Population based Local Search for University Course Timetabling Problems

Anmar Abuhamdah[1], Masri Ayob[1], Graham Kendall[2, 3] and Nasser R. Sabar[1]

[1] Data Mining and Optimisation Research Group (DMO), Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, 43600 UKM, Bangi Selangor, Malaysia
anmar@ftsm.ukm.my, masri@ftsm.ukm.my, naserdolayme@yahoo.com
[2]ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK.
Graham.Kendall@nottingham.ac.uk
[3]University of Nottingham, Malaysia Campus,
Graham.Kendall@nottingham.edu.my

Abstract. Population based algorithms are better in exploring wider areas of the search space than local search algorithms (i.e. a single based heuristic). However, the limitation of many population based algorithms is in exploiting the search space. Thus, we proposed a population based Local Search (PB-LS) heuristic that embedded with a local search algorithm (as a mechanism to exploit the search space). PB-LS employs two operators. The first is applied to a single solution to determine the force between the incumbent solution and the trial current solution (i.e. a single direction force). Whilst, the second operator is applied to all solutions, to determine the force in all directions. The progress of the search is governed by the forces either in a single direction or in all directions. The aim of our work is to produce an effective algorithm that has more capability in diversification and intensification compared to common local searchers and population based algorithms. We use university course timetabling Socha benchmark datasets as a test domain. In order to evaluate the effectiveness of PB-LS, we perform a comparison between the performances of PB-LS with other approaches in the literatures. Results showed that PB-LS is able to produce statistically significant higher quality solutions that outperforms many other approaches (with regards to Socha dataset).

Keywords: Course Timetabling Problem; Metaheuristics; Population Based Algorithm; Hybrid Methods; Gravitational Emulation.

1 Introduction

University course timetabling problems involve assigning a set of courses (events), teachers and students to a fixed number of timeslots and rooms subject to a variety of constraints [28]. Constraints in a timetabling problem can be classified as *hard* and *soft* [28]. The goal, when solving timetabling problems, is to satisfy all hard constraints and attempt to accommodate the soft constraints as much as possible (in order to produce a high-quality timetable). All hard constraints must be satisfied in order to obtain a feasible timetable, whilst soft constraints can be violated if necessary, but each is penalized. The smaller the overall penalty value, the better the quality of the timetable. University course timetabling problems have been classified as an NP-hard problem; therefore it is difficult (in general) to find an optimal solution (for larger size instances) in a reasonable time [30]. Finding good quality solutions to these problems depends on the methodology used and the problem representation employed during the search [30].

In recent years, various approaches have been applied to university course timetabling problems. These approaches include great deluge [14], simulated annealing [20], tabu search [18], randomized descent [31] and honey bee mating algorithm [29]. The disadvantage of the local search methodologies is that they are incapable of escaping from local optima [31].

Therefore, in [8], we proposed an adaptive randomized descent algorithm (ARDA) that employs an adaptive criterion to escape from local optima. We then hybridized MPCA (multi-neighbourhood particle collision algorithm [7]) and ARDA to complement their strength and weaknesses [9]. However, MPCA-ARDA has no diversification strategy, which motivated us to use the concept of a population based algorithm due to their ability to explore wider areas of the search space than local search algorithms [13]. Generally, the limitation of many population based algorithms is in exploiting the search space [13], as most of them are good at exploration rather than exploitation. In this work, we propose a population based heuristic (Population Based Local Search, PB-LS) to overcome the limitations of a population based

algorithm by increasing the intensification mechanism. This algorithm is motivated by the idea of Gravitational Emulation Local Search (GELS), which was proposed and developed by Webster [38].

The proposed PB-LS is also motivated by the ideas of memetic algorithm and hyper-heuristics. In the scientific literature, population based algorithm are usually hybridized with local search algorithm (i.e. a single based heuristic) in order to exploit their complementary character. This is known as a memetic algorithm in which genetic algorithms are hybridized with local search algorithm. Hyper-heuristics [] [] [] is an emergent research trend that aims to control the selection of which heuristic to be applied at each decision point. The proposed PB-LS share some features with both memetic algorithm and hyper-heuristics as follows: i) PB-LS operate on single and population of solutions. Thus, it could be classified as a memetic algorithm. However, the main difference between PB-LS and the memetic algorithm is that PB-LS did not apply the local search neither at the initial population nor every iteration. This feature can control solution diversity because in most existing memetic algorithm local search is applied at every iteration which might lead to fast convergence. ii) PB-LS utilize the round-robin strategy to select the suitable neighborhood structure which can help the search process in exploring different area because different neighborhood structure will generate different search space.

The aim of our work is to investigate that if PB-LS with MPCA-ARDA has more capability in intensifying and diversifying the search, then it will produce better quality solution compared to other local search (single based). In order to evaluate the effectiveness of PB-LS we compare its performance with MPCA, ARDA, MPCA-ARDA as well as other approaches. We use Socha's university course timetabling datasets, in line with many other researchers [33-34].

2 Problem Description

The eleven standard benchmark instances that were introduced by Socha et al. [33] are used in this work, which seek to optimise the students' satisfaction for the university course timetabling problem. The problem consists of:

- A set of Rooms *R* in which events can take place.
- A set of Events (courses) *E* to be scheduled in 45 timeslots (5 days of 9 hours each with one hour for each timeslot).
- A set of Features *F* characterizing the rooms.
- A set of Students *S* who attend the events.

These datasets are categorized into three groups, small (*small 1, small 2, small 3, small 4* and *small 5*), medium (*medium 1, medium 2, medium 3, medium 4* and *medium 5*) and large (*large*) (see Table 1 and [33] for a detailed description). Table 1 also shows, the number of students, events, room and features as well as the conflict density (*CD*) for each dataset (representing the complexity), approximation number of students enrolled in each event (Students/ Events), and the approximation number of available rooms for each event (Rooms/ Events) which are calculated as in [17].

Table 1. The course timetabling datasets [33]

Dataset	# Students	# Events	# Rooms	# Features	CD	Students/ Events	Rooms/ Events
Small 1	80	100	5	5	10.96	4.98	0.82
Small 2	80	100	5	5	13.92	5.36	0.79
Small 3	80	100	5	5	9.71	4.65	1.00
Small 4	80	100	5	5	7.16	3.45	1.39
Small 5	80	100	5	5	15.10	5.99	1.17
Medium 1	200	400	10	5	37.38	8.85	2.23
Medium 2	200	400	10	5	37.66	8.84	1.91
Medium 3	200	400	10	5	40.44	8.85	1.91
Medium 4	200	400	10	5	37.50	8.81	1.88
Medium 5	200	400	10	5	28.27	8.66	1.37
large	400	400	10	10	45.57	8.92	0.76

These datasets have three hard constraints (Hc1, Hc2 and Hc3) and three soft constraints (Sc1, Sc2 and Sc3), as follows:

(a) Hard constraints

Hc1: No student attends more than one event at the same time.
Hc2: The room has to be large enough for all the attending students and has all the features required by the event.
Hc3: Only one event takes place in each room in any timeslot.

(b) Soft constraints

Sc1: A student should not have a class in the last timeslot of the day.
Sc2: A student should not have more than two classes consecutively.
Sc3: A student should not have a single class on a day.

The quality of a timetable is measured based on the number of soft constraint violations (penalty cost). Each violation of a soft constraint will be penalized '1' for each student who is involved in this situation [33]. All hard constraints must be satisfied since we only deal with feasible solutions, which is usually the case for the majority of research in this domain.

3 Methodology

We propose a population based local search algorithm (PB-LS) for university course timetabling problems. PB-LS starts with an initial solution and iteratively explores its neighbour solutions, seeking a better one. The neighbour solution is obtained by modifying the current solution using one or more neighbourhood structures.

3.1 Initial Solution

In this work, we use a constructive heuristic that was proposed in [25] to generate initial solutions for the course timetabling problems. The constructive heuristic has three phases: largest degree heuristic, neighborhood search and tabu search. The constructive heuristic starts with an empty timetable and successively invokes three phases to generate a feasible timetable. In the first phase, all unscheduled courses are sorted based on the number of students they have in conflict with other courses. Then, the one that has the highest number of conflicts compared to the other courses is selected first. The selected course is then assigned to a feasible timeslot-room which is selected randomly. If there is no feasible room for this course, it will be assigned to any room. If all courses have been scheduled to feasible timeslot-rooms, we ignore phases 2 and 3. Otherwise, phases 2 and 3 are invoked to achieve feasibility.

Phase 2, employs a simple decent algorithm to reduce the number of hard constraint violations. The neighbourhood solution is generated by either moving one course from its current timeslot-room into another timeslot-room, selected randomly, or it randomly selects two courses and swaps their timeslots and rooms. In both cases, the new solution is accepted if the move does not violate any hard constraints and the quality of the generated timetable in terms of hard constraints violation is better than the previous solution. Phase 2 is terminated after ten non-improving iterations. If the solution is feasible, we ignore phase 3, otherwise, phase 3 is invoked.

Phase 3, employs a tabu search algorithm that explores neighbouring solutions in a similar way to phase 2, but it also maintains a tabu list to prevent certain moves being made for a certain number of iterations. The size of the tabu list is calculated by $tl = rand(10)+\delta * nc$, where $rand(10)$ is a random number between 0 and 10, nc is the number of events that violate the hard constraints and δ is a constant which is set to 0.6 [25]. This phase will stop after 1000 non-improving iterations. If the generated solution is infeasible, re-call the constructive heuristic to generate a new solution from scratch until a feasible solution is found.

3.2 Neighbourhood Structures

The proposed algorithm starts with an initial solution and iteratively improves it by generating a neighbour solution using a set of neighbourhood structures. We use eight neighbourhood structures (NS1-NS8) which have been widely used in university course timetabling problem. These neighbourhood structures are classified into two groups: i) common neighbourhoods (NS1-NS5 as in [1]) and ii) shaking neighbourhoods (NS6 and NS8 as in [1], and NS7 as in [35]). Five of them (NS1-NS5) are the same neighbourhoods utilised in ARDA [8]. In addition, we use three other neighbourhood structures (NS6-NS8) to diversify the search (shake the timetable). The neighbourhood structures are:

NS1: Randomly select two courses and swap their rooms and timeslots if feasible. Otherwise swap their timeslots only (if feasible) [1].

NS2: Randomly select two timeslots and swap all the courses in one timeslot with all the courses in the other timeslot [1].

NS3: Randomly select four courses and swap timeslots and rooms of the first and second courses with the third and fourth courses (if feasible).

NS4: Randomly select a course, timeslot and room, and then move the course (reassign) to the new timeslot and room (if feasible) [1].

NS5: Randomly choose a course from the top 15% of the list (ordered based on the penalty value in descending order) and randomly assign to other timeslot. Abdullah [1] used this neighbourhood (NS5) but with 10% selection. The reason behind for using 15% is to widen the search space.

NS6: Choose 15% of the courses in the list and randomly assign them to other feasible timeslots. We use the same idea as NS5 but the difference is that, in NS5 we assign one course only whereas in NS6 we assign 15% of the courses to diverse the search (shake the timetable).

NS7: Randomly select two timeslots (*t1* and *t2*) based on the largest enrolled (conflict) events. Select the most conflicting event in *t1* and then apply a kempe chain move [35] to move the select even from *t1* to *t2*. The idea of the kempe chain neighbourhood is to move a chain of courses within the timetable while maintaining the feasibility by swapping conflicting courses between the selected timeslots until achieving feasibility.

NS8: Rotate two timeslots: Randomly select two timeslots (t_i and t_j), where i and j are the timeslot index, $i < j$ and the timeslots are ordered $t_0, t_1 \ldots t_n$ where n is the total number of timeslots. Take all the courses in t_i and allocate them to t_j. Now take the courses that were in t_j and allocate them to t_{j-1}. Then allocate those that were in t_{j-1} to t_{j-2} and so on until those courses that were in t_{i+1} are allocated to t_i. This neighbourhood structure was introduced in [1].

4 Population based Local Search Algorithm

This work is motivated by the strength of population based algorithms to explore wider areas of the search space than when using a local search algorithm [13]. However, some limitations of population based algorithms are [22]:

- Ineffective exploitation of the solution space (intensification process), in which there is no significant solution improvement.
- Solution combination methods to generate new solutions (e.g. crossover in genetic algorithm) and mutation operator usually rely on randomization and require a repair mechanism to use them effectively in constrained problems.
- The process of updating the population is usually performed randomly.
- Some algorithms do not have a memory as guidance for the search (e.g. genetic algorithm and memetic algorithm).

The idea of the PB-LS is to enhance the performance of population based algorithms by:

a) Increasing the ability of the intensification process by using population of solutions in local search.
b) Using a memory to guide PB-LS to search around more promising region of the search space which is also contains some elite solutions and a useful information about them
c) Systematically update the population.

This heuristic is motivated by the idea of gravitational emulation local search (GELS). The GELS algorithm is based on the natural principles of gravitational attraction [38]. The reason for using gravity is to cause objects to be pulled towards each other. The more massive an object, the more gravitational "pull" it exerts on other objects. Also the closer two objects to each other, the stronger the gravitational forces are between them. This means that a given object will be more strongly attracted to larger and closer objects.

Previously the GELS algorithm was known as gravitational local search algorithm (GLSA) and consisted of two versions: one that was based on the gravitational attraction between two objects, and allowed navigation only to adjacent positions within the solution space; the other was based on gravitational field attractions among objects and allowed navigation to non-adjacent positions [38].

Both versions of GELS use the same gravitational force equation (see eq. 1) but in slightly different ways [38]. The first version applies the equation to a single solution (vector) within the local search neighbourhood to determine the gravitational force between that solution and the current solutions. Whilst, the second version applies the equation to all solutions (vectors) within the neighbourhood and calculates the gravitational force between each of them and the current solution.

GELS simulate Newton's equation of gravitational force between two objects [38] (eq. 1).

$$F= G (CU - CA) / R^2 \quad (1)$$

Where F is the Force value representing the value of the gravitational force between two objects (i.e. to enhance the difference between the quality of solutions), G = 6.672, CU = objective function value of the incumbent solution, CA = objective function value of the trial solution and R is the value of the parameter radius, representing the middle point in the distance between two objects (which requires parameter tuning).

Therefore, this work proposed the PB-LS heuristic that eliminated parameters radius and G to overcome the parameter tuning issue. In PB-LS, we propose a new equation to calculate the force value (eq. 2) to overcome the issue of setting the *radius* parameter in GELS.

$$F= CU - CA \quad (2)$$

Where F is the Force value (assuming a minimization problem), CU = objective function value of the incumbent solution and CA = objective function value of the trial solution.

Equation 2 differs from the equation 1, as GELS obtains a real force value, whilst, equation 2 does not employ any static parameters. Indeed there is no relation between G (i.e. 6.672) and university course timetabling problem or the problems that GELS has been applied to. We also omitted parameter R (radius) in the gravitational force equation since in PBLS the gravitational force is increased or decreased solely based on the quality of solution.

Fig. 1 shows the pseudo code for PB-LS approach. Let S_o be a given initial solution, S_{best} be the best obtained solution, $f(S_{best})$ be the quality (penalty cost) of S_{best}; $f(S_o)$ be the quality of S_o; *N.iters* be the number of maximum iterations; *reset.iter* be the number of non-improving iterations required to reset the directions and update solutions; *force* be the force value, vv_i be the i^{th} velocity vector and N be the number of shaking neighbourhoods. In addition, we will also need to initialize the required parameters for the local search method (in our case, we use MPCA-ARDA, see Table 2).

PB-LS start from zero velocity (i.e. direction value is zero) and it is updated during the search process. In PB-LS, we only apply the first method. The procession of the search through the search space is governed by the forces in a single direction as determined by equation 2. We use MPCA-ARDA (as a local search) to intensify the search since we have already proposed MPCA-ARDA in (Abuhamdah and Ayob [9]). Therefore, we can compare the result of MPCA-ARDA against the PB-LS with MPCA-ARDA in order to demonstrate the effectiveness of our proposed PB-LS approach.

In the initialization phase, we initialize all the parameters (see Table 2), generate initial velocity vectors (by applying shaking neighbourhoods; NS6, NS7 and NS8 on S_o). For our first iteration vv_1, vv_2 and vv_3 are equal to the solutions generated by NS6, NS7 and NS8 accordingly. Initially, the direction for all velocity vectors is reset to 0 (see example in Fig. 2-a).

In the improvement phase (Step 2 in Fig. 1), at each iteration, we rearrange the solutions in vv in descending order based on their direction values. Higher direction value indicates better potential for improving the solution rather than other solutions in vv. If all direction values are different (Step 2-1 in Fig.1), we choose the solution that has the largest direction value, vv_1 (e.g. see Fig. 2-b) and set $S_0=vv_1$. The local search (in our work, we use MPCA-ARDA) is applied on S_0 to produce S_0* until stopping condition is met. In MPCA-ARDA, we use common neighbourhood structures (NS1-NS5). The force value for the solution returned by the local search is calculated using equation (2) with $CU=S_o$ and $CA=S_o$*; and is added (positive or negative) to the direction of vv_1.

The S_{best} will be updated with S_0* if $f(S_0$*$)$ is better than $f(S_{best})$. If S_0* has better quality than S_0, we replace vv_1 with S_0^*. Otherwise, we will increase the unimproved counter of the selected solution ($UnImprove_1$) by one. If $UnImprove_1$ is equal to the predetermined successive unimproved iterations, *reset.iter* (i.e. 10 in this work), then we reset the direction of vv_k to zero and replace vv_1 with the best neighbour generated from S_{best} by randomly generating some neighbours (i.e. 5 in this work) from shaking

neighbourhoods. Otherwise, we proceed with the next iteration. This mechanism attempts to escape from a local optimum and to diversify the search.

```
Procedure PB-LS
Step 1: Initialization Phase
  Given an initial candidate solution So with f(So) as the quality of So;
  Set Sbest = So , f(Sbest)= f(So), N = number of shaking neighbourhood;
  Generate N neighbours of Sbest (one neighbour from each shaking neighbourhood);
  Assign each generated neighbour to the velocity vector, vv;
  Set all the directions and un-improve counters (UnImprove) for each vector in vv = 0;
  Set N.iters; //stopping condition (e.g. see Table 2);
  Set reset.iter; // the number of iterations to reset the direction & update solutions (e.g. see
                  Table 2);
 Initialize the required parameters for the Local Search Method (e.g. see Table 2);
   //depend on what local search is used
Step 2: Improvement (Iterative) Phase
Iterations =1;
repeat
     Arrange the vector in vv in descending order based on their direction value;
     2.1 If the direction is clear // No duplication in the direction values
           Set So = vv1; // Select the best direction
           Set k=1;
       a) Apply Local Search on So to produce So*; //any local search can be applied
           Calculate the force value for the vvk using equation 2; //positive or negative force
                with So as a current solution and So*as a candidate solution.
           Update the direction value by adding force value to the direction of the vvk;
           If f(So*) is better than f(Sbest), then Sbest= So*;
           If f(So*) is better than f(S0), then set vvk = So*;
           Else
                Increase the UnImprovek by one;
                If UnImprovek == reset.iter;
                     Set the direction of vvk = 0;
                     UnImprovek=0;
                     Randomly generate N neighbours of Sbest (one neighbour from each
                     shaking neighbourhood) and set vvk = Sbest* (the best neighbour of
                     Sbest);
                End If
           End Else;
        End if
     2.2 Else ; // Improve all solutions that has similar direction value
           While (direction not clear) //some vv have similar direction value
                Select the vv* (the vv that have similar direction values with other vv), set
                k=index of vv* , set So = vv* and apply Step 2.1a);
           End while
        End Else;
   iterations=iterations+1;
until iterations> N.iters (termination condition is met)
Step 3: Termination phase (N.iters termination condition is met)
     Return the best found solution Sbest
```

Fig. 1. Pseudo code for PB-LS approach to solve university course timetabling problem

If some direction values are the same (*Step2.2 in Fig. 1*), e.g. vv_2 and vv_3 in Fig. 2-c, we then perform step 2.1a) in Fig.1 on the appropriate *vv* (e.g. vv_2 and vv_3 in Fig. 2-c), to differentiate that directions for all solutions that have similar direction value. This attempts to maintain a set of diverse solutions.

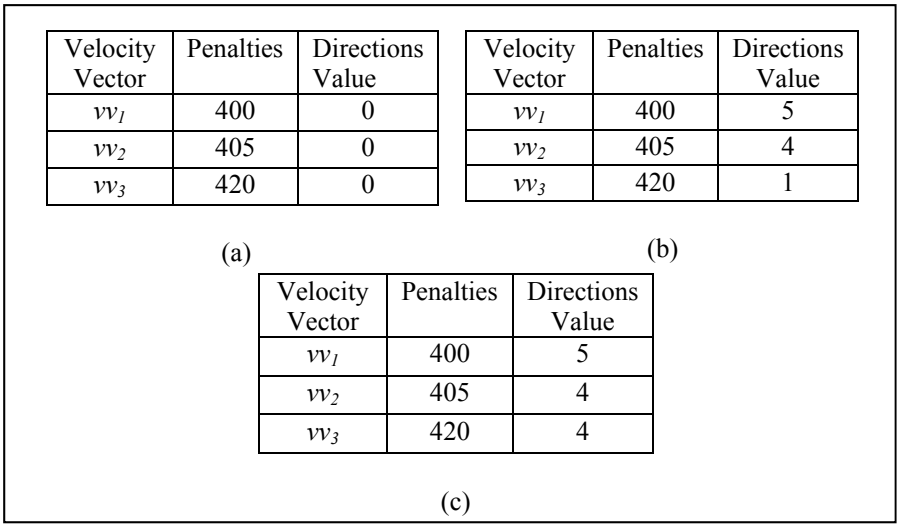

Fig. 2. Example on initialising velocity vectors in PB-LS approach

5 Experimental Results and Discussion

In this work, we have run our algorithm 20 times across 11 instances that were introduced by Socha et al. [33]. The algorithm is run on PC with an Intel dual core 1.8 MHz, 1GB RAM. PB-LS parameters are shown in Table 2. For the small datasets, PB-LS obtain results within 2 to 10 minutes. Whilst for the medium and large datasets, PB-LS took between 10 to 13 hours to achieve the results.

Table 2. Parameters settings used in our PB-LS

Parameter	Value
N.iters	Termination condition (Number of Iterations) = 500,000.
reset.iter	The number of iterations to reset the directions and update the solutions= 10
N	The number of shaking neighbourhood structures = 3 (i.e. NS6, NS7 and NS8)
Local Search	MPCA-ARDA (number of iterations=10)

Table 2 shows that PB-LS employ four parameters as follows: the first parameter (*N.iters*) is determined based on the literature [37]. Whereas, the second parameter (*reset.iter*) is determined based on preliminary experiments. In this experiment, we performed 5 runs for values of 5, 10, 15 and 20 and found that the *reset.iter*=10 produces the best results. The third parameter (*N*) is determined as three based on the number of shaking neighbourhoods (in our case three neighbourhoods used; NS6, NS7 and NS8). The fourth parameter is the selection of local search method.

In order to investigate the performance differences between PB-LS with MPCA and MPCA-ARDA, a Wilcoxon test is carried out with 95% confidence level. The null hypothesis assumes that there is no difference between the compared methods. The *p*-value less than 0.05 mean that, there is a significant difference between these methods. Table 3 shows the comparison between MPCA-ARDA and PB-LS. It illustrates the best score (*fmin*), the average score (*favg*), the standard deviation (σ) for PB-LS and MPCA-ARDA algorithms as well as the *p*-value of PB-LS against MPCA-ARDA (abbreviated as PB-LS vs. MPCA-ARDA).

Table 3 shows that for small instances (Small 1 to Small 5), the performance of both algorithms (PB-LS and MPCA-ARDA) are the same (*p*-value greater than 0.05). This is because small instances are easy to solve and both algorithms manage to solve them effectively. However, PB-LS with MPCA-ARDA outperformed MPCA-ARDA in all medium and large datasets (*p*-value is less than 0.05).

Fig. 3 shows the box and whisker plot of the PB-LS with MPCA-ARDA. It shows that in most cases, PB-LS is capable of producing good quality solutions in all datasets (the medians are close to the best solutions), except in small 3, small 4 and medium 2 datasets, the median is close to the worst. For

example, in small 2, more than 75% of runs obtained solutions that are closed to the minimum penalty value.

In general the results in Table 3 shows that, PB-LS outperformed MPCA-ARDA across all instances (with regard to *fmin* and *favg*). This demonstrates that the use of a population of solution helps the algorithm in diversifying the search space to obtain better quality solutions.

Table 3. Statistical analysis of the results obtained by PB-LS with MPCA-ARDA algorithm and MPCA-ARDA algorithm out of 20 runs.

Data Set	***fmin***		***favg***		***Std. Dev.(σ)***		**PB-LS vs. MPCA-ARDA**
	PB-LS	***MPCA-ARDA***	***PB-LS***	***MPCA-ARDA***	***PB-LS***	***MPCA-ARDA***	***p-value***
Small 1	0	0	0.65	1.00	0.75	0.86	0.071
Small 2	0	0	0.55	1.00	0.83	0.79	**0.007**
Small 3	0	0	0.95	1.15	0.89	0.81	0.392
Small 4	0	0	0.75	0.90	0.72	0.79	0.414
Small 5	0	0	0.60	0.95	0.68	0.83	0.008
Medium 1	41	64	52.75	75.35	7.55	6.99	**0.000**
Medium 2	39	65	54.80	78.05	9.01	8.04	**0.000**
Medium 3	60	91	80.85	106.00	14.73	8.65	**0.000**
Medium 4	39	66	49.50	79.35	7.74	8.32	**0.000**
Medium 5	55	89	65.05	104.05	7.54	9.99	**0.000**
Large	463	576	483.20	593.50	16.69	11.89	**0.000**

Note: Bold in p-value indicate that PB-LS is significantly better than MPCA-ARDA.
The value for fmin and favg are the minimum and average penalty cost, respectively.

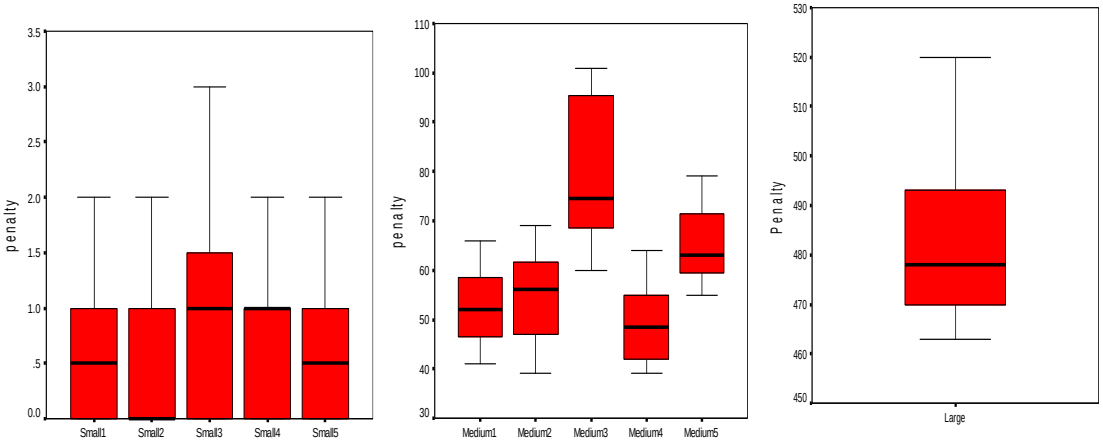

Fig. 3. Box and whisker plot of results obtained by PB-LS (for 20 runs) for all datasets

Table 4 shows the comparison between our PB-LS and other metaheuristic searches that were tested on the Socha benchmark datasets and we also report the rank of our algorithm compared to the others. The best results are presented in bold.

Results in Table 4 shows that of our algorithm outperforms other methods on medium 1, medium 2, and medium 3 instances. The best result of medium 4 is obtained by Turabieh and Abdullah [36] whilst, the best results of medium 5 and large instance is obtained by Turabieh et al. [37]. If we consider an individual comparison with these two methods, our algorithm outperformed Turabieh and Abdullah [36] on 5 instances and ties on the small instances (obtained the same results for all small instances) and outperformed Turabieh et al. [37] on 3 instances and also ties on the small instances (obtained same results for all small instances with regard to the best obtained soltuions). Also, the percentage deviation ($\Delta(\%)$) of our algorithm for medium 4, medium 5 and large are 0.21, 0.122 and 0.13, which are very close to the best known results for these instances (third column in Table 4).

Table 4. Comparison between our PB-LS with MPCA-ARDA and other approaches in the literature

Data Set	**Rank**	**Δ(%)**	**PB-LS with MPCA-ARDA**	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M17	M18	M19	M20	M21	M22	M23	M24
Small1	Same	0	**0**	**0**	**0**	**0**	**0**	5	10	2	**0**	6	3	1	1	8	5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Small2	Same	0	**0**	**0**	**0**	**0**	**0**	5	9	4	**0**	7	4	3	2	11	3	**0**	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Small3	Same	0	**0**	**0**	**0**	**0**	**0**	3	7	2	**0**	3	6	1	**0**	8	2	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Small4	Same	0	**0**	**0**	**0**	**0**	**0**	3	17	**0**	**0**	3	6	1	1	7	3	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Small5	Same	0	**0**	**0**	**0**	**0**	**0**	**0**	7	4	**0**	4	**0**	**0**	**0**	5	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Medium1	1	*	**41**	317	175	80	221	176	243	254	242	372	140	195	146	199	316	55	126	71	88	168	45	84	117	105	82	64
Medium2	1	*	**39**	313	197	105	147	154	225	258	161	419	130	184	173	202.5	243	70	123	82	88	160	40	82	108	108	78	65
Medium3	1	*	**60**	357	216	139	246	191	249	251	265	359	189	248	267	-	255	102	185	137	112	176	61	123	135	156	136	91
Medium4	3	0.21	39	247	149	88	165	148	285	321	181	348	112	164.5	169	177.5	235	**32**	116	55	84	144	35	62	75	84	73	66
Medium5	2	0.122	55	292	190	88	130	166	132	276	151	171	141	219.5	303	-	215	61	129	106	103	71	**49**	75	160	141	103	89
large	3	0.13	463	926	912	730	529	798	1138	1027	-	1068	876	851.5	1166	-	-	653	821	777	915	417	**407**	690	589	719	680	576

Note: M1: (Abdullah et al. [6]). M2: (Abdullah and Turabieh [2]). M3: (McMullan [26]). M4: (Abdullah et al. [4]). M5: (Ejaz and Javed [19]). M6 (Asmuni et al. [12]). M7: (Abdullah and Turabieh [3]). M8: (Abdullah et al. [5]). M9: (Burke et al. [16]). M10: (Landa-Silva and obit [25]). M11: (Socha et al. [33]). M12: (Burke et al. [15]). M13: (Socha et al. [34]). M14: (Al-Betar et al. [10]). M15: (Turabieh and Abdullah [36]). M16: (Landa-Silva and obit [29]). M17: (Obit et al. [27]). M18: (Al-Betar et al. [11]). M19: (Turabieh et al. [37]). M20: (Jaradat and Ayob [23]). M21: (Shaker and Abdullah [32]). M22: (Abuhamdah and Ayob [7]). M23: (Abuhamdah and Ayob [8]). M24: (Abuhamdah and Ayob [9]).

- *The value for PL-LS, M1, M2, etc. are the minimum penalty cost obtained by each approach.*

In order to find out whether the performance of the PB-LS is different in terms of solution quality when compared to other methods in the literature, again we carried out a Wilcoxon test between PB-LS and other methods (Garcia et al. [21]). Since none of the compared method report the full details of the runs, the reported average value by other method were used in our test. Please note that only those that reported the average values are considered in the comparisons. All methods are compared by means of pairwise comparisons. The confidence level is 95%. The null hypothesis assumes there is no difference between the compared methods. A *p*-value less than 0.05 means that, there is a significant difference between these methods. Table 5 show the results of Wilcoxon test (*p*-value).

Table 5 Wilcoxon test results (P-value) between PB-LS and other approaches with 95% confidence level.

PB-LS Vs.	P-value
M3	0.010
M4	0.110
M7	**0.003**
M11	**0.010**
M12	**0.004**
M15	0.110
M17	**0.026**
M22	**0.003**
M23	**0.004**
M24	**0.003**

According to the *p*-values in Table 5, almost all compared algorithm, the reported *p*-value is less than 0.05 which mean that the performance of PB-LS is significantly different from other methods (except M4 [26] and M15 [36]). Although, our algorithm is not better than M4 and M15, according to Wilcoxon test, the results reported in Table 5 shows that in terms of solution quality, our method outperformed M4 on 6 instances and ties on the small instances (obtained same results for all small instances) and outperformed M15 on 5 instances and ties on the small instances (obtained same results for all small instances).

The positive result reported in Tables 4 and 5 reveal that our method obtains competitive results compared to the best known methodologies and also outperforms them on some instances (medium 1 to medium 3). As a result, we conclude that the use of a population helps PB-LS in obtaining good quality solutions. Indeed, PB-LS did not employ a complex operator such as crossover operator which usually requires a repair mechanism to maintain feasibility.

6 Conclusions and discussion

This work has proposed a new population based local search algorithm (PB-LS) that is motivated by the idea from Gravitational Emulation Local Search algorithm (GELS). PB-LS was embedded with MPCA-ARDA (multi-neighbourhood particle collision algorithm and adaptive randomized descent algorithm) as a local search to overcome the limitations of population based algorithms (in fact it can be embedded within any local search algorithm too). In order to evaluate the effectiveness of PB-LS with MPCA-ARDA, we tested PB-LS with MPCA-ARDA on the Socha course timetabling benchmark dataset [33]. Results indicate that, PB-LS with MPCA-ARDA outperform MPCA-ARDA and some other methods in the literature, the results are generally statistically significant. Thus we can conclude that PB-LS with MPCA-ARDA has more capability in intensifying and diversifying the search.

The limitation of PB-LS approach is that, we need to determine the number of *reset.iter* to reset the directions and update the solutions. Smaller *reset.iter* indicate more exploration, whilst, larger values promote more exploitation.

References

1. Abdullah S. (2006) Heuristic approaches for university timetabling problems. PhD Thesis, The University of Nottingham. The School of Computer Science and Information Technology.
2. Abdullah S. and Turabieh H. (2008) Electromagnetic Like Mechanism and Great Deluge for Course Timetabling Problems," In the first seminar on Data Mining and Optimization DMO, volume I, ISBN 9778-967-5048-36-4, pages 21-25.
3. Abdullah S. and Turabieh H. (2008) Generating University Course Timetable Using Genetic Algorithms and Local Search. In Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology , volume 1, pages 254-260.
4. Abdullah S., Burke E. K. and McCollum B. (2007) A hybrid evolutionary approach to the university course timetabling problem. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), Singapore, pages. 1764–1768.
5. Abdullah S., Burke E. K. and McCollum B. (2007) Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for the University Course Timetabling Problem. Metaheuristics - Progress in Complex Systems Optimization. Operations Research/Computer Science Interfaces, volume 39, pages. 153–169. Springer US.
6. Abdullah S., Burke E.K. and McCollum B. (2005) An Investigation of Variable Neighbourhood Search for University Course Timetabling. In the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications (MISTA), July 18th-21st, pages. 413–427. New York, USA.
7. Abuhamdah A. and Ayob M. (2009). Multi-neighbourhood particle collision algorithm for solving course timetabling problems. In Proceeding of 2nd Conference On Data Mining and Optimization, October, pp.21-27, Selangor, Malaysia, IEEE, 2009.
8. Abuhamdah A. and Ayob M. (2010) Adaptive Randomized Descent Algorithm for Solving Course Timetabling Problems. International Journal of the Physical Sciences (IJPS - 2010), volume 5(16), pp.2516-2522.
9. Abuhamdah A. and Ayob M. (2011) MPCA-ARDA for Solving Course Timetabling Problems. In Proceeding of the 3nd Conference On Data Mining and Optimization (2011), pp, 171-177, June 2011, Selangor, Malaysia.
10. Al-Betar M. A., Khader A. T, and Gani T. A. (2008) A harmony search algorithm for university course timetabling. In Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, Canada, August 18-22, 2008.
11. Al-Betar M. A., Khader A. T., and Liao I. Y. (2010) A harmony search with multi-pitch adjusting rate for the university course timetabling. Recent Advances In Harmony Search Algorithm. Studies in Computational Intelligence, volume 270, pages147-161.
12. Asmuni H., Burke E. K. and Garibaldi J. M. (2005) Fuzzy multiple heuristic ordering for course timetabling. In Proceedings of the 5th UK Workshop on Computational Intelligence (UKCI05), London, UK, pages 302-309.
13. Blum C. and Roli A. (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR), 35(3) 268-308.
14. Burke E. K., Bykov Y., Newall J., and Petrovic S. (2003) A time-predefined approach to course timetabling," Yugoslav Journal of Operations Research (YUJOR), 13(2)139-151.
15. Burke E. K., Kendall G. and Soubeiga E. (2003) A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics, 9(6)451–470.
16. Burke E. K., Meisels A., Petrovic S. and Qu R. (2007) A Graph-Based Hyper-Heuristic for Timetabling Problems," European Journal of Operational Research, 176(1) 177-192.
17. Chiarandini M., Birattari M., Socha K., and Rossi-Doria O. (2006) An effective hybrid algorithm for university course Timetabling. Journal of Scheduling, 9(5) 403-432.
18. Costa D. (1994) A tabu search for computing an operational timetable. European Journal of Operational Research. 76 (1) 98-110.
19. Ejaz N. and. Javed M (2007) A Hybrid Approach for Course Scheduling Inspired by Die-Hard Co-Operative Ant Behavior. In Proceedings of the IEEE International Conference on Automation and Logistics, August, pages 3095 – 3100, Jinan, China, 2007.
20. Elmohamed MAS., Coddington P., and Fox G. (1998) A comparison of annealing techniques for academic course scheduling. Selected Papers from 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT II), Toronto, Canada, Lecture Notes in Computer Science, Springer-Verlag, volume 1408, pages 92-112.

21. Garcia S., Fernandez A., Luengo J., and Herrera F., "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," Information Sciences, vol. 180, pp. 2044-2064, 2010.
22. Hoos H. H. and Stutzle T. (2004) Stochastic Local Search: Foundations and Applications. Morgan Kaufmann /Elsevier.
23. Jaradat G. M. and Ayob M. (2010) An elitist-ant system for solving the post-enrolment course timetabling problem. In the 2010 International Conference on Database Theory and Application (DTA 2010), pages167-176, Springer-Verlag Berlin Heidelberg.
24. Landa-Silva D. and Obit J. H. (2009) Evolutionary nonlinear great deluge for university course timetabling. In Proceedings of the 2009 International Conference on Hybrid Artificial Intelligence Systems (HAIS 2009), Lecture Notes in Computer Science, volume 5572/2009, pages 269-276.
25. Landa-Silva D. and Obit J.H. (2008) Great deluge with non-linear decay rate for course timetabling problems. In proceedings of 4th International IEEE Conference Intelligent Systems. Pages 8.11-8.18.
26. McMullan P. (2007) An extended implementation of the great deluge algorithm for course timetabling," ICCS. In Proceedings of the 7th International Conference of Computational Science, Part I, Lecture Note in Computer Science, Springer-Verlag Berlin Heidelberg, Germany, volume 4487, pages 538-545.
27. Obit J. H., Landa-Silva D., Ouelhadjand D. and Sevaux M. (2009) Non-Linear Great Deluge with Learning Mechanism for Solving the Course Timetabling Problem. MIC 2009: The VIII Metaheuristics International Conference, Hamburg, Germany, pages 1-10.
28. Petrovic S. and Burke E. K. (2004) Educational Timetabling. in Joseph Leung (Ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman & Hall/CRCRC Press, 2004, pages 45.1 – 45.23.
29. Sabar N. R., Ayob M., Kendall G. and Qu R. (2012) A honey-bee mating optimization algorithm for educational timetabling problems. Eur J Oper Res. 216(3), 533-543, 2012.
30. Schaerf A. (1999a) A survey of automated timetabling. Artificial Intelligence Review, 13(2):87-127.
31. Schaerf A. (1999b) Local search techniques for large high-school timetabling problems. Systems, Man and Cybernetics, Part A: IEEE Transactions on Systems and Humans, 29(4) 368-377.
32. Shaker K. and Abdullah S. (2010) Controlling multi algorithms using round robin for university course timetabling problem. In the 2010 International Conference on Database Theory and Application (DTA 2010), Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, pages 47-55, December 2010.
33. Socha K., Knowles J. and Samples M. (2002) A max-min ant system for the university course timetabling problem. In proceedings of the 3rd International Workshop on Ant Algorithms, ANTS 2002, Lecture Notes in Computer Science Vol 2463 (10), pages 1-13.
34. Socha K., Sampels M., and Manfrin M. (2003) Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In the third European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2003), Lecture Notes in Computer Science, 2003, volume 2611/2003, pages. 334-345.
35. Thompson J. M. and Dowsland K. A. (1996) Variants of simulated annealing for the examination timetabling problem. Annals of Operations Research Journal, 63(1)105–128.
36. Turabieh H. and Abdullah S. (2009) Incorporating Tabu Search into Memetic approach for enrolment-based course timetabling problems (TS-MA). In Proceeding of 2nd Conference On Data Mining and Optimization, October, pages. 115-119, Selangor, Malaysia.
37. Turabieh H., Abdullah S., McCollum B., and McMullan P. (2010) Fish swarm intelligent algorithm for the course timetabling problem. In Rough Set and Knowledge Technology Conference (RSKT), Lecture Notes in Computer Science, volume 6401/2010, pp.588-595.
38. Webster B.L. (2004) Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction. PhD Thesis, College of Engineering at Florida Institute of Technology.