tests - Small Instances

	ENLGD-2	ENLGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.041	0.000	0.000
ENLGD-1	0..041	-	0.000	0.000
ENLGD-M	0.000	0.000	-	0.000
NLGD	0.000	0.000	0.000	-

Table 8: Post Hoc Tests - Small Instances

	ENLGD-2	ENLGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.000	0.000	0.000
ENLGD-1	0..000	-	0.001	0.019
ENLGD-M	0.000	0.000	-	0.649
NLGD	0.000	0.019	0.649	-

Table 9: Post Hoc Tests - large Instances

	ENLGD-2	ENLGD-1	ENLGD-M	NLGD
ENLGD-2	-	0.000	0.000	0.000
ENLGD-1	0..000	-	0.003	0.697
ENLGD-M	0.000	0.003	-	0.063
NLGD	0.000	0.697	0.063	-

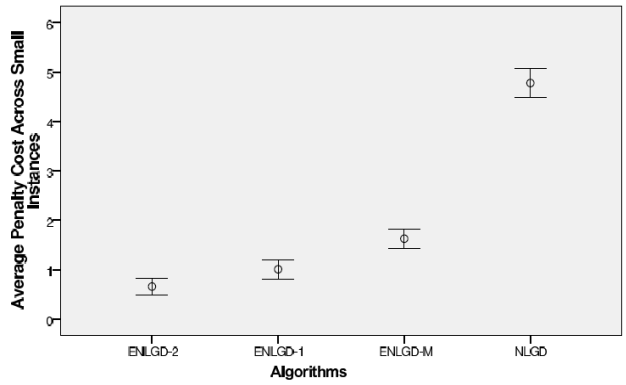

Figure 8: Mean Plot and LSD Intervals (Small Instances).

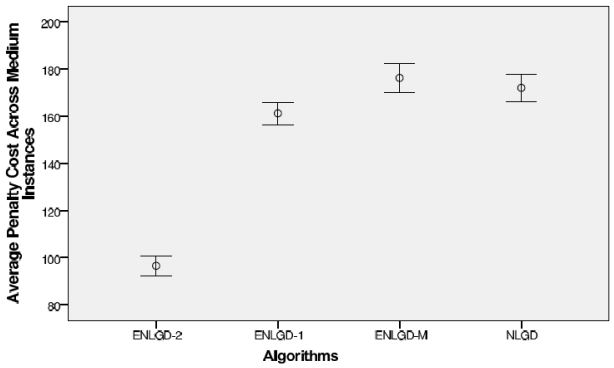

Figure 9: Mean Plot and LSD Intervals (Medium Instances).

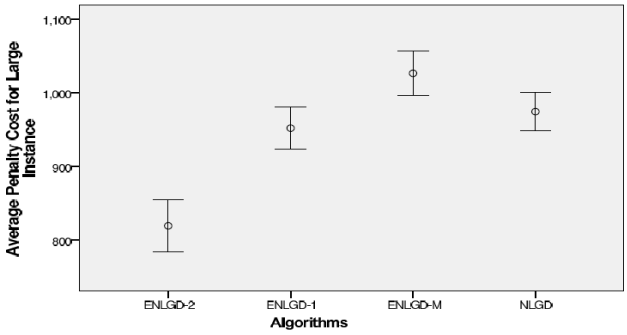

Figure 10: Mean Plot and LSD Intervals (Large Instance).

7. Conclusions

The overall endeavour of this paper was to extend our previous approach, a non-linear great deluge algorithm, towards an evolutionary variant by incorporating some key operators like a population of solutions, tournament selection, a mutation operator and a steady-state replacement strategy. The performances of the various versions of evolutionary non-linear great deluge were compared along with the single-solution NLGD algorithm. Preliminary comparisons illustrate that ENLGD-2 outperforms the results produced by other versions of ENLGD and NLGD algorithms. The results from our experiments also provide evidence that our hybrid evolutionary algorithm is capable of producing best known solutions for a number of the test instances used here. Obtaining the best timetables (with penalty equal to zero) for the medium and large instances is still a challenge. However, when compared to the results obtained by ENLGD-2 to the best know results reported in the literature, obviously, ENLGD-2 outperform all the results of medium instances and produced comparable ones for large instance.

References

[1] S. Abdullah, E. K. Burke and B. McCollum, "A Hybrid Evolutionary Approach to the University Course Timetabling problem". in proceedings of CEC: The IEEE Congress on Evolutionary Computation, 2007, pp. 1764-1768.

[2] S. Abdullah, E.K. Burke and B. McCollum, "Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling", Metaheuristics-Progress in Complex Systems Optimization, 2007, pp. 153-172.

[3] S. Abdullah, E. K. Burke and B. McCollum, " An Investigation of Variable Neighbourhood Search for University Course Timetabling", in The 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, NY, USA, 2005, pp. 413-427.

[4] S. Abdullah, H. Turabieh, " Generating University Course Timetable Using Genetic Algorithms and Local Search", in The Third International Conference on Convergence and Hybrid Information Technology (ICCIT), 2008, pp. 254-260.

[5] D.H. Ackley, A Connectionist Machine for Genetic Hill Climbing, Kluwer Academic Press, Boston, 1987.

[6] H. Asmuni, E.K. Burke and 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.

[7] T. Back, F. Hoofmeister and H. Schwefel, "A survey of Evolution Strategies", in Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 2-9.

[8] E. Burke, G. Kendall and E. Soubeiga, "A Tabu Search Hyperheuristic for Timetabling and Rostering", Journal of Heuristics, 2003, vol. 9, pp. 451-470.

[9] E. Burke, B. McCollum, A. Meisels, S. Petrovic and Q. Rong, "A Graph based Hyper-heuristic for Educational Timetabling Problems", European Journal of Operational Research, 2007, vol. 176, pp. 177-192.

[10] E. Burke, J. Newall and R. Weare, "A Memetic Algorithm for University Exam Timetabling" , in Burke, E. Ross. P.(eds), The Practice Theory of Automated Timetabling: Selected Papers PATAT95, Napier University, Lecture Notes in Computer Science, Springer, New York, 1996, vol. 1153, pp. 241-25.

[11] E. K. Burke and J. P. Newall, "A Multi-Stage Evolutionary Algorithm for the Timetable Problem", IEEE Transactions On Evolutionary Computation, 1999, vol. 13(1), pp. 63-74.

[12] T. Cooper and H. Kingston, "The Complexity of Timetable Construction Problems", in Selected paper from the 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT'95), LNCS, Springer, 1996, vol. 1153, pp. 283-295.

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

[14] A. Eiben and J. E Smith, Introduction to Evolutionary Computing, Natural Computing Series. Springer first edition, 2003.

[15] M. Gorges-Schleuter, "ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy", in proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann (San Mateo), 1989, pp. 422-427.

[16] G. Gutin and D. Karapetyan, " A Memetic Algorithm for the Generalized Travelling Salesman Problem", Natural Computing, 2010, vol. 9(1), pp. 47-60.

[17] D. Haibin and X. Yu, "Hybrid ant Colony Optimization Using Memetic Algorithm for Travelling Salesman Problem", in proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), 2007, pp. 92-95.

[18] D. Landa-Silva and J. Henry Obit, "Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems", in proceedings of the IEEE Conference on Intelligent Systems, IEEE Press, 2008, pp. 8.11-8.18.

[19] P. McMullan, "An Extended Implementation of the Great Deluge Algorithm for Course Timetabling", Springer-Verlag Berlin Heidelberg, Part I, LNCS, 2007, vol. 4487, pp. 538-545.

[20] P. Moscato and M.G. Norman, "A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems", in proceedings of the International Conference on Parallel Computing and Transporter Applications, 1992, vol. 28, pp. 177-186.

[21] E. Ozcan and A. Alkan, "A Memetic Algorithm for Solving a Timetabling Problem: An Incremental Strategy", in P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, in proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA): Paris, France, 2007, pp. 394-401.

[22] B. Paechter , A.P. Cumming, M. Norman and H. Luchian, " Extensions to a Memetic Timetabling System", The Practice and Theory of Automated Timetabling I: Selected Papers from 1st International Conference on the Practice and Theory of Automated Timetabling (PATAT I), Springer-Verlag: Edinburgh, UK, 1996, vol. 1153, pp. 251-265.

[23] N. J. Radclife and P. D. Surry, "Formal Memetic Algorithms", in Evolutionary Computing: AISB Workshop Workshop, Ed: T.C. Fogarty, Springer-Verlag LNCS, 1994, vol. 865, pp. 1-16.

[24] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L.C. Gambardella, J. Knowles, M. Manfrin. M. Mastrolilli, B. Paechter, B. Paquete and T. Stutzle, "A Comparison of the Performance of Different Metaheuristics on the Timetabling Problems", selected papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT), 2003, vol. 2740, pp. 330-352.

[25] K. Sastry, D. Goldberg and G.Kendall, "Genetic Algorithms", in E. Burke and G. Kendall, editors, Search Methodology, Springer, 2005, pp. 97-125.

[26] K. Socha, K.. Knowles and J. Samples, M, "A Max-Min Ant System for the University Course Timetabling Problems. in Ant Algorithms", in proceedings of The 3rd

International Workshop (ANTS), 2002, vol.2463, pp. 1-13.

[27] K. Socha, M. Sampels and M. Manfrin, "Ant Algorithms for the University Course Timetabling Problems with Regard to the State-of-the-Art", in Applications of Evolutionary Computing, proceedings of the EvoWorkshops, Springer, LNCS, 2003, vol. 2611, pp. 334-345.

First Author Dr. Joe Henry Obit is a Senior Lecturer in the School of Informatics Science in E-Commerce Department at the Universiti Malaysia Sabah, Labuan International Campus. His main research interest lies at the interface of Operational Research and Computer Science. In particular, the exploration and development of innovative Operational Research, Artificial Intelligence, and Distributed Artificial Intelligence models and methodologies for automatically producing high quality solutions to a wide range of real world combinatorial optimisation and scheduling problems. Dr. Joe Obtained his Bachelor Degree in Finance at Universiti Kebangsaan Malaysia in 1999, an MSc Information Technology from Universiti Putra Malaysia in 2001and a PhD in Computer Science from the School of Computer Science at the University of Nottingham. His PhD thesis is Developing a Novel Meta-heuristic, Hyper-heuristic and Cooperative Search, and it was under the supervision of Associate Professor Dr. Dario Landa-Silva.

Second Author Dr. Djamila Ouelhadj is a Senior Lecturer in Operational Research Department of Mathematics at the University of Portsmouth. Her main research interest lies at the interface of Operational Research and Computer Science. In particular, the exploration and development of innovative Operational Research, Artificial Intelligence, and Distributed Artificial Intelligence models and methodologies for automatically producing high quality solutions to a wide range of real world combinatorial optimisation and scheduling problems. Dr. Djamila Ouelhadj obtained her PhD in Computer Science from the School of Computer Science at the University of Nottingham in 2002.

Third Author Dr. Dario Landa-Silva is an Associate Professor in Computer Science for the School of Computer Science at the University of Nottingham. He is a member of the Automated Scheduling, Optimisation and Planning (ASAP) research group. He is also a member of the Institute for Operations Research and Management Sciences (INFORMS), the Operational Research Society (ORS) and a member of the editorial board for the Neural Computing and Application Journal. Dario Landa-Silva obtained a Technical Professional Qualification in Electro-mechanics from the CBTis 13 (Mexico) in 1987, a BEng in Industrial Electronic Engineering from Instituto Tecnologico de Veracruz (Mexico) in 1991, an MSc in Engineering-Computer Science from DEPI in the Instituto Tecnologico de Chihuahua in 1997 and a PhD in Computer Science from the School of Computer Science at the University of Nottingham in 2003.

Fourth Author Dr. Rayner Alfred is a Senior Lecturer in Software Engineering Department for the School of Engineering and Information Technology at Universiti Malaysia Sabah. His main research interest lies at the Machine Learning in Knowledge Discovery. Dr. Rayner Alfred obtained his BSc in Computer Science at Polytechnic University of Brooklyn, New York, United States of America in 1994, an MSc in Computer Science from Western Michigan University, Michigan, United States of America in 1997 and a PhD in Computer Science from the School of Computer Science at the University of York, UK in 2008.