Applied Intelligence (2021) 51:467–492
https://doi.org/10.1007/s10489-020-01833-x

A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search

Amin Rezaeipanah[1] · Samaneh Sechin Matoori[2] · Gholamreza Ahmadi[3]

Published online: 19 August 2020

Abstract
Scheduling is one of the problems that has attracted the attention of many researchers over the years. The University Course Timetabling Problem (UCTP) is a highly constrained real-world combinatorial optimization task. Designing course timetables for academic institutions has always been challenging, because it is a non-deterministic polynomial-time hardness (NP-hard) problem. This problem attempts to assign specific timeslots and rooms to the events considering a number of hard and soft constraints. All hard constraints must be satisfied to achieve a feasible solution, whereas satisfying all soft constraints is not necessary. Although the quality of the solution is directly related to the number of soft constraints that are satisfied. One of the recent innovative methodologies for solving UCTP is the hybrid algorithm, which attempts to automate the timetabling design process so that it would be able to work with different instances of problem domains. In this paper, we present a hybrid method based on the Improved Parallel Genetic Algorithm and Local Search (IPGALS) to solve the course timetabling problem. The Local Search (LS) approach is used to strengthen the Genetic Algorithm (GA). The IPGALS has applied a representation of the timetable, which ensure the hard constraints would never be violated. Hard constraints are measured by Distance to Feasibility (DF) criterion. In fact, applying the DF criterion leads to achieving feasible solutions and promotes the performance of our algorithm. Due to the wide range of problem constraints, the proposed algorithm is performed in parallel to improve the GA searching process. The IPGALS algorithm is tested over BenPaechter and ITC-2007 standard benchmarks and compared with the state-of-the-art techniques in this literature. The experimental results confirm the effectiveness and the superiority of the proposed algorithm compared to other prominent methods for solving UCTP.

Keywords Genetic algorithm · Local search · University course timetabling problem · Distance to feasibility

1 Introduction

The scheduling and timetabling of the courses in universities is a multi-stage process which requires proper coordination and communication among several groups. The planning groups of universities often consist of students, instructors, department head and education officer [1]. Basically, there are various constraints in a University Course Timetabling Problem (UCTP) [2–5]. Some of these constraints must necessarily be satisfied, which are called hard constraints. On the contrary, another category of constraints known as soft constraints helps to improve the timetabling quality [6]. Therefore, satisfying soft constraints leads to approach the optimal solution. The UCTP attempts to conduct a conflict-free assignment of all classes and laboratory sessions to instructors, rooms, and timeslots considering the hard and soft constraints [7]. Each institution pursues unique sets of hard and soft constraints based on its resources and facilities.

Generally, three categories of education timetabling are considered such as school timetabling [8, 9], course timetabling [10, 11], and examination timetabling [12, 13]. All above-mentioned categories resemble in some aspects, but they mainly vary in their unique constraints that require

✉ Amin Rezaeipanah
amin.rezaeipanah@gmail.com

Samaneh Sechin Matoori
samanehmatoori@gmail.com

Gholamreza Ahmadi
grahmadi@pgu.ac.ir

1 Department of Computer Engineering, University of Rahjuyan Danesh Borazjan, Bushehr, Iran

2 Department of Computer Engineering, Bushehr Branch, Islamic Azad University, Bushehr, Iran

3 Department of Computer Engineering, Persian Gulf University, Bushehr, Iran

to be satisfied. In this paper, course timetabling is investigated in which courses/events are assigned to certain timeslots and rooms such that a set of requirements and constraints are satisfied. This problem bears a high resemblance to the graph coloring problem in which an identical color could not be assigned to two neighbor nodes (i.e. connected by an edge) [14]. Formerly, the university timetables were manually formed. Needless to mention that forming a course timetable manually is a highly time-consuming and complicated task. Nonetheless, it is mostly unable to form a conflict-free timetable even after several reforming iterations. The UCTP is a hybrid optimization problem and has been shown to be NP-hard [10–12]. The complexity of UCTP is due to the diversity of requirements of different institutions. Additionally, various approaches have been proposed to address the problem due to its importance [15, 16]. Most of these approaches use optimization methods to create an automatic timetable that is able to produce an optimal or best near-optimal schedule. However, there are still opportunities to generate more solid results.

The BenPaechter and ITC-2007 instances have been widely used as UCTP benchmarks for algorithmic comparison [16]. In this literature, numerous approaches have been presented on these instances to form course timetables so far. Scheduling tasks including UCTP are recently conducted applying various exact and meta-heuristic algorithms [17, 18]. Integer Linear Programming (ILP) settles in the category of exact algorithms which applies mathematical models to solve an objective function. ILP is capable of solving scheduling problems including the UCTP [17]. In addition, a large number of researches have discussed various meta-heuristic techniques for the UCTP. Some of the meta-heuristic approaches are Hybrid Evolutionary Algorithm [19], Hybrid GA with LS [20], Hybrid Evolutionary Approach with Nonlinear Great Deluge [21] and Hybrid Electromagnetism-like Mechanism with Great Deluge [22]. The evolutionary and heuristic algorithms suffer from two primary drawbacks such as early convergence to and getting stuck in the local optimum. Hence, hybrid algorithms are presented to cope with these drawbacks [10]. In fact, hybrid algorithms are a kind of approximate optimization algorithms that utilize mechanisms for escaping the local optimum and preventing early convergence as well [23].

In this paper, a hybrid approach based on the Improved Parallel Genetic Algorithm (GA) and Local Search (LS) is developed for solving UCTP, which we briefly called IPGALS. Due to the simultaneous investigation of several solutions and random survey of case space, which is suitable for the scheduling problem, the basis of the proposed hybrid algorithm in this study is the GA. We propose a two-stage method to deal with the UCTP. In the first stage, feasible solutions are generated. In the second stage, the algorithm attempts to improve the generated solutions progressively. The IPGALS used a representation of the timetable, which ensure the hard constraints would never be violated. In fact, hard constraints are measured by Distance to Feasibility (DF) criterion. GA tries to find feasible solutions based on the DF criterion. Then, we improve the feasible solutions in terms of soft constraints violation by using intelligent and elitist operators. Here, the LS approach is used to strengthen the GA. In addition, due to the wide range of problem constraints, the proposed algorithm is performed in parallel to improve the GA searching process.

The rest of the paper is organized as follows. The details of the UCTP are presented in Section 2. Section 3 is dedicated to literature review. The structure of proposed IPGALS algorithm is elaborated in Section 4 and the experimental results and discussion are presented in Section 5. Finally, Section 6 concludes and suggests possible future works.

2 The university course timetabling problem

The University Course Timetabling Problem (UCTP) is a classic and famous problem in the field of optimization problems. This problem is an abstraction of a real-world timetabling problem. The purpose of UCTP is to schedule a number of events (courses) in proper timeslots and rooms. This problem consists of a set of events to be scheduled in a number of timeslots, a set of rooms in which events can take place, a set of the attended students in each event, and a set of features satisfied by rooms and required by events [10]. Therefore, The UCTP is an abstraction of a real-world timetabling problem. The problem consists of a set of N_e events $E = \{e_1, e_2, \ldots, e_{N_e}\}$ scheduled in a set of 45 timeslots $T = \{t_1, t_2, \ldots, t_{45}\}$ (5 days in a week, 9 timeslots per day), and a set of N_r rooms $R = \{r_1, r_2, \ldots, r_{N_r}\}$, in which events can take place (rooms capacity is determined according to the input instances). Additionally, there is a set of N_s students $S = \{s_1, s_2, \ldots, s_{N_s}\}$, which attend in each event, and a set of N_f features $F = \{f_1, f_2, \ldots, f_{N_f}\}$, which is intended for each room and is required to satisfy each event (e.g. computer, video projector, etc.). Meanwhile, each room has a capacity, and each student attends a number of events.

In addition, there are some hard and soft constraints in the UCTP. A feasible timetable ensures that the timeslot and room are assigned to all events while all hard constraints are satisfied. In addition, the UCTP attempts to maximize the number of satisfied soft constraints because violating each soft constraint leads to a penalty.

2.1 Constraints

Constraints in UCTP are classified into two hard and soft constraints. All hard constraints must be satisfied, so that the generated solution would be feasible. Soft constraints are related to the objective function. The objective function attempts to maximize the number of satisfied soft constraints. Unlike hard constraints, soft constraints are not necessarily required to be satisfied. However, the more the number of satisfied soft constraints increases, the more the quality of solutions of objective function promotes. In the following, a list of extracted hard and soft constraints from the literature is presented [24]. We use these constraints to solve the UCTP.

2.1.1 Hard constraints

- HC_1: No student can have more than one enrolled course in a timeslot.
- HC_2: The room must be capable of satisfying the required features of the course.
- HC_3: The number of enrolled students in the course must be less than or equal to the capacity of the room.
- HC_4: No more than one course is allowed to be held during a timeslot in each room.

2.1.2 Soft constraints

- SC_1: A student should not have a single course during a day.
- SC_2: A student should not have more than two consecutive courses.
- SC_3: A student should not have a course which is scheduled in the last timeslot of the day.

The UCTP aims to form feasible timetables with the lowest penalty. Obviously, no hard constraint is violated in a feasible timetable, and the violation of soft constraint impose penalties.

2.2 Benchmarks

A set of benchmark instances from two different sources such as BenPaechter [25] and ITC-2007 Track 2 [26] is used for performance evaluation and comparison work. Although these instances have ignored multiple real-world problem constraints, they have empowered us to compare our approach with the state-of-the-art techniques. The instances in the BenPaechter benchmark include scheduling 100–400 events into a timetable with 45 timeslots (5 days and 9 h per day) as well as satisfying the room features and the room capacity constraints. These instances are divided into three categories: small, medium and large. We deal with 12 instances including five small, five medium and two large ones. We utilize ‘S1’ to ‘S5’ notations to show small instance 1 to 5 respectively, ‘M1’ to ‘M5’ to distinguish medium instance 1 to 5, and ‘L1’ and ‘L2’ notations to identify large instance 1 and 2. More details on the BenPaechter benchmark are given in Table 1. The instances with the identical characteristics (e.g. number of events, number of students, etc.), possess various internal parameters. Therefore, the time complexity of solving them is different, and they need different timetabling. The ITC-2007 benchmark is composed by real-world instances and was created by the University of Udine (Italy). This ITC-2007 benchmark contains 24 instances with 100–600 events. Similarly, we need to schedule the events into 45 timeslots. More details on the ITC-2007 benchmark are given in Table 2. In the experimental studies on this dataset, ‘C01’ to ‘C24’ notations are utilized to represent the instance COMP01 COMP24, respectively.

Tables 1 and 2 consist of nine columns in which the first column is a list of the instances and the other columns represent the specifications of scheduling resources. Specifications include the number of events, the number of rooms, the number of features and the number of students. In addition, rest of columns represent the maximum students per event, maximum events per student, mean features per room and mean features per event, respectively. The reported statistics are rounded numbers in the table.

All instances have at least one feasible solution (no hard constraint violations), although it is not known which is the optimal value for the soft constraints. Also, each instance comes in a single file, containing a file header and four sections: room’s capacity, student/event, room/feature, and event/feature. The header contains number of events, number of rooms, number of features and number of students. The capacity for each room represents the maximum number of seats. The student/event means that the student does attend the event or not. The room/feature means that the room does satisfy the feature or not, and finally, event/feature means the event does require the feature or not.

3 Literature review

In the past few years, researchers have focused on hyper-heuristic, hybridization and meta-heuristic approaches such as Harmony Search (HS) [27], Ant Colony Optimization (ACO) [28], Particle Swarm Optimization (PSO) [29], Genetic Algorithm (GA) [30], Great Deluge (GD) [31], Simulated Annealing (SA) [32], Hill Climbing (HC) [33], Local Search (LS) [2] and Tabu Search (TS) [34]. Susan and Bhutani [35], have compared the performance of SA and GA for forming the university timetables after mining the students’ preferences in selecting courses from the online repository of the university. Abuhamdah et al. [2] proposed a population-based LS algorithm for UCTP, which is a subcategory of LS (i.e. a mechanism to exploit the search space). This method implements two operators. One of them operates on a single solution to determine the force by comparing the current generating solution and the most recent generated

Table 1 Details of the BenPaechter benchmark instances

Instances	Specifications							
	Number of events	Number of rooms	Number of features	Number of students	Max. students per event	Max. events per students	Mean features per room	Mean features per event
Small1	100	5	5	80	15	15	3	2
Small2	100	5	5	80	13	17	3	2
Small3	100	5	5	80	20	13	3	2
Small4	100	5	5	80	12	12	4	3
Small5	100	5	5	80	17	19	4	3
Medium1	400	10	5	200	11	20	3	2
Medium2	400	10	5	200	11	20	3	2
Medium3	400	10	5	200	12	20	3	3
Medium4	400	10	5	200	11	20	3	2
Medium5	400	10	5	200	20	20	3	3
Large1	400	10	10	400	30	20	5	4
Large2	400	10	10	400	25	20	5	5

Table 2 Details of the ITC-2007 Track 2 benchmark instances

Instances	Specifications							
	Number of events	Number of rooms	Number of features	Number of students	Max. students per event	Max. events per students	Mean features per room	Mean features per event
COMP01	400	10	10	500	33	25	3	1
COMP02	400	10	10	500	32	24	4	2
COMP03	200	20	10	1000	98	15	3	2
COMP04	200	20	10	1000	82	15	3	2
COMP05	400	20	20	300	19	23	2	1
COMP06	400	20	20	300	20	24	3	2
COMP07	200	20	20	500	43	15	5	3
COMP08	200	20	20	500	39	15	4	3
COMP09	400	10	20	500	34	24	3	1
COMP10	400	10	20	500	32	23	3	2
COMP11	200	10	10	1000	88	15	3	1
COMP12	200	10	10	1000	81	15	4	23
COMP13	400	20	10	300	20	24	2	1
COMP14	400	20	10	300	20	24	3	1
COMP15	200	10	20	500	41	15	2	3
COMP16	200	10	20	500	40	15	5	3
COMP17	100	10	10	500	195	23	4	2
COMP18	200	10	10	500	65	23	4	2
COMP19	300	10	10	1000	55	14	3	1
COMP20	400	10	10	1000	40	15	3	1
COMP21	500	20	20	300	16	23	3	1
COMP22	600	20	20	500	22	25	3	2
COMP23	400	20	30	1000	69	24	5	3
COMP24	400	20	30	1000	41	15	5	3

solution. Another one operates on all generated solutions to determine the force in all directions.

In evolutionary-based optimization methods, GA is very popular and various GA-based strategies have been examined for UCTP. Goh et al. [36] utilized a hybrid local search algorithm which hybridizes TS and SAR (SA with Reheating) to solve the UCTP. Chen et al. [23] developed a TS algorithm with controlled randomization for solving the UCTP. Here, a random acceptation strategy with threshold mechanism is proposed to cater for subsequent algorithm design. Muklason et al. [37] proposed a method called Tabu-Variable Neighborhood Search to mitigate the drawbacks of meta-heuristic optimization methods (i.e. requiring the parameters tuning for every instance). Their method has selected heuristics at each decision point instead of directly solving the problem. Goh et al. [32] applied a highly tuned SA on the instances and reported superior results. Mazlan et al. [28] have recently utilized ACO on these instances. Their method has yielded more solid results such that it is still considered as the current state-of-the-art method. Hossain et al. [29] investigated a novel PSO-based method for solving highly constrained UCTP in which the basic PSO operations are transformed to solve combinatorial optimization task of UCTP and introduced couple of novel operations to PSO to solve UCTP efficiently. In this method, swap sequence-based velocity calculation and its application is implemented to transform each particles in order to improve them.

Matias et al. [38] investigates the GA with guided strategies and a self-adaptive mechanism to solve the UCTP. These data structures have used to guide or direct the searching process to an available resource and enhance the solutions produced by the genetic operators. In addition, a self-adaptive mechanism has integrated with choosing a neighborhood structure. Gozali and Fujimura [39] have applied the localized island model GA with dual dynamic migration policy (DM-LIMGA) for solving the UCTP. Thepphakorn and Pongcharoen [3] applied Particle Swarm Optimisation based Timetabling (PSOT) to solve the real-world datasets of the UCTP. The PSO, the standard PSO (SPSO), and the Maurice Clerc PSO (MCPSO) were embedded in the PSOT program for optimising the desirable objective function. The MCPSO outperformed the other variants of PSO for most datasets. Pintér and Dávid [5] proposed a two-stage heuristic method for solving UCTP. Here, an initial solution is created using a recursive search algorithm, then its quality is improved using local search heuristic. In addition, they use tabu list of forbidden transformations at the end of each iteration.

The hybrid approaches (the combination of evolutionary mechanisms with TS) have provided highly efficient solutions for the UCTP [40]. Yusoff and Roslan [33] proposed a hybrid GA-HC with the elitist for UCTP considering a bunch of constraints and a repair mechanism for all infeasible solutions. Akkan and Gülcü [6] proposed a hybrid bi-criteria GA algorithm for scheduling in UCTP. Gülcü and Akkan [41] exposed the problem of a robust UCTP to single and multiple disruptions. In their study, two versions of a hybrid Multi-Objective-GA (MOGA) have been developed, i.e. MOGA-Single Disruption (MOGA-SD) for problems with single disruptions and MOGA-Sample Average Approximation (MOGA-SAA) for problems with multiple disruptions. In fact, these two differ in the robustness level of the generated solution resulted in the MOGA. These algorithms aimed to identify an appropriate Pareto Archive which has been determined by the quality and robustness of the solution (imposed penalties due to soft constraints violations). A novel GA for UCTP has been presented by Susan and Bhutani [42]. They have hybridized the global search strategies of GA with the LS heuristics of SA, and implemented a greedy randomized LS mutation in GA.

Niknamian [37] has provided a survey on hybrid and meta-heuristic based approaches for UCTP. Kostuch [7] won the International Timetabling Competition 2002 (ITC-2002) using a meta-heuristic algorithm. Rezaeipanah et al. [10] proposed a parallel hybrid approach for generating initial timetables, and applied GA to improve the quality of the generated timetables. Some other hybrid approaches have been developed such as the multi-population hybrid GA in [11], the hybrid fuzzy evolutionary approach in [15] and the fuzzy-GA with LS in [16]. Hambali et al. [1] proposed a combination of GA and SA to form a heuristic approach for solving UCTP in the Federal University Wukari. Bashab et al. [4] proposed a meta-heuristic algorithm for UCTP which has combined grey-wolf optimizer and cat swarm optimization. Habashi et al. [43] suggested an adaptive diversifying meta-heuristic based approach for UCTP. In their study, a competitive iterated-LS approach has been proposed which has been strengthened by an add-delete meta-heuristic. The meta-heuristic has utilized an adaptive heuristic mechanism for generating solutions by using a variable-length list of add and delete operations.

4 IPGALS algorithm for UCTP

It is very difficult to find a general and effective solution for UCTP due to the diversity of the problem, variation of constraints, and different needs in universities and institutions planning. Despite presenting various approaches, many researchers use hybrid methods for solving the UCTP [6, 11, 15]. In addition, extensive researches have shown that GAs is an effective method for solving the UCTP [44–47]. The GA is a population-based algorithm which explores the entire search space without concentrating on the individuals bearing the best fitness function within a population. The GA may not always be able to find the optimal solution due to the possibility of premature convergence and getting stuck into local optima. Therefore, LS applies changes in the generated solutions. In fact, it performs local changes in a valid solution to achieve another valid solution in the search space until certain conditions are satisfied. The process iterated until achieving an optimal solution or in a time-bound. The hybrid of GA and LS (GALS) improves the performance by properly adjusting the global search [20, 33]. Therefore, GA is a well-suited tool for solving the UCTP which has increasingly attracted the attention of researchers.

The key superiority of GA is the searching for the solution with a population of solutions. However, GA requires an improvement due to random searches within all the problem space. This has been inferred after analyzing the solutions yielded from applying the classic GA on different instances (i.e. instances with a large number of events). Therefore, we have proposed a hybrid parallel method to solving the UCTP using the GA and LS and called it IPGALS. The IPGALS is an improved GA algorithm with a parallel architecture, where it is combined with LS for better convergence. Here, GA has been developed with some reforms such as initial population creation, offspring production, parallel structure and some intelligent operators. Meanwhile, the proposed structure for timetables does not violate any hard constraints in UCTP. Hard constraints are measured by Distance to Feasibility (DF) criterion in IPGALS. This leads to achieving feasible solutions and promoting the performance of our algorithm.

Parallel Genetic Algorithm (PGA) is such an algorithm that uses multiple GAs to solve a single task [10]. In the paper, we use 'island model' for PGA, because populations are isolated from each other. In the PGA, each algorithm performs over its population. Accordingly, the individuals of populations may differ from one conducting algorithm to another. Therefore, PGA may take a little more time than a non-parallel one. For that reason, PGA solutions are expected to produce more solid results than non-parallel GA. In addition, if the population of an algorithm does not progress after a number of generations, chromosomes are exchanged between algorithms. Higher genetic diversity, finding the global optima with higher probability, accelerating convergence, search in high-dimensional spaces, and so on, are the advantages of this work. The architecture of IPGALS algorithm is depicted in Fig. 1.

In the following, each component of the proposed IPGALS algorithm is described in details.

4.1 Genetic algorithm

Researchers have applied numerous techniques to find the most suitable and fastest way of solving UCTP. In proposed IPGALS algorithm, timetables are formed through combination of an improved PGA and LS. Several studies have shown that GA is an effective and executable method for solving UCTP [44–47]. In computer science, GA is a search heuristic that is inspired by Charles Darwin's theory of natural evolution [33]. This algorithm inspired by the natural selection of chromosomes in which the fittest chromosomes candidates are combined to produce subsequent offspring. The GA and developed models of it have been successfully applied to solve a number of scheduling problems such as UCTP [10, 15]. We have applied the IPGALS to take the advantages of both GA and LS. Here are the details of the improved GA.

4.1.1 Chromosome representation

Each chromosome in the GA population is represented by a chromosome representation. The representation scheme reflects the structure of the GA problem and either does determine the applied genetic operators. The proposed chromosome representation includes a two-dimensional vector of gene sequences. The proposed chromosome length in UCTP is related to the number of events. In addition, each event in chromosome structure includes two genes characterized by timeslot (event time) and room number (event location). Here, chromosome representation is subtly designed to make the use of genetic operators very simple. Structure of the proposed chromosome is shown in Fig. 2, where $t_i \in T$ and $r_i \in R$ represent the *j*-th timeslot and *k*-th room number assigned to *i*-th event, respectively.

4.1.2 Initial population

Population initialization is the first step in the GA Process. Population is a subset of chromosomes in the current generation. Generally, chromosomes in GA are generated randomly or by heuristic methods from the search space. The initial population for the GA is usually generated randomly. Applying a random method for generating the initial population often leads to producing the infeasible solutions due to the lack of satisfying the problem constraints. In fact, the infeasible solutions are achieved due to the absence of any preconditions to generate the population. In the following, we will describe a three-step approach to generate an initial population. This approach generates the initial population based on a random-heuristic method.

– *Step 1:* Generate all states to set an event (gene) of the chromosome. All states are determined by the timeslots and the room's number. For example, all states of a timetable with 5 timeslots and 3 rooms are 15 (3×5). To better understand this issue, see Fig. 3. Each state represents a gene from the chromosome that could be assigned to an event.
– *Step 2:* Shuffling technique assists to promote evolution, the genetic diversity and population variety [10]. Therefore, all the states generated in the first step are shuffled for producing each new chromosome. This is done in order to select chromosomes from the entire

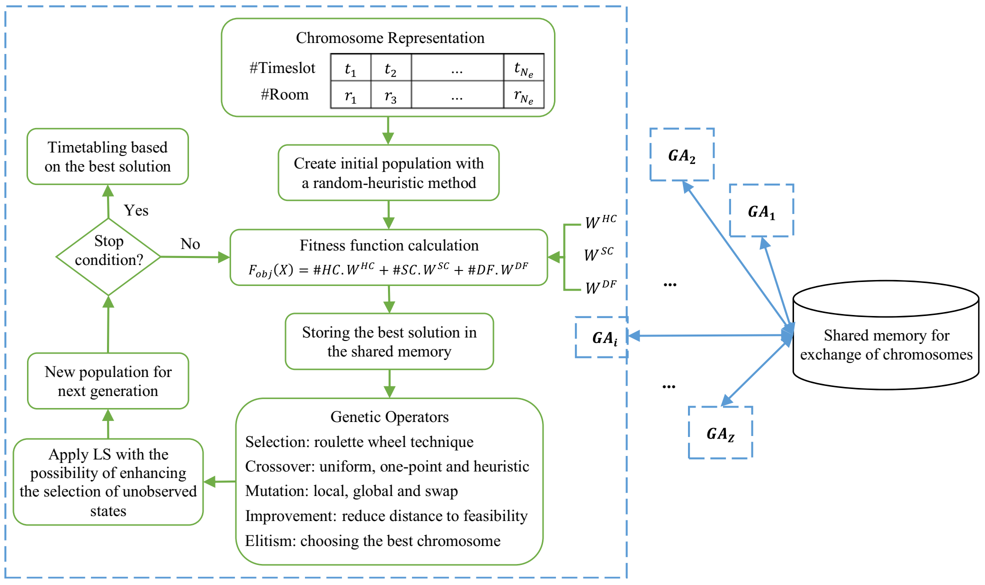

Chromosome Representation
#Timeslot
#Room
t_1 t_2 … t_{N_e}
r_1 r_3 … r_{N_e}
Create initial population with a random-heuristic method
Fitness function calculation
$F_{obj}(X) = \#HC.W^{HC} + \#SC.W^{SC} + \#DF.W^{DF}$
W^{HC}
W^{SC}
W^{DF}
Storing the best solution in the shared memory
Genetic Operators
Selection: roulette wheel technique
Crossover: uniform, one-point and heuristic
Mutation: local, global and swap
Improvement: reduce distance to feasibility
Elitism: choosing the best chromosome
Apply LS with the possibility of enhancing the selection of unobserved states
New population for next generation
Stop condition?
No
Yes
Timetabling based on the best solution
GA_2
GA_1
GA_i
GA_Z
Shared memory for exchange of chromosomes

Fig. 1 The architecture of the IPGALS algorithm

search space and prevent generating of duplicated chromosomes. Preventing the production of duplicate chromosomes is accomplished by considering the value of an event according to the set of possible states for that event. Meanwhile, all states are shuffled to generate each new chromosome. In addition, this technique adds a random property to the production of the initial population. Figure 4 shows the effect of using the shuffling technique of the given example in Fig. 3.

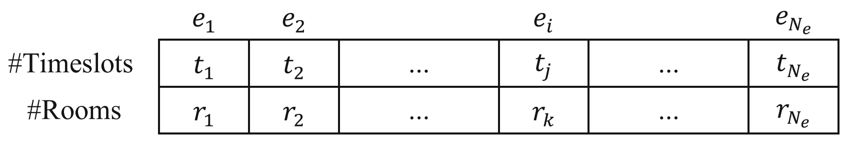
– *Step 3:* Here, each event of the chromosome is assigned a state (among the states of the second step). The proposed heuristic method provides the best state for each event. Hence, it is necessary to select a subset of all states assigned to an event. Finally, a state among the subsets of the selected states is assigned to the event. The subsets of states for each event is selected so that all hard constraints are not violated in the chromosome. Accordingly, all the states for each event are investigated. The first state with no violated hard constraint is assigned to the event. Then, the selected state is eliminated from the set of all states, since it should not be selected for other events. That way, a subset of states is determined for other events. However, it may not be acceptable state for some events, especially in the final events of the chromosome. Therefore, some events may violate hard constraint by selecting any remaining state. In this paper, these events are denoted by the '-1' symbols, and we call them conflicting events. Figure 5 shows an example of a chromosome with six events according to Fig. 4.

In this example, 10th and 14th states are assigned to first and second events. Since selecting any remaining state for the third event violates the hard constraint, this event is set to '-1' symbol. Also, the fourth and fifth events are set to 6th and 15th states. The eighth state with content {2, 1} (2 for timeslot and 1 for room number), violates the hard constraint in the sixth event. Therefore, it is impossible to assign this state to the sixth event. Here, the sixth event has been set to third state (with content {1, 3}).

4.1.3 Fitness function

The input data for each instance of the problem include number of events, number of students, number of features, room's capacity, attended students in each event, features required for each event and features satisfied by rooms. Based on the data of each instance as well as the constraints defined in Section 2.1, the fitness function for each chromosome is calculated. A fitness function is a particular type of objective function and shows how close a given design solution is to achieving the set aims. Here, each chromosome contains a set of events and each event that causes the constraints violation penalizes the fitness function. At first, the constraints violation

Fig. 2 The structure of proposed chromosome representation



	e_1	e_2		e_i		e_{N_e}
#Timeslots	t_1	t_2	…	t_j	…	t_{N_e}
#Rooms	r_1	r_2	…	r_k	…	r_{N_e}

Fig. 3 All states of a timetable with 5 timeslots and 3 rooms

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
#Timeslots	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5
#Rooms	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3

in the solution X is calculated based on the fitness function defined in the Eq. (1) [44, 45].

$$F_{obj}(X) = \sum_{i=1}^{N_{sc}} w_i^{sc} \times P_i^{sc} + \sum_{j=1}^{N_{hc}} w_j^{hc} \times P_j^{hc} \quad (1)$$

Where, w_i^{sc} and w_j^{hc} represent the weight of each soft and hard constraint, respectively. P_i^{sc} and P_j^{hc} number of soft and hard constraints violation, respectively. Also, N_{sc} and N_{hc} is the number of soft and hard constraints, respectively. The constraints weight is determined according to their importance in the timetable. However, $F_{obj}(X)$ represents the fitness function for the solution X.

In the IPGALS algorithm, the value of hard constraints is always equal to zero, i.e. the hard constraints violation is impossible. Because we use the '-1' symbol for conflicting events. Therefore, in Eq. (1) it is always $\sum_{j=1}^{N_{hc}} w_j^{hc} \times P_j^{hc} = 0$. However, IPGALS does not allow any of the hard constraint to be violated in the optimization process. In this paper, instead of hard constraints violation, DF criterion is defined for conflicting events. Therefore, if there are events with a '-1' symbol in the solution, their penalty is calculated by DF for the fitness function. DF for a solution expresses the distance to feasibility. Consequently, the value of fitness function in each solution considering soft constraints violation and DF criterion is calculated by Eq. (2).

$$F_{obj}(X) = \sum_{i=1}^{N_{sc}} w_i^{sc} \times P_i^{sc} + \sum_{j=1}^{N_{df}} w^{df} \times P_j^{df} \quad (2)$$

Where, w^{df} is the weight of DF criterion, P_j^{df} shows the DF for j-th conflicting event, and N_{df} is the number of conflicting events (events with '-1' symbol in the chromosome). The following is considered to calculate the fitness function of a solution:

– DF criterion (P^{df}): This criterion is calculated for a conflicting event based on the number of attended students. In fact, DF for a solution is the overall of the number of students in conflicting events. As an example, if the solution contains three conflicting events, and the number of students in each case is 7, 14 and 3. Hence, the DF for this solution is equal to 24 (7 + 14 + 3).
– Number of violated soft constraints (P^{sc}): The number of violated soft constraints is equal to the overall of the three soft constraints defined in Section 2.1.2.

4.1.4 Selection operator

Selection operator is the stage of GA, which chromosomes with good fitness function are preferred and allows them to pass on their genes to the successive generations. So, this operator is for selecting combining a pair of parents to produce offspring in the next generation. Parent selection plays an important role in GA converging duration such a way that proper parents direct chromosomes to fitter solutions. In IPGALS, the roulette wheel technique is used to select parents [48]. This technique is a common method for selecting an item proportional to its probability. In addition, shared memory chromosomes are used to aid convergence. Here the first parent is selected from the population of the current GA. Also, the second parent with a 50% probability is the best chromosome from the shared memory, and with a 50% probability is selected from the chromosomes of shared memory.

4.1.5 Crossover operator

The performance of GA mainly depends on the type of genetic operators involve in crossover and mutation. Crossover is a genetic operator used to vary the genome of chromosomes from one generation to the next. The IPGALS algorithm consists of three crossover operators: Uniform, One-Point and Heuristic Operators. According to the two selected parents, one chromosome is generated as an offspring in each crossover operator. Crossover operator is usually applied in a GA with the highest C_R probability. Otherwise, offspring is generated by repeating the parents in case the crossover is not conducted on a chromosome pair. However, based on the three crossover operators, three offspring are generated. The details of the proposed crossover operators are discussed below.

Fig. 4 Shuffling technique on the example given in Fig. 3

	10	14	6	15	8	3	11	9	12	5	7	12	1	2	13
#Timeslots	4	5	2	5	2	1	4	3	4	2	3	3	1	1	5
#Rooms	1	2	3	3	1	3	2	3	3	2	1	2	1	2	1

	10	14	-	6	15	3
#Timeslots	4	5	-1	2	5	1
#Rooms	1	2	-1	3	3	3

Fig. 5 Selecting states for a chromosome with six events according to Fig. 4

Uniform crossover operator Generally, the uniform crossover is more effective for many problems, especially for numerical optimization problems [45, 46]. At first, the chromosome of offspring is randomly generated with '0' and '-1' genes in the proposed uniform crossover operator. Then, the equivalent genes are duplicated from the first parent for each gene with the content of '0'. Afterwards, the genes are duplicated from the second parent instead of genes with the symbol '-1', provided they do not violate the hard constraints. Finally, some genes may not be assigned any value with the content of '-1'. Hence, these genes are considered as conflicting events. An example of a uniform crossover operator is shown in Fig. 6.

In this example, events 2, 4, and 5 of the offspring chromosome with '0' content and other events (1, 3 and 6) with '-1' are initially set. Then, events 2, 4, and 5 are duplicated by the equivalent genes of the first parent. That is, {14, 1}, {2, 3} and {3, 1}. Similarly, the events 1, 3 and 6 are set by the second parent, where the hard constraints have been violated for the third event with {14, 1} content. Therefore, the third event is a conflicting event.

One-point crossover operator In one-point crossover, a random crossover point is selected and the genes of its two parents are swapped to get new offspring. At first, the offspring chromosome with the content of '-1' is generated by default. Then a random number is generated and the parents are cut from the same point. Afterwards, the genes swap is done between two parent chromosomes. The first part of the first parent is placed directly on the offspring chromosome. The genes of the second part of the second parent are used to complete offspring chromosome, provided they do not violate the hard constraints. If an event violates a hard constraint, it is set by the '-1' symbol. An example of a one-point crossover operator is shown in Fig. 7.

In this example, crossover point is randomly selected with the value of 3. In this case, events 1 and 2 of the first parent and other events (3, 4, 5 and 6) of the second parent are used to generate the offspring chromosome. Here, the third event of the second parent violates the hard constraints. Therefore, this event is a conflicting event in the offspring chromosome and it is denoted by the '-1' symbol.

Heuristic crossover operator The heuristic crossover operator also fills the offspring chromosome genes with the content of '-1' by default to avoid hard constraints violation at first. Then, the shared genes between the parents are placed directly on the offspring chromosome. Afterwards, other genes of the offspring chromosome are set according to the remaining events of parents, provided they do not violate the hard constraints. An example of the heuristic crossover operator is illustrated in Fig. 8.

In this example, events 1, 3, and 6 are share to both parents. Therefore, these events are duplicated in the offspring chromosome with {7, 3}, {5, 2} and {4, 2}, respectively. Then, according to the remaining events of parents (namely, {14, 1}, {2, 3}, {3, 1}, {6, 1}, {13, 3} and {8, 4}), other genes of the offspring chromosome are set. Here, none of the remaining events has been able to complete the fifth event without the violating hard constraints. Hence, the fifth event is a conflicting event and setting by the '-1' symbol.

4.1.6 Mutation operator

The mutation operator aims to produce a little modification to a chromosome to produce the next offspring stochastically. This operator alters one or more gene values in the offspring chromosomes from its initial state. One of the main advantages of this operator is the ability to access the entire search space. Mutation operator is used to maintain and introduce diversity in the genetic population and is usually applied in a GA with the lowest M_R probability. It has been observed that mutation is essential to the convergence of the GA while crossover is not [48]. The IPGALS algorithm consists of three mutation operators: Local, Global and Swap Operators. These operators apply on three offspring produced by crossover operators. Therefore, the IPGALS algorithm produces nine offspring. The details of the proposed mutation operators are discussed below.

Fig. 6 An example of uniform crossover operator

First Parent						Second Parent					
7	14	5	2	3	4	5	6	14	13	8	5
3	1	2	3	1	2	4	1	1	3	4	5
Offspring Initialization						**Final Offspring**					
-1	0	-1	0	0	-1	5	14	-1	2	3	4
-1	0	-1	0	0	-1	4	1	-1	3	1	2

Fig. 7 An example of one-point crossover operator

First Parent						Second Parent					
7	14	5	2	3	4	5	6	14	13	8	5
3	1	2	3	1	2	4	1	1	3	4	5
Offspring Initialization						Final Offspring					
-1	-1	-1	-1	-1	-1	7	14	-1	13	8	5
-1	-1	-1	-1	-1	-1	3	1	-1	3	4	5

Local mutation operator In this operator, two events are randomly selected from the contradictory events of the offspring chromosome. Where, contradictory events include a list of events that violate soft constraints. The selected events are replaced, provided does not violate the hard constraints and improve the fitness function. The probability of changing the timeslot and room is as follows: with a 50% probability both timeslot and room, with a 25% probability only the timeslots and with 25% probability only the room. In this operator, if the offspring chromosome had conflict events, one of the events is randomly selected from them. Therefore, according to internal events investigate of the offspring chromosome, the efficiency of this mutation operator is local.

Global mutation operator This operator first compiles a list of all possible states that can be added to the chromosome. According to Fig. 3, each state consists of the timeslot and the room number. The possible states are the result of sharing all the states and states of the offspring chromosome. Then, an event is randomly selected from the list of possible states and a random event from the offspring chromosome. The event related to the offspring chromosome is selected from the conflicting events. The selected state is replaced, provided does not violate the hard constraints and improve the fitness function. The probability of changing timeslot and room is similar to the local mutation operator. In general, there is a probability of selecting any conflicting event from the offspring chromosome in this operator. But, if the offspring chromosome contains conflicting events, the focus of selection is only on those events. Therefore, the efficiency of this mutation operator is global according to investigate of unseen events in the chromosome.

Swap mutation operator This operator randomly selects two events among all events within the chromosome. Then, the events are swapped in case fitness function produce better results.

4.1.7 Improvement operator

According to the structure of the proposed chromosome, some events might not be scheduled in a timetable. These are conflicting events which are represented by the '-1' symbols. The purpose of the improvement operator is to find the timeslot and the room for conflicting events so that the timetable becomes a feasible solution. In fact, this operator has been designed to improve the timetable by reducing the DF criterion in which we tend to minimize DF in the offspring population. This operator may drastically increase the number of soft constraints violation, but it reduces the DF criterion without violating hard constraints. Meanwhile, this operator applies to all nine offspring produced by mutation operator. Improvement operator performs in following steps for each conflicting event.

- Creating all possible states for events in E_S parameter
- Eliminate the states in the offspring chromosome from E_S
- The remaining states in E_S are examined in order to be replaced by conflicting event. The first state that does not violate any hard constraints for the offspring chromosome is replaced with the conflicting event.

4.1.8 Elitism operator

The interesting point of GA is the possibility of generating highly suitable chromosomes in median generations (yielding solid results at fitness function). These chromosomes might be destroyed in the result of the crossover and mutation operators

Fig. 8 An example of heuristic crossover operator

First Parent						Second Parent					
7	14	5	2	3	4	7	6	5	13	8	4
3	1	2	3	1	2	3	1	2	3	4	2
Offspring Initialization						Final Offspring					
-1	-1	-1	-1	-1	-1	7	2	5	6	-1	4
-1	-1	-1	-1	-1	-1	3	3	2	1	-1	2

and not generated anymore. Elitism technique identifies such cases and uses them in subsequent generations. Therefore, this technique can help IPGALS to find optimal solutions for UCTP. The elitism operator of this paper selects the best chromosome from the shared memory in each generation and transfers it directly to the next-generation population (for all PGAs).

4.1.9 New population for next generation

In the IPGALS, nine offspring chromosomes are produced at each generation by operators. Three offspring are produced by crossover operators (C_1, C_2 and C_3) which three offspring ($\{C_{1,\ 1}$, $C_{1,\ 2}$ and $C_{1,\ 3}\}$, $\{C_{2,\ 1}$, $C_{2,\ 2}$ and $C_{2,\ 3}\}$ and $\{C_{3,\ 1}$, $C_{3,\ 2}$ and $C_{3,\ 3}\}$) are produced by each offspring through mutation operators. Thus, GA created a new population sequence by designed operators. Therefore, only nine new offspring are produced at each generation and the population is updated based on them. For the next population, the best chromosomes (based on the fitness function) from the old population and the offspring population are selected for the next-generation population. Then, the new generation of candidate solutions is used in the next generation of the algorithm. Following the above steps, the genetics cycle is transferred to the next generation. The reason for producing only nine offspring chromosomes is to reduce the inter-dependency of the population size and the algorithm running time in each generation. Accordingly, the population size can be increased due to genetic diversity without affecting the running time of the algorithm. This generation process is repeated until a termination condition has been reached. In particular, the termination condition in IPGALS is to reach the maximum number of the generation G_{max}.

4.2 Local search

Local Search (LS) is the basis of many heuristic methods for combinatorial optimization problems. This algorithm performs local changes in a valid solution to achieve another valid solution in the search space until certain conditions are satisfied. In general, LS is introduced to find a solution in order to maximize a fitness function among a number of ongoing solutions. In UCTP, there might be the genes of a chromosome that are less repeated or never been used. For this reason, it is necessary to increase the probability of choosing unselected states for each event. In the IPGALS algorithm, LS applies to the population of the offspring in each generation. The LS randomly swaps some of the unused genes in the chromosome. Unused genes include states that have not been used as events in the solution. LS changes only the amount of one gene in each iteration and changes are applied if the quality (fitness function) of timetabling increases. The maximum number of iterations to complete the LS process is LS_{iter}.

4.3 Shared memory

Shared memory contains the best chromosomes produced in all PGAs (based on the fitness function). The shared memory members are updated after calculating fitness function in each generation. Each running GA, stores Q chromosomes with the best fitness function in the shared memory. Therefore, the size of the shared memory is $Q \times Z$, where Z represents the number of running PGAs.

4.4 Structure of the PGA

As a matter of fact, the parallel implementation has been introduced for finding the solution in the NP-hard problems with the emergence of multi-core systems. Parallel computing techniques can be used to improve the efficiency of GA by exploiting the concurrency of calculations performed in GAs. One of the important features of the GA is the capability of running in parallel and searching for complicated spaces. In general, the possibility of getting stuck in a local optimum trap is much low using PGAs. PGA uses multiple genetic algorithms to solve a single task. In PGA, different populations are generated and processed in parallel instead of using a population and evolving it. In UCTP, developing a high-performance GA which always solves the problem is almost impossible due to the diversity of constraints. In IPGALS, a GA with multi-population has been proposed to solve this problem. In addition, in IPGALS, the parallelization of GA is done applying the shared memory programming and island model. In a parallel implementation of an island model each processor executes a GA. The processors perform the evolutionary process by periodically exchanging a portion of their populations.

The chromosome structure is identical in all populations, so that each population has its own unique operators. Meanwhile, each population is processed in parallel and independently. In addition, the initial population and fitness function are calculated in parallel, since this was the most time-consuming part of the algorithm. Although the selection, crossover and mutation operators can be performed in parallel, these are not computationally intensive tasks. Here, the process of exchanging chromosomes between populations of PGAs is performed after number of generations using elitist methods. Number of generations for the exchanging the chromosomes is determined based on th_1 threshold. According to this threshold, each th_1 generation of chromosomes is exchanged. In the exchange process, L item of the worst chromosomes are eliminated from the current population and the best chromosomes of the shared memory replace L items. The worst and best chromosomes are determined by fitness function. The number of exchanging chromosomes should be selected based on population size. Therefore, choosing a very high or very low value for L may have negative influences on the evolution of population, genetic diversity, and disrupting the final convergence.

4.5 Pseudo-code for the proposed IPGALS algorithm

The sequence of pseudo-code presented in Fig. 9 shows the proposed algorithm.

5 Results and discussion

In this section, we conduct an experimental study to evaluate the performance of the proposed algorithm. All experiments are carried out on an Intel Core i5 CPU at 2.2GHz and 8GB of memory and Windows 10 64-bit operating system. Furthermore, MATLAB R2016a is used for implementation. In order to achieve more reliable results, all experiments have been reported an average of 10 independent executions. The proposed algorithm is evaluated on the BenPaechter [25] and ITC-2007 Track 2 [26] benchmark instances. Best-known solutions is available at the website of the Timetabling Research Group at the University of Udine (http://satt.diegm.uniud.it), which researchers can upload their best solutions and results.

In this paper, all considered constraints for the comparing algorithms have been identified according to Section 2.1. Due to the same constraints and instances, all comparisons between algorithms have been performed in fair conditions. This section divided into five subsections. The first subsection discusses the parameters setting. The second subsection reports the soft constraints violation for all instances. The third subsection shows the convergence of IPGALS algorithm in reducing constraints. The fourth subsection investigates the effectiveness of each operator in producing the final solutions. The fifth subsection proves the superiority of IPGALS algorithm versus IGA and IGALS methods. The sixth subsection compares the IPGALS algorithm with the state-of-the-art techniques. The discussion on the time complexity of the IPGALS algorithm is presented in last subsection.

5.1 Parameters setting

The efficiency and effectiveness of hybrid and meta-heuristic algorithms highly depend on the appropriate adjustment of the parameters. The IPGALS algorithm has several parameters, which are presented in Table 3. Parameters are set into three categories according to the complexity (number of events) of the instances such as the events fewer than 200 ($N_e \leq 200$), between 200 and 400 ($200 < N_e \leq 400$) and more than 400 ($N_e > 400$). Some of the initial parameter values are derived from the studies of [47, 49], including control parameters for the weight adjustment mechanism, i.e. N_{pop}, C_R, M_R and G_{max}. Other parameters (i.e. Z, LS_{iter}, th_1, L and Q) are determined using Taguchi method to achieve the best solution [50]. This method ensures the identification of effective parameters and levels with fewer experiments by providing balance among

```
IPGALS Algorithm: The Improved Parallel Genetic Algorithm And Local Search
input: An instance of BenPaechter or ITC-2007 benchmarks and $N_{pop}$, $C_R$, $M_R$, $G_{max}$, $LS_{iter}$, $Z$, $th_1$, $L$ and $Q$ parameters.
parfor $i = 1$ to $Z$ do   // Parallel Running
    Parallel initialization: Create an initial population with the size equal to $N_{pop}$ based on the random-heuristic method.
    Parallel evaluation: Calculate fitness function for each $X_i$ chromosome.
    Shared memory: Storing the best chromosome in the shared memory.
    for $j = 1$ to $G_{max}$ do
        Selection operator: Selecting two parents ($P_1$ and $P_2$) with roulette wheel method.
        Crossover operator: Apply uniform, one-point and heuristic crossovers and production of three offspring ($C_1$, $C_2$ and $C_3$) based on $C_R$.
        Improvement operator: Improvement of three offspring $C_1$, $C_2$ and $C_3$.
        Calculate fitness function for three offspring $C_1$, $C_2$ and $C_3$.
        for $k = 1$ to $3$ do   // For three offspring
            if $DF_{C_k} = 0$ then
                Mutation operator: Apply local and global mutations and production of two offspring ($C_{1,k}$ and $C_{2,k}$) based on $M_R$.
            else
                Mutation operator: Apply local and global mutations and production of two offspring ($C_{1,k}$ and $C_{2,k}$) base on $M_R$ only conflicting events (genes with '-1' symbol).
            end
            Mutation operator: Apply swap mutation and to produce an offspring ($C_{3,k}$) base on $M_R$.
            Calculate fitness function for three offspring $C_{1,k}$, $C_{2,k}$ and $C_{3,k}$.
        end
    end
    Local search: Applying local search with $LS_{iter}$ iterations on the offspring population.
    Next generation: Select the best chromosomes from the old population and offspring population for the next-generation population.
    Elitism operator: Select the best chromosome from the shared memory and transferring it to the next-generation population.
    Shared memory: Storing the best chromosome in the shared memory with a limited memory capacity equal to $Q$.
    Chromosome exchange: Transfer of $L$ chromosomes from shared memory to the population, based on $th_1$ threshold.
end
$X_{best}$ = Chromosome with the best fitness function from shared memory.
output: The timeslots and rooms on each event ($X_{best}$).
```

Fig. 9 Pseudo-code of the proposed IPGALS algorithm

Table 3 Parameter setting for IPGALS algorithm

Parameters	Description	Values based on the number of events		
		$N_e \leq 200$	$200 < N_e \leq 400$	$N_e > 400$
N_{pop}	Population size	50	30	30
C_R	Crossover rate	0.75	0.85	0.85
M_R	Mutation rate	0.35	0.1	0.15
G_{max}	Maximum number of generation	75	50	50
Z	Number of GAs in parallel structure	3	5	10
LS_{iter}	Maximum number of iteration in LS	2000	1000	500
th_1	Threshold limit for transfer of chromosomes	10	5	5
L	Number of chromosomes in the exchange	2	2	2
Q	Number of chromosomes allowed to store in shared memory	3	3	3

the orthogonal index, parameters, and levels. The aim of Taguchi method is to maximize the S/N ratio (signal-to-noise) which in the paper is calculated by Eq. (3).

$$S/N_{ij} = -10\log_{10}\left(\frac{1}{m}\sum_{i=1}^{m} F_{obj}(i,j)^2\right), \qquad \forall j \in level \tag{3}$$

Where, $F_{obj}(i,j)$ is the objective function value using the parameter i on j-th level and m is the number of times j-th level of the parameter i is repeated over the runs of all trials.

The results obtained through various parameters of 4 levels are estimated based on standard table of orthogonal arrays L_{16} [50]. Because of the similarity in reasoning procedure, the parameters setting process is only shown for instances with 100 event numbers. Table 4 presents the value of different parameters at each level in Taguchi method.

In this Table, rows denote the level of parameters in each experimental scheme and columns indicate a specific level of a parameter that is changeable in each scheme. The S/N ratio obtained from the experiment (illustrated in Fig. 10) indicates the order of importance of the variables. Rank shows the level according to which variables should be used in order to receive the best solution. The level with the highest S/N value is the most suitable level. According to the S/N ratio, it is inferred that Z, LS_{iter}, th_1, L and Q, which are 3, 2000, 10, 2, and 3, respectively, is an appropriate solution based on Taguchi method.

Table 4 Level values considered for parameters

Parameters	Level 1	Level 2	Level 3	Level 3
Z	2	3	4	5
LS_{iter}	1000	2000	3000	4000
th_1	5	10	15	20
L	1	2	3	4
Q	1	2	3	4

In IPGALS, the weight for the DF criterion is set to be 1.0 ($w^{df} = 1$). This weight prevents the selection of chromosomes with conflicting events. In general at UCTP, the weight of soft constraints (w^{sc}) is very effective for determining the optimal solutions in UCTP. Therefore, the weight of soft constraints is evaluated by different states to find the most appropriate values for them. In each state, we assign different weights in the range [0–1] to the soft constraints w_1^{sc}, w_2^{sc} and w_3^{sc}. The performance of the IPGALS is investigated through setting the weights on the BenPaechter and ITC-2007 benchmarks. The results of this comparison have been shown in Table 5.

The bold row of the table indicates the relative optimum values. Here, average of the constraints violation is presented for all instances. The results show that the best values of weights are when w_1^{sc}, w_2^{sc} and w_3^{sc} are 0.25, 0.75 and 0.50, respectively. Meanwhile, selecting chromosomes based on this weights leads to increase efficiency and genetic diversity.

5.2 The IPGALS algorithm results

We applied the IPGALS algorithm on the instances of BenPaechter and ITC-2007 benchmarks and the results are shown in Tables 6 and 7, respectively. The results are depicted based on the best and mean cases of soft constraints violation. For each experiment, the average of 10 runs of IPGALS algorithm is reported in the results. IPGALS does not cause the hard constraints of the input instance. Therefore, the presented results are only related to soft constraints and DF criterion. In these tables, the number of violations for each constraint is individually calculated and has been reported for each instance. DF in Tables 6 and 7 represents the distance to feasibility criterion. In fact, DF indicates how far a solution is from a feasible solution. The last column of these tables represents the sum of all soft constraints plus DF criterion. So, the

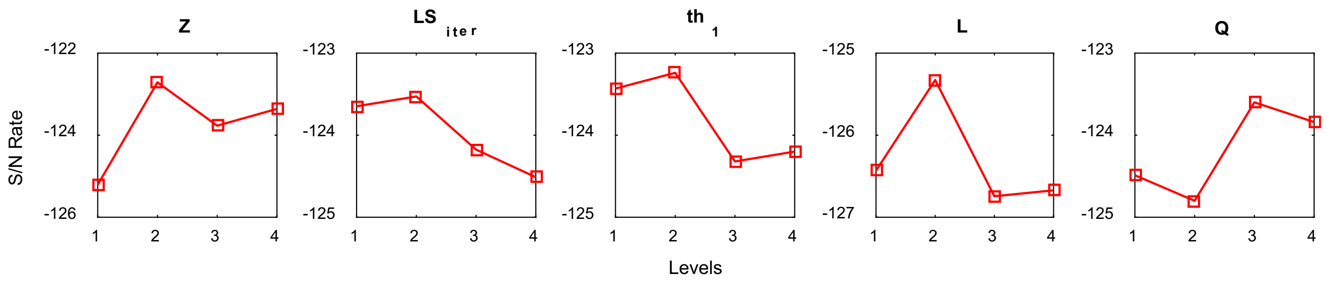


Fig. 10 The S/N ratio obtained in the Taguchi method

column is entitled 'Total' equals $SC_1 + SC_2 + SC_3 + DF$, where SC_i is the i-th soft constraint in the fitness function.

The results of Table 6 show that the IPGALS generates solutions without constraints for all small instances in the BenPaechter benchmark. Medium instances have little soft constraints. However, IPGALS is able to present a feasible timetable for all medium instances (the DF criterion for all medium instances equals zero). In large instances, the number of constraints violation is greater than in small instances. For example, the total number of violated constraints in the Medium1 instance is 84, while this value is 516 for Large1 instance (best case). In addition, the performance of the IPGALS is satisfactory based on DF criterion, since feasible solutions are presented for all the 12 instances of the BenPaechter benchmark (best case). DF criterion results are 193 and 386 on average for Large1 and Large2 instances, respectively. These results indicate that the IPGALS does not always provide the feasible timetable for large instances. Moreover, the results of SC_1, SC_2 and SC_3 constraints show that reducing SC_2 is more difficult for IPGALS. This constraint is equivalent to the number of times a student has taken more than two consecutive classes.

Results of Table 7 for the ITC-2007 benchmark show that the IPGALS algorithm for some instances always presents the timetable without any violated constraint, i.e. C05-C09, C13-C18 and C21. The results of other instances are also satisfactory, since feasible solutions are presented for all 24 instances of the ITC-2007 benchmark (best case). However, the average DF results for C01, C10, C22 and C23 instances show that the IPGALS has not always been able to always provide a feasible timetable. In addition, the results show the relatively high difference between the average and the best value, which represents the scattering in the solution space. It is believed that there is a close relationship among the soft constraints. Note that satisfy soft constraint 1, increases the violation on soft constraint 2 and 3, because more events will be scheduled in a day to avoid having a single event on a day. For example, in Table 7, satisfaction of soft constraints 2 and 3 is weaker in most instances.

5.3 Observing the convergence of IPGALS

GA results are usually converged when there is no significant improvement in the values of the fitness function of the population from one generation to the next. It causes evolution halting because all chromosomes in the population become identical. IPGALS convergence analysis for reducing the constraints is covered in this subsection.

In order to solve constrained optimization problems through GA, the algorithm must generate feasible solutions and select a near-optimal solution among all the feasible solutions. IPGALS algorithm benefits from being able to escape local optima utilizing intelligent operators. In IPGALS, accelerating the convergence is performed using improvement operator and LS. These operators reproduce valid chromosomes in each generation. Moreover, IPGALS accelerates convergence by embedding the process of reasoning with reducing

Table 5 The Performance results of the IPGALS with different weights

Soft constraints weight			Average constraints violated	
w_1^{sc}	w_2^{sc}	w_3^{sc}	BenPaechter	ITC-2007
0.25	0.50	0.75	148.20	179.63
0.25	**0.75**	**0.50**	**132.83**	**166.04**
0.50	0.25	0.75	155.72	188.13
0.50	0.75	0.25	146.27	180.51
0.75	0.25	0.50	155.25	187.67
0.75	0.50	0.25	151.35	183.09

Table 6 The performance of IPGALS algorithm on BenPaechter benchmark

Instances	Constraints									
	SC_1		SC_2		SC_3		DF		Total	
Mean/ Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
Small1	0	0	0	0	0	0	0	0	0	0
Small2	0	0	0	0	0	0	0	0	0	0
Small3	0	0	0	0	0	0	0	0	0	0
Small4	0	0	0	0	0	0	0	0	0	0
Small5	0	0	0	0	0	0	0	0	0	0
Medium1	0	0	82	76	11	8	0	0	93	84
Medium2	0	0	105	96	6	3	0	0	111	99
Medium3	4	2	126	118	24	22	0	0	154	142
Medium4	1	0	82	78	7	6	0	0	90	84
Medium5	4	3	97	87	31	22	0	0	132	112
Large1	129	142	298	353	16	21	193	0	636	516
Large2	16	26	309	388	122	143	386	0	833	557

Table 7 The performance of IPGALS algorithm ITC-2007 benchmark

Instances	Constraints									
	SC_1		SC_2		SC_3		DF		Total	
Mean/ Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
COMP01	43	36	270	256	124	117	133	0	570	409
COMP02	89	87	205	198	103	96	0	0	397	381
COMP03	54	50	91	83	69	62	0	0	214	195
COMP04	40	35	119	100	83	76	0	0	242	211
COMP05	0	0	0	0	0	0	0	0	0	0
COMP06	0	0	0	0	0	0	0	0	0	0
COMP07	0	0	0	0	0	0	0	0	0	0
COMP08	0	0	0	0	0	0	0	0	0	0
COMP09	0	0	0	0	0	0	0	0	0	0
COMP10	63	57	299	281	150	138	142	0	654	476
COMP11	37	32	66	63	46	40	0	0	149	135
COMP12	39	35	83	77	47	41	0	0	169	153
COMP13	0	0	0	0	0	0	0	0	0	0
COMP14	0	0	0	0	0	0	0	0	0	0
COMP15	0	0	0	0	0	0	0	0	0	0
COMP16	0	0	0	0	0	0	0	0	0	0
COMP17	0	0	0	0	0	0	0	0	0	0
COMP18	0	0	0	0	0	0	0	0	0	0
COMP19	18	13	67	44	26	18	0	0	111	75
COMP20	34	30	191	178	93	87	0	0	318	295
COMP21	0	0	0	0	0	0	0	0	0	0
COMP22	93	81	367	328	131	124	176	0	767	533
COMP23	148	133	439	417	330	306	382	0	1299	856
COMP24	44	31	158	141	71	63	0	0	273	266

the distance to feasibility. Because IPGALS does not waste its time on the infeasible solutions.

In Figs. 11 and 12 the convergence of IPGALS for the BenPaechter and ITC-2007 benchmarks is shown based on the number of violated constraints. Meanwhile, the results have been reported on average for all small, medium and large instances in the BenPaechter benchmark. For example, convergence in small instances is calculated based on the average S1-S5 instances. Similarly, the results have been reported on average for instances with a number of events of less than 200, between 200 and 400, and more than 400 in the ITC-2007 benchmark.

DF criterion in small instances converges to zero rapidly which means a feasible solution has been produced. For these instances, due to the tiny size of the search space and the random-heuristic method for initial population, the hard constraints are not violated. This means that there is no gene with the '-1' symbol in the solution. Therefore, operators focus on reducing soft constraints based on the mutation operator structure. In fact, mutation operators for solutions without DF constraints only apply to conflicting events. Obviously, all three soft constraints SC_1, SC_2 and SC_3 are rapidly reduced and the best result is achieved after the 60th generation. However, DF values are not zero in the medium and large instances, due to the extent of the problem dimensions in these instances. Therefore, IPGALS focuses on reducing DF constraints according to the designed operators and IPGALS algorithm strategy against DF criterion. For this reason, other soft constraints are sometimes increased by DF decrement. That is clearly indicated in the Fig. 11 for medium and large instances. For example, DF value is about 100 in the first generation of the medium instances and after 17 generations converges to an optimum value of zero. However, SC_2 is about 150 in the first generation and its value reaches more than 160 after 17 generations. In general, SC_2 constraint value is higher than any other constraints. The number of SC_2 constraint is much higher than SC_1 and SC_3 constraints due to the high number of students and events for each student.

In all experiments, only the value of DF criterion equals zero for IPGALS except for the small instances of BenPaechter. It demonstrates that reducing DF constraints is more important than the soft constraints for IPGALS. Because, IPGALS algorithm allocates higher priority to reducing DF, through allocating higher weight and the behavior of some operators. Figure 12 confirms that IPGALS for the instances on the ITC-2007 benchmark performs similar to the BenPaechter benchmark.

5.4 Observing the effectiveness of operators

IPGALS algorithm defines different operators for reproducing and improving the solutions. These operators consist of uniform crossover, one-point crossover, heuristic crossover, local mutation, global mutation, swap mutation, improvement, elitism and local search. Each of these operators has different responsibilities in creating a new population. In Figs. 13 and 14 the effectiveness of each operator in producing the final solutions is shown for BenPaechter and ITC-2007 instances. The effectiveness of an operator in producing final solutions is measured based on performing IPGALS with deactivating that operator. Therefore, greater importance for an operator is achieved when the efficiency of IPGALS is reduced without its use.

Here, the number of violated constraints with deactivating each operator is reported based on 10 runs of an instance on the best case. In Figs. 13 and 14, the term 'S1' means the first instance of small data, 'S2' means the second instance of small data, and so on. Likewise, the term 'M1' denotes the first instance of the medium data and 'L1' denotes the first instance of large data and so on. Similarly, the term 'C01' means COMP01 instance, 'C02' means COMP02 instance, and so on. The effectiveness of improvement, local search, and heuristic crossover operators is clearly demonstrated in various instances. The importance of these operators (especially improvement operator) is in reducing the DF criterion and creating feasible solutions. The improvement operator is

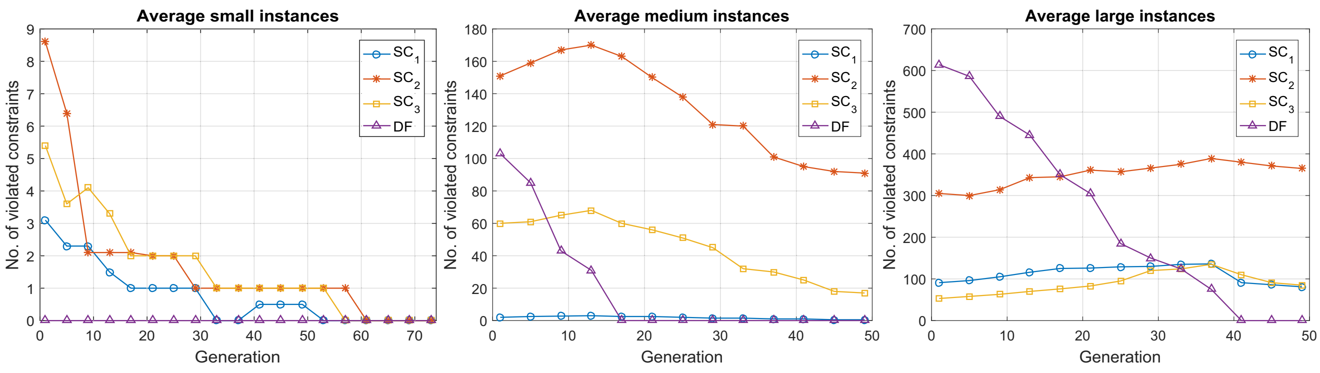


Fig. 11 The convergence of proposed algorithm on the BenPaechter instances

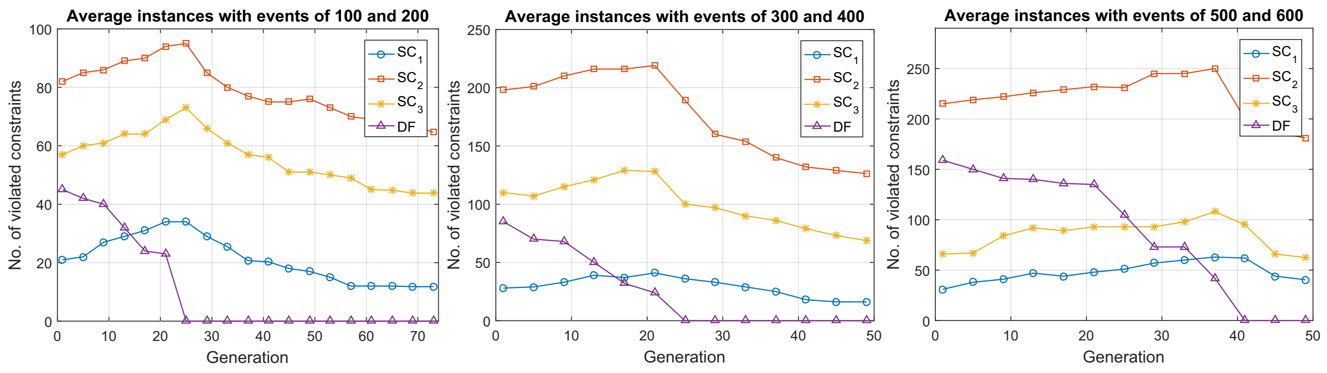


Fig. 12 The convergence of proposed algorithm on the ITC-2007 instances

more efficient than all operators except M4 instance of BenPaechter and C06 and C09 instances of ITC-2007.

5.5 Comparing IPGALS with IGA and IGALS

In this paper, we proposed a hybrid method for solving UCTP using IPGALS. IPGALS is a hybrid of GA and LS that use a parallel architecture. Here, we compare the results of IPGALS with the results of IGA and IGALS methods. IGA is an improved GA, and IGALS is a combined algorithm of IGA and LS. The IGA and IGALS methods details are similar to the proposed IPGALS algorithm. The performance of IPGALS versus IGA and IGALS is shown by the total average of violated soft constraints (for all instances) in Table 8. The results are reported an average of 10 runs for each instance of BenPaechter and ITC-2007 benchmarks.

As evident in Table 8, each algorithm recorded the average of violated soft constraints for all instances of a benchmark. The averages produced by IGALS and IPGALS methods are comparable for these benchmarks. Meanwhile, IPGALS clearly outperforms IGA on the both benchmarks. Overall, IPGALS seems to be the most effective algorithm.

Here, we compare the performance of IGA, IGALS and IPGALS methods in minimizing the number of violated soft constraints. For BenPaechter instances, the comparison between IPGALS and IGALS is shown in Table 9. In order to compare, the best values of soft constraints violation are depicted for 10 runs. Moreover, a paired *t*-test has been used to verify the effectiveness of the proposed algorithm. The paired *t*-test has been performed between IPGALS and each of the IGA and IGALS methods. A value of $p < 0.05$ is considered to determine statistical significance. The *t*-test values constitute of the soft constraints violation achieved with

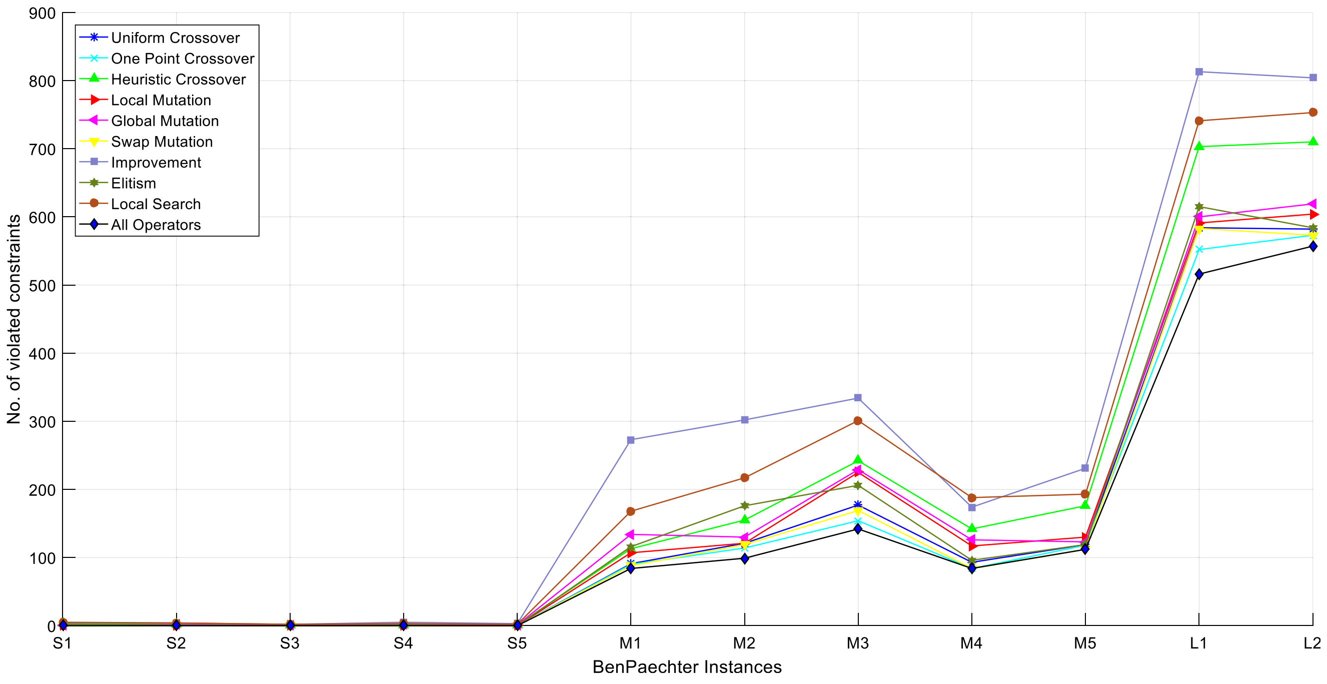


Fig. 13 The effectiveness of the proposed algorithm operators on BenPaechter instances

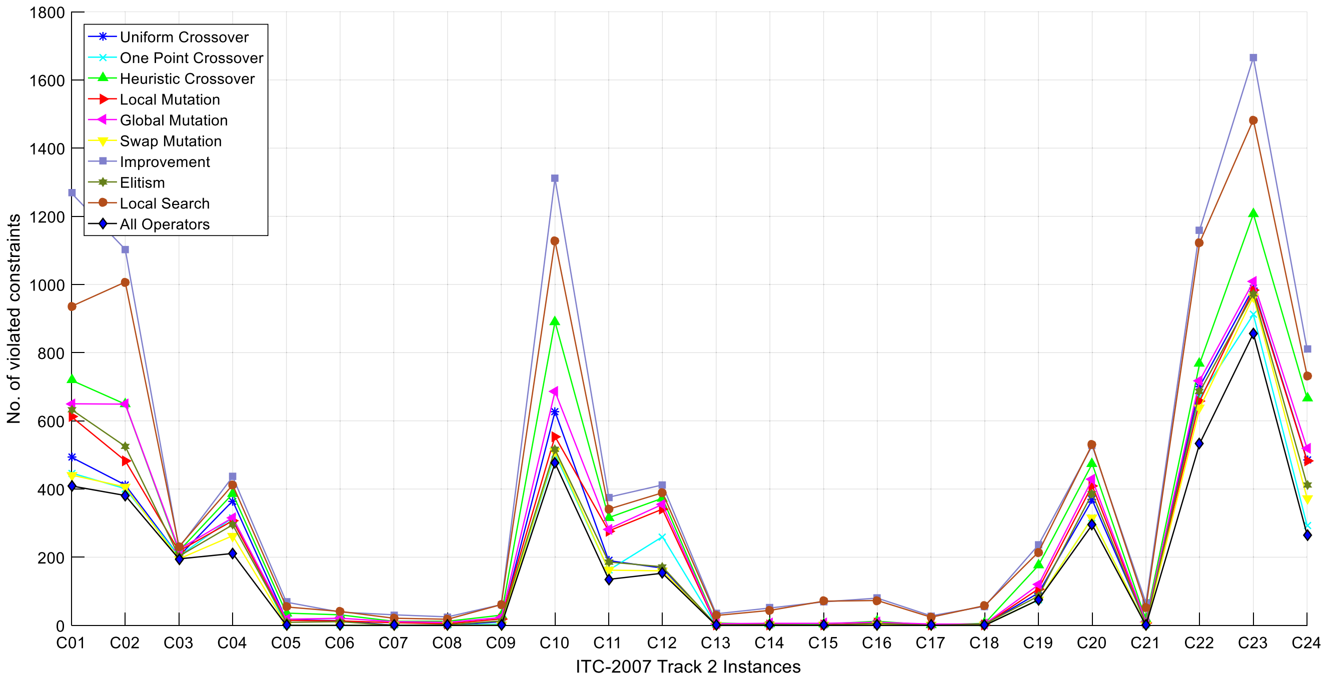


Fig. 14 The effectiveness of the proposed algorithm operators on ITC-2007 instances

methods as one distribution. The p values reveal that there is no significant difference between the means of IGALS and IPGALS for all the instances except M4 where IGALS is better than IPGALS. In addition, IPGALS performance is clearly better than IGA for all instances.

Results of comparison between IGA, IGALS and IPGALS methods for ITC-2007 instances have been shown in Table 10. The t-tests show that IGALS only performed better than IPGALS for instances C03, C04 and C11. However, IPGALS is clearly more effective for instances C02 and C19. The difference between both methods for the rest of the instances is negligible on average. In addition, IPGALS has performed significantly better compared to IGA for instances C02, C06, C10, C19, C22, C23 and C24. IGA has performed better than IPGALS for instance C03 on the best case. The difference between both methods for the rest of the instances is negligible on average. Totally, IPGALS outperforms IGALS on 4 instances. Also, IPGALS is significantly outperforming IGA on 12 instances. The t-tests to indicate significant statistical differences between the means of the three methods for the rest of the instances.

Table 8 Comparing average of soft constraints violation between IGA, IGALS and IPGALS algorithms

Benchmarks	Algorithms		
	IGA	IGALS	IPGALS
BenPaechter	178	142	133
ITC-2007	193	176	166
Total	371	318	299

5.6 Comparing IPGALS with other methods

The existence of different methods for solving UCTP has opened a vast opportunity for researchers in this field. Therefore, choosing the best method or following its sequence could provide relatively better successes for future approaches. In this respect, the BenPaechter and ITC-2007 instances have been widely used as UCTP benchmarks for algorithmic comparison. Here, the performance of the proposed IPGALS algorithm is compared with other methods in solving UCTP. The average numbers of constraints violation of 10 independent executions for BenPaechter instances are shown in Table 11. In addition, comparison results for ITC-2007 instances are reported in Table 12. Meanwhile, A_i ($i = 1, 2, \ldots, 38$) denote the compared methods which A_{38} denotes the proposed IPGALS algorithm. Appendix is dedicated to present a list of compared methods.

In Table 11, the number of violated soft constraints is presented on the mean and best cases for all small (S1–S5), medium (M1–M5) and large (L1 and L2) instances. The terms B and M refer to the best and mean cases, respectively. The reported results confirm the superiority of the IPGALS algorithm in comparison with other methods in most of the instances. According to Table 11, A_2, A_8, A_{11}, A_{18} and A_{21} suffer from inefficiency for small instances. Hereby, the

Table 9 Comparison between IGA, IGALS and IPGALS methods on BenPaechter instances

Methods	Instances											
	S1	S2	S3	S4	S5	M1	M2	M3	M4	M5	L1	L2
IGA	0	4	0	1	2	116	119	176	98	166	767	687
IGALS	0	0	0	0	0	87	107	146	83	117	590	574
IPGALS	0	0	0	0	0	84	99	142	84	112	516	557
t-test (Vs. IGA)	–	0.107	–	0.088	0.110	0.181	0.576	0.243	0.074	0.290	0.826	0.0573
t-test (Vs. IGALS)	–	–	–	–	–	0.074	0.121	0.057	0.008	0.085	0.532	0.130

performance of algorithms is degraded over medium instances. Here, the order of superiority of methods is A_{38}, A_{23}, A_{12}, A_4 and A_{17}. In general, A_3, A_{12}, A_{23} and A_{38} methods have appropriate efficiency over large instances. The methods A_1, A_3, A_5, A_{12}, A_{15}, A_{16}, A_{17}, A_{19}, A_{22}, A_{23} and A_{38} methods in small instances present best performances, because they do not violate any soft constraints. However, the worst performance belongs to both A_{11} and A_{21} methods.

We have obtained the performance of the above methods on medium instances, where the A_{38} method (IPGALS algorithm) presents the best efficiency and A_{12} and A_{23} methods have an acceptable performance. However, the worst performance belongs to A_2 and A_{18} methods. Finally, we have obtained the performance of the abovementioned methods on large instances in which the A_{23} method presents the best performance and A_3 and A_{38} methods have an acceptable performance. However, the worst performance belongs to both A_{11} and A_{13} methods.

Similarly, the number of violated soft constraints has been presented for ITC-2007 instances in Table 12. In general, A_{24}, A_{30} and A_{38} methods have appropriate efficiency over C05-C08 and C13-C18 instances because they do not violate any soft constraints. In addition, the performance for these methods is degraded over other instances. Here, the order of superiority of methods is A_{38}, A_{25} and A_{30}. Hereto, the method A_{32} yields the best performance while A_{38}, A_{27} and A_{36} methods show appropriate performances. Meanwhile, we realized that the A_{23}, A_{32} and A_{38} methods outperformed other methods in all instances.

Table 13 presents a summary of the results to evaluate the performance of the IPGALS algorithm in comparison with other methods. The number of violated soft constraints in the best case is presented for the average of all instances in each method. Here, results for both BenPaechter and ITC-2007 benchmarks are reported based on the difference in the number of violated soft constraints. In addition, comparisons have also been conducted in terms of the number of instances, which the IPGALS algorithm performance is superior to other methods. Negative values refer to the no-superiority of the IPGALS algorithm over any of the compared methods. It is worth-mentioning the methods whose results are not reported in this comparison have not been used to calculate results in some instances.

The results of this comparison show that in all instances except A_{23} (for BenPaechter benchmark) and A_{32}, A_{37} (for ITC-2007 benchmark), the IPGALS algorithm performs better than the other methods. This superiority is clearly evident in both the average of violated soft constraints and the number of instances with lower number of violated soft constraints. The best IPGALS results are obtained compared to A_2 in the BenPaechter, where 187 is less violated constraints and is superior in all 10 instances. In addition, IPGALS reports the best results compared to the A_{31} at ITC-2007, where the number of violated constraints is 551 less, and it performs better in 23 out of the 24 instances.

5.7 Time complexity for IPGALS

Studenovský [51] showed that the time complexity of UCTP was initially $O(n^4)$. In general, UCTP parameters include: set of events, $E = \{e_1, e_2, \ldots, e_{N_e}\}$, set of timeslots, $T = \{t_1, t_2, \ldots, t_{45}\}$, set of rooms, $R = \{r_1, r_2, \ldots, r_{N_r}\}$, set of students, $S = \{s_1, s_2, \ldots, s_{N_s}\}$ and set of features $F = \{f_1, f_2, \ldots, f_{N_f}\}$. It is clear that these conditions can be verified in a time $O(n^4)$. However, the proposed IPGALS algorithm for solving UCTP bears less time complexity. The time complexity of IPGALS highly depends on GA. In some cases, GAs are not chaotic, they are stochastic. IPGALS complexity depends on the genetic operators, their implementation (which may have a very significant effect on overall complexity), the representation of the chromosomes, the population size, and obviously on the fitness function. According to the applied operators, the time complexity for the roulette wheel selection is $O(N_{pop})$, the uniform crossover is $O(N_e)$, the one-point crossover is $O(N_e)$, the heuristic crossover is $O(N_e^2)$, the local mutation is $O(N_e)$, the global mutation is $O(N_e)$, the swap mutation is $O(1)$, the improvement is $O(N_e^2)$, the elitism is $O(1)$, the local search is $O(LS_{iter})$, and finally the fitness function is $O(N_e^2 N_s)$. Hence, the time complexity of IPGALS

Table 10 Comparison between IGA, IGALS and IPGALS methods on ITC-2007 instances

AM	Instances																							
	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24
IGA	423	436	187	221	0	21	8	7	2	501	147	174	3	9	7	0	17	11	199	302	0	616	1021	313
IGALS	417	473	178	203	0	7	0	0	0	489	124	166	0	0	0	0	4	0	182	302	0	542	867	281
IPGALS	409	381	195	211	0	0	0	0	0	476	135	153	0	0	0	0	0	0	75	295	0	533	856	266
t-test (Vs. IGA)	.327	.878	.017	.051	–	.066	.052	.110	.001	.088	.107	.241	.007	.058	.066	–	.241	.119	.874	.00	–	.097	.635	.237
t-test (Vs. IGALS)	.097	.605	.001	.005	–	–	–	–	–	.066	.047	.446	–	–	–	–	.002	–	.375	.001	–	.588	.063	.074

Table 11 Comparison of IPGALS performance with other methods on the BenPaechter instances, all solutions are possible, and i.e. the hard constraints are 0

Algorithms	Instances																							
	S1		S2		S3		S4		S5		M1		M2		M3		M4		M5		L1		L2	
Med/Best	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B	M	B
A_1	0	0	0	0	0	0	0	0	0	0	245	242	263	161	268	265	184	181	153	151	–	–	–	–
A_2	–	6	–	7	–	3	–	3	–	3	–	419	–	359	–	348	–	171	–	1068	–	–	–	–
A_3	–	0	–	0	–	0	–	0	–	0	–	221	–	147	–	246	–	165	–	135	–	529	–	–
A_4	–	3	–	4	–	6	–	6	–	0	–	140	–	130	–	189	–	112	–	141	–	876	–	–
A_5	0	0	0	0	0	0	0	0	0	0	229	227	185	180	238	235	155	142	203	200	–	–	–	–
A_6	–	0	–	0	–	0	–	0	–	0	–	317	–	313	–	357	–	247	–	292	–	–	–	–
A_7	–	1	–	2	–	0	–	1	–	0	–	146	–	173	–	267	–	169	–	303	–	–	–	–
A_8	8	–	11	–	8	–	7	–	5	–	199	–	202	–	–	–	177	–	–	–	–	–	–	–
A_9	–	0	–	3	–	0	–	0	–	0	–	280	–	188	–	249	–	247	–	232	–	–	–	–
A_{10}	1	–	3	–	1	–	1	–	0	–	195	–	184	–	248	–	164	–	219	–	851	–	–	–
A_{11}	–	10	–	9	–	7	–	17	–	7	–	243	–	325	–	249	–	285	–	112	–	1138	–	–
A_{12}	0	0	0	0	0	0	0	0	0	0	143	139	69	62	124	122	101	98	119	116	622	615	–	–
A_{13}	–	2	–	4	–	2	–	0	–	4	–	254	–	258	–	251	–	321	–	276	–	1027	–	–
A_{14}	–	0	–	3	–	0	–	0	–	0	–	280	–	188	–	249	–	247	–	232	–	–	–	–
A_{15}	–	0	–	0	–	0	–	0	–	0	–	180	–	176	–	219	–	150	–	196	–	–	–	–
A_{16}	–	0	–	0	–	0	–	0	–	0	–	242	–	161	–	265	–	181	–	151	–	–	–	–
A_{17}	–	0	–	0	–	0	–	0	–	0	–	221	–	147	–	246	–	165	–	130	–	–	–	–
A_{18}	–	6	–	7	–	3	–	3	–	4	–	372	–	419	–	359	–	348	–	171	–	–	–	–
A_{19}	–	0	–	0	–	0	–	0	–	0	–	317	–	313	–	357	–	247	–	292	–	–	–	–
A_{20}	–	1	–	2	–	0	–	1	–	0	–	146	–	173	–	267	–	169	–	303	–	–	–	–
A_{21}	–	10	–	9	–	7	–	17	–	7	–	243	–	325	–	249	–	285	–	132	–	–	–	–
A_{22}	0	0	0	0	0	0	0	0	0	0	242	240	258	160	245	242	161	158	126	124	822	801	–	–
A_{23}	0	0	0	0	0	0	0	0	0	0	131	106	131	107	151	132	93	72	125	107	555	505	524	486
A_{38}	0	0	0	0	0	0	0	0	0	0	93	84	111	99	154	142	90	84	132	112	603	516	833	557

algorithm for UCTP is $O(G_{max}N_{pop}(N_e^2N_s + LS_{iter}))$. Where, G_{max} is the number of generations, N_{pop} is the population size and LS_{iter} is the number of LS iterations.

However, most GAs are inherently chaotic [50]. So, calculating the time complexity could not be useful and even might be misleading. A better way to measure the time complexity is to actually measuring the run-time and averaging. The performed studies related to the run-time of the compared hybrid methods for UCTP reveal that the algorithms A_{15}, A_{24} and A_{38} have less average run-time compared to other methods. The run-time of A_{15} and A_{38} algorithms is 825 s and 780 s respectively in the BenPaechter benchmark and the run-time of A_{24} and A_{38} algorithms is 482 s and 546 s in the ITC-2007 benchmark (average for all instances). Since many algorithms have not been investigated the run-time factor in their studies, this comparison has been conducted only among some of the algorithms.

6 Conclusion and future work

In this study, a new method has been presented for solving UCTP based on a hybrid approach. We have presented an enhanced variant of GA and called it IPGALS. This algorithm is an improved PGA combined with LS. In IPGALS, the hard and soft constraints have been applied for determining the feasible solutions. In addition, the DF criterion has been applied to identify the distance to the feasibility of solutions. DF criterion reduces the number of violated hard constraints and improves the quality of solutions. GA is an efficient solution for the UCTP due to its multi-directional search characteristic. Since GA focuses more on the extraction, it might be failed by getting stuck in the local optimum. In this paper, LS and the elitism operator are implemented after applying crossover and mutation operators in GA to enhance its performance and prevent getting stuck in the local optimum trap. The use of

Table 12 Comparison of IPGALS performance with other methods on the ITC-2007 instances, all solutions are possible, and i.e. the hard constraints are 0

Algorithms	Instances																							
	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24
A_{24}	501	342	377	234	0	0	0	0	989	499	246	172	0	0	0	0	0	0	84	297	0	1142	963	274
A_{25}	571	993	164	310	5	0	6	0	1560	2163	178	146	0	1	0	2	0	0	1824	445	0	29	238	21
A_{26}	1482	1635	288	385	559	851	10	0	1947	1741	240	475	675	804	0	1	5	3	1868	396	602	1364	688	822
A_{27}	15	0	391	239	34	87	0	4	0	0	547	32	166	0	0	41	68	26	22	2735	33	0	1275	30
A_{28}	1861	2174	272	425	8	28	13	6	2733	2797	263	804	285	110	5	132	72	70	2268	878	40	889	436	372
A_{29}	630	450	300	602	6	0	0	0	640	663	344	198	0	35	0	140	0	0	400	150	0	32	238	640
A_{30}	523	342	379	234	0	0	0	0	1102	515	246	241	0	0	0	0	0	0	121	304	36	1154	963	274
A_{31}	1166	1665	251	424	47	412	6	65	1819	2091	288	474	298	127	108	138	0	25	2146	625	308	787	3101	841
A_{32}	59	0	148	25	0	0	0	0	0	3	142	267	1	0	0	0	0	0	0	543	5	5	1292	0
A_{33}	650	470	290	600	35	20	30	0	630	2349	350	480	46	80	0	0	0	20	360	150	0	33	1007	0
A_{34}	61	547	382	529	5	0	0	0	0	0	548	869	0	0	379	191	1	0	786	1215	0	0	438	720
A_{35}	19	338	238	49	81	253	603	55	215	285	0	1135	276	67	379	784	391	228	283	1098	215	–	–	–
A_{36}	82	48	155	254	0	0	4	0	59	6	140	33	0	0	0	2	0	1	617	482	0	35	1083	1
A_{37}	209	10	188	321	3	55	15	2	15	31	202	304	90	26	13	46	1	8	11	664	26	6	714	78
A_{38}	409	381	195	211	0	0	0	0	0	476	135	153	0	0	0	0	0	0	75	295	0	533	856	266

Table 13 Summary of the results to evaluate the performance of the IPGALS algorithm compared to other methods

Benchmarks	Superiority	Algorithms																						
		A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}
BenPaechter	Constraints	48	187	37	52	46	100	54	–	68	–	124	10	124	67	40	47	38	117	100	54	76	63	–7
	Instances	5	10	6	10	5	5	8	–	6	–	10	2	10	6	5	5	5	10	5	8	10	5	–3
		Algorithms																						
		A_{24}	A_{25}	A_{26}	A_{27}	A_{28}	A_{29}	A_{30}	A_{31}	A_{32}	A_{33}	A_{34}	A_{35}	A_{36}	A_{37}									
ITC-2007	Constraints	88	194	535	73	539	61	102	551	–62	150	111	190	348	–39									
	Instances	11	7	20	6	22	10	12	23	–2	13	8	11	2	8									

parallel structure also significantly accelerates GA convergence and diversifies the genetic populations. Experimental results reveal that the proposed IPGALS algorithm provides higher-quality solutions compared to other similar methods.

Unfortunately, our IPGALS has not always been able to form a feasible timetable for the some large instances. Therefore, the major shortcoming of IPGALS is that it is unable to assure the offering a feasible timetable in large instances. However, the proposed hybrid algorithm has yielded acceptable results in creating the feasible timetable for UCTP. Further analyzing the contribution of individual components (local search and guided search) for enhancing the performance of IPGALS is considered as a future direction. Additionally, improvement of genetic operators and new neighborhood techniques based on different constraints can be investigated. It is demonstrated that the performance of GA for UCTP would be improved by applying advanced genetic and heuristic operators. The correct relationship between these techniques and their proper arrangement in a GA might lead to higher performance.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix

Full names of applied algorithms (A_i) in Tables 11, 12 and 13. A_1: (RIICN: Randomize Iterative Improvement with Composite Nears [52]), A_2: (GBHH: Graph Based Hyper Heuristic [53]), A_3: (HEA: Hybrid Evolutionary Algorithm [54]), A_4: (NLGD: Non-Linear Great Deluge [55]), A_5: (MA: Memetic Algorithm [56]), A_6: (VNS: Variable Neighborhood Search [52]), A_7: (THH: Taboo based Hyper Heuristics [44]), A_8: (LS: Local Search [57]), A_9: (EA: Evolutionary Algorithm [58]), A_{10}: (ACO: Ant Colony Optimization [59]), A_{11}: (FA: Fuzzy Approach [60]), A_{12}: (EGSGA: Extended Guided Search with Genetic Algorithm [20]), A_{13}: (GA w LS: Genetic Algorithm with Local Search [61]), A_{14}: (GA: Genetic Algorithm [25]), A_{15}: (HGA: Hybrid Genetic Algorithm [62]), A_{16}: (RIIA: Randomised Iterative Improvement Algorithm [44]), A_{17}: (HEA: Hybrid Evolutionary Approach [63]), A_{18}: (GHH: Graph-based Hyper Heuristic [53]), A_{19}: (VNS: Variable Neighborhood Search [64]), A_{20}: (TSH: Tabu-Search Hyperheuristic [65]), A_{21}: (FMH: Fuzzy Multiple Heuristic [66]), A_{22}: (GSGA: Guided Search Genetic Algorithm [48]), A_{23}: (HA: Hybrid Algorithm [67]), A_{24}: (HGATS: Hybrid Genetic Algorithm and Tabu Search [68]), A_{25}: (MMH: Mixed Meta-Heuristic [69]), A_{26}: (HA: Hybrid Algorithm [24]), A_{27}: (ACO w ILS: Ant Colony Optimization with Iterative Local Search [59]), A_{28}: (LS: Local Search [26]), A_{29}: (HHADL: Hyper-Heuristic with Add Delete Lists [43]), A_{30}: (GAGLS: Genetic Algorithms with Guided and Local Search [20]), A_{31}: (TDMH: Time-Dependent Meta-Heuristic [70]), A_{32}: (SA: Simulated Annealing [71]), A_{33}: (HH: Hyper-Heuristics [72]), A_{34}: (TS-ILS: Tabu Search And Iterated Local Search [73]), A_{36}: (CB-CTT: Curriculum-Based Course TimeTabling [74]), A_{37}: (RPNS: Random Partial Neighborhood Search [75]), A_{38}: (SAIRL: Simulated Annealing with Improved Reheating and Learning [32]), A_{39}: (IPGALS: Improved Parallel Genetic Algorithm and Local Search (Proposed Algorithm)).

References

1. Hambali AM, Olasupo YA, Dalhatu M (2020) Automated university lecture timetable using heuristic approach. Niger J Technol 39(1):1–14. https://doi.org/10.4314/njt.v39i1.1
2. Abuhamdah A, Ayob M, Kendall G, Sabar NR (2014) Population based local search for university course timetabling problems. Appl Intell 40(1):44–53. https://doi.org/10.1007/s10489-013-0444-6
3. Thepphakorn T, Pongcharoen P (2019) Variants and parameters investigations of particle swarm optimisation for solving course timetabling problems. In: International conference on swarm intelligence. Springer, Cham, pp 177–187. https://doi.org/10.1007/978-3-030-26369-0_17
4. Bashab A, Ibrahim AO, AbedElgabar EE, Ismail MA, Elsafi A, Ahmed A, Abraham A (2020) A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms. Neural Comput & Applic 32(11):1–36. https://doi.org/10.1007/s00521-020-05110-3

5. Pintér M, Dávid B (2019) A two-stage heuristic for the university course timetabling problem. In: Proceedings of the 2019 6th student computer science research conference-StuCoSReC. Univerza na Primorskem, Inštitut Andrej Marušič, pp 27–30. https://doi.org/10.26493/978-961-7055-82-5.27-30
6. Akkan C, Gülcü A (2018) A bi-criteria hybrid genetic algorithm with robustness objective for the course timetabling problem. Comput Oper Res 90:22–32. https://doi.org/10.1016/j.cor.2017.09.007
7. Kostuch P (2003) Timetabling competition-SA-based heuristic. International Timetabling Competition. http://www.idsia.ch/ttcomp2002/docs
8. Pillay N (2014) A survey of school timetabling research. Ann Oper Res 218(1):261–293. https://doi.org/10.1007/s10479-013-1321-8
9. Saviniec L, Santos MO, Costa AM (2018) Parallel local search algorithms for high school timetabling problems. Eur J Oper Res 265(1):81–98. https://doi.org/10.1016/j.ejor.2017.07.029
10. Rezaeipanah A, Abshirini Z, Zade MB (2019) Solving University course timetabling problem using parallel genetic algorithm. International Journal of Scientific Research in Computer Science and Engineering 7(5):5–13
11. Fajrin AM, Fatichah C (2020) Multi-parent order crossover mechanism of genetic algorithm for minimizing violation of soft constraint on course timetabling problem. Register: Jurnal Ilmiah Teknologi Sistem Informasi 6(1):43–51. https://doi.org/10.26594/register.v6i1.1663
12. Soghier A, Qu R (2013) Adaptive selection of heuristics for assigning time slots and rooms in exam timetables. Appl Intell 39(2):438–450. https://doi.org/10.1007/s10489-013-0422-z
13. Mansour N, Isahakian V, Ghalayini I (2011) Scatter search technique for exam timetabling. Appl Intell 34(2):299–310. https://doi.org/10.1007/s10489-009-0196-5
14. Assi M, Halawi B, Haraty RA (2018) Genetic algorithm analysis using the graph coloring method for solving the university timetable problem. Procedia Computer Science 126:899–906. https://doi.org/10.1016/j.procs.2018.08.024
15. Babaei H, Karimpour J, Hadidi A (2018) Applying hybrid fuzzy multi-criteria decision-making approach to find the best ranking for the soft constraint weights of lecturers in UCTP. International Journal of Fuzzy Systems 20(1):62–77. https://doi.org/10.1007/s40815-017-0296-z
16. June TL, Obit JH, Leau YB, Bolongkikit J, Alfred R (2020) Sequential constructive algorithm incorporate with fuzzy logic for solving real world course timetabling problem. In: Computational science and technology. Springer, Singapore, pp 257–267. https://doi.org/10.1007/978-981-15-0058-9_25
17. Phillips AE, Walker CG, Ehrgott M, Ryan DM (2017) Integer programming for minimal perturbation problems in university course timetabling. Ann Oper Res 252(2):283–304. https://doi.org/10.1007/s10479-015-2094-z
18. AlHadid I, Kaabneh K, Tarawneh H (2018) Hybrid simulated annealing with meta-heuristic methods to solve UCT problem. Mod Appl Sci 12(11):366–375. https://doi.org/10.5539/mas.v12n11p366
19. Abdullah S, Burke EK, McCollum B (2007) A hybrid evolutionary approach to the university course timetabling problem. In: 2007 IEEE congress on evolutionary computation, pp 1764–1768. https://doi.org/10.1109/CEC.2007.4424686
20. Yang S, Jat SN (2010) Genetic algorithms with guided and local search strategies for university course timetabling. IEEE Trans Syst Man Cybern Part C Appl Rev 41(1):93–106. https://doi.org/10.1109/TSMCC.2010.2049200
21. Landa-Silva D, Obit JH (2009) Evolutionary non-linear great deluge for university course timetabling. In: International conference on hybrid artificial intelligence systems. Springer, Berlin, pp 269–276. https://doi.org/10.1007/978-3-642-02319-4_32
22. Turabieh H, Abdullah S, Mccollum B (2009) Electromagnetism-like mechanism with force decay rate great deluge for the course timetabling problem. In: International conference on rough sets and knowledge technology. Springer, Berlin, pp 497–504. https://doi.org/10.1007/978-3-642-02962-2_63
23. Chen M, Tang X, Song T, Wu C, Liu S, Peng X (2020) A Tabu search algorithm with controlled randomization for constructing feasible university course timetables. Comput Oper Res 123(105007):1–31. https://doi.org/10.1016/j.cor.2020.105007
24. Al-Betar MA, Khader AT, Zaman M (2012) University course timetabling using a hybrid harmony search metaheuristic algorithm. IEEE Trans Syst Man Cybern Part C Appl Rev 42(5):664–681. https://doi.org/10.1109/TSMCC.2011.2174356
25. Paechter B (2002) A local search for the timetabling problem. In: Proceedings of the 4th international conference on the practice and theory of automated timetabling. PATAT, pp 21–23
26. Müller T (2009) ITC2007 solver description: a hybrid approach. Ann Oper Res 172(1):429–446. https://doi.org/10.1007/s10479-009-0644-y
27. Wahid J (2017) Hybridizing harmony search with local search based metaheuristic for solving curriculum based university course timetabling. In: The doctoral research abstracts, Institute of Graduate Studies, UiTM, Shah Alam 11(11). http://ir.uitm.edu.my/id/eprint/19762
28. Mazlan M, Makhtar M, Khairi AFKA, Mohamed MA (2019) University course timetabling model using ant colony optimization algorithm approach. Indonesian Journal of Electrical Engineering and Computer Science 13(1):72–76. https://doi.org/10.11591/ijeecs.v13.i1.pp72-76
29. Hossain SI, Akhand MAH, Shuvo MIR, Siddique N, Adeli H (2019) Optimization of university course scheduling problem using particle swarm optimization with selective search. Expert Syst Appl 127:9–24. https://doi.org/10.1016/j.eswa.2019.02.026
30. Gozali AA, Kurniawan B, Weng W, Fujimura S (2020) Solving university course timetabling problem using localized island model genetic algorithm with dual dynamic migration policy. IEEJ Trans Electr Electron Eng 15(3):389–400. https://doi.org/10.1002/tee.23067
31. Junn KY, Obit JH, Alfred R (2017) Comparison of simulated annealing and great deluge algorithms for university course timetabling problems (UCTP). Adv Sci Lett 23(11):11413–11417. https://doi.org/10.1166/asl.2017.10295
32. Goh SL, Kendall G, Sabar NR (2019) Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem. J Oper Res Soc 70(6):873–888. https://doi.org/10.1080/01605682.2018.1468862
33. Yusoff M, Roslan N (2019) Evaluation of genetic algorithm and hybrid genetic Algorithm-Hill climbing with elitist for Lecturer University timetabling problem. In: International conference on swarm intelligence. Springer, Cham, pp 363–373. https://doi.org/10.1007/978-3-030-26369-0_34
34. Islam T, Shahriar Z, Perves MA, Hasan M (2016) University timetable generator using tabu search. Journal of Computer and Communications 4(16):28–37. https://doi.org/10.4236/jcc.2016.416003
35. Susan S, Bhutani A (2018) Data mining with association rules for scheduling open elective courses using optimization algorithms. In: International conference on intelligent systems design and applications, Springer, Cham, pp 770–778. https://doi.org/10.1007/978-3-030-16660-1_75
36. Goh SL, Kendall G, Sabar NR, Abdullah S (2020) An effective hybrid local search approach for the post enrolment course timetabling problem. *Opsearch* 57(3):1–33. https://doi.org/10.1007/s12597-020-00444-x
37. Muklason A, Irianti RG, Marom A (2019) Automated course timetabling optimization using Tabu-variable neighborhood search

based hyper-heuristic algorithm. Procedia Computer Science 161: 656–664. https://doi.org/10.1016/j.procs.2019.11.169

38. Matias JB, Fajardo AC, Medina RM (2018) Examining genetic algorithm with guided search and self-adaptive neighborhood strategies for curriculum-based course timetable problem. In: IEEE fourth international conference on advances in computing, communication & automation, pp 1–6. https://doi.org/10.1109/ICACCAF.2018.8776728
39. Gozali AA, Fujimura S (2020) Solving University course timetabling problem using multi-depth genetic algorithm-solving UCTP using MDGA. In: SHS web of conferences. EDP Sciences, pp 1–18. https://doi.org/10.1051/shsconf/20207701001
40. Vianna DS, Martins CB, Lima TJ, Vianna MDFD, Meza EBM (2020) Hybrid VNS-TS heuristics for university course timetabling problem. Brazilian Journal of Operations & Production Management 17(2):1–20. https://doi.org/10.14488/BJOPM.2020.014
41. Gülcü A, Akkan C (2020) Robust university course timetabling problem subject to single and multiple disruptions. Eur J Oper Res 283(2):630–646. https://doi.org/10.1016/j.ejor.2019.11.024
42. Susan S, Bhutani A (2019) A novel memetic algorithm incorporating greedy stochastic local search mutation for Course scheduling. In: 2019 IEEE international conference on computational science and engineering, pp 254–259. https://doi.org/10.1109/CSE/EUC.2019.00056
43. Habashi SS, Salama C, Yousef AH, Fahmy HM (2018) Adaptive diversifying hyper-Heuristic based approach for timetabling problems. In: 2018 IEEE 9th annual information technology, electronics and mobile communication conference, pp 259–266. https://doi.org/10.1109/IEMCON.2018.8615035
44. Babaei H, Karimpour J, Hadidi A (2015) A survey of approaches for university course timetabling problem. Comput Ind Eng 86:43–59. https://doi.org/10.1016/j.cie.2014.11.010
45. Civicioglu P (2013) Backtracking search optimization algorithm for numerical optimization problems. Appl Math Comput 219(15): 8121–8144. https://doi.org/10.1016/j.amc.2013.02.017
46. Saruhan H, Rouch KE, Roso CA (2004) Design optimization of tilting-pad journal bearing using a genetic algorithm. International Journal of Rotating Machinery 10(4):301–307. https://doi.org/10.1155/S1023621X04000314
47. Karami AH, Hasanzadeh M (2012) University course timetabling using a new hybrid genetic algorithm. Computer and Knowledge Engineering, IEEE, pp 144–149. https://doi.org/10.1109/ICCKE.2012.6395368
48. Jat SN, Yang S (2009) A guided search genetic algorithm for the university course timetabling problem. In: The 4th multidisciplinary international scheduling conference: theory and applications, pp 180–191. http://bura.brunel.ac.uk/handle/2438/5880
49. Shaker K, Abdullah S, Hatem A (2012) A differential evolution algorithm for the university course timetabling problem. In: 2012 IEEE 4th conference on data mining and optimization, pp 99–102. https://doi.org/10.1109/DMO.2012.6329805
50. Azadeh A, Elahi S, Farahani MH, Nasirian B (2017) A genetic algorithm-Taguchi based approach to inventory routing problem of a single perishable product with transshipment. Comput Ind Eng 104:124–133. https://doi.org/10.1016/j.cie.2016.12.019
51. Studenovský J (2009) Polynomial reduction of time–space scheduling to time scheduling. Discret Appl Math 157(7):1364–1378. https://doi.org/10.1016/j.dam.2008.10.014
52. Aladag CH, Hocaoglu G, Basaran MA (2009) The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem. Expert Syst Appl 36(10):12349–12356. https://doi.org/10.1016/j.eswa.2009.04.051
53. Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007) A graph-based hyper-heuristic for educational timetabling problems. Eur J Oper Res 176(1):177–192. https://doi.org/10.1016/j.ejor.2005.08.012
54. Rogalska M, Bożejko W, Hejducki Z (2008) Time/cost optimization using hybrid evolutionary algorithm in construction project scheduling. Autom Constr 18(1):24–31. https://doi.org/10.1016/j.autcon.2008.04.002
55. Kifah S, Abdullah S (2015) An adaptive non-linear great deluge algorithm for the patient-admission problem. Inf Sci 295:573–585. https://doi.org/10.1016/j.ins.2014.10.004
56. Lei Y, Gong M, Jiao L, Zuo Y (2015) A memetic algorithm based on hyper-heuristics for examination timetabling problems. International Journal of Intelligent Computing and Cybernetics 8(2):139–151. https://doi.org/10.1108/IJICC-02-2015-0005
57. Soria-Alcaraz JA, Özcan E, Swan J, Kendall G, Carpio M (2016) Iterated local search using an add and delete hyper-heuristic for university course timetabling. Appl Soft Comput 40(13):581–593. https://doi.org/10.1016/j.asoc.2015.11.043
58. Beligiannis GN, Moschopoulos CN, Kaperonis GP, Likothanassis SD (2008) Applying evolutionary computation to the school timetabling problem: the Greek case. Comput Oper Res 35(4): 1265–1280. https://doi.org/10.1016/j.cor.2006.08.010
59. Nothegger C, Mayer A, Chwatal A, Raidl GR (2012) Solving the post enrolment course timetabling problem by ant colony optimization. Ann Oper Res 194(1):325–339. https://doi.org/10.1007/s10479-012-1078-5
60. Cavdur F, Kose M (2016) A fuzzy logic and binary-goal programming-based approach for solving the exam timetabling problem to create a balanced-exam schedule. International Journal of Fuzzy Systems 18(1):119–129. https://doi.org/10.1007/s40815-015-0046-z
61. Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Trans Evol Comput 7(2): 204–223. https://doi.org/10.1109/TEVC.2003.810752
62. Feng X, Lee Y, Moon I (2017) An integer program and a hybrid genetic algorithm for the university timetabling problem. Optimization Methods and Software 32(3):625–649. https://doi.org/10.1080/10556788.2016.1233970
63. Abdullah S, Turabieh H, McCollum B, McMullan P (2012) A hybrid metaheuristic approach to the university course timetabling problem. J Heuristics 18(1):1–23. https://doi.org/10.1007/s10732-010-9154-y
64. Mladenović N, Dražić M, Kovačevic-Vujčić V, Čangalović M (2008) General variable neighborhood search for the continuous optimization. Eur J Oper Res 191(3):753–770. https://doi.org/10.1016/j.ejor.2006.12.064
65. Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6): 451–470. https://doi.org/10.1023/B:HEUR.0000012446.94732.b6
66. Asmuni H, Burke EK, Garibaldi JM, McCollum B, Parkes AJ (2009) An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. Comput Oper Res 36(4):981–1001. https://doi.org/10.1016/j.cor.2007.12.007
67. Badoni RP, Gupta DK, Mishra P (2014) A new hybrid algorithm for university course timetabling problem using events based on groupings of students. Comput Ind Eng 78:12–25. https://doi.org/10.1016/j.cie.2014.09.020
68. Jat SN, Yang S (2011) A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. J Sched 14(6):617–637. https://doi.org/10.1007/s10951-010-0202-0
69. Cambazard H, Hebrard E, O'Sullivan B, Papadopoulos A (2012) Local search and constraint programming for the post enrolment-based course timetabling problem. Ann Oper Res 194(1):111–135. https://doi.org/10.1007/s10479-010-0737-7

70. Lewis R (2012) A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. Ann Oper Res 194(1):273–289. https://doi.org/10.1007/s10479-010-0696-z
71. Ceschia S, Di Gaspero L, Schaerf A (2012) Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Comput Oper Res 39(7):1615–1624. https://doi.org/10.1016/j.cor.2011.09.014
72. Soria-Alcaraz JA, Ochoa G, Swan J, Carpio M, Puga H, Burke EK (2014) Effective learning hyper-heuristics for the course timetabling problem. Eur J Oper Res 238(1):77–86. https://doi.org/10.1016/j.ejor.2014.03.046
73. Lü Z, Hao JK (2010) Adaptive tabu search for course timetabling. Eur J Oper Res 200(1):235–244. https://doi.org/10.1016/j.ejor.2008.12.007
74. Banbara M, Inoue K, Kaufmann B, Okimoto T, Schaub T, Soh T, Wanko P (2019) *teaspoon*: solving the curriculum-based course timetabling problems with answer set programming. Ann Oper Res 275(1):3–37. https://doi.org/10.1007/s10479-018-2757-7
75. Nagata Y (2018) Random partial neighborhood search for the post-enrollment course timetabling problem. Comput Oper Res 90:84–96. https://doi.org/10.1016/j.cor.2017.09.014

Amin Rezaeipanah received his B.E. in Computer Science & Engineering from Faculty of Engineering at Ferdows University, Mashhad, Iran in 2010 and M.E. in Artificial intelligence from Shiraz University, Shiraz, Iran in 2013. He is currently researcher and lecturere at Rahjuyan Danesh University, Borazjan, Iran. His main research interests consist of recommender systems, social network analysis, wireless sensor networks and large-scale data mining.

Samaneh Sechin Matoori obtained her B.E in Information Technology from Payame Noor University, Abadan, Iran in 2015. She received her M.E in Information Technology from the Islamic Azad University of Bushehr, Iran in 2018. She is currently a Ph.D. student at the Islamic Azad University of Najafabad. She is now working as a lecturer at Islamic Azad University of Abadan, Abadan, Iran. She is working on topics related to mobile networks, machine learning and data mining.

Gholamreza Ahmadi received his Bs degree in computer engineering from the University of Tehran, fanni faculty in 2000, and has received his M.Sc. degree in information technology engineering from Amir kabir University 2010. He is working in Persia Gulf university(jam branch) now. He has taught in the areas of computer and network and his research interests include data mining, optimization algorithms, network security and genetic algorithm.