**Journal of Scheduling manuscript No.**
(will be inserted by the editor)

# A Hybrid Genetic Algorithm and Tabu Search Approach for Post Enrolment Course Timetabling

**Sadaf Naseem Jat · Shengxiang Yang**

Received: 23 November, 2009 / Revised: 06 July, 2010 / Accepted: 22 September, 2010

**Abstract** The post enrolment course timetabling problem (PECTP) is one type of university course timetabling problems, in which a set of events has to be scheduled in time slots and located in suitable rooms according to the student enrolment data. The PECTP is an NP-hard combinatorial optimization problem and hence is very difficult to solve to optimality. This paper proposes a hybrid approach to solve the PECTP in two phases. In the first phase, a guided search genetic algorithm is applied to solve the PECTP. This guided search genetic algorithm, integrates a guided search strategy and some local search techniques, where the guided search strategy uses a data structure that stores useful information extracted from previous good individuals to guide the generation of offspring into the population and the local search techniques are used to improve the quality of individuals. In the second phase, a tabu search heuristic is further used on the best solution obtained by the first phase to improve the optimality of the solution if possible. The proposed hybrid approach is tested on a set of benchmark PECTPs taken from the international timetabling competition in comparison with a set of state-of-the-art methods from the literature. The experimental results show that the proposed hybrid approach is able to produce promising results for the test PECTPs.

Sadaf Naseem Jat
Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
E-mail: snj2@mcs.le.ac.uk

Shengxiang Yang
Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, United Kingdom
E-mail: shengxiang.yang@brunel.ac.uk

**Keywords** Post enrolment course timetabling problem · University course timetabling problem · Guided search genetic algorithm · Local search · Tabu search

## 1 Introduction

Timetabling is one of the common scheduling problems, which can be described as the allocation of resources for tasks under predefined constraints so that it maximizes the possibility of allocation or minimizes the violation of constraints [57]. Timetabling problems are often complicated by the details of a particular timetabling task. A general algorithm approach to a problem may turn out to be incapable, because of certain special constraints required in a particular instance of that problem. Typical timetabling areas are educational timetabling [1,60], sports timetabling [37], transport timetabling [11], employee timetabling [14], and so on. Educational timetabling can be divided into school timetabling, exam timetabling, and course timetabling.

In the university course timetabling problem (UCTP), events (subjects, courses) have to be allocated into a number of time slots and rooms while satisfying various constraints. It is very difficult to find a general and effective solver for the UCTP due to the diversity of the problem, variance of constraints, and particular requirements from university to university according to the characteristics. There is no known deterministic polynomial time algorithm for the UCTP. That is, the UCTP is an NP-hard combinatorial optimization problem [26]. The UCTP can be further divided into two categories as proposed by the organizers of the 2007 International Timetabling Competition (ITC-2007)[1]: the

[1] For more details, see the official competition website at http://www.cs.qub.ac.uk/itc2007

post enrolment course timetabling problem (PECTP) [41] and the curriculum base course timetabling problem (CBCTP). The main difference between them lies in that in the CBCTP, courses are scheduled according to the curricula published by the university while in the PECTP, courses are scheduled according to the student enrolment data [44]. In this paper, we deal with the PECTP, which is similar to the 2003 timetabling competition UCTP problem except two new hard constraints. According to Lewis [41], these two new hard constraints make it very difficult to find the feasible solution in the search space and make the PECTP much similar to the real-world timetabling problem.

The research on timetabling has a long history of over forty years, starting with Gotlieb in 1963 [32]. Over the past forty years, researchers have proposed various timetabling approaches by using constraint-based methods, population-based approaches (e.g., genetic algorithms (GAs), ant colony optimization (ACO), and memetic algorithms), meta-heuristic methods (e.g., tabu search (TS), simulated annealing, and great deluge), variable neighbourhood search, and hybrid and hyper-heuristic approaches, etc. A comprehensive review on timetabling can be found in [17,19,39,41, 50,54,4]. Several researchers have used GAs to solve course timetabling problems [5,25,40,48]. Rossi-Doria *et al.* [51] compared different meta-heuristics to solve the UCTP. They concluded that conventional GAs do not give good results among a number of approaches developed for the UCTP. Hence, conventional GAs need to be enhanced to solve the UCTP.

Recently, a guided search genetic algorithm, denoted *GSGA*, has been proposed for solving the UCTP [35]. The GSGA consists of a guided search strategy and a local search (LS) technique. One important concept of GAs is the notion of population. Unlike traditional search methods, GAs rely on a population of candidate solutions [31,53]. In GSGA, a guided search strategy is used to generate offspring into the population-based on an extra data structure. This data structure is constructed from the best individuals from the population and hence stores useful information that can be used to guide the generation of good offspring into the next population. The main advantage of this data structure is that it provides diversity by maintaining part of good solutions, which otherwise would have been lost in the selection process. In GSGA [35], a LS technique is also used to improve the quality of individuals through searching in three kinds of neighbourhood structures. In [35], GSGA has shown some promising results based on some preliminary experiments.

In this paper, a hybrid approach is proposed based on the GSGA [35] and a TS heuristic to solve the PECTP. The proposed hybrid approach works in two phases. In the first phase, the GSGA developed in [35] for the UCTP is adapted and applied to solve the PECTP. In addition to the original LS strategy used in GSGA [35], some new neighbourhood structures and relevant LS strategies are integrated into GSGA for the PECTP. Given that finding a feasible solution for the PECTP can be a challenging task [41], the hybrid approach employs a second phase, where a TS heuristic is further used on the best solution obtained by GSGA in the first phase to improve the optimality of the solution if possible. In order to investigate the effect of parameters on the performance of the hybrid approach for the PECTP, the sensitivity analysis of key parameters of GSGA is carried out by systematical experiments based on a set of ITC-2007 benchmark PECTPs. In order to test the performance of the proposed hybrid approach, experiments are also carried out to compare it with other variants of GAs and TS and a set of state-of-the-art methods from the literature on the benchmark PECTP instances. The experimental results show that the proposed hybrid approach is better than or comparable to all other tested methods.

The rest of this paper is organized as follows. The next section briefly describes the related work. Section 3 describes the PECTP in details. Section 4 presents the proposed hybrid approach for the PECTP. Experimental results of comparing the proposed hybrid approach and other algorithms from the literature are reported and discussed in Section 5. Finally, Section 6 concludes this paper with discussions on the future work.

## 2 Related work

Many algorithms have been introduced to solve timetabling problems. The earliest algorithms are based on graph colouring heuristics methods. These algorithms show a great efficiency in small instances of timetabling problems, but are not efficient in large instances [33,38,60]. Later, stochastic search methods, such as GAs, simulated annealing, and TS, etc., were introduced to solve timetabling problems.

Generally speaking, there are two types of meta-heuristic algorithms [10]: local area based algorithms and population-based algorithms. Each type has some advantages and disadvantages. A local area based algorithm starts from an initial state/solution and tries to find a better solution in the space of candidate solutions until a stopping criterion is met [28]. Local area based algorithms differ from each other in the method that is used to find a neighborhood solution in the search space and the criterion to stop the search. Local area based algorithms include simulated annealing

[6,58], very large neighborhood search [1,2], TS [9,44], and many more. Usually, local area based algorithms more focus on exploitation rather than exploration [10, 21]. They usually work in a non-systematic way that may lead to find a solution in one direction without performing a wider scan of the search space [10,28]. Population-based algorithms start with many solutions and refine them to obtain optimal solution(s) in the whole search space and, hence, are global area based algorithms. Population-based algorithms that are commonly used to tackle timetabling problems include evolutionary algorithms (EAs) [22], ant colony algorithms [56], and artificial immune systems [45], etc.

In recent years, GAs have been used to solve the UCTP. Erben and Keppler [25] proposed a GA for weekly course timetables. They used a problem-specific chromosome representation and knowledge-augmented genetic operators. These operators can avoid building illegal timetables. Their approach was tested on real data. Sigl *et al.* [55] used 3D cubes, corresponding to room, day, and time slot, to model the timetable. They enhanced the performance of GAs by using modified genetic operators and tested their algorithm on small and large problem instances. Generally speaking, when a simple GA is employed, it may generate illegal timetables that have duplicate and/or missing events. Usually, the quality of a solution produced by population-based algorithms is not better than local area based algorithms. There are many reasons behind this, such as the premature convergence problem. In that situation , the solving procedure is trapped in the suboptimal state and is unable to generate offspring that are superior to their parents. The main reason for this premature convergence problem in population-based algorithms is that they are more concerned with exploration than exploitation [10]. Population-based algorithms perform search in the whole search space without strictly focusing on the good part of an individual within a population, which may lead to the lose of useful information in a good individual [8]. Population-based algorithms have another drawback of requiring more time [21]. However, GAs have several advantages when compared with other optimization techniques [48]. For example, GAs perform multiple directional search using a set of candidate solutions [29].

Various combinations of local area and global area based algorithms have been reported to solve timetabling problems in the literature [15,51,57,36,3]. In addition, it is also being increasingly realized that EAs without incorporation of problem-specific knowledge do not perform as well as mathematical programming based algorithms on certain classes of timetabling problems [13]. In this paper, we aim to combine the good properties of local and global area based algorithms to solve the PECTP. We try to make a balance between the exploration ability (global improvement) of GAs and the exploitation ability (local improvement) of LS operators (to be described in Section 4.1.5) and TS. In addition, an external memory data structure (to be described in Section 4.1.2) is used to store parts of previous good solutions. These stored parts are reintroduced into offspring in order to enable the proposed GAs to quickly locate the optimum of a PECTP.

## 3 Post enrolment course timetabling

### 3.1 Problem description

According to Carter and Laporte [19], the UCTP is a multi-dimensional assignment problem, in which students and teachers (or faculty members) are assigned to courses, course sections, or classes and events (individual meetings between students and teachers) are assigned to classrooms and time slots. The PECTP consists of assigning university courses to time slots and rooms according to student enrolment data, where each assignment has to fulfil various constraints. Among these constraints, some are hard constraints and some are soft constraints. Hard constraints must not be violated under any circumstances, e.g., a student cannot attend two classes at the same time. Soft constraints should preferably be satisfied, but can be accepted with a penalty associated to their violation, e.g., students should not attend more than two classes in a row. The PECTP deals with the following hard constraints [41]:

- H1: No student attends more than one event at the same time;
- H2: The room is big enough for all the attending students and satisfies all the features required by the event;
- H3: Only one event is in a room at any time slot;
- H4: Events should only be assigned to time slots that are predefined as available for those events;
- H5: Where specified, events should be scheduled to occur in the correct order.

A solution is feasible if all hard constraints are satisfied by the solution. According to the organizers of ITC-2007, a timetable is required to satisfy all hard constraints. Due to the fact that it may be very difficult to achieve this hard-constraint feasibility, they suggested that some events may be left unassigned in a timetable if necessary, and they introduced the notion of "distance to feasibility ($Df$)" [41], which is defined as the number of students that are affected by unassigned events. Given a solution, if there is any event that causes any

hard-constraint violation, it needs to be removed (i.e., unassigned) from the timetable, and as an effect, some students that have to take this event will suffer from its removing. The $Df$ of the given solution is calculated by identifying the number of students that are required to attend each of the unassigned events and then simply adding these values together. For example, if a solution has three events that need to be unassigned to prevent any violation of the hard constraints, and the number of students that need to attend each of these events is 2, 3, and 1, then the $Df$ of the solution is $(2+3+1) = 6$. With the notion of "distance to feasibility", an infeasible solution can be characterized by its $Df$.

In the PECTP, there are also soft constraints, which are penalized equally by their occurrences

- S1: A student has a class in the last time slot of a day;
- S2: A student has more than two classes in a row;
- S3: A student has a single class on a day.

The soft-constraint penalty value is denoted as *SCP* in this paper.

### 3.2 Problem formulation

In a PECTP, we assign an event (courses, lectures) into a time slot and also assign a number of resources (students, rooms) in such a way that there is no conflict between the rooms, time slots, and events. The PECTP consists of a set of $n$ events (classes, subjects) $E = \{e_1, e_2, ..., e_n\}$ to be scheduled in a set of 45 time slots $T = \{t_1, t_2, ..., t_{45}\}$ (i.e., nine for each day in a five day week), a set of $m$ available rooms $R = \{r_1, r_2, ..., r_m\}$ in which events can take place, a set of $k$ students $S = \{s_1, s_2, ..., s_k\}$ who attend the events, a set of $l$ available features $F = \{f_1, f_2, ..., f_l\}$ that are satisfied by rooms and required by events.

In addition, interrelationships between these sets are given by the following seven matrices:

- The first matrix $A_{k\times n}$, called the *Student-Event* matrix, shows which event is attended by which students. The value of a cell $a_{ij} \in A_{k\times n}$ is 1 if student $s_i \in S$ should attend event $e_j \in E$; otherwise, the value is 0.
- The second matrix $B_{n\times n}$, called the *Event-Conflict* matrix, indicates whether two events can be scheduled in the same time slot or not. It helps to quickly identify events that can be potentially assigned to the same time slot.
- The third matrix $C_{m\times l}$, called *Room-Feature* matrix, gives the features that each room possesses, where the value of a cell $c_{ij}$ is 1 if $r_i \in R$ has a feature $f_j \in F$; otherwise, the value is 0.
- The fourth matrix $D_{n\times l}$, called *Feature-Event* matrix, gives the features required by each event. It means that event $e_i \in E$ needs feature $f_j \in F$ if and only if $d_{ij} = 1$.
- The fifth matrix $G_{n\times m}$, called the *Event-Room* matrix, lists the possible rooms to which each event can be assigned. Through this matrix, we can quickly identify all rooms that are suitable in size and feature for each event. Usually, a matrix is used for assigning each event to a room $r_i$ and a time slot $t_i$. Each pair of $(r_i, t_i)$ is assigned a particular number which corresponds to an event. If a room $r_i$ in a time slot $t_i$ is free or no event is placed, then "-1" is assigned to that pair. In this way, we assure that there will be no more than one event assigned to the same pair so that one of the hard constraint will always been satisfied.
- The six matrix $H_{n\times t}$ is called the *Event-Availability* matrix, where the value of a cell $m_{ij}$ is 1 if event $e_i \in E$ should take place at time slot $t_j \in T$; otherwise, the value is 0.
- The last matrix $I_{n\times n}$ is called the *Event-Preference* matrix, where the value of cell $n_{ij}$ is 1 if $e_i \in E$ has to be schedule before $e_j \in E$, or -1 if $e_i \in E$ has to place in timetable after $e_j \in E$; otherwise, the value is 0 if no restriction of precedence between $e_i \in E$ and $e_j \in E$.

In addition to the above matrices, we create an array $EE$ of lists. Each element $EE_i \in EE$ is a list of events that have to be scheduled in a timetable after event $e_i$. This information helps to satisfy the hard constraint H5 in the execution of an algorithm. We also create a list $ET$ of event time slots. Each element $ET_i \in ET$ is a list of possible time slots where event $e_i$ has to be scheduled. There is also a set $E'_e$ of events that are not subject to any time restriction. Similarly, there is a set $T'_s$ of time slots that have no restriction of any event.

For room assignment, we use a matching algorithm described by Rossi-Doria *et al.* [51]. For every time slot, there is a list of events taking place in it and a preprocessed list of possible rooms to which events can be assigned. The matching algorithm uses a deterministic network flow algorithm and gives the maximum cardinality matching between rooms and events.

In general, the solution to a PECTP can be represented in the form of an ordered list of pairs $(r_i, t_i)$, of which the index of each pair is the identification number of an event $e_i \in E$ $(i = 1, 2, \cdots, n)$. For example, the time slots and rooms are allocated to events in an ordered list of pairs like:

$$(2, 4),\ (3, 30),\ (1, 12),\ \cdots,\ (2, 7),$$

where room 2 and time slot 4 are allocated to event 1, room 3 and time slot 30 are allocated to event 2, and so on.

The goal of the PECTP is to minimize the number of hard- and soft-constraint violations. The objective function $f(s)$ for a timetable $s$ is the weighted sum of the number of hard-constraint violations $\#hcv$ and soft-constraint violations $\#scv$, which was also used in [52], as defined below:

$$f(s) := \#hcv(s) * C + \#scv(s) \quad (1)$$

where $C$ is a constant, which is larger than the maximum possible number of soft-constraint violations.

## 4 Proposed hybrid approach for the PECTP

GAs are a class of powerful general purpose optimization tools that model the principles of natural evolution [31,53]. GAs are population-based heuristic methods, which start from an initial population of random solutions for a given problem. Each solution in the population is called an individual. Each individual is evaluated according to a problem-specific objective function, usually called the fitness function. After evaluation, there is a selection phase in which possibly good individuals will be chosen by a selection operator to undergo the recombination process. In the recombination phase, crossover and mutation operators are used to create new individuals in order to explore the solution space. The newly created individuals replace old individuals, usually the worst ones, of the population-based on the fitness. This process is repeated until a stopping criterion is reached, which may be the maximum number of generations or a time limit. GAs were first used for timetabling in 1990 [22]. Since then, there have been a number of papers investigating and applying GAs for the UCTP [19,59].

In this paper, we propose a hybrid approach that hybrdizes the GSGA proposed in [35] with new LS techniques and a TS heuristic for the PECTP. The pseudocode of the proposed hybrid GA and TS approach, denoted *HGATS*, for the PECTP is shown in Algorithm 1. HGATS works in two phases. In the first phase, the GSGA that uses genetic operators, a guided search strategy, and two powerful LS techniques, is used to evolve a population of candidate solutions toward better and better solutions, ideally finding the optimal solution. Usually, GAs are able to locate promising regions for global optima in the search space, but sometimes have difficulties in finding the exact optimum of highly constrained problems [31]. Several examples can be found from the literature where a solution obtained from a GA is improved by another optimization technique [34]. In this paper, we also use this technique

**Algorithm 1** Proposed Hybrid Approach—HGATS

1: **input** : A problem instance **I**
2: set the generation counter $g := 0$
3: **for** $i := 1$ to population size **do**
4: $s_i \leftarrow InitializeIndividual(i)$
5: $s_i \leftarrow$ solution $s_i$ after applying LS operator 1 (LS1)
6: $s_i \leftarrow$ solution $s_i$ after applying LS operator 2 (LS2)
7: **end for**
8: **while** the termination condition is not reached **do**
9: **if** (g mod $\tau$) == 0 **then**
10: apply $ConstructMEM()$ to construct $MEM$
11: **end if**
12: $s \leftarrow$ child solution by applying $GuidedSearchByMEM()$ or $Crossover()$ with a probability $\gamma$
13: $s \leftarrow$ child solution after mutation with a probability $P_m$
14: $s \leftarrow$ child solution after applying LS operator 1 (LS1)
15: $s \leftarrow$ child solution after applying LS operator 2 (LS2)
16: replace the worst individual of the population by $s$
17: $g := g + 1$
18: **end while**
19: **if** $s$ is an optimal solution **then**
20: go to line 24
21: **else**
22: $s \leftarrow$ Apply $TabuHeuristic()$ on the best solution obtained in the first phase
23: **end if**
24: **output** : The best solution $s_{best}$ achieved for the problem instance **I**

in HGATS, trying to find an optimal solution for the PECTP. Considering the hardness of the PECTP, if only feasible solutions are found during the first phase of HGATS, the second phase is executed, which uses a TS heuristic inspired by [51] to improve the feasible solution toward the optimal solution. Below we describe the two phases of HGATS in detail, respectively.

### 4.1 The enhanced GSGA—Phase I of HGATS

The first phase of HGATS uses the GSGA, which is adapted and enhanced according to the PECTP, to solve the PECTP. The framework of GSGA is based on a steady state GA, where one child solution is generated per iteration (or per generation) [27,49]. GSGA starts from an initial population of individuals that are randomly generated by $InitializeIndividual()$, where events are assigned to rooms and time slots for each solution based on the property of each event. Usually, for GAs, the quality of the initial solutions affects the final solutions and researchers have shown that good initial solutions usually produce good or required results within less computational time [24,42,53]. Hence, we want to create a good initial population that would help GSGA evolve toward the optimal solution quickly. For this purpose, two LS methods are applied to each individual of the initial population. The LS methods use six neighborhood structures, which will be described in

**Algorithm 2** *InitializeIndividual*$(i)$

1: **input** : The index $i$ of individual $I_i$
2: **for** each event $e_j$ of $I_i$ **do**
3: **if** event $e_j \in E'_e$ **then**
4: assign a random time slot from $T'_s$ to $e_j$
5: assign a random room from a list of suitable rooms
6: **else**
7: assign a random time slot from $ET_j$ to $e_j$
8: assign a random room from a list of suitable rooms
9: **end if**
10: **end for**
11: **output** : The generated individual $I_i$

Section 4.1.5, to first move events to time slots and then use the matching algorithm to allocate rooms and time slots to events.

After the initialization of the population, a data structure (denoted $MEM$ in this paper) is constructed, which stores a list of room and time slot pairs $(r, t)$ for all the events in the set $E'_e$ that have zero penalty (i.e., no hard- and soft-constraint violation at these events) of good individuals selected from the population. After that, $MEM$ can be used to guide the generation of offspring for the following generations. The $MEM$ data structure is re-constructed regularly, e.g., every $\tau$ generations. In each generation of GSGA, one child is first generated either by using $MEM$ or by applying the crossover operator, depending on a probability $\gamma$. Then, the child will undergo the mutation operation followed by the LS methods for potential improvement. Finally, the worst member in the population is replaced with the newly generated child individual. This iteration continues until one termination condition is reached, e.g., a preset time limit $t_{max}$ is reached or the best solution found has no soft- and hard-constraint violation.

In the following sub-sections, we will describe in details the key components of the adapted GSGA respectively, including the initialization of the population, the $MEM$ data structure and its construction, the guided search strategy, the crossover and mutation operators, and the two LS methods.

### *4.1.1 Initialization of the population*

Each individual $I_i$ of the initial population is created by Algorithm 2. We divide the set of events $E$ into two classes: events in $E'_e$ and events not in $E'_e$. If an event has no particular time slot restriction, it is allocated a random time slot $t$ from the set $T'_s$ of not restricted time slots and a suitable room; otherwise, the event is allocated a random time slot from the element time slot list of $ET$ corresponding to the event and randomly allocated a room among suitable rooms. This way, each individual generated will satisfy the hard constraints H2 and H4. However, it is not guaranteed to be feasible. An infeasible individual will be checked by the following LS operations, which will try to make it feasible.

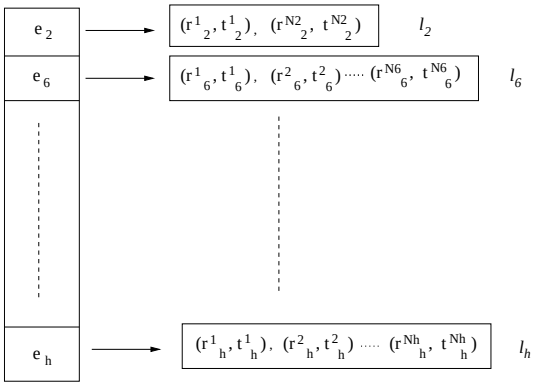


**Fig. 1** Illustration of the data structure $MEM$.

**Algorithm 3** *ConstructMEM*()

1: **input** : The whole population $P$ with the population size $N$
2: sort the population $P$ according to the fitness of individuals
3: $Q \leftarrow$ select the best $\alpha \times N$ individuals in $P$
4: **for** each individual $I_j$ in $Q$ **do**
5: **for** each event $(e_i \in E'_e)$ in $I_j$ **do**
6: calculate the penalty value of event $e_i$ from $I_j$
7: **if** $e_i$ is feasible (i.e., $e_i$ has zero penalty) **then**
8: add the room and time slot pair $(r_i, t_i)$ assigned to $e_i$ into the list $l_i$
9: **end if**
10: **end for**
11: **end for**
12: **output** : The data structure $MEM$

### *4.1.2 The MEM data structure*

There have been a number of researches in the literature on using extra data structure or memory to store useful information in order to enhance the performance of GAs and other meta-heuristic methods for optimization and search [8,7,43]. In GSGA, we also use a data structure $MEM$ to guide the generation of offspring by re-introducing the best part of individuals from previous generations. This $MEM$ data structure is used to provide further direction of exploration and exploitation in the search space.

Fig. 1 shows the details of the $MEM$ data structure, which is a two-level structure. The first level is a list of events and the second level is a list $l_i$ of room and time slot pairs corresponding to each event $e_i$ in the first level list. In Fig. 1, $N_i$ represents the total number of pairs in the second level list $l_i$.

The $MEM$ data structure is regularly re-constructed every $\tau$ generations. Algorithm 3 shows the outline of constructing $MEM$. When $MEM$ is due to be re-constructed, we first select $\alpha \times N$ best individ-

**Algorithm 4** *GuidedSearchByMEM*()

1: **input** : The $MEM$ data structure
2: $E_s$ := randomly select $\beta * |E'_e|$ events from $E'_e$
3: **for** each event $e_i$ in $E_s$ **do**
4: randomly select a room and time slot pair from the list $l_i$
5: assign the selected pair to event $e_i$ for the child
6: **end for**
7: **for** each remaining event $e_i$ not in $E_s$ **do**
8: **if** $e_i \in ET$ **then**
9: assign a particular time slot and suitable room to $e_i$
10: **else**
11: assign a random time slot and room to $e_i$
12: **end if**
13: **end for**
14: **output** : A new child generated using $MEM$

**Algorithm 5** *Crossover*()

1: **input** : The current population
2: Select parents $P1$ and $P2$ by the tournament selection
3: **for** each event $e_i$ of the child $Ch$ **do**
4: **if** penalty value of $e_i$ of $P1$ < penalty value of $e_i$ of $P2$ **then**
5: $e_i$ of $Ch \leftarrow$ the time slot and room allocated to $e_i$ of $P1$
6: **else**
7: $e_i$ of $Ch \leftarrow$ the time slot and room allocated to $e_i$ of $P2$
8: **end if**
9: **end for**
10: **output** : A new child generated using crossover

uals from the population $P$ to form a set $Q$, where $N$ denotes the population size. After that, for each individual $I_j \in Q$, we check each event $e_i \in E'_e$[2] by its penalty value, i.e., the hard- and soft-constraint violations associated with this event. If an event has a zero penalty value, then we store the information corresponding to this event into $MEM$. For example, if the event $e_2$ of an individual $I_j \in Q$ is assigned room 2 at time slot 13 and has a zero penalty value, then we add the pair $(2, 13)$ into the list $l_2$. Similarly, the events of the next individual $I_{j+1} \in Q$ are also checked by their penalty values. If the event $e_2$ in $I_{j+1}$ has a zero penalty, then we add the pair of room and time slot assigned to $e_2$ in $I_{j+1}$ into the existing list $l_2$. If for an event $e_i$, there is no list $l_i$ existing yet, then the list $l_i$ is added into the $MEM$ data structure.

Similar process is carried out for the selected $Q$ individuals and finally $MEM$ stores pairs of room and time slot corresponding to those events with zero penalty of the best individuals of the current population. This newly re-constructed $MEM$ data structure is then used to guide the generation of offspring for the next $\tau$ generations. We update $MEM$ every $\tau$ generations instead of every generation in order to make a balance between the solution quality and the computational time cost of GSGA.

#### *4.1.3 Generating a child by the guided search strategy*

In GSGA, a child is created through the guided search by $MEM$ or a crossover operator with a probability $\gamma$. That is, when a new child is to be generated, a random number $\rho \in [0.0, 1.0]$ is first generated. If $\rho$ is less than $\gamma$, *GuidedSearchByMEM*() (as shown in Algorithm 4) will be used to generate the new child; otherwise, a crossover operation *Crossover*() (as shown in Algorithm 5) will be used to generate the new child.

If a child is to be created using the $MEM$ data structure, we first select a set $E_s$ of $\beta * |E'_e|$ random events from $E'_e$ to be generated from $MEM$. Here, $\beta$ is a percentage value and $|E'_e|$ is the size of the set $E'_e$. We randomly select a pair of $(r_i^j, t_i^j)$, $j = 1, \cdots, N_i$, from the list $l_i$ that corresponds to the event $e_i$ and assign the selected pair to $e_i$ for the child. If there is an event $e_i$ in $E_s$ but there is no list $l_i$ in $MEM$, then we randomly assign a room and time slot from possible rooms and time slots to $e_i$ for the child. This process is carried out for all the events in $E_s$. For those remaining events that are not present in $E_s$, they are assigned time slots and room according their particular requirements for the child.

If a child is to be generated using the crossover operator, we first select two individuals from the current population as the parents by the tournament selection with a tournament size 2. Then, a child is generated as follows: for each event, we first select the parent that has the smaller penalty value corresponding to that event, and then allocate the corresponding room and time slot pair to the event of the child.

#### *4.1.4 Mutation*

After a child is generated by using either $MEM$ or crossover, a mutation operator is used with a probability $P_m$. The mutation operator first randomly selects one from four neighborhood structures N1, N2, N3, and N4, which will be described in Section 4.1.5, and then makes a move within the selected neighborhood structure.

#### *4.1.5 Local search methods*

After the mutation operation, two LS operators, denoted *LS1* and *LS2*, respectively, are applied on the child solution for possible improvement. Algorithm 6

[2] We only check those events that are not involved directly with H4 because other events must have been assigned in pre-specified time slots. It is worthless to assign and evaluate those events since they do not help to increase the diversity of the GSGA.

**Algorithm 6** Local Search Operator 1 (LS1)

```
1: input : Individual I from the population
2: for i := 1 to n do
3:    if event e_i is infeasible then
4:       if there is untried move left then
5:          calculate the moves: first in N1, then in N2 if N1 fails,
            then in N3 if N2 also fails, and finally in N4 if N3 also
            fails
6:          apply the matching algorithm to the time slots af-
            fected by the move and delta evaluate the result.
7:          if moves reduce hard-constraint violations then
8:             make the moves and go to line 4
9:          end if
10:      end if
11:   end if
12: end for
13: if no any hard-constraint violations remain then
14:   for i := 1 to n do
15:      if event i has soft-constraint violations then
16:         if there is untried move left then
17:            calculate the moves: first in N1, then in N2 if N1
               fails, then in N3 if N2 also fails, and finally in N4
               if N3 also fails
18:            apply the matching algorithm to the time slots
               affected by the move and delta evaluate the result
19:            if moves reduce soft-constraint violations then
20:               make the moves and go to line 16
21:            end if
22:         end if
23:      end if
24:   end for
25: end if
26: output : A possibly improved individual I
```

**Algorithm 7** Local Search Operator 2 (LS2)

```
1: input : Individual I after LS1 is applied
2: for each event e_i of I in EE do
3:    for each event e_j in EE_i do
4:       try to place event e_j in the timetable after the time slot
         of e_i by calculating a move of e_i in the neighborhood
         N1 and N2
5:       apply the matching algorithm to the time slots affected
         by the move
6:       compute the penalty of e_i and delta evaluate the result
7:       apply the move if it reduces hard- or soft-constraint vi-
         olations
8:    end for
9: end for
10: S :=randomly pick a percentage of occupied time slots from T
11: for each time slot t_i ∈ S do
12:    for each event e_j in the time slot t_i do
13:       calculate the penalty value of event e_j
14:    end for
15:    sum the total penalty value of events in the time slot t_i
16: end for
17: select the time slot w_t with the biggest penalty value from S
18: for each event e_i in w_t do
19:    calculate a move of e_i in the neighborhood N1
20:    apply the matching algorithm to the time slots affected by
       the move
21:    compute the penalty of e_i and delta evaluate the result
22: end for
23: if all the moves of events in w_t together reduce hard- or
    soft-constraint violations then
24:    apply the moves
25: else
26:    delete the moves
27: end if
28: output : A possibly improved individual I
```

and Algorithm 7 summarize the LS1 and LS2 schemes, used in the proposed algorithm, respectively. LS1 works on all events while LS2 works on a set of events. Here, we suppose that each event is involved with soft- and hard-constraint violations. The LS methods are based on six neighborhood structures, denoted as N1, N2, N3, N4, N5, and N6, respectively. They are described as follows:

- N1: the neighborhood defined by an operator that moves one event from a time slot to a different one.
- N2: the neighborhood defined by an operator that swaps the time slots of two events.
- N3: the neighborhood defined by an operator that permutes three events in three distinct time slots in one of the two possible ways other than the existing permutation of the three events.
- N4: the neighborhood defined by an operator that takes two random events from the set $E'_e$ and replaces their time slots by random ones from $T'_s$.
- N5: the neighborhood defined by an operator that takes each event $e_i$ from the list of $EE$ and try to find a place in the timetable before all the events in $EE_i$.
- N6: the neighborhood defined by an operator that takes a subset of time slots among all occupied time slots. Among this subset, the worst time slot (that contain events that collectively have the highest penalty value) is selected and its events are moved to another randomly chosen time slot in the subset.

As mentioned before, LS1 works on all events and is based on two steps. In the first step (line 2-12 in Algorithm 6), LS1 checks the hard-constraint violations of each event while ignoring its soft-constraint violations. If there are hard-constraint violations for an event, LS1 tries to resolve them by applying moves in the neighborhood structures N1, N2, N3, and N4 orderly[3] until

[3] For the event being considered, potential moves are calculated in a strict order. First, we try to move the event to the next time slot, then the next, then the next, etc. If this search in N1 fails, we then search in N2 by trying to swap the event with the next one in the list, then the next one, and so on. If the search in N2 also fails, we try a move in N3 by using one different permutation formed by the event with the next two events, then with the next two, and so on. If the search in N3 also fails, we try a move in N4 by replacing the time slots of two random events that are in the set $E'_e$ with random time slots from $T'_s$, then the next two, and so on.

a termination condition is reached, e.g., an improvement is reached or the maximum number of steps $s_{max}$ is reached, which is set to different values for different problem instances in the experimental study.

After each move, we apply the matching algorithm to the time slots affected by the move and try to resolve the room allocation disturbance and delta evaluate the result of the move (i.e., calculate the hard- and soft-constraint violations before and after the move). If there is no untried move left in the neighborhood for an event, LS1 continues to the next event. After applying all neighborhood moves on each event, if there is still any hard-constraint violation, then LS will stop; otherwise, LS1 will perform the second step (lines 13-25 in Algorithm 6). In the second step, after reaching a feasible solution, the LS1 method is used to deal with soft constraints. LS1 performs a similar process as in the first step on each event to reduce its soft-constraint violations. For each event, LS1 tries to make moves in the neighborhood N1, N2, N3 and/or N4 orderly without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta evaluated.

Algorithm 7 describes the second LS operator, LS2, used in GSGA. LS2 works on a set of events with N5 (corresponding to lines 2-9 in Algorithm 7) and N6 (corresponding to lines 10-27 in Algorithm 7). The basic idea of LS2 is that it first tries to place an event $e_i$ (involved in the precedence constraint H5) in a time slot before the corresponding list of events $EE_i$. After moving a concerned event into a new time slot in the neighborhood structures N1 and N2 every time, the new penalty value of the event is calculated. If the move reduces the penalty value, then it is saved; otherwise, it is not saved.

After applying N5, LS2 applies N6. It first randomly selects a percentage of time slots[4] (e.g., 20% as used in this paper) from the total time slots in $T$. Then, the penalty value of each selected time slot is calculated and the time slot $w_t$ that has the biggest penalty value is selected for local search. This way, LS2 aims to help improve the existing result. After taking the worst time slot, LS2 tries a move in the neighborhood structure N1 for each event of $w_t$ and checks the penalty value of each event before and after applying the move. If all moves in $w_t$ together reduce the hard- and/or soft-constraint violations, then we apply all the moves; otherwise, we do not make the moves. This way, LS2 can not only place the events according to their precedence but also check the worst time slot and reduce the penalty value for some events by moving them to other time slots. In general, LS2 is expected to enhance the individuals of the population and increase the quality of the feasible solution by reducing the number of constraint violations. When LS2 finishes, we get a possibly improved and feasible individual.

At the end of each generation of GSGA, the obtained child solution replaces the worst member of the population to make a better population in the next generation. By the end of Phase I, GSGA may produce several different optimal or near-optimal solutions.

[4] Rather than choosing the worst time slot out of all time slots, we randomly select a set of time slots and then choose the worst time slot from the set. This is because for each selected time slot we need to calculate its penalty value, which costs time. By selecting a set of time slots instead of all time slots, we try to balance between the computational time and the quality of the algorithm.

**Algorithm 8** $TabuHeuristic()$—Phase II of HGATS

```
1: input : The best solution s_best from Phase I (GSGA)
2: s ← s_best
3: if s is not feasible then
4:    remove all events that involve hard-constraint violations
5: end if
6: TL ← ∅
7: while the termination condition is not reached do
8:    for i := 0 to 10% of the neighbors do
9:       s_i ← s after the i-th move
10:      compute the objective value f(s_i)
11:   end for
12:   if ∃s_j|f(s_j) < f(s) and f(s_j) ≤ f(s_i)∀i then
13:      s ← s_j
14:      TL ← TL ∪ E_i where E_i is the set of events moved to
         get s_j
15:   else
16:      s ← the best non-tabu moves among all s_i
17:      TL ← TL ∪ E_b where E_b is the set of events moved by
         the best non-tabu move
18:   end if
19:   s_best ← the best solution so far
20: end while
21: output : The optimized solution s_best
```

## 4.2 The tabu search heuristic – Phase II of HGATS

We try to find an optimal solution using the above proposed GSGA. However, due to the hardness of the PECTP, after the first phase of HGATS, sometimes an optimal or feasible solution may not be obtained. In order to further improve the quality of the solution obtained by GSGA, a simple TS heuristic $TabuHeuristic()$, which is shown in Algorithm 8, is applied as the second phase of HGATS in the hope to get an improved and feasible solution from the best solution obtained from phase I. TS is a kind of heuristic methods, which has the advantage of having internal memory [30]. This internal memory prevents TS from revisiting previously visited areas of the search space. Therefore, it is easier to escape from local optimum and

approach the global or near-global optimum in a short time [23]. TS is usually known be a powerful tool for all types of timetabling problems [16].

The TS heuristic used in HGATS is similar to the TS scheme described in [51]. We first check the best solution obtained from the first phase. If it is optimal, Phase II will not be executed. Otherwise, if it is a feasible solution, then we improve the solution by applying the TS heuristic; if a solution is not feasible, we first remove all events that involve hard-constraint violations and re-consider them if and only if they satisfy all hard constraints during the neighborhood search.

We apply N1, N2, and N4 as neighborhood structures for moving a solution. A move of a solution is defined by moving one random event of the solution using N1, swapping two random events of the solution using N2, or swapping two specific events of the solution to time slots using N4, orderly. The reason for not applying N5 and N6 in the move lies in that using N5 and N6 takes time and extra work on removing a hard-constraint violation.

A move is a tabu move if at least one of the events involved has been moved less than $l$ steps before, where $l$ is the length of the tabu list $TL$. The tabu list length is set to the number of events divided by a constant $K$ ($K = 100$ as described in [51]). In order to decrease the probability of generating cycles of moves and enhance the exploration, a variable neighborhood set is applied, as suggested in [51], where every move uses the neighborhood N1, N2, or N4 with a probability 0.1. In order to explore the search space more efficiently, we accept a tabu move if it improves the best so far solution. In summary, the TS heuristics considers a variable neighborhood set and performs the best move that improves the best so far solution; otherwise, it performs the best non-tabu move chosen among those that belong to the current variable set of neighbors. The TS heuristics continues until a time limit is reached or the best so far solution has no soft- and hard-constraint violation (i.e., an optimal solution is obtained).

## 5 Experimental study

In this section, we experimentally investigate the performance of the proposed hybrid approach for the PECTP in comparison with several other algorithms. All algorithms were coded in GNU C++ under version 4.1 and run on a 3.20 GHz PC. We use 24 benchmark PECTP instances to test the algorithms, which were proposed for the ITC-2007 [41]. Table 1 presents the features of these PECTP instances[5].

Two sets of experiments were carried out in this study. The first set of experiments are devoted to analyze the sensitivity of key parameters for the performance of HGATS for the PECTP. The second set of experiments compare the performance of HGATS with two relevant algorithms on the test PECTP instances. In the end, we compare our experimental results with current state-of-the-art methods from the literature on the tested instances.

### 5.1 Sensitivity analysis of key parameters of HGATS

The performance of the proposed hybrid approach depends on the parameters and operators used, especially in GSGA. Through our previous work [35], we found that $\alpha$, $\beta$, $\gamma$, and $\tau$ are key parameters that can greatly affect the performance of GSGA for the UCTP, where $\alpha$ is the percentage of best individuals selected from the current population for creating the data structure $MEM$, $\beta$ is the percentage of the total number of events that are used to create a child through the data structure $MEM$, $\gamma$ is the probability that indicates whether a child is created through $MEM$ or crossover, and $\tau$ decides the frequency of updating $MEM$ (i.e., $MEM$ is updated every $\tau$ generations). Hence, we test our algorithm HGATS with different settings of these parameters. Table 2 shows different parameters and their settings that were tested in our experiments. Some other parameters for HGATS were set as follows: the population size $N$ was set to 50 and the mutation probability $P_m$ was set to 0.5.

In order to find out which parameter settings have a great effect on the performance of HGATS, we run HGATS 50 times for all parameter combinations in Table 2. Here, we report some typical results in Fig. 2, where the dynamic performance of HGATS regarding the average objective value against the number of evaluations over 50 runs with one parameter changing while the other parameters kept constant on different PECTP instances is shown. Fig. 2(a) shows the effect of changing $\alpha$ on the 2007-16 problem instance with $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$. Fig. 2(b) shows the effect of changing $\beta$ on 2007-17 with $\alpha = 0.2$, $\gamma = 0.8$, and $\tau = 20$. Fig. 2(c) shows the effect of changing $\gamma$ on 2007-11 with $\alpha = 0.2$, $\beta = 0.3$, and $\tau = 20$. Fig. 2(d) shows the effect of changing $\tau$ on 2007-3 with $\alpha = 0.2$, $\beta = 0.3$, and $\gamma = 0.8$.

[5] Details about these PECTP instances can be found at http://www.cs.qub.ac.uk/itc2007/postenrolcourse/course_post_index.htm.

**Table 1** ITC-2007 problem instances

| ITC-2007 Instance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of events | 400 | 400 | 200 | 200 | 400 | 400 | 200 | 200 | 400 | 400 | 200 | 200 |
| Number of rooms | 10 | 10 | 20 | 20 | 20 | 20 | 20 | 20 | 10 | 10 | 10 | 10 |
| Number of features | 10 | 10 | 10 | 10 | 20 | 20 | 20 | 20 | 20 | 20 | 10 | 10 |
| Number of students | 500 | 500 | 1000 | 1000 | 300 | 300 | 500 | 500 | 500 | 500 | 1000 | 1000 |
| Max students per event | 33 | 32 | 98 | 82 | 19 | 20 | 43 | 39 | 34 | 32 | 88 | 81 |
| Max events per student | 25 | 24 | 15 | 15 | 23 | 24 | 15 | 15 | 24 | 23 | 15 | 15 |
| Mean features per room | 3 | 4 | 3 | 3 | 2 | 3 | 5 | 4 | 3 | 3 | 3 | 4 |
| Mean features per event | 1 | 2 | 2 | 2 | 1 | 2 | 3 | 3 | 1 | 2 | 1 | 23 |
| ITC-2007 Instance | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Number of events | 400 | 400 | 200 | 200 | 100 | 200 | 300 | 400 | 500 | 600 | 400 | 400 |
| Number of rooms | 20 | 20 | 10 | 10 | 10 | 10 | 10 | 10 | 20 | 20 | 20 | 20 |
| Number of features | 10 | 10 | 20 | 20 | 10 | 10 | 10 | 10 | 20 | 20 | 30 | 30 |
| Number of students | 300 | 300 | 500 | 500 | 500 | 500 | 1000 | 1000 | 300 | 500 | 1000 | 1000 |
| Max students per event | 20 | 20 | 41 | 40 | 195 | 65 | 55 | 40 | 16 | 22 | 69 | 41 |
| Max events per student | 24 | 24 | 15 | 15 | 23 | 23 | 14 | 15 | 23 | 25 | 24 | 15 |
| Mean features per room | 2 | 3 | 2 | 5 | 4 | 4 | 3 | 3 | 3 | 3 | 5 | 5 |
| Mean features per event | 1 | 1 | 3 | 3 | 2 | 2 | 1 | 1 | 1 | 2 | 3 | 3 |

**Table 2** Parameter settings in HGATS

| Parameter | Settings | | | |
|---|---|---|---|---|
| $\alpha$ | 0.2 | 0.4 | 0.6 | 0.8 |
| $\beta$ | 0.1 | 0.3 | 0.5 | 0.7 |
| $\gamma$ | 0.2 | 0.4 | 0.6 | 0.8 |
| $\tau$ | 20 | 40 | 60 | 80 |

From Fig. 2, several results can be observed and are analyzed below. First, the parameter $\alpha$ has a significant effect on the performance of HGATS for the PECTP. The performance of HGATS drops when the value of $\alpha$ increases from 0.2 to 0.8, see Fig. 2(a) for reference. This occurs because when we choose a small part of population to create the $MEM$ data structure, $MEM$ can provide a strong guidance during the genetic operations and help HGATS exploit the area of the search space that corresponds to the best individuals of the population sufficiently. This sufficient exploitation can ensure that HGATS quickly achieves better solutions. In the contrast, when a large part of the population is taken to create or update $MEM$, then $MEM$ will lose its effect of guiding HGATS to exploit promising areas of the search space. In other words, when $\alpha$ is set to large values, HGATS tends to be GALS and, hence, the performance will drop or be weak. This can be observed from Fig. 2(a): when the value of $\alpha$ increases, the best solution of HGATS can not improve after a certain number of evaluations, e.g., after about 4000 evaluations when $\alpha = 0.6$ and after about 2000 evaluations when $\alpha = 0.8$.

Second, regarding the effect of $\beta$, an interesting behaviour of HGATS can be observed on the 2007-17 problem instance with $\alpha = 0.2$, $\gamma = 0.8$, $\tau = 20$, and different $\beta$ values in Fig. 2(b). From Fig. 2(b), it can be seen that when the value of $\beta$ increases from 0.1 to 0.3, the performance of HGATS improves due to the enhanced effect of the $MEM$ data structure. However, when the value of $\beta$ is further raised, the performance of HGATS drops. This occurs because if a large portion of individuals is created through $MEM$, e.g., when $\beta = 0.7$, the chance of creating a similar child may be increased every generation and after a few generations, HGATS may be trapped in a sub-optimal state and hence can not obtain the optimal solution. From Fig. 2(b), it can be seen that setting the value of $\beta$ to 0.5 or 0.7 leads to an earlier stagnation in the performance of HGATS during the solving process.

Third, regarding the effect of $\gamma$, from Fig. 2(c), it can be easily seen that increasing the value of $\gamma$ results in better solutions. The reason lies in the fact that the small value of $\gamma$ leads to the proposed GSGA acting as the conventional GA. The effect of $\gamma$ also shows the importance of introducing the $MEM$ data structure.

Fourth, regarding the effect of $\tau$, it can be seen from Fig. 2(d) that updating $MEM$ every 20 generations gives a better performance for HGATS than updating $MEM$ every 80 generations. This is due to the fact that in the former case the search space is explored more than in the latter case, which increases the diversity and gives a greater chance to create better individuals. The difference is significant when $\tau$ is set to 20 over 100. This is because increasing the value of $\tau$ slows down the updating of the $MEM$ data structure and hence degrades the efficiency of $MEM$.

Based on the above parameter analyses, in the following experiments, we set the parameters for HGATS as follows: $\alpha = 0.2$, $\beta = 0.3$, $\gamma = 0.8$, and $\tau = 20$.

### 5.2 Comparison with relevant algorithms

This set of experiments compares the performance of HGATS with two relevant algorithms: one is the same as the proposed GSGA except that the guided search technique is switched off. That is, it is the standard

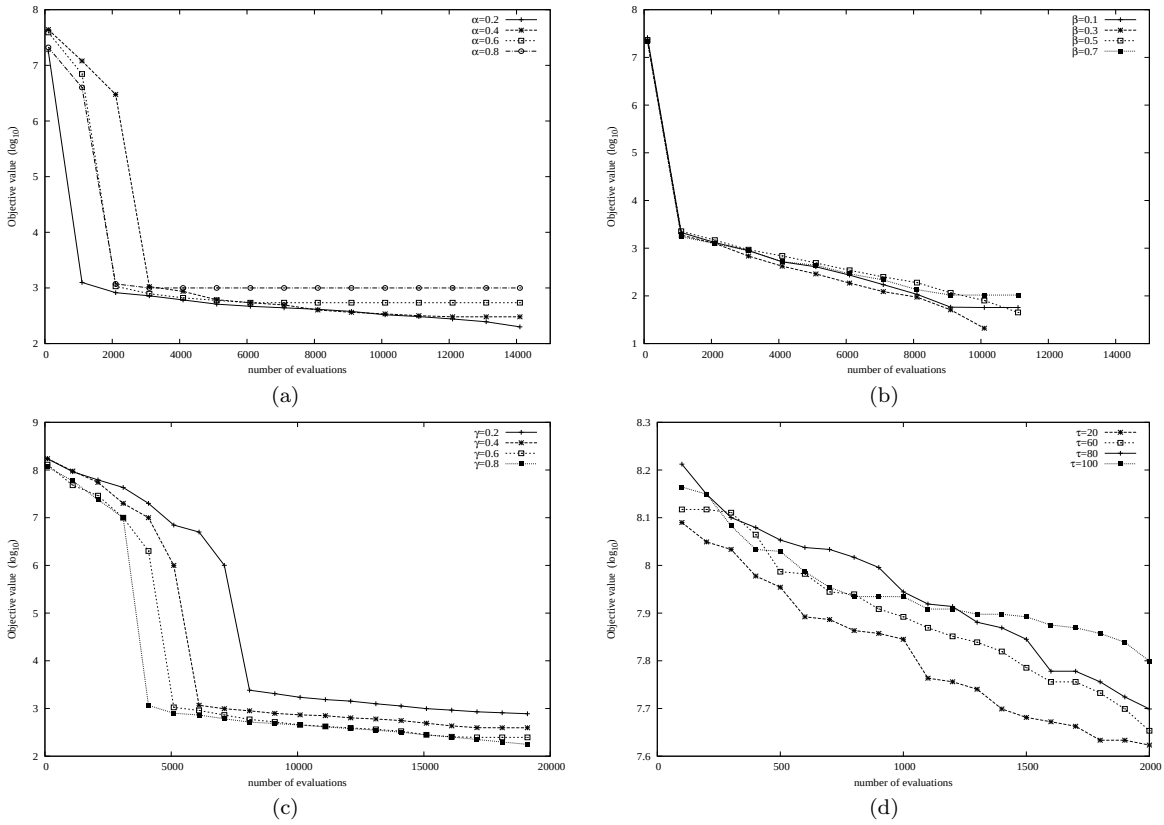


**Fig. 2** Comparison on the effect of parameters on the performance of HGATS on different problem instances: (a) 2007-21 with $\beta = 0.3$, $\gamma = 0.8$ and $\tau = 20$, (b) 2007-17 with $\alpha = 0.2$, $\gamma = 0.8$ and $\tau = 20$, (c) 2007-11 with $\alpha = 0.2$, $\beta = 0.3$ and $\tau = 20$, and (d) 2007-03 with $\alpha = 0.2$, $\beta = 0.3$ and $\gamma = 0.8$.

steady state GA with LS1 and LS2, denoted *GALS* in this study. For GALS, the crossover operator is applied with a crossover probability $P_c = 0.8$. The second algorithm is the TS algorithm. The basic framework of the TS algorithm tested is inspired by [51] with the same new neighborhood operators as the tabu heuristic used in HGATS. The parameter settings identified for HGATS by the previous experiments were used in HGATS and GALS (if relevant) in this section. The $InitializeIndividual()$ is used for initial solutions for all algorithms in order to have a fair comparison of the performance of algorithms. There were 50 runs of each algorithm on each problem instance. The run time for each run of an algorithm on each problem instance was set to $t_{max} = 600$ seconds based on the time allocation used by the ITC-2007. Other parameter settings are as follows: the population size $N$ was set to 50 and the mutation probability $P_m$ was set to 0.5.

Algorithms are evaluated on the basis of two values, Df and SCP. Table 3 presents the results of algorithms in terms of the best, worst, average, and standard deviation of Df and SCP values over the 50 runs on the 24 problem instances. From Table 3, it can be seen that HGATS produces a lower average and standard deviation of the objective value on most of the PECTP instances. HGATS produces good solutions due to the usage of the $MEM$ data structure and LS schemes. As mentioned earlier, this is due to the fact that we assign to an event a pair of room and time slot that was extracted from one of the best individuals of previous populations. This means that the pair satisfies different constraints that are suitable to that event. The local and tabu search techniques further helps find the local optimum of an individual. By doing so, we increase the chance of getting better and better solutions during the solving process.

Fig. 3 shows the performance of different algorithms regarding the objective value in the log scale against the number of evaluations. From Fig. 3, it can be seen that on the 2007-14 and 2007-17 problem instances, HGATS and TS reach a solution as the number of evaluations increases. HGATS remarkably decreases in the objective value and gives an optimal solution after 9000 and 4000 evaluations, respectively.

The $t$-test results of statistically comparing investigated algorithms with 98 degrees of freedom at a 0.05 level of significance are shown in Table 4. In Table 4, the $t$-test result is shown as "$s+$", "$s-$", "$+$", "$-$", or "$\sim$"

**Table 3** Comparison of algorithms on different problem instances

| | | Best | | Worse | | Average | | Std | |
|---|---|---|---|---|---|---|---|---|---|
| PECTP | Alg | Df | SCP | Df | SCP | Df | SCP | Df | SCP |
| 2007-1 | TS | 0 | 1069 | 51 | 1732 | 11.33 | 1202.6 | 17.55 | 257.04 |
| | GALS | 0 | 641 | 12 | 976 | 3.44 | 704.89 | 5.13 | 152.33 |
| | HGATS | 0 | 501 | 12 | 842 | 0 | 587 | 1.84 | 108.61 |
| 2007-2 | TS | 0 | 989 | 72 | 2213 | 25.87 | 1191.22 | 17.44 | 386.77 |
| | GALS | 0 | 747 | 50 | 2311 | 8.78 | 1005.11 | 16.87 | 504.79 |
| | HGATS | 0 | 342 | 0 | 695 | 0 | 476.2 | 0 | 96.94 |
| 2007-3 | TS | 0 | 756 | 18 | 821 | 5.89 | 794.33 | 5.84 | 21.9 |
| | GALS | 0 | 509 | 0 | 801 | 0 | 697.44 | 0 | 129.53 |
| | HGATS | 0 | 3770 | 432 | 0 | 0 | 407.78 | 0 | 19.73 |
| 2007-4 | TS | 0 | 794 | 76 | 1130 | 18.33 | 910.5 | 29.44 | 141.45 |
| | GALS | 0 | 521 | 11 | 791 | 2 | 669 | 4.09 | 46 |
| | HGATS | 0 | 234 | 4 | 524 | 0 | 369 | 0.33 | 26.72 |
| 2007-5 | TS | 0 | 496 | 65 | 678 | 22 | 544.2 | 9.62 | 884 |
| | GALS | 0 | 98 | 20 | 310 | 8.62 | 154 | 9.62 | 78 |
| | HGATS | 0 | 0 | 0 | 325 | 0 | 118 | 0 | 88.05 |
| 2007-6 | TS | 0 | 218 | 0 | 788 | 0 | 428 | 0 | 272.93 |
| | GALS | 0 | 10 | 0 | 430 | 0 | 207 | 0 | 134.18 |
| | HGATS | 0 | 0 | 0 | 342 | 0 | 201 | 0 | 139.5 |
| 2007-7 | TS | 0 | 84 | 198 | 508 | 82 | 258 | 66.59 | 183.74 |
| | GALS | 0 | 275 | 70 | 489 | 25.5 | 381.75 | 35.20 | 89.531 |
| | HGATS | 0 | 0 | 2 | 543 | 0.53 | 418 | 1.62 | 98.404 |
| 2007-8 | TS | 0 | 0 | 0 | 751 | 0 | 481 | 0 | 315.7 |
| | GALS | 0 | 0 | 0 | 424 | 0 | 322 | 0 | 193.1 |
| | HGATS | 0 | 0 | 0 | 309 | 0 | 257.12 | 0 | 120.78 |
| 2007-9 | TS | 0 | 1711 | 152 | 2361 | 40.12 | 1797 | 53.04 | 294.85 |
| | GALS | 0 | 1547 | 115 | 2141 | 28.87 | 1237 | 39 | 412.37 |
| | HGATS | 0 | 989 | 42 | 1183 | 4.5 | 1002 | 20.09 | 81.12 |
| 2007-10 | TS | 0 | 763 | 0 | 1978 | 0 | 999 | 0 | 406 |
| | GALS | 4 | 548 | 26 | 1040 | 5 | 850 | 9 | 154 |
| | HGATS | 0 | 499 | 0 | 810 | 0 | 614 | 0 | 117 |
| 2007-11 | TS | 0 | 680 | 0 | 1980 | 0 | 968 | 0 | 414 |
| | GALS | 0 | 701 | 0 | 984 | 0 | 897 | 0 | 84 |
| | HGATS | 0 | 246 | 0 | 691 | 0 | 452 | 0 | 121 |
| 2007-12 | TS | 0 | 373 | 56 | 1563 | 17 | 702 | 23 | 393 |
| | GALS | 0 | 444 | 0 | 984 | 0 | 576 | 0 | 178 |
| | HGATS | 0 | 172 | 13 | 546 | 1.625 | 226 | 0.59 | 129 |
| 2007-13 | TS | 0 | 624 | 20 | 1873 | 6.37 | 1230 | 9.1 | 380 |
| | GALS | 0 | 201 | 0 | 1639 | 0 | 852 | 0 | 392 |
| | HGATS | 0 | 0 | 0 | 717 | 0 | 616 | 0 | 249 |
| 2007-14 | TS | 0 | 241 | 17 | 416 | 4.75 | 287 | 7.62 | 76 |
| | GALS | 0 | 61 | 0 | 104 | 0 | 78.2 | 0 | 17 |
| | HGATS | 0 | 0 | 0 | 19 | 0 | 4.125 | 0 | 7.29 |
| 2007-15 | TS | 0 | 101 | 0 | 164 | 0 | 135 | 0 | 33 |
| | GALS | 0 | 14 | 0 | 97 | 0 | 69 | 0 | 21 |
| | HGATS | 0 | 0 | 0 | 37 | 0 | 26 | 0 | 6.54 |
| 2007-16 | TS | 0 | 109 | 0 | 1158 | 0 | 563 | 0 | 161 |
| | GALS | 0 | 168 | 0 | 771 | 0 | 377 | 0 | 195 |
| | HGATS | 0 | 0 | 0 | 270 | 0 | 168 | 0 | 115.27 |
| 2007-17 | TS | 0 | 0 | 0 | 42 | 0 | 32 | 0 | 10 |
| | GALS | 0 | 0 | 0 | 21 | 0 | 5 | 0 | 7.4 |
| | HGATS | 0 | 0 | 0 | 11 | 0 | 2.5 | 0 | 4.65 |
| 2007-18 | TS | 0 | 0 | 0 | 1241 | 0 | 924 | 0 | 420.52 |
| | GALS | 0 | 0 | 0 | 842 | 0 | 631 | 0 | 270 |
| | HGATS | 0 | 0 | 0 | 572 | 0 | 446 | 0 | 108 |
| 2007-19 | TS | 147 | 1078 | 346 | 1867 | 138 | 1372 | 110 | 334 |
| | GALS | 0 | 1015 | 430 | 2693 | 174 | 1612 | 154 | 673.65 |
| | HGATS | 0 | 84 | 319 | 1900 | 133 | 810 | 115 | 513.7 |
| 2007-20 | TS | 40 | 348 | 113 | 1192 | 71 | 1100 | 29 | 133 |
| | GALS | 0 | 318 | 138 | 1942 | 67 | 1199 | 86 | 439 |
| | HGATS | 0 | 297 | 234 | 2305 | 75 | 1274 | 95 | 622 |
| 2007-21 | TS | 0 | 137 | 261 | 1162 | 69.5 | 805 | 96 | 267 |
| | GALS | 0 | 0 | 10 | 621 | 22.5 | 305 | 4.6 | 241 |
| | HGATS | 0 | 0 | 15 | 1359 | 2.5 | 780 | 2 | 422 |
| 2007-22 | TS | 91 | 1742 | 102 | 2439 | 97.37 | 2051 | 4.47 | 260.6 |
| | GALS | 42 | 1579 | 188 | 2466 | 94 | 1715 | 42 | 396 |
| | HGATS | 0 | 1142 | 73 | 1315 | 33.125 | 1196 | 38 | 243 |
| 2007-23 | TS | 0 | 2062 | 34 | 5556 | 362 | 1604 | 16 | 8.75 |
| | GALS | 11 | 1001 | 43 | 1291 | 81 | 1193 | 13 | 20 |
| | HGATS | 0 | 963 | 16 | 1896 | 1.2 | 1152 | 3.6 | 2 |
| 2007-24 | TS | 0 | 629 | 0 | 2309 | 0 | 1407 | 0 | 541 |
| | GALS | 0 | 368 | 9 | 2007 | 2.25 | 1112 | 4.16 | 463.6 |
| | HGATS | 0 | 274 | 0 | 2142 | 0 | 1002 | 0 | 519 |

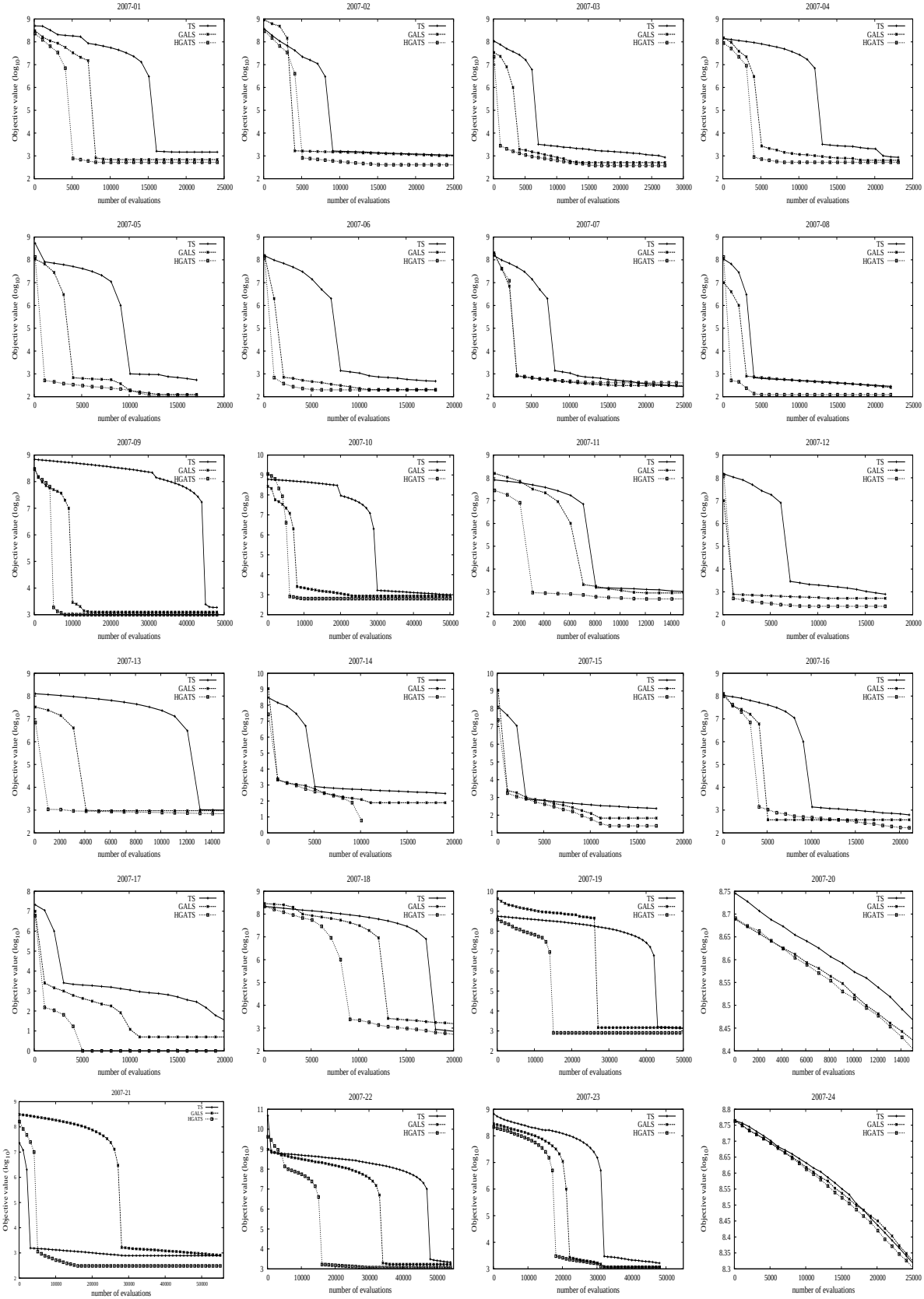


**Fig. 3** Dynamic performance of algorithms on different problem instances.

**Table 4** The t-test values of comparing algorithms on different problem instances

| PECTP | Df | | | SCP | | |
|---|---|---|---|---|---|---|
| | HGATS−GALS | HGATS−TS | GALS−TS | HGATS−GALS | HGATS−TS | GALS-TS |
| 2007-1 | − | $s+$ | $s+$ | + | $s+$ | $s+$ |
| 2007-2 | $s+$ | $s+$ | $s+$ | $s+$ | $s+$ | + |
| 2007-3 | ∼ | + | + | $s+$ | $s+$ | $s+$ |
| 2007-4 | + | $s+$ | $s+$ | $s+$ | $s+$ | + |
| 2007-5 | ∼ | ∼ | ∼ | + | + | + |
| 2007-6 | $s+$ | $s+$ | $s+$ | $s+$ | $s+$ | + |
| 2007-7 | ∼ | ∼ | ∼ | + | $s+$ | $s+$ |
| 2007-8 | $s+$ | $s+$ | + | + | $s+$ | + |
| 2007-9 | + | ∼ | − | + | $s+$ | $s+$ |
| 2007-10 | ∼ | ∼ | ∼ | $s+$ | $s+$ | + |
| 2007-11 | $s-$ | $s+$ | $s+$ | $s+$ | $s+$ | + |
| 2007-12 | ∼ | $s+$ | $s+$ | $s+$ | $s+$ | + |
| 2007-13 | ∼ | $s+$ | $s+$ | + | $s+$ | $s+$ |
| 2007-14 | ∼ | $s+$ | $s+$ | $s+$ | $s+$ | $s+$ |
| 2007-15 | ∼ | ∼ | ∼ | $s+$ | $s+$ | $s+$ |
| 2007-16 | ∼ | ∼ | ∼ | $s+$ | $s+$ | $s+$ |
| 2007-17 | ∼ | ∼ | ∼ | + | $s+$ | $s+$ |
| 2007-18 | ∼ | ∼ | ∼ | + | $s+$ | + |
| 2007-19 | + | + | + | $s+$ | $s-$ | $s-$ |
| 2007-20 | + | + | + | $s-$ | $s-$ | + |
| 2007-21 | $s+$ | $s+$ | $s+$ | $s-$ | + | $s+$ |
| 2007-22 | $s+$ | $s-$ | $s-$ | $s+$ | $s+$ | $s+$ |
| 2007-23 | $s+$ | ∼ | $s+$ | $s+$ | $s+$ | $s+$ |
| 2007-24 | $s+$ | ∼ | $s-$ | + | $s+$ | + |

**Table 5** Percentage of feasible solutions obtained by HGATS after phase I and phase II over 50 runs on each ITC-2007 problem instances

| | Problem Instances | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HGATS Phase | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Phase I (GSGA) | 52% | 64% | 92% | 96% | 82% | 78% | 78% | 100% | 48% | 52% | 100% | 80% |
| Phase II (TS) | 92% | 100% | 100% | 98% | 100% | 100% | 94% | 100% | 82% | 100% | 100% | 96% |
| | Problem Instances | | | | | | | | | | | |
| HGATS Phase | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Phase I (GSGA) | 86% | 86% | 100% | 90% | 100% | 100% | 20% | 26% | 30% | 48% | 48% | 50% |
| Phase II (TS) | 100% | 100% | 100% | 100% | 100% | 100% | 54% | 68% | 94% | 70% | 96% | 100% |

when the first algorithm is significantly better than, significantly worse than, insignificantly better than, insignificantly worse than, or statistically equivalent to the second algorithm, respectively.

From Table 4, it can be seen that the performance of HGATS is significantly better than the performance of the other two algorithms on most problem instances. It can also been observed that the performance of GALS is significantly better than the performance of TS on most problem instances. This result indicates that a single heuristic is not enough for solving a PECTP. These results show that the integration of proper LS with guided search techniques can greatly improve the performance of GAs for the PECTP.

In order to show the benefit of introducing the second phase (i.e., the TS heuristic) in HGATS, we also recorded the percentage of feasible solutions obtained by HGATS after Phase I and Phase II over 50 runs on each ITC-2007 PECTP instances. The results are shown in Table 5. From Table 5, it can be seen that the TS heuristic is beneficial to the performance of HGATS on most test PECTP instances.

### 5.3 Comparison with algorithms from the literature

In this section, in order to justify the performance of our proposed algorithm, we compare the experimental results of HGATS with the available results of other algorithms on the ITC-2007 PECTP instances. Another reason for comparing our results to the available results is that we are interesting to see the behaviour of GAs for highly constraint PECTPs among different heuristic and optimization methods, which has not been investigated yet in the literature. The algorithms compared are described as follows:

– HGATS: The hybrid approach proposed in this paper.

**Table 6** Comparison of algorithms on different problem instances

| | HGATS | | CTI | | MMA | | HA | | ACO | | LSA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PECTP | df | BSCP | df | BSCP | df | BSCP | df | BSCP | df | BSCP | df | BSCP |
| 2007-1 | 0 | 523 | 0 | 61 | 0 | 571 | 0 | 1482 | 0 | 15 | 0 | 1861 |
| 2007-2 | 0 | 342 | 0 | 547 | 0 | 993 | 0 | 1635 | 0 | 0 | 39 | 2174 |
| 2007-3 | 0 | 379 | 0 | 382 | 0 | 164 | 0 | 288 | 0 | 391 | 0 | 272 |
| 2007-4 | 0 | 234 | 0 | 529 | 0 | 310 | 0 | 385 | 0 | 239 | 0 | 425 |
| 2007-5 | 0 | 0 | 0 | 5 | 0 | 5 | 0 | 559 | 0 | 34 | 0 | 8 |
| 2007-6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 851 | 0 | 87 | 0 | 28 |
| 2007-7 | 0 | 0 | 0 | 0 | 0 | 6 | 0 | 10 | 0 | 0 | 0 | 13 |
| 2007-8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 6 |
| 2007-9 | 0 | 1102 | 0 | 0 | 0 | 1560 | 0 | 1947 | 0 | 0 | 162 | 2733 |
| 2007-10 | 0 | 515 | 0 | 0 | 0 | 2163 | 0 | 1741 | 0 | 0 | 161 | 2697 |
| 2007-11 | 0 | 246 | 0 | 548 | 0 | 178 | 0 | 240 | 0 | 547 | 0 | 263 |
| 2007-12 | 0 | 241 | 0 | 869 | 0 | 146 | 0 | 475 | 0 | 32 | 0 | 804 |
| 2007-13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 675 | 0 | 166 | 0 | 285 |
| 2007-14 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 864 | 0 | 0 | 0 | 110 |
| 2007-15 | 0 | 0 | 0 | 379 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 |
| 2007-16 | 0 | 0 | 1 | 91 | 0 | 2 | 0 | 1 | 0 | 41 | 0 | 132 |
| 2007-17 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 5 | 0 | 68 | 0 | 72 |
| 2007-18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 26 | 0 | 70 |
| 2007-19 | 0 | 121 | 267 | 1862 | 0 | 1824 | 0 | 1868 | 0 | 22 | 197 | 2268 |
| 2007-20 | 0 | 304 | 0 | 1215 | 0 | 445 | 0 | 596 | 665 | 2735 | 0 | 878 |
| 2007-21 | 0 | 36 | 0 | 0 | 0 | 0 | 0 | 602 | 0 | 33 | 0 | 40 |
| 2007-22 | 0 | 1154 | 0 | 0 | 0 | 29 | 0 | 1364 | 0 | 0 | 0 | 889 |
| 2007-23 | 0 | 963 | 0 | 430 | 0 | 238 | 0 | 688 | 11 | 1275 | 0 | 436 |
| 2007-24 | 0 | 274 | 0 | 720 | 0 | 21 | 0 | 822 | 0 | 30 | 0 | 372 |

- Mixed meta-heuristic approach (MMA): In the paper [18], Hadrien *et al.* proposed the MMA, which includes TS and simulated annealing used in conjunction with various neighborhood operators.
- CTI: Mitsunori *et al.* [12] proposed a technique that is the combination of a general purpose constraint satisfaction solver, TS, and iterated LS techniques.
- Hybrid algorithm (HA): In the paper [20], Chiarandini *et al.* proposed a HA that combines a constructive procedure for achieving the feasibility, followed by LS and simulated annealing for satisfying the soft constraints.
- ACO: In the paper [47], Nothegger *et al.* proposed an ACO algorithm in conjunction with a local improvement search routine.
- LS based algorithm (LSA): Müller [46] used an LSA with routines taken from the Constraint Solver Library. Various neighborhood search algorithms are also used to eliminate violations of hard and soft constraints.

Table 6 gives the comparison results, where the term "Df" represents the distance to feasibility and "BSCP" means the best SCP value over 10 runs. One thing to note is that the ITC-2007 competition results of other algorithms were based on 10 runs per instance. For fair comparison, we also show our results based on 10 runs per instance here.

From Table 6, it can be seen that our proposed HGATS achieved the feasibility on all of the problem instances over 10 runs. It can also be seen that the chance of HGATS getting optimal solutions is higher than other algorithms. HGATS achieved the optimal solution on 10 out of 24 problem instances. It gives the best result on problem instances 2007-4, 2007-5, 2007-16, and 2007-20 over all the compared algorithms. From the results, we can see that the guided search strategy and appropriate combination of local and tabu search approaches can help to minimize the objective values and give better results for the PECTP compared to other population-based and heuristic-based algorithms in the literature.

## 6 Conclusion and future work

This paper presents a hybrid approach, which combines a guided search genetic algorithm (GSGA) and a tabu search heuristic, to solve the post enrolment course timetabling problem (PECTP). In the GSGA, a guided search strategy uses a data structure to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and has a zero penalty value. This data structure is used to guide the generation of offspring into the next population. The main advantage of this data structure lies in that it provides

parts of former good solutions, which otherwise would have been lost in the selection process, and reuses the stored information in the following generations. This can enable the algorithm to quickly retrieve the best solutions corresponding to the previous and new populations. In the proposed HGATS algorithm, two LS techniques are used to improve the quality of individuals through searching six neighborhood structures. As the second phase of HGATS, a TS scheme is used to further improve the best solution obtained by GSGA in the first phase.

In order to test the performance of HGATS for the PECTP, experiments were carried out to analyze the sensitivity of parameters and the effect of the guided search strategy for the performance of HGATS based on a set of benchmark ITC-2007 PECTP instances. The experimental results of HGATS were also compared with several state-of-the-art methods from the literature on these benchmark ITC-2007 PECTP instances. The experimental results show that the proposed hybrid algorithm is competitive and work well across all test PECTP instances in comparison with other approaches studied in the literature. Generally speaking, with the help of the guided, local and tabu search strategies, HGATS is able to efficiently find optimal or near-optimal solutions for the PECTP and hence can act as a powerful tool for the PECTP.

There are several works to pursue in the future. One future work will be to further analyze the neighborhood techniques toward the performance of HGATS and make it more powerful for highly constrained problem. We also intend to test our approach particularly on ITC-2007 examination competition benchmarks and other problem instances that are available in the literature. Improvement of genetic operators and new neighborhood techniques based on different problem constraints will also be investigated. We believe that the performance of HGATS for the PECTP can be improved by applying advanced genetic operators and heuristics.

**Acknowledgements** The authors would like to thank the anonymous reviewers for their thoughtful comments and constructive suggestions. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/01 and Grant EP/E060722/02.

## References

1. Abdullah, S., Burke, E. K., & McCollum, B. (2005). An investigation of variable neighborhood search for university course timetabling. *Proc. of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, 413–427.
2. Abdullah, S., Burke, E. K., & McCollum, B. (2007). Using a randomised iterative improvement algorithm with composite neighborhood structures. *Proc. of the 6th Int. Conf. on Metaheuristic*, 153–169.
3. Abdullah, S., Shaker, K., McCollum, B., & McMullan, P. (2010). Incorporating Great Deluge with Kempe Chain Neighborhood Structure for the Enrolment-based Course Timetabling Problem. *The Fifth International Conference on Rough Set and Knowledge Technology*, LNAI 6401, 70–77.
4. Abdullah, S., Turabieh, H., McCollum, B., & McMullan, P. (2010). A Multi-objective Post Enrolment Course Timetabling Problems: A New Case Study. *IEEE Congress on Evolutionary Computation*,Barcelone, Spain.
5. Abdullah, S., & Turabieh, H. (2008). Generating university course timetable using genetic algorithm and local search. *Proc. of the 3rd Int. conf. on Hybrid Information Technology*, 254–260.
6. Abramson, D. (1991). Constructing school timetables using simulated annealing: equential and parallel algorithms. *Management Science*, 37(1), 98–113.
7. Acan, A. (2004). An External Memory Implementation in Ant Colony Optimization. *Proc. of the 4th Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, 73–82.
8. Acan, A. & Tekol, Y. (2003). Chromosome Reuse in Genetic Algorithms *Proc. of the 2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*, 695–705.
9. Aladğ, Ç. H., & Hocaoğlu, G. (2007). A tabu search algorithm to solve a course timetabling problem. *Hacettepe Journal of Mathematics and Statistics*, 36(1), 53–64.
10. Al-Betar, M. A., Khader, A. T., & Gani, A. T. (2007). A harmony search algorithm for university course timetabling. *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling.*
11. Atkin, J. A., Burke, E. K., Greenwood, J., & Reeson, D. (2007). Hybrid meta-heuristics to aid runway scheduling at london heathrow airport. *Transportation Science*, 41(1), 90–106.
12. Atsuta, M., Nonobe, K., & Ibaraki, T. (2008). ITC2007 Track 2, An Approach using general CSP solver. *www.cs.qub.ac.uk/itc2007*
13. Bonissone, P. P., Subbu, R., Eklund, N., & Kiehl, T. R. (2006). Evolutionary algorithms + domain knowledge = real-world evolutionary computation. *IEEE Transactions on Evolutionary Computation*, 10(3), 256–280.
14. Burke, E. K., Causmaecker, P. D., Berghe G. V., & Landeghem, H. V. (2004). The state of the art of nurse rostering *Journal of Scheduling* , 7(6), 441-499
15. Burke, E. K., Elliman, D. G., & Weare, R. F. (1995). A hybrid genetic algorithm for highly constrained timetabling problems. *Proc. of 6th Int. Conf. on Genetic Algorithms*, 605–610.
16. Burke, E. K., Kendall, G., & Soubeiga, E. (2003). A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(6), 451–470.
17. Burke, E. K., & Petrovic, S. (2002). Recent research directions in automated timetabling. *European Journal of Operation Research*, 140(2), 266–280.
18. Cambazard, H., Hebrard, E., OŚullivan, B., & Papadopoulos, A. (2008). Local search and constraint programming for the post enrolment-based course timetabling problem. *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT 2008).*
19. Carter, M. W., & Laporte, G. (1998). Recent developments in practical course timetabling. *Proc. of the 2nd Int. Conf. on Practice and Theory of Automated Timetabling*, LNCS 1408, 3–19.

20. Chiarandini, M., Fawcett, C., & Hoos, H. H. (2008). A modular multiphase heuristic solver for post enrollment course timetabling. *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT 2008)*.
21. Chiarandini, M., Birattari, M., Socha, K., & Rossi-Doria, O. (2006). An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5), 403–432.
22. Colorni, A., Dorigo, M., & Maniezzo. (1990). Genetic algorithms - A new approach to the timetable problem. *NATO ASI Series, Combinatorial Optimization*, LNCS, F(82), 235–239.
23. Chu, S. C., & Frang, H. L. (1999) Genetic algorithm vs. tabu search in timetabling scheduling *Proc. of the 3rd Int. Conf. on Knowledge-Based Intelligent Information Engineering System*.
24. Datta, D., Deb, K., & Fonseca, C. M. (2007). Multi-objective evolutionary algorithm for university class timetabling problem. In Dahal, K. P., Tan, K. C., & Cowling, P. I. (eds.), *Evolutionary Scheduling*, Springer, 197–236.
25. Erben, W., & Keppler, J. (1995). A genetic algorithm solving a weekly course timetabling problem. *Proc. of the 1st Int. Conf. on Practice and Theory of Automated Timetabling*, LNCS 1153, 198–211.
26. Even, S., Itai, A., & Shamir, A. (1997). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4), 691–703.
27. Freisleben, B., & Merz, P. (1996). A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. *Proc. of IEEE Int. Conf. on Evolutionary Computation*, 616–621.
28. Gaspero, L. D., & Schaerf, A. (2001) Tabu search eechniques for examination timetabling. *Practice and Theory of Automated Timetabling III*, Springer, LNCS 2079, 104–117.
29. Gen, M., & Cheng, R. (1997). *Genetic Algorithms and Engineering Design*. Wiley-IEEE.
30. Glover, F., & Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.
31. Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.
32. Gotlie, C. C. (1963). The construction of class-teacher timetables. *IFIP Congress*, 62, 73–77.
33. Gunadhi, H., Anand, V. J., & Yong, Y. W. (1996) Automated timetabling using an object-oriented scheduler *Expsert Systems with Applications*, 10(2), 243–256.
34. Hageman, J. A., Wehrens, R., Sprang, H. A., & Buydens, L. M. C. (2003). Hybrid genetic algorithmtabu search approach for optimizing multilayer optical coatings. *Analytica Chimica Acta*, 490, 211–222.
35. Jat, S. N., & Yang, S. (2009). A guided search genetic algorithm for the university course timetabling problem. Proc. of the 4th Multidisciplinary Int. Scheduling Conf: Theory and Applications, 180–191.
36. Jat, S. N., & Yang, S. (2008). A memetic algorithm for the university course timetabling problem. Proc. of the 20th IEEE Int. Conf. Tools with Artif. Intell., 427–433.
37. Kendall, G., Knust, S., Ribeiro, C. C., & Urrutia, S. (2010). Scheduling in Sports: An Annotated Bibliography *Computers and Operations Research*, 37(1), 1–19.
38. Knauer, B. A. (1974). Solutions of a timetable problem *Computers and Operations Research*, 1(3-4), 363–375.
39. Lewis, R. (2008). A survey of metaheuristic based techniques for university timetabling problems. *OR Spectrum*, 30(1), 167–190.
40. Lewis, R., & Paechter, B. (2005). Application of the grouping genetic algorithm to university course timetabling. *Proc. of the 5th European Conf. on Evol. Comput. in Combinatorial Optimization (EvoCOP 2005)*, LNCS 3448, 144–153.
41. Lewis, R., Paechter, B., & McCollum, B. (2007). Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. *Technical Report*, Cardiff University, 2007.
42. Liu, Y. H. (2010). Different initial solution generators in genetic algorithms for solving the probabilistic traveling salesman problem. *Applied mathematics and computation*, 216(1), 125–137.
43. Louis, S., & Li, G. (1997). Augmenting genetic algorithms with memory to solve traveling salesman problem. *Proc. of the 1997 Joint Conference on Information Sciences*, 108–111.
44. Lü, Z., & Hao, J. K. (2010). Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1), 235–244.
45. Malim, M. R., Khader, A. T., & Mustafa, A. (2006). Artificial Immune Algorithms for University Timetabling. In Burke, E. K., and Rudova, H. (Eds.), *Proc of the 6th Int. Conf. on Practice and Theory of Automated Timetabling*, 234–245.
46. Müller, T. (2008). ITC2007 Solver Description: A Hybrid Approach *Proc. of the 7th Int. Conf. on the Practise and Theory of Automated Timetabling (PATAT 2008)*.
47. Nothegger, C., Mayer, A., Chwatal, A., & Raidl, G. (2008). Solving the post enrolment course timetabling problem by ant colony optimization *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT 2008)*.
48. Pongcharoen, P., Promtet, W., Yenradee, P., & Hicks, C. (2008). Schotastic optimization timetabling tool for university course scheduling. *International Journal of Production Economics*, 112, 903–918.
49. Prestwich, S., Tarim, A., Rossi, R., & Hnich, B. (2008). A steady-state genetic algorithm with resampling for noisy inventory control. *Proc. of the 10th Int Conf on Parallel problem solving from Nature*, LNCS 5199, 559–568.
50. Qu, R., Burke, E. K., McCollum, B., & Merlot, L. T. G. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1), 55–89.
51. Rossi-Doria, O., Sampels, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., & Stützle, T. (2002). A comparison of the performance of different metaheuristics on the timetabling problem. *Lecture Notes in Computer Science 2740*, 329–351.
52. Rossi-Doria, O., & Paechter, B. (2004). A memetic algorithm for university course timetabling. *Proc. of Combinatorial Optimization (CO 2004)*, 56.
53. Sastry, K., Goldberg, D., & Kendall, G. (2005). Genetic algorithms. In Burke, E. K., and Kendall, G. (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Chapter 4, 97–125, Springer.
54. Schearf, A. (1999). A survey of automated timetabling. *Artificial Intelligence Review*, 13(2), 87–127.
55. Sigl, B., Golub, M., & Mornar, V. (2003). Solving timetable scheduling problem using genetic algorithms. *Proc. of the 25th Int. Conf. on Information Technology Interfaces*, 519–524.
56. Socha, K., Knowles, J., & Samples, M. (2002). A max-min ant system for the university course timetabling problem. *Proc. of the 3rd Int. Workshop on Ant Algorithms (ANTS)*, LNCS 2463, 1–13.
57. Thanh, N. D. (2006). Solving timetabling problem using genetic and heuristics algorithms *Journal of Scheduling*, 9(5), 403–432.
58. Tuga, M., Berretta, R., & Mendes, A. (2007) A Hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. *Proc. of the 6th IEEE/ACIS Int. Conf. on Computer and Information Science*, 400–405.

59. Turabieh, H. & Abdullah, S. (2009) Incorporating Tabu Search into Memetic Approach for enrolment-based course timetabling problems. *2nd Data Mining and Optimization Conference*, 122–126.
60. Werra, D. (1986). An introduction to timetabling *European Journal of Operational Research*, 19(2), 151–162.