Computers & Industrial Engineering 78 (2014) 12–25

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A new hybrid algorithm for university course timetabling problem using events based on groupings of students[☆]

Rakesh P. Badoni, D.K. Gupta[*], Pallavi Mishra

Department of Mathematics, Indian Institute of Technology Kharagpur, Kharagpur 721302, India

ARTICLE INFO

Article history:
Received 28 February 2014
Received in revised form 23 July 2014
Accepted 23 September 2014
Available online 6 October 2014

Keywords:
Timetabling
Genetic algorithm
Demand-driven
Local search
Grouping
Distance to feasibility

ABSTRACT

In this paper, a new hybrid algorithm (NHA) combining genetic algorithm with local search and using events based on groupings of students is described to solve the university course timetabling problem. A list of events such as lectures, tutorials, laboratories and seminars are ordered and mutually disjoint groups of students taking them are formed in such a way that once a student is selected in any group, he is excluded from further selection in other groups. The union of all the events taken by all the students of each group is formed. The number of events in each group is termed as its group size whose upper bound is restricted by the total number of timeslots and can be reduced to the maximum number of events per student. The above process of forming groups is repeated till the size of each group is reduced within this bound by not choosing those events which are common for all the students in the group. Now, the genetic algorithm with local search (GALS) is applied on a number of benchmark problems. The experimental results show that our algorithm, NHA, is able to produce promising results when compared with the results obtained by using GALS and other existing algorithms.

1. Introduction

The development of automated timetables also known as the allocation of the resources for tasks under predefined constraints, reduces time for its creation, minimizes errors and satisfies as nearly as possible a set of desirable objectives. These problems involve timetabling for educational (course and examination), employees, sports events, transportation, etc. These are important and challenging problems encountered in computer science (CS), operation research (OR) and artificial intelligence (AI). Here, we are concerned with the university course timetabling problem which maximizes the possibility of allocations and minimizes the violation of constraints. These are high dimensional multi-objectives combinatorial optimization problems belonging to a class of NP-complete problems. A university course timetabling problem consists in assigning a set of events (lectures, tutorials, laboratories, seminars, etc.) into a limited number of time slots and rooms in such a way as to minimize violation of a predefined set of constraints. A general and effective solution for timetabling is very difficult as it involves problem diversity, constraints variances and changed requirements from university to university. As a result, the construction of their solution involving a weekly schedule of events acceptable to all people involved and satisfying hard as well as soft constraints as efficiently as possible is extremely difficult and grows exponentially with size. Hard constraints are those which can not be violated under any circumstances whereas soft constraints can be relaxed. For example, a hard constraint implies that a student can not be physically present in two different events at the same time. An example of a soft constraint can be to avoid the last time slot of the day for an event. Accordingly, a violation of soft constraints requires penalizing the underlying solution with some penalty value that is added to the cost of the solution. Thus, the aim of timetabling problem is usually to obtain a feasible solution that does not violate any hard constraints and minimizes its cost when penalty of soft constraints violations is also taken into consideration. As many combinatorial optimization problems, their solutions are generally obtained in the construction and the improvement phases. The algorithm starts with an empty timetable in construction phase and gradually constructs a timetable by adding one event into it. In general, the initial timetable is usually of poor quality and consists of many constraints violations. The improvement phase is used after achieving a complete timetable from construction phase and tries to gradually improve the quality of complete timetable. In the

[☆] This manuscript was processed by Area Editor T.C. Edwin Cheng.
[*] Corresponding author. Tel.: +91 3222 283652 (O); fax: +91 3222 255303x282700.
E-mail addresses: rakeshbadoni@gmail.com (R.P. Badoni), dkg@maths.iitkgp.ernet.in (D.K. Gupta), mshrpal@gmail.com (P. Mishra).

improvement phase, some events of the schedule may be altered hoping to achieve a better timetable.

The university course timetabling problems are extensively studied by a number of researchers (Cambazard, Hebrard, O'Sullivan, & Papadopoulos, 2012; Carter, 1986; Lewis, 2008; Nothegger, Mayer, Chwatal, & Raidl, 2012; Yang & Jat, 2011) and a number of approaches are proposed for their solutions. They have systematically categorized these problems, presented their mathematical formulations and described both exact and heuristic algorithms for their solutions. Some of the most important methods used are sequential methods, cluster methods, constraint-based methods and generalized search methods. The hybrid evolutionary algorithms, metaheuristics, multi-criteria approaches, case based reasoning techniques, hyper-heuristics and adaptive approaches to solve them are described in (Petrovic & Burke, 2004). One approach (Abdullah & Turabieh, 2008; Wijaya & Manurung, 2009) converted it to a graph in which the nodes correspond to lectures and the edges between the nodes correspond to the constraints and used graph coloring algorithms. The graph coloring algorithm assigns a limited number of colors to the nodes of the graph in such a way that no two nodes connected by an edge have the same color. The number of colors correspond to the number of available time slots. The techniques of constraint based reasoning (Cambazard et al., 2012; Wijaya & Manurung, 2009) were also successfully used for the solution of the university course timetabling problem. These problems are represented as the Constraint Satisfaction Problems (CSPs) and then solved by using techniques used for solving CSPs. An approach in which constraints were iteratively added to the CSPs and solved by backtrack algorithm is described in Banks, Van Beek, and Meisels (1998). Several metaheuristic approaches inspired from nature and apply nature-like processes to solutions or populations of solutions to get optimal solutions of these problems are ant colony optimization (Rossi-Doria et al., 2003; Nothegger et al., 2012), evolutionary algorithms (Rossi-Doria et al., 2003; Yang & Jat, 2011), simulated annealing (Abdullah, Shaker, McCollum, & McMullan, 2010; De Causmaecker, Demeester, & Vanden Berghe, 2009) and tabu search (Lü & Hao, 2010; Wilke & Ostler, 2008). Burke, McCollum, Meisels, Petrovic, and Qu (2007) employed tabu search within a graph based hyper-heuristic and applied it to the course timetabling benchmark datasets with the aim of raising the level of generality by operating on different problem domains. In Abdullah et al. (2010), a Dual-sequence Simulated Annealing algorithm is employed as an improvement algorithm. The Round Robin algorithm is used to control the selection of neighborhood structures within it. A genetic algorithm combined with a sequential local search for the curriculum based course timetabling problem which also used the two phased approach, i.e., the construction phase and the improvement phase is presented in Abdullah, Turabieh, McCollum, and Burke (2009). A guided search genetic algorithm for their solutions is discussed in Yang and Jat (2011). By grouping similar lectures in a timetabling problem, a decomposed heuristic is described in De Causmaecker et al. (2009). In Rossi-Doria et al. (2003), the performance of the implementations of five different metaheuristics on a university course timetabling problem are compared unbiasedly. For fairness, all the algorithms implementing them use a common solution representation and a common local search. Lewis and Paechter (2005) presented a grouping genetic algorithm for feasible solution of university course timetabling problem. They used the definition of grouping (Falkenauer, 1998) as one where the task is to partition a set of objects U into a collection of mutually disjoint subsets u_i of U such that $\cup u_i = U$ and $u_i \cap u_j = \phi,\ i \neq j$, and according to a set of problem-specific constraints that define valid and legal groupings. The NP-hard bin packing problem defined for a finite set of items of various sizes is a well known example of it. Here, the task is to partition all of the items into various bins such that the total size of all the items in any one bin does not exceed the bin's maximum capacity with the minimum number of bins used. Further, when genetic algorithm with grouping is applied to solve university course timetabling problem, the representations and resulting genetic operators are used in such a way that allow the groupings of objects to be propagated as they are the building blocks of the problem and not the particular positions of any one object on its own. They also considered feasibility and optimality as two separate subproblems. They suggested that the performance of any algorithms satisfying hard and soft constraints might be different. This means, what may be a good approach for finding feasibility may not necessarily be good for optimality of solutions. They further suggested that algorithms comprising two stages, the first to find feasibility and the second to optimize soft constraints whilst staying in feasible regions of the search space might be the more promising approach. In their work, the set of events represents the set of objects to partition and the groups are defined by the timeslots. So, a feasible solution is therefore one in which all the events $|E|$ are partitioned into $|T|$ feasible timeslots $t_1, t_2, \ldots, t_{|T|}$. A feasible timeslot t_i $(1 \leqslant i \leqslant |T|)$ is one in which none of the events in t_i conflict and all the events in t_i can be placed in their own suitable room. We have considered the set of students as the set of objects to partition into mutually disjoint student groups and then events taken by these student groups are assigned to timeslots and rooms by using GALS. This is applied in two phases, construction phase and improvement phase. In the first phase, the goal is to achieve feasibility by satisfying all hard constraints whereas in the second phase it is to optimize soft constraints whilst maintaining the feasibility of the solution.

In this paper, a new hybrid algorithm (NHA) combining genetic algorithm with local search and using events based on groupings of students is described to solve the university course timetabling problem. A list of events such as lectures, tutorials, laboratories and seminars are ordered and mutually disjoint groups of students taking them are formed in such a way that once a student is selected in any group, he is excluded from further selection in other groups. The union of all the events taken by all the students of each group is formed. The number of events in each group is termed as its group size whose upper bound is restricted by the total number of timeslots and can be reduced to the maximum number of events per student. The above process of forming groups is repeated till the size of each group is reduced within this bound by not choosing those events which are common for all the students in the group. Now, the genetic algorithm with local search (GALS) is applied on a number of benchmark problems. The experimental results show that our algorithm, NHA, is able to produce promising results when compared with the results obtained by using GALS and other existing algorithms.

This paper is organized as follows. Section 1 is the introduction. In Section 2, our university course timetabling problem and its mathematical formulation is described. Genetic algorithm with local search (GALS) is discussed in Section 3. The proposed new hybrid algorithm (NHA) combining GALS and using events based on grouping of students, instead of individual student, to get an optimal solution is given in Section 4. In Section 5, the results of the newly designed algorithm along with its comparison with the GALS and the other state-of-the-art algorithms considered from the literature are displayed. Finally, conclusions are included in Section 6.

2. University course timetabling problem

In this section, a university course timetabling problem and its mathematical formulation is described. It is a multidimensional assignment problem, in which events (lectures, tutorials, laboratories, etc.) taken by students are to be scheduled in 45 timeslots (5 days of 9 timeslots each) and rooms in such a way that the violation of predefined set of constraints is minimum. It is assumed that each student takes a number of events and each room is of specified size with a set of features. A feasible solution assigns all the events to timeslots and rooms so that the following hard constraints.

1. Only one event is attended by a student at any timeslot.
2. All events are to be assigned to rooms having adequate seating capacity and all the required features.
3. Only one event is assigned to any one room in any timeslot

are satisfied. In addition, the timetable is penalized equally for each occurrence of the following soft constraint violations

1. Events should not be scheduled in the last timeslot of a day.
2. No student should attend more than two events in consecutive time slots in a day.
3. All student should have more than one event in a day.

The mathematical formulation of university course timetabling problem can be described as follows. It consists of a set of events E to be scheduled in 45 timeslots (5 days of 9 timeslots each), a set of rooms R in which events can take place, a set of students S who attend the events and a set of features F satisfied by rooms and required by events. Each student attends a number of events and each room has a capacity.

- $E = \{e_1, e_2, \ldots, e_n\}$ is a set of n events.
- $R = \{r_1, r_2, \ldots, r_m\}$ is a set of m rooms.
- $T = \{t_1, t_2, \ldots, t_{45}\}$ is a set of 45 timeslots.
- $S = \{s_1, s_2, \ldots, s_p\}$ is a set of p students.
- $F = \{f_1, f_2, \ldots, f_q\}$ is a set of q rooming features.
- $G(r_l)$ is a set of features of room r_l.
- r_l.*capacity* is the capacity of room r_l.
- x_{ijkl} is a boolean variable representing student s_i attending an event e_j in timeslot t_k and room r_l and defined for $i = 1, 2, \ldots, p,\ j = 1, 2, \ldots, n,\ k = 1, 2, \ldots, 45$ and $l = 1, 2, \ldots, m$, as
$$x_{ijkl} = \begin{cases} 1 & \text{if the above combination holds true, and} \\ 0 & \text{otherwise.} \end{cases}$$
- y_{jlg} is a boolean variable representing an event e_j held in room r_l requires feature f_g and defined for $j = 1, 2, \ldots, n,\ l = 1, 2, \ldots, m$ and $g = 1, 2, \ldots, q$, as
$$y_{jlg} = \begin{cases} 1 & \text{if the above combination holds true, and} \\ 0 & \text{otherwise} \end{cases}$$
- z_{lg} is a boolean variable representing room r_l with room feature f_g and defined for $l = 1, 2, \ldots, m$ and $g = 1, 2, \ldots, q$, as,
$$z_{lg} = \begin{cases} 1 & \text{if room } r_l \text{ have feature } f_g, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

In a similar way, the hard and the soft constraints can be mathematically described. The hard constraints formulation is as follows.

1. Only one event is attended by a student at any timeslot:
$$\sum_{j=1}^{n}\sum_{l=1}^{m} x_{ijkl} \leqslant 1, \quad i = 1, 2, \ldots, p;\ k = 1, 2, \ldots, 45.$$
2. All events are to be assigned to rooms having adequate seating capacity and all the required features:
$$\sum_{g=1}^{q} y_{jlg} \leqslant |G(r_l)|, \quad j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, m; \quad \text{and}$$
$$y_{jlg} \leqslant z_l g, \quad j = 1, 2, \ldots, n;\ l = 1, 2, \ldots, m;\ g = 1, 2, \ldots, q.$$
$$r_l.capacity \geqslant \sum_{i=1}^{p} x_{ijkl}, \quad j = 1, 2, \ldots, n;\ k = 1, 2, \ldots, 45;$$
$$l = 1, 2, \ldots, m.$$
3. Only one event is assigned to any one room in any timeslot:
$$\sum_{j=1}^{n} x_{ijkl} \leqslant 1, \quad i = 1, 2, \ldots, p;\ k = 1, 2, \ldots, 45;\ l = 1, 2, \ldots, m.$$

In order to achieve a feasible timetable, it is necessary that each event $e_1, e_2, \ldots, e_n$ is assigned to exactly one room $r_1, r_2, \ldots, r_m$ and exactly one of the 45 timeslots $t_1, t_2, \ldots, t_{45}$, such that the above mentioned hard constraints are satisfied. The mathematical formulation of the soft constraints is as follows.

1. Events should not be scheduled in the last timeslot of a day:
$$\sum_{j=1}^{n}\sum_{l=1}^{m} x_{ijkl} = 0, \quad i = 1, 2, \ldots, p;\ k = 9, 18, \ldots, 45.$$
2. No student should attend more than two events in consecutive time slots in a day.
$$\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{k=a}^{a+2} x_{ijkl} \leqslant 2, \quad i = 1, 2, \ldots, p;$$
$$a = 1, 2, \ldots, 7, 10, 11, \ldots, 16, \ldots, 37, 38, \ldots, 43.$$
3. All student should have more than one event in a day:
$$\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{k=d+1}^{d+9} x_{ijkl} \geqslant 1, \quad i = 1, 2, \ldots, p;\ d = 0, 9, 18, 27, 36.$$

If these soft constraints are also considered then the objective is to minimize the violations of the soft constraints in order to get an optimal solution for the university course timetabling problem. An example of a feasible solution and an optimal solution for the university course timetabling problem is given as follows.

Example 2.1. Consider a university course timetabling problem involving 100 events $\{e_1, e_2, \ldots, e_{100}\}$ taken by 80 students $\{s_1, s_2, \ldots, s_{80}\}$. These events are assigned to 5 rooms $\{r_1, r_2, \ldots, r_5\}$ and 45 timeslots $\{t_1, t_2, \ldots, t_{45}\}$. Each room takes approximately 3 features out of total 5 features $\{f_1, f_2, \ldots, f_5\}$. The events are randomly assigned to students restricting maximum and minimum number of events per student by 12 and 3 respectively. Also, maximum and minimum number of students per event are restricted to 12 and 0 respectively. The timetable for feasible solution is shown in Table 1. In this table an ordered pair of an event e_i and a room r_j denoted by (e_i, r_j) is assigned to a timeslot t_k.

Table 2 gives an optimal solution for the above mentioned problem.

One can see that the first soft constraint is violated in timeslots t_{18} and t_{36} but no student is enrolled in events e_5 and e_{61}.

Keeping things in simplest possible manner, a direct solution representation is chosen. A solution consists of an integer valued ordered lists of length $|E|$, say $a[i]$, where $1 \leqslant a[i] \leqslant 45$ and $1 \leqslant i \leqslant |E|$. Here, $a[i]$ denotes the timeslot for event e_i. The room assignments are generated by matching algorithm. For every timeslot there is a list of events taking place in it and a preprocessed list

Table 1
The feasible timetable.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
(e_{26}, r_1)	(e_{79}, r_1)	(e_{89}, r_1)	(e_{33}, r_1)	(e_{64}, r_1)	(e_{63}, r_1)	(e_1, r_5)	(e_2, r_5)	
(e_{38}, r_2)	(e_{83}, r_2)		(e_{81}, r_2)	(e_{88}, r_2)		(e_{65}, r_1)		
(e_{40}, r_3)								
t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
(e_7, r_1)	(e_{13}, r_1)	(e_{28}, r_1)	(e_{30}, r_5)	(e_{23}, r_1)	(e_{35}, r_1)	(e_4, r_4)	(e_{14}, r_1)	
(e_{12}, r_5)	(e_{25}, r_2)	(e_{73}, r_2)	(e_{43}, r_1)	(e_{44}, r_2)	(e_{85}, r_2)	(e_{21}, r_5)	(e_{71}, r_5)	
(e_{29}, r_2)	(e_{47}, r_3)	(e_{82}, r_3)		(e_{95}, r_5)	(e_{92}, r_3)	(e_{56}, r_1)	(e_{98}, r_2)	
(e_{51}, r_3)					(e_{96}, r_5)	(e_{60}, r_2)		
						(e_{66}, r_3)		
t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}
(e_9, r_1)	(e_{16}, r_5)	(e_{15}, r_2)	(e_3, r_5)	(e_{50}, r_1)	(e_{18}, r_4)	(e_{39}, r_1)	(e_{10}, r_1)	
(e_{78}, r_2)	(e_{34}, r_1)	(e_{24}, r_1)	(e_{93}, r_2)	(e_{77}, r_2)	(e_{31}, r_2)		(e_{17}, r_5)	
(e_{99}, r_3)	(e_{62}, r_2)	(e_{67}, r_5)	(e_{100}, r_1)	(e_{91}, r_3)	(e_{46}, r_3)		(e_{37}, r_2)	
	(e_{72}, r_4)	(e_{84}, r_3)			(e_{49}, r_1)			
	(e_{94}, r_3)				(e_{69}, r_5)			
t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}
	(e_{57}, r_5)	(e_{97}, r_1)	(e_{41}, r_1)	(e_{22}, r_5)	(e_{58}, r_2)	(e_{76}, r_1)	(e_{53}, r_1)	
				(e_{27}, r_1)				
t_{37}	t_{38}	t_{39}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}
(e_{20}, r_1)	(e_{11}, r_1)	(e_{19}, r_1)	(e_{55}, r_1)	(e_6, r_1)	(e_{32}, r_1)	(e_{36}, r_1)	(e_5, r_1)	
(e_{42}, r_2)	(e_{45}, r_2)	(e_{48}, r_3)	(e_{70}, r_2)	(e_8, r_5)	(e_{68}, r_2)	(e_{74}, r_2)	(e_{52}, r_2)	
(e_{59}, r_3)	(e_{87}, r_3)	(e_{86}, r_2)	(e_{80}, r_5)	(e_{54}, r_2)			(e_{75}, r_3)	
(e_{61}, r_4)								
(e_{90}, r_5)								

Table 2
The optimal timetable.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
(e_{83}, r_2)	(e_3, r_5)	(e_{34}, r_2)	(e_1, r_5)	(e_{63}, r_1)	(e_{51}, r_5)	(e_{21}, r_5)	(e_{23}, r_1)	
	(e_{20}, r_1)	(e_{75}, r_1)	(e_{38}, r_2)	(e_{100}, r_2)	(e_{77}, r_1)		(e_{33}, r_2)	
					(e_{89}, r_3)			
t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
(e_{13}, r_4)	(e_{10}, r_1)	(e_{44}, r_1)	(e_{41}, r_1)	(e_{43}, r_1)	(e_2, r_5)	(e_{56}, r_1)	(e_7, r_1)	(e_{61}, r_1)
(e_{17}, r_5)	(e_{25}, r_2)	(e_{74}, r_3)	(e_{58}, r_2)	(e_{85}, r_2)	(e_9, r_1)	(e_{73}, r_3)	(e_{49}, r_2)	
(e_{26}, r_2)	(e_{40}, r_3)	(e_{99}, r_2)			(e_{24}, r_2)	(e_{82}, r_4)		
(e_{64}, r_3)					(e_{36}, r_3)	(e_{86}, r_2)		
(e_{94}, r_1)								
t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}
(e_{57}, r_5)	(e_{35}, r_1)	(e_{14}, r_1)	(e_{19}, r_1)	(e_8, r_5)	(e_6, r_1)	(e_{42}, r_2)	(e_{66}, r_2)	
(e_{70}, r_1)	(e_{48}, r_3)	(e_{29}, r_3)	(e_{69}, r_2)	(e_{39}, r_1)	(e_{79}, r_3)			
(e_{72}, r_2)	(e_{52}, r_2)	(e_{53}, r_2)	(e_{98}, r_3)	(e_{54}, r_2)	(e_{93}, r_2)			
				(e_{55}, r_3)				
t_{28}	t_{29}	t_{30}	t_{31}	t_{32}	t_{33}	t_{34}	t_{35}	t_{36}
(e_{15}, r_2)	(e_{46}, r_1)	(e_{27}, r_2)	(e_{18}, r_1)	(e_{12}, r_5)	(e_{16}, r_5)	(e_{45}, r_1)	(e_{71}, r_5)	(e_5, r_1)
(e_{22}, r_5)	(e_{81}, r_2)	(e_{68}, r_1)	(e_{59}, r_4)		(e_{78}, r_1)		(e_{97}, r_2)	
			(e_{76}, r_3)					
			(e_{90}, r_2)					
t_{37}	t_{38}	t_{39}	t_{40}	t_{41}	t_{42}	t_{43}	t_{44}	t_{45}
(e_{28}, r_1)	(e_{65}, r_2)	(e_{31}, r_2)	(e_4, r_1)	(e_{67}, r_5)	(e_{62}, r_2)	(e_{11}, r_1)	(e_{37}, r_2)	
(e_{30}, r_5)	(e_{84}, r_1)	(e_{80}, r_5)	(e_{47}, r_3)		(e_{88}, r_1)	(e_{96}, r_5)	(e_{50}, r_1)	
(e_{32}, r_3)		(e_{87}, r_1)	(e_{60}, r_2)				(e_{91}, r_3)	
(e_{92}, r_2)							(e_{95}, r_5)	

of possible rooms to which these events can be assigned according to size and features. Further, a bipartite matching algorithm gives a maximum cardinality matching between these two sets using a deterministic network flow algorithm considered by Papadimitriou and Steiglitz (1998). If there are still unplaced events left, it takes them in label order and puts each one onto the room of correct type and size which is occupied by the fewest events. If two or more rooms are tied, it takes the one with the smallest label. This leads to a complete assignment of timeslots and rooms to all events.

3. Genetic algorithm with local search (GALS)

In this section, a hybrid algorithm combining a genetic algorithm with a local search algorithm using the solution representation as described in the previous section for university course timetabling problem is described. Genetic algorithms and their modifications have been successfully used to solve a number of combinatorial optimization problems in particular timetabling problems. GALS has been selected because it uses the merits of both genetic algorithm and local search algorithm. Genetic algorithm is a population-based algorithm which uses exploration of the entire search space without concentrating on the individuals of good fitness within a population. They may experience premature convergence and get trapped into local optima. Thus, they take more time. On the other hand, local search algorithm emphasizes on exploitation and move in one direction without performing a wider scan of the search space. GALS improve their performance by fine tuning the global search. The term genetic algorithm (GA) was first used by John (1992). It is developed based on Darwin's theory of survival of the fittest. It is an optimization procedure used in finding approximate solutions to many difficult problems arising in various fields of applications. It is a probabilistic search algorithm used to transform iteratively a population of objects, each with an associated fitness value into a new population of offspring objects using the natural selection and two operations termed as crossover and mutation operations. The encoding of potential solution to a specific problem in the form of chromosome like data structure is required. Better offsprings preserving the critical information are generated from these structures by applying the recombination operators. Thus, variables are represented as genes on chromosomes. Chromosomes with better fitness are obtained through natural selection and the genetic operators. Natural selection guarantees that chromosomes with better fitness will propagate in future populations. Recombination or crossover operator combines genes from two parents chromosomes to form a new chromosome with high probability of having better fitness than the parents. Mutation alters one or more genes values in a chromosome from its initial state with the aim of finding a better solution. It can help to prevent population from stagnating at any local optima. Thus, a genetic algorithm operates on a population of randomly generated potential solutions across the search space and comprise three major stages namely selection, reproduction and replacement. Selection stage allocates more copies of those solutions with higher fitness and thus imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to give higher chance to the fittest individuals than those less fit of being chosen as parents for the next generation as in natural selection. Reproduction is performed by means of crossover and mutation operators applied to the selected parents. Crossover combines parts of each of two parental solutions to create new, possibly better solutions, while mutation modifies a solution locally, i.e., mutation performs a random walk in the vicinity of a candidate solution. Finally, the population of offsprings created by selection and reproduction replaces the original parental population, usually trying to keep the best individuals and removing the worst ones. The selection stage ensures the better utilization of the healthier offspring, while the reproduction stage ensures the proper exploration of the search space based on the fact that the replacement policy allows the acceptance of new solutions that do not necessarily improve existing ones.

GALS was first introduced by Moscato et al. (1989) and Moscato, Cotta, and Mendes (2004) as a form of population-based genetic algorithms hybridized with an individual learning procedure capable of fine tuning of global search. Many researchers (Abdullah, Burke, & McCollum, 2007a; Abdullah & Turabieh, 2008; Abdullah et al., 2009; Abdullah, Turabieh, McCollum, & McMullan, 2012; Rossi-Doria et al., 2003; Wijaya & Manurung, 2009) have purposed various methods of hybridization to get improve performance for the university course timetabling problem. In our work, we have used a basic implementation of genetic algorithm that uses only the problem specific heuristic information coming from the local search. It is characterized by a steady-state evolution process, i.e. at each generation only one couple of parent individuals is selected for reproduction. Tournament selection strategy is used. A number of individuals are chosen randomly from the current population and the best in terms of fitness function is selected as parent. Since university course timetabling problem involved both hard and soft constraints, where all hard constraints are to be satisfied and soft constraints violations are to be minimized as much as possible. A fitness function $f(I)$ is designed for estimating the violations of soft constraints. Hence, the fitness function $f(I)$ for an individual solution I is given by

$$f(I) = \gamma \times hcv(I) + scv(I),$$

where $hcv(I)$, $scv(I)$ and γ are the number of hard constraint violations, the number of soft constraint violations and a constant that is always set larger than the maximum possible number of soft constraint violations respectively. A uniform crossover operator is used on the solution representation, where for each event, a timeslot's assignment is inherited either from the first or the second parents with equal probability. Mutation is just a random move in the neighborhood defined by the local search extended with three-cycle permutations of the timeslots of three distinct events, which corresponds to the complete neighborhood of the local search algorithm. This complete neighborhood is defined as the union of three types of neighborhood move. Type 1 move takes one event from a timeslot to a different timeslot, type 2 move swaps two events in two different timeslots and type 3 move permutes three events in three distinct timeslots in one of the two possible ways. The offspring replaces the worst member of the population at each generation. The initial population is built randomly by assigning a timeslot to each event for each individual using uniform distribution. The bipartite matching is used for room assignments to ensure that each event-timeslot assignment corresponds uniquely to one timetable, i.e., a complete assignment of events and timeslots to all the rooms. Local search is then applied to each individual in two phases, the construction phase and the improvement phase. The population size and the tournament size is chosen 10 and 5 respectively. The termination criteria of the algorithm is either time limit, or number of iterations, or optimal solution achieved with zero fitness function value.

Algorithm 1. Genetic algorithm

```
Input: A problem instance $I$;
Output: an optimal solution $y_{best}$ for $I$.
1: begin
2: for ($i \leftarrow 1$ to $max$) do    // generate an initial random
   population of solutions of size $max$.
3:     $y_i \leftarrow$ random initial solution;
4:     $y_i \leftarrow$ solution after applying Local Search;
5:     calculate fitness function value of $y_i$;
6: sort population of solutions based on increasing order
   of their fitness function values;
7: repeat
8:     select two parents from population by tournament
   selection;
9:     $y \leftarrow$ child solution created after crossover with
    probability $\alpha$;
10:    $y \leftarrow$ child solution created after mutation with
    probability $\beta$;
11:    $y \leftarrow$ child solution created after applying the
    local search;
12:    $y_{max} \leftarrow y$;    // $y$ replaces the worst solution $y_{max}$
   in the population of sorted solutions.
13:    generate population of solutions sorted based on
   increasing order of their fitness function value;
14:    $y_{best} \leftarrow y_1$;    // $y_1$ is the best solution in the
   population.
15: until (termination criteria not reached);
16: end
```

The local search algorithm is a stochastic first improvement local search based on the neighborhood and applied to each individual solution in two phases, namely the construction phase and the improvement phase. The construction phase goes through the list of all the events in a random order and tries all possible moves in the neighborhood for every event involved in hard constraint violations (hcv) while ignoring all soft constraint violations (scv). If there are hcv for an event, it tries to resolve them by applying all the possible neighborhood moves for every event involved in hcv until a termination criteria is reached. The termination criteria is either an improvement in solution or the pre-specified number of iterations used. The neighborhood is defined as a union of two smaller neighborhoods N_1 and N_2. N_1 defines an operator that chooses a single event at random and move to a different timeslot that can generate the lowest penalty. N_2 defines an operator that selects two events at random and swap their timeslots. N_2 is applied only when N_1 fails. After each move, we apply the matching algorithm to the timeslots affected by the neighborhood move and try to resolve the room allocation disturbance and delta-evaluate the result of the move. A delta evaluation means that we only calculate the hard constraint violations on those events involved in a move within a solution and work out the change in the fitness value of the solution before and after the move. The construction phase of local search algorithm continues to the next event if either there is no hcv or no untried move left in the neighborhood for the current event. If there is still any hcv after applying all neighborhood moves to all the events, the construction phase stops working with no possible feasible solution of the considered problem. The improvement phase is used only after a feasible solution is reached. It is similar to the construction phase with the only difference of using soft constraints instead of the hard constraints. For each event $e_i,\ i = 1, 2, \ldots, n$, the improvement phase tries to make moves in the neighborhood N_1 and N_2 in order to reduce the scv without violating the hard constraints. In short, we can say that the construction phase gives us the feasible solution by satisfying all the hard constraints and the improvement phase moves towards optimality by satisfying as many as possible soft constraints.

Algorithm 2. Construction phase of local search algorithm

```
Input: Individual $I$ from the population;
Output: Either a feasible solution $I$ or no possible solution.
1: begin
2: generate a circular randomly-ordered list $(e_1, e_2, \ldots, e_n)$ of $n$ events;
3: $i \leftarrow 1$;    // $i$ is the event counter;
4: choose the event $e_i$;
5: select event $e_i$ after $i \leftarrow i + 1$;    // move the pointer to the next event.
6: if (all neighborhood moves to all the events have been applied) then
7:     if ($\exists$ any hcv remaining in $I$) then
8:         END LOCAL SEARCH;
9:     else
10:        give a feasible solution $I$ and END the construction phase;
11: if ((current event $e_i$ is feasible) $\bigvee$ ($\nexists$ an untried move left for the event $e_i$)) then
12:     goto 5;
13: $CheckSolution(e_i, I)$;    // returns the solution $I$ after neighborhood moves.
14: if (move reduced number of hcv in $I$) then
15:     make the move;
16:     $i \leftarrow 1$;
17:     goto 5;
18: else
19:     goto 11;
20: end
```

Algorithm 3. Improvement phase of local search algorithm

```
Input: Feasible solution $I$ obtained from construction phase;
Output: An optimal solution $I$.
1: begin
2: take the circular randomly-ordered list $(e_1, e_2, \ldots, e_n)$ of $n$ events generated in construction phase;
3: $i \leftarrow 1$;    // $i$ is the event counter.
4: choose the event $e_i$;
5: select event $e_i$ after $i \leftarrow i + 1$;    // move the pointer to the next event.
6: if (all neighborhood moves to all the events have been applied) then    // $i.e, i = n + 1$
7:     END LOCAL SEARCH with an optimal solution $I$;
8: if ((current event $e_i$ NOT involved in any scv) $\bigvee$ ($\nexists$ an untried move left for the event $e_i$)) then
9:     goto 5;
10: $CheckSolution(e_i, I)$;    // returns the solution $I$ after neighborhood moves.
11: if (move reduced number of scv in $I$ without making $I$ infeasible) then
12:     make the move;
13:     $i \leftarrow 1$;
14:     goto 5;
15: else
16:     goto 8;
17: end
```

Procedure *CheckSolution*(e_i, I)

```
Taking event $e_i$ and solution $I$ as arguments, it returns the solution $I$ after neighborhood moves
Input: $T$: the set of 45 timeslots; $R$: the set of $m$ rooms;
1: begin
2: apply neighborhood search $N_1$ to solution $I$;
3: if (move, apply neighborhood search $N_1$ to solution $I$, successful) then
4:     generate resulting potential solution $I$;
5: else
6:     apply move neighborhood search $N_2$ to solution $I$ and generate resulting potential solution $I$;
7: for ($k \leftarrow 1$ to 45) do
8:     if timeslot $t_k$ is effected by either of the move $N_1$ or $N_2$ then
9:         apply the matching algorithm to allocate rooms for events held in $t_k$;
10: delta-evaluate the result of the move;
11: return $I$;
12: end
```

The description of GALS is illustrated by the following flow chart given in Fig. 1.

4. A new hybrid algorithm (NHA)

In this section, a new hybrid algorithm (NHA) combining genetic algorithm with a local search algorithm and using events based on grouping of students for the solutions of the university course timetabling problem is described. It uses the definition of grouping given in Falkenauer (1998). Falkenauer (1998) developed the grouping genetic algorithms to solve clustering

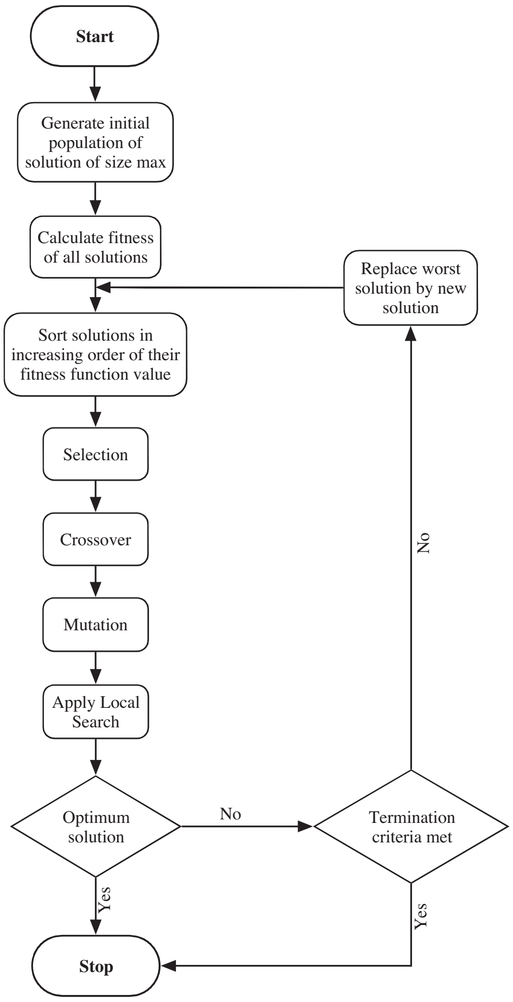

Start
Generate initial population of solution of size max
Calculate fitness of all solutions
Sort solutions in increasing order of their fitness function value
Selection
Crossover
Mutation
Apply Local Search
Optimum solution
No
Termination criteria met
Yes
Yes
No
Replace worst solution by new solution
Stop

Fig. 1. Flow chart of GALS.

problems which is based on the following definition of grouping problem. The grouping problem is defined as one where the task is to partition a set of objects U into a collection of mutually disjoint subsets u_i of U such that $\cup u_i = U$ and $u_i \cap u_j = \phi,\ i \neq j$, and according to a set of problem-specific constraints that define valid and legal groupings. In other words, the aim of a grouping problem is to group the members of a set of objects into one or more groups, where each object is in exactly one group. Grouping genetic algorithm is not a ready-for-use approach; it has to be explicitly tailored to each grouping problem. However, the structure of the problems is the same in the sense that their cost functions depend on the composition of these groups. In our approach, we have used events based on groupings of students to solve the university course timetabling problem. Thus, leading to a innovative way of using grouping approach of Falkenauer (1998). Here, set of students S is considered as a set of objects and is partitioned into a collection of k mutually disjoint groups $G_i\ (i = 1, 2, \ldots, k)$ such that $\cup G_i = S$ and $G_i \cap G_j = \phi,\ i \neq j$. The mutually disjoint groups of students are formed from the given set of students S for all the events in the given order, i.e., starting from first event, we select all those students who are taking this event and formed a group. A selected student is excluded from further selection in other groups. This procedure is repeated for all the events until all the students are not selected. Let groups $G_1, G_2, \ldots, G_k$, where k be an integer are formed. We also formed group of events $E(G_k)$ corresponding to group G_k, where $E(G_k) = \cup_j\{E(s_j) | s_j \in G_k\}$, i.e. the union of the events taken by all the students of group G_k. The number of events $|E(G_k)|$ in each group G_k is termed as its group size. Once, the students and the events groups are formed, the group size of each events groups is reduced within a given upper bound. Initially, the upper bound of the group size is taken as the total number of timeslots. The second soft constraint reduces this bound to 30. This can be further reduced to the maximum number of events per student which is taken as 20 for all the problem instances of Rossi-Doria et al. (2003). In order to reduce the group size, the students groups are divided into a number of subgroups. This is done by selecting only those events which are not common for all the students in a group. This is repeated till the group size is not within the bounds. The modified problem is now solved by GALS of previous section by considering each student group $G_i\ (i = 1, 2, \ldots, k)$ as a student, where $k \leqslant p$. The upper bound of the group size is successively reduced by one till either the optimal solution is not reached or becomes equal to the number of maximum events per student. Since, the university course timetabling problem is the problem of assignment of events into timeslots and rooms in such a way that there is no hard constraint violation and the number of soft constraint violations should be as minimum as possible. Hence, we assigned events taken by student groups, not by students, into timeslots and rooms.

Algorithm 4. New grouping algorithm

```
Input: S: the set of p students; E: the set of all n events;
  E(s_i): the set of events taken by student s_i;
Output: Groups of students F_Group with their respective
  group of events of desired group size.
1: begin
2: $X \leftarrow \max_{1 \leqslant i \leqslant p}\{|E(s_i)|\}$;
3: $MAX \leftarrow$ (maximum possible cardinality of group
  $|X \leqslant MAX \leqslant 45$);
4: $G \leftarrow S$;
5: $i \leftarrow 1$;
6: $k \leftarrow 1$;
7: $F\_Group \leftarrow \phi$;     // F_Group is a set consisting of the group
  of students, say $G_k$, as its elements.
8: while $(G \neq \phi)$ do
9:     $G_k \leftarrow \phi$;     // set $G_k$ consists the elements (students)
  of $k^{th}$ group.
10:    $E(G_k) \leftarrow \phi$;     // set $E(G_k)$ consists all the events
  taken by $k^{th}$ group.
11:    for $(j \leftarrow 1$ to $p)$ do
12:      if $((e_i \in E(s_j)) \wedge (s_j \in G))$ then
13:        $G_k \leftarrow G_k \cup \{s_j\}$;
14:        $G \leftarrow G \setminus \{s_j\}$;
15:        $E(G_k) \leftarrow E(G_k) \cup E(s_j)$;
16:    if $(G_k \neq \phi)$ then
17:        $CheckSize(G_k, E(G_k), MAX)$;     // returns the
  group of students Group with desired group size.
18:        $F\_Group \leftarrow F\_Group \cup Group$
19:        $k \leftarrow k + 1$;
20:    $i \leftarrow i + 1$;
21: end
```

Procedure *CheckSize*$(G_k, E(G_k), MAX)$

```
Taking $G_k, E(G_k)$ and $MAX$ as arguments, it returns the group of
  students $Group$ within required group size
1: begin
2: $Group \leftarrow \phi$;    // $Group$ is a set consisting of the students
  subgroup of $G_k$ within required group size.
3: if $(|E(G_k)| \leqslant MAX)$ then
4:     $Group \leftarrow \{G_k\}$;
5:     return $Group$;
6: else
7:     repeat
8:         $ReduceGroupSize(G_k, E(G_k))$;    // returns a
  subgroup $G_k^j$ of $G_k$ with reduced group size.
9:         $E(G_k^j) \leftarrow \bigcup_y \{E(s_y|s_y \in G_k^j\}$
10:        if $(|E(G_k^j)| \leqslant MAX)$ then
11:            $Group \leftarrow Group \bigcup \{G_k^j\}$;
12:    until $(|E(G_k^j)| > MAX$ for any $G_k^j \subseteq G_k | \bigcup_j (G_k^j) = G_k)$;
13:    return $Group$;
14: end
```

Procedure *ReduceGroupSize*$(G_K, E(G_k))$

```
Taking $G_K$ and $E(G_k)$ as arguments, it returns a number of
  partitioned subgroups $G_k^j$ of $G_k$
1: begin
2: $i \leftarrow 1$;
3: $j \leftarrow 1$;
4: while $((e_i \notin \cap\{E(s_y)|s_y \in G_k\}) \wedge (G_k \neq \phi))$ do
5:     $G_k^j \leftarrow \phi$;
6:     $E(G_k^j) \leftarrow \phi$;
7:     for $((l \leftarrow 1top) \wedge (e_i \in E(s_l)) \wedge (s_l \in G_k))$ do
8:         $G_k^j \leftarrow G_k^j \cup \{s_l\}$;
9:         $G_k \leftarrow G_k \setminus \{s_l\}$;
10:        $E(G_k^j) \leftarrow E(G_k^j) \cup E(s_l)$;
11:    if $(G_k^j \neq \phi)$ then
12:        return $G_k^j$;
13:        $j \leftarrow j+1$;
14:    $i \leftarrow i+1$;
15: end
```

After constructing disjoint groups of students, room-features and event-features are modified in order to satisfy the constraint requiring all the events are to be assigned to rooms having adequate seating capacity and all the required features. There are total m rooms and initially there are q features per room. If there are multiple rooms with the same seating capacity and there are total y distinct room capacities, where $y \leqslant m$. Then a total of $(q+y)$ features will be there for each one of the room. Also, the rooms with the same seating capacity will have the same set of new features in addition to the given q features per room. The following procedure will give us a total of $(q+y)$ features for each one of the m rooms of the new problem instance after construction of student groups.

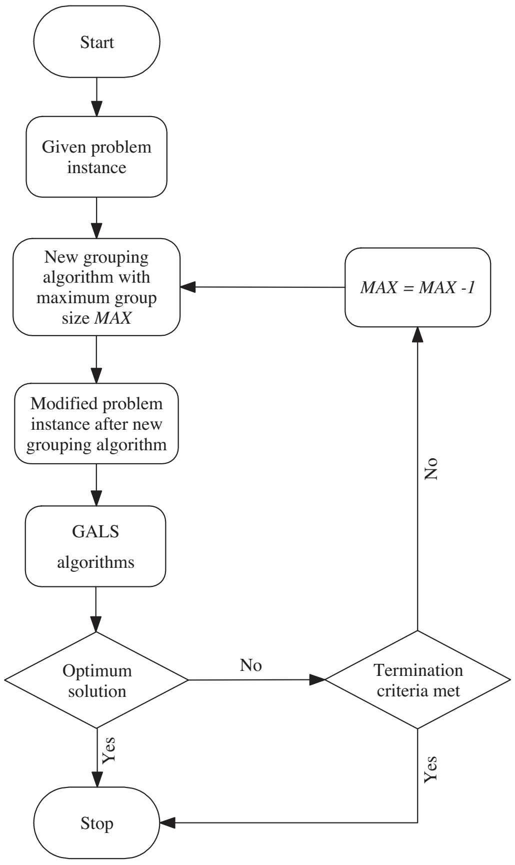


Fig. 2. Flow chart of NHA.

Procedure Room-features for the new problem instance

```
Input: All $m$ rooms $r_1, r_2, \ldots, r_m$ with their seating capacity
  and room-features;
Output: Room-features for the new problem instance.
1: begin
2: for $(i \leftarrow 1$ to $m)$ do
3:     $seat\_cap[i] \leftarrow$ seating capacity of room $r_i$;
4: sort all room seating capacities (distinct) in their increasing
  order;
5: $cap[1] \leftarrow$ least room seating capacity;
6: $cap[y] \leftarrow$ highest room seating capacity;    // Assume,
  there are total $y$ distinct room capacities, where $y \leqslant m$;
7: $rf[i][q] \leftarrow$ all $q$ features for room $r_i$;
8: for $(i \leftarrow 1$ to $m)$ do    // for $m$ rooms;
9:     give all the room-features $rf[i][q]$ for room $r_i$;
10:    for $(j \leftarrow 1toy)$ do
11:        if $(seat\_cap[i] \leqslant cap[j])$ then
12:            $rf[i][q+j] \leftarrow 1$;
13:        else
14:            $rf[i][q+j] \leftarrow 0$;
15: end
```

Similarly, we can modify the event-features for the new problem instance after construction of student groups by the following procedure. Here, the number of students enrolled in a particular event is also required.

Procedure Event-features for the new problem instance

```
Input: All $m$ rooms $r_1, r_2, \ldots, r_m$ with their seating capacity;
    event-features for all $n$ events $e_1, e_2, \ldots, e_n$; number of
    students in each event;
Output: Event-features for the new problem instance.
1: begin
2: $cap[0] \leftarrow 0$;
3: sort all room seating capacities (distinct) in their increasing
    order;
4: $cap[1] \leftarrow$ least room seating capacity;
5: $cap[y] \leftarrow$ highest room seating capacity;    // Assume,
    there are total $y$ distinct room capacities, where $y \leqslant m$;
6: $se[i] \leftarrow$ number of students taking event $e_i$;
7: $ef[i][q] \leftarrow$ all $q$ features for event $e_i$;
8: for ($i \leftarrow 1$ to $n$) do    // for $n$ events;
9:      give all the event-features $ef[i][q]$ for event $e_i$;
10:     for ($j \leftarrow 1$ to $y$) do
11:         if ($cap[j-1] < se[i] \leqslant cap[j]$) then
12:             $ef[i][q+j] \leftarrow 1$;
13:         else
14:             $ef[i][q+j] \leftarrow 0$;
15: end
```

The description of NHA is illustrated by the following flow chart given in Fig. 2.

The following algorithm describes the NHA.

Algorithm 5. New hybrid algorithm (NHA)

```
Input: The given problem instance;
Output: Either an optimal solution or no solution.
1: begin
2: take initial upper bound $MAX$ of group size;
3: apply the new grouping algorithm with the upper bound
    $MAX$;    // given in Algorithm 4
4: modify the new problem instance according to new student
    groups;
5: apply the GALS algorithms    // given in Algorithms 1–3
6: if (termination criteria met) then
7:      STOP the algorithm;
8: else
9:      $MAX \leftarrow MAX - 1$;
10:     goto 3;
11: end
```

Here, the termination criteria used is either time limit, or number of iterations, or optimal solution achieved with zero fitness function value.

5. Experimental results

In this section, the performance of the proposed NHA combining GALS and using events based on grouping of students is investigated by comparing it with GALS and a number of existing algorithms for solving university course timetabling problem. The performance measure depends on the parameters and operators such as population size, crossover and mutation used and is given in terms of fitness function values. All algorithms are programmed in C++ on GNU compiler GCC version 4.5.2 running on a 3.10 GHz PC. Datasets of problems instances are taken from the benchmark problems proposed by Metaheuristic Network[1] in Rossi-Doria et al. (2003) and the first international timetabling competition (ITC2002).[2] Although these problem instances lack many of the real-world problem constraints and issues, they allow the comparison of NHA with GALS and other approaches. The first set of problem instances involves the scheduling of 100–400 events into a timetable with 45 timeslots corresponding to 5 days of 9 h each whilst satisfying room features and room capacity constraints. The problem instances are divided into three small, medium and large categories. The parameters used consist of number of events, rooms, students, features, approximate features for each room, percentage of features used in each sized category, maximum number of events for each student and maximum number of students in each event are listed in Table 3.

Table 3
Parameter values for the problem instances of Rossi-Doria et al. (2003).

Class	Small	Medium	Large
Number of events	100	400	400
Number of rooms	5	10	10
Number of students	80	200	400
Number of features	5	5	10
Approximate features per room	3	3	5
Percentage feature use	70	80	90
Maximum events per student	20	20	20
Maximum students per event	20	50	100

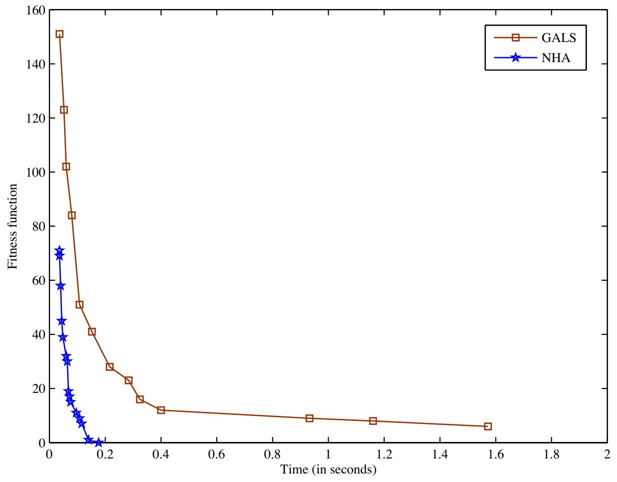


Fig. 3. Fitness function values versus time for small01 by GALS and NHA.

The first experiment is carried out on 5 instances each of small and medium sized and 2 instances of large sized standard benchmark problems. The second experiment is performed on twenty datasets taken from ITC2002. The parameters used are the population size, the mutation probability and the crossover probability taken as 10, 0.5 and 0.8 respectively. The value of the constant γ is taken as 10^6 in the fitness function. In all problem instances, the group size of events is taken slightly larger than the maximum number of events per students.

5.1. Rossi-Doria et al. datasets

In the first experiment, the termination criteria is either the maximum number of iterations taken as 200, 1000 and 2000 or the time limit of 2 s, 10 s and 15 s respectively for the small, medium and large sized problem instances. The experiment can also terminate when the value of fitness function becomes equal to zero.

[1] http://www.metaheuristics.org/.

[2] http://www.idsia.ch/Files/ttcomp2002.

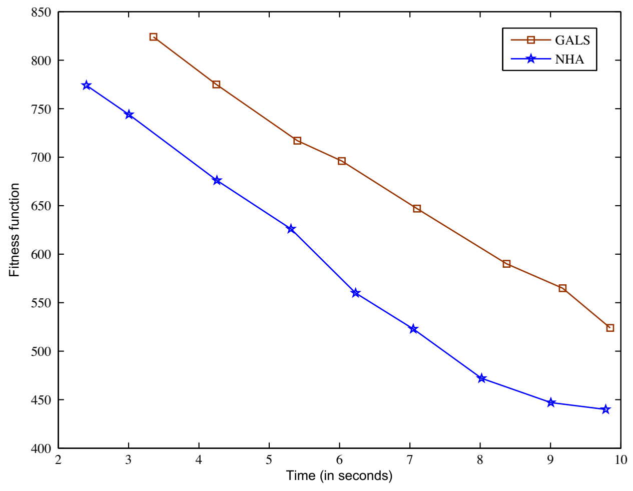


Fig. 4. Fitness function values versus time for medium01 by GALS and NHA.

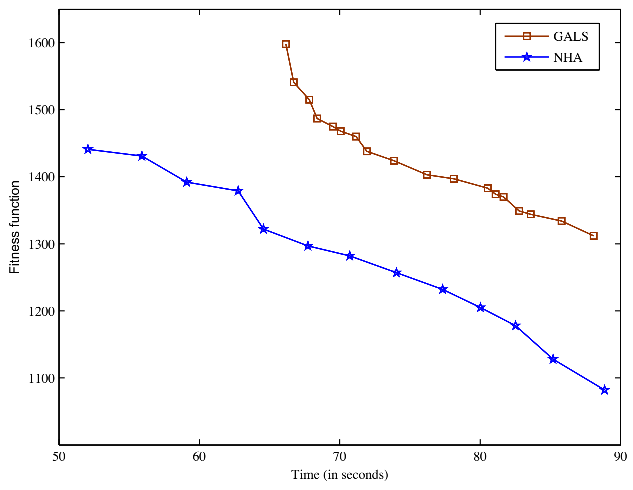


Fig. 5. Fitness function values versus time for hard02 by GALS and NHA.

Each instance of each problem is run for 20 times and the smallest fitness function value among them is taken as the best value for the solution of the problem instance. The fitness function values versus time taken by GALS and NHA are compared. This comparison is shown in Figs. 3 and 4 for small01 and medium01 instances.

It is evident that our algorithm NHA in comparison to GALS gives zero fitness function values for each instances of the small class of problems. It takes the minimum fitness function values for each instances of the medium class of problems. No feasible solutions were obtained for all instances of large class of problems with the considered parameters. If we define the distance to feasibility as the total number of hard constraint violations in an unfeasible solution, it is very less for NHA in comparison to GALS. To get the optimal solutions for all instances of large class of problems, the time limits as well as the number of iterations are increased incrementally. It is observed that for the time limit and number of iterations of 90 s and 8000, the optimal solution of hard02 problem instance is obtained. The fitness function values versus time taken by GALS and NHA are compared by the graph given in Fig. 5 for the hard02 problem instance.

However, no optimal solution of hard01 problem is obtained for all its instances. In order to get suitable combination of crossover probability (cr) and mutation probability (mr) for medium03 problem instance, the Table 4 lists the fitness function values for various combination of cr and mr for medium03 problem instance. The comparison of fitness function values versus time for different combinations of cr and mr are plotted in Fig. 6. It is observed that the optimal fitness function value is obtained for $cr = 0.8$ and $mr = 0.5$. It is further found that this combination also gives optimal solutions for all other problems instances.

The fitness function values for 12 problem instances of first experiment obtained by GALS and NHA are listed in Table 5. Next, our NHA compared with some other algorithms used in the literature. The algorithms compared and the conditions under which there results are reported is described as follows.

NHA The results were reported out of twenty runs with 100,000 evaluations per run. The small problems instances are not considered since global optima for them was already obtained. The time limits for each run of medium and large instances are restricted by 900 and 9000 s respectively.

A1 A genetic algorithm with a repair function and local search proposed by Abdullah and Turabieh (2008). The results were reported out of five runs.

A2 A composite neighborhood structure with randomized iterative improvement algorithm proposed by Abdullah, Burke, and McCollum (2007b). The results were reported out of five runs with each run lasting for 200,000 evaluations.

A3 A tabu search with graph-based hyperheuristic proposed by Burke et al. (2007). The results were reported out of five runs with 12,000 evaluations per run for small instances, 1200 evaluations per run for medium instances, and 5400 evaluations per run for the large instance.

A4 A variable neighborhood search based on a random descent local search with Monte-Carlo acceptance criterion proposed by Abdullah, Burke, and Mccollum (2005). The results were reported out of five runs with each run lasting for 200,000 evaluations.

A5 A hybrid evolutionary algorithm consisting of an evolutionary algorithm using a light mutation operator followed by a randomized iterative improvement algorithm proposed by Abdullah et al. (2007a). The results were reported out of five runs with 200,000 evaluations per run.

A6 An extended great deluge algorithm proposed by McMullan (2007). The results were reported out of ten runs with 200,000 evaluations per run. The time taken to achieve the best solutions for small instances was ranged between 15 to 60 s.

A7 A modified great deluge algorithm by using a nonlinear decay of water level proposed by Landa-Silva and Obit (2008). They successfully improved the performance of the great deluge algorithm on medium instances. The results were reported out of ten runs with each run lasting for 3600, 4700 and 6700 s respectively for small, medium and large instances.

Table 4
Fitness function values of medium03 instance corresponding to different crossover and mutation probabilities by NHA.

Crossover rate	0.0	0.0	0.2	0.2	0.2	0.4	0.4	0.6	0.6	0.8	0.8	0.8	1.0	1.0
Mutation rate	0.5	1.0	0.2	0.5	1.0	0.5	1.0	0.5	1.0	0.5	0.8	1.0	0.5	1.0
Fitness function	568	637	599	574	584	574	629	608	605	535	567	673	602	634

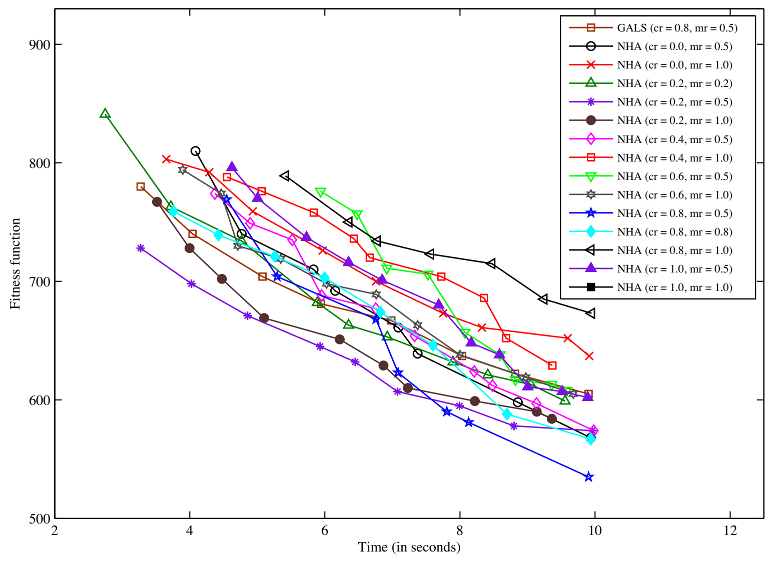


Fig. 6. Fitness function values versus time for medium03 by GALS and NHA for different *cr* and *mr*.

Table 5
Fitness function values for problem instances of Rossi-Doria et al. (2003) by GALS and NHA.

Method	small01	small02	small03	small04	small05	medium01	medium02	medium03	medium04	medium05	hard01	hard02
GALS	6	9	2	4	0 (0.704 s)	524	441	605	441	599	∞	1312
NHA	0 (0.176 s)	0 (0.112 s)	0 (0.520 s)	0 (0.548 s)	0 (0.080 s)	440	324	535	362	478	∞	1082

Table 6
Comparison of fitness function values for problem instances of Rossi-Doria et al. (2003).

Instance	NHA				A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13
	Best	Avg.	Worst	Std. dev.													
small01	0	0	0	0	2	0	6	0	0	0	3	8	1	10	0	0	0
small02	0	0	0	0	4	0	7	0	0	0	4	11	3	9	3	0	0
small03	0	0	0	0	2	0	3	0	0	0	6	8	1	7	0	0	0
small04	0	0	0	0	0	0	3	0	0	0	6	7	1	17	0	0	0
small05	0	0	0	0	4	0	4	0	0	0	0	5	0	7	0	0	0
medium01	106	131.45	147	12.96	254	242	372	317	221	80	140	199	195	243	280	139	175
medium02	107	126.70	140	9.40	258	161	419	313	147	105	130	202.5	184	325	188	92	197
medium03	132	151	185	17.11	251	265	359	357	246	139	189	77.5% Inf.	248	249	249	122	216
medium04	72	92.8	121	16.23	321	181	348	247	165	88	112	177.5	164.5	285	247	98	149
medium05	107	124.8	140	10.97	276	151	171	292	130	88	141	100% Inf.	219.5	132	232	116	190
hard01	505	555	655	36.59	1026	100% Inf.	1068	100% Inf.	529	730	876	100% Inf.	851.5	1138	100% Inf.	615	912
hard02	486	523.65	598	38.67	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

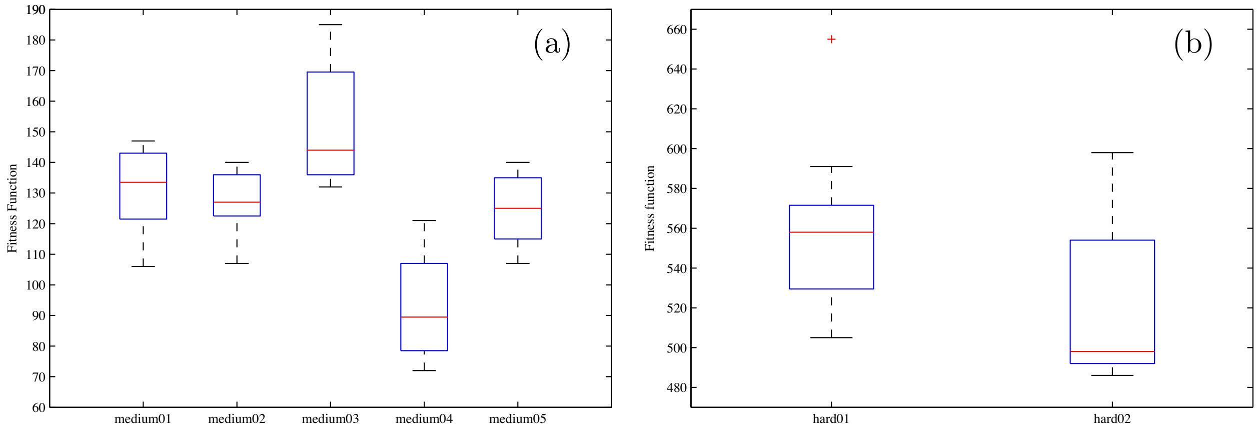
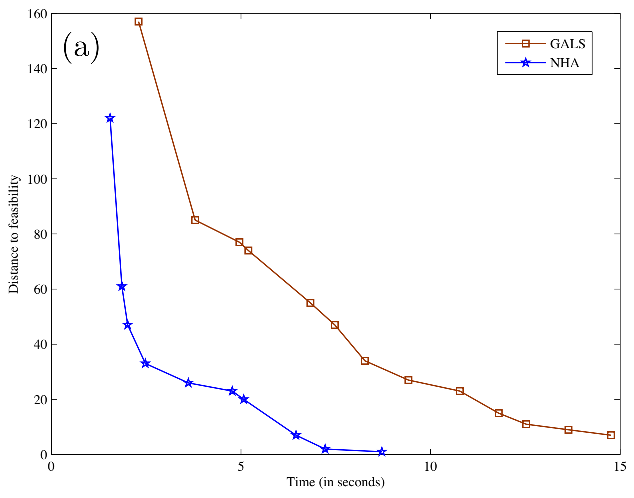


Fig. 7. Box and whisker plot of results obtained by NHA (a) for medium problem instances (b) for large problem instances.

Table 7
Description of problem instances of ITC2002

Instance	No. of events	No. of rooms	No. of features	No. of students	Avg. events/student	Avg. students/event	Avg. features/event	Avg. features/room
com01	400	10	10	200	17.755	8.8775	4.435	5.5
com02	400	10	10	200	17.23	8.615	3.0775	4.4
com03	400	10	10	200	17.705	8.8525	2.26	2.5
com04	400	10	5	300	17.433	13.075	2.19	2.7
com05	350	10	10	300	17.777	15.237	4.7486	5.4
com06	350	10	5	300	17.773	15.234	2.6743	3.3
com07	350	10	5	350	17.477	17.477	2.5086	3.2
com08	400	10	5	250	17.584	10.99	2.4825	3.1
com09	440	11	6	220	17.359	8.6795	2.2909	2.8182
com10	400	10	5	200	17.78	8.89	2.6	2.6
com11	400	10	6	220	17.414	9.5775	3.135	3.5
com12	400	10	5	200	17.575	8.7875	2.005	2.1
com13	100	10	6	250	17.688	11.055	2.175	2.4
com14	350	10	5	350	17.417	17.417	1.5629	1.9
com15	350	10	10	300	17.58	15.0686	2.6629	2.9
com16	440	11	6	220	17.455	8.8772	1.8159	2.2727
com17	350	10	10	300	17.67	15.1457	4.3571	4.8
com18	400	10	10	200	17.56	8.78	3.7375	4.7
com19	400	10	5	300	17.707	13.28	2.4125	3.0
com20	350	10	5	300	17.487	14.9886	2.5943	2.9



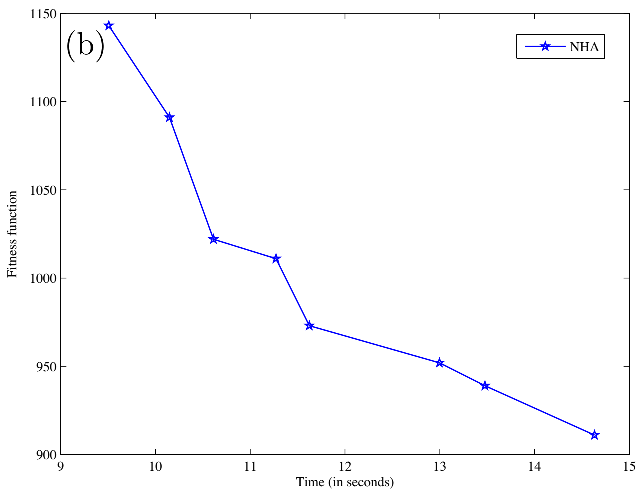


Fig. 8. (a) Distance to feasibility versus time for competition05 by GALS and NHA. (b) Fitness function values versus time for competition05 by NHA.

A8 A random restart local search method proposed by Socha, Knowles, and Sampels (2002). The results were reported out of 50 runs with each run lasting for 90 s for small instances, 40 runs with each run lasting for 900 s for medium instances and 10 runs with each run lasting for 9000 s for the large instance.

A9 Ant algorithm proposed by Socha et al. (2002). The results were reported out of 50 runs with each run lasting for 90 s for small instances, 40 runs with each run lasting for 900 s for medium instances and 10 runs with each run lasting for 9000 s for the large instance.

A10 A fuzzy algorithm proposed by Asmuni, Burke, and Garibaldi (2005). The results were reported out of one run for each problem instance and the stopping criterion for each run on a problem instance was not clearly mentioned in the paper.

A11 A genetic algorithm with a local search method proposed by Rossi-Doria et al. (2003). The results were reported out of 50 runs with each run lasting for 90, 900 and 9000 s respectively for small, medium and large instances.

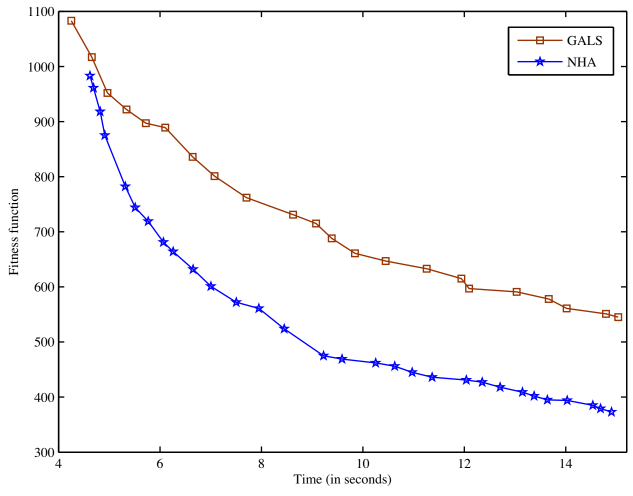


Fig. 9. Fitness function values versus time for competition08 by GALS and NHA.

Table 8
Fitness function values for problem instances of ITC2002 by GALS and NHA.

Method	com01	com02	com03	com04	com05	com06	com07	com08	com09	com10	com11	com12	com13	com14	com15	com16	com17	com18	com19	com20
GALS	658	474	531	1004	∞	651	702	545	550	873	594	∞	827	1151	743	495	∞	456	825	572
NHA	544	429	493	832	911	437	536	373	491	592	508	661	776	696	509	475	687	431	718	511

A12 An extended guided search genetic algorithm proposed by Yang and Jat (2011). The results were reported out of 50 runs with each run lasting for 90, 900 and 9000 s respectively for small, medium and large instances.
A13 A hybrid metaheuristic approach proposed by Abdullah et al. (2012). The results were reported out of five runs with 100,000 evaluations per run. Also, the time limit for each run is restricted by 90, 7200 and 21,600 s respectively for small, medium and large instances.

Table 6 shows the comparison of the best fitness function values obtained. Other algorithms have considered only the first 11 problem instances. Here, hard01 problem is corresponding to large problem whereas none of them used hard02 problem. Hence, the fitness function value of hard02 problem instance is shown as *not available (NA)* in all the compared methods. The term $x\%$ Inf in Table 6 indicates a percentage of runs that failed to produce feasible solutions. It is observed that our algorithm NHA is capable of finding feasible solutions for all the problem instances. The optimal solutions of all small sized problem instances are obtained within a second by our method. In fact, for small05 problem instance we get optimal solution within 0.08 s. Also, our algorithm NHA outperforms all the other compared algorithms in medium04 and hard01 (or large instance of Socha et al. (2002)) instances. Results obtained for all medium and large instances from all the 20 runs are summarized by boxplot in Fig. 7(a) and (b). In the boxplot, a box shows the range between 25% and 75% quantile of the data. The median is indicated by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box. Outliers are indicated as plus.

5.2. ITC2002 datasets

The second experiment is performed based on ITC2002 datasets. The detailed description of these datasets are given in Table 7. Here, each of the 20 problem instances is run for 20 times by both GALS and NHA algorithms and the best results obtained by them are compared. The time limit and the number of iterations are fixed by 15 s and 1500 per generation. The graphs of two randomly selected instances competition05 and competition08 are given in Figs. 8 and 9. Fig. 8(a) and (b) are respectively showing distance to feasibility versus time and fitness function values versus time for competition05 instance. It can be seen that GALS does not give any feasible solution, whereas, NHA gives quite impressive result with the same parameters. Similarly, from Fig. 9 showing fitness function values versus time for competition08 instance, it is observed that NHA gives significant improvement compare to GALS.

The fitness function values for all the twenty problem instances of ITC2002 obtained by GALS and NHA are given in Table 8. It is observed that NHA shows improved performance in comparison to GALS.

In GALS, the events taken by students are assigned to timeslots and rooms. On the other hand, NHA using modified grouping problem takes groups of students for this purpose. Thus, results in the reduction of selection time taken by the selection procedure in NHA compared to GALS.

6. Conclusions

A new hybrid algorithm, NHA, combining GALS and using events based on groupings of students is described to solve the university course timetabling problem. The experiments are carried out on datasets taken from Rossi-Doria et al. (2003) and ITC2002. The performance measure depending on the parameters and operators such as population size, crossover and mutation is given in terms of fitness function values. It is shown that in all small sized problem instances, NHA gives optimal solutions with zero fitness function values within 0.6 s. In all other datasets of benchmark problems, NHA outperforms GALS. In some problem instances, it is observed that GALS is not able to provide any feasible solution whereas NHA is giving quite significant solutions with the same set of parameters. For medium04 and hard01 problem instances, NHA is outperforming all the other state-of-the-art algorithms considered.

Acknowledgments

The authors thank the anonymous referees for their valuable comments which have improved the presentation of the paper. First author also acknowledges financial support from CSIR through CSIR (NET) with application number 09/081(1093)/2010-EMR-I.

References

Abdullah, S., Burke, E. K., & McCollum, B. (2007a). A hybrid evolutionary approach to the university course timetabling problem. In *IEEE congress on evolutionary computation, 2007, CEC 2007* (pp. 1764–1768). IEEE.
Abdullah, S., Burke, E. K., & McCollum, B. (2007b). Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In *Metaheuristics* (pp. 153–169). Springer.
Abdullah, S., Burke, E.K., & Mccollum, B. (2005). An investigation of variable neighbourhood search for university course timetabling. In *The second multidisciplinary international conference on scheduling: theory and applications (MISTA)* (pp. 413–427).
Abdullah, S., Turabieh, H., McCollum, B., & Burke, E.K. (2009). An investigation of a genetic algorithm and sequential local search approach for curriculum-based course timetabling problems. In *Proc. multidisciplinary international conference on scheduling: Theory and applications (MISTA 2009), Dublin, Ireland* (pp. 727–731).
Abdullah, S., Shaker, K., McCollum, B., & McMullan, P. (2010). Dual sequence simulated annealing with round-robin approach for university course timetabling. In *Evolutionary computation in combinatorial optimization* (pp. 1–10). Springer.
Abdullah, S., & Turabieh, H. (2008). Generating university course timetable using genetic algorithms and local search. *Third international conference on convergence and hybrid information technology, 2008, ICCIT'08* (Vol. 1, pp. 254–260). IEEE.
Abdullah, S., Turabieh, H., McCollum, B., & McMullan, P. (2012). A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics, 18*(1), 1–23.
Asmuni, H., Burke, E. K., & Garibaldi, J. M. (2005). Fuzzy multiple heuristic ordering for course timetabling. In *Proceeding of the fifth United Kingdom workshop on computational intelligence* (pp. 302–309). London: Citeseer.
Banks, D., Van Beek, P., & Meisels, A. (1998). A heuristic incremental modeling approach to course timetabling. In *Advances in artificial intelligence* (pp. 16–29). Springer.
Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research, 176*(1), 177–192.

Cambazard, H., Hebrard, E., O'Sullivan, B., & Papadopoulos, A. (2012). Local search and constraint programming for the post enrolment-based course timetabling problem. *Annals of Operations Research, 194*(1), 111–135.

Carter, M. W. (1986). Or practice-a survey of practical applications of examination timetabling algorithms. *Operations Research, 34*(2), 193–202.

De Causmaecker, P., Demeester, P., & Vanden Berghe, G. (2009). A decomposed metaheuristic approach for a real-world university timetabling problem. *European Journal of Operational Research, 195*(1), 307–318.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley & Sons, Inc.

John, H. (1992). *Adaptation in natural and artificial systems*. Cambridge, MA: MIT Press.

Landa-Silva, D., & Obit, J. H. (2008). Great deluge with non-linear decay rate for solving course timetabling problems. *Fourth international IEEE conference intelligent systems, 2008, IS'08* (Vol. 1, pp. 8–11). IEEE.

Lewis, R. (2008). A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum, 30*(1), 167–190.

Lewis, R., & Paechter, B. (2005). Application of the grouping genetic algorithm to university course timetabling. In *Evolutionary computation in combinatorial optimization* (pp. 144–153). Springer.

Lü, Z., & Hao, J.-K. (2010). Adaptive tabu search for course timetabling. *European Journal of Operational Research, 200*(1), 235–244.

McMullan, P. (2007). An extended implementation of the great deluge algorithm for course timetabling. In *Computational science – ICCS 2007* (pp. 538–545). Springer.

Moscato, P., Cotta, C., & Mendes, A. (2004). Memetic algorithms. In *New optimization techniques in engineering* (pp. 53–85). Springer.

Moscato, P. et al. (1989). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program*. C3P Report 826.

Nothegger, C., Mayer, A., Chwatal, A., & Raidl, G. R. (2012). Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research, 194*(1), 325–339.

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial optimization: Algorithms and complexity*. Courier Dover Publications.

Petrovic, S., & Burke, E.K. (2004). University timetabling. *Handbook of scheduling: Algorithms, models, and performance analysis* (Vol. 45, p. 1–23).

Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., et al. (2003). A comparison of the performance of different metaheuristics on the timetabling problem. *Practice and theory of automated timetabling IV* (Vol. 2740, pp. 329–351). Springer.

Socha, K., Knowles, J., & Sampels, M. (2002). A max-min ant system for the university course timetabling problem. In *Ant algorithms* (pp. 1–13). Springer.

Wijaya, T., & Manurung, R. (2009). Solving university timetabling as a constraint satisfaction problem with genetic algorithm. In *Proceedings of the international conference on advanced computer science and information systems (ICACSIS 2009), Depok*.

Wilke, P., & Ostler, J. (2008). Solving the school time tabling problem using tabu search, simulated annealing, genetic and branch & bound algorithms. In *Seventh international conference on the practice and theory of automated timetabling, PATAT2008*.

Yang, S., & Jat, S. N. (2011). Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 41*(1), 93–106.