University Course Timetabling Using a New Hybrid Genetic Algorithm

Amir Hossein Karami[1], *Member, IEEE* and Maryam Hasanzadeh[2]
Department of Engineering
Shahed University
Tehran, Iran
[1]amir.k1369@gmail.com, [2]hasanzadeh@shahed.ac.ir

***Abstract*—The university course timetabling problem (UCTP) is a classical, old and famous problem in the field of optimization problems. The purpose of UCTP is to schedule a number of events (courses) in proper timeslots and suitable rooms. In this problem, there are some hard and soft constraints. A feasible timetable must satisfy all hard constraints. In addition, each soft constraint violation causes a penalty. As far as UCTP is an NP-complete problem, it is reasonable to use population-based metaheuristic algorithms and evolutionary algorithms (EA). Although various methods have been presented the best results are referred to as hybrid evolutionary algorithms (HEA) and metaheuristics. The proposed method is a hybrid genetic algorithm (HGA). In our innovative HGA, the initial population which comes from heuristics is stored into red-black tree data structure. After that, our HGA creates new offsprings from previous individuals by its operators. Moreover, to improve local exploitation, we used hill climbing. The results were compared with other available ones using the 11 datasets of Socha *et al*. The results were promising and showed that the proposed HGA method is a good method to solve UCTP.**

Keywords-Hybrid Genetic Algorithm (HGA); Local Search; university course timetabling problem (UCTP); optimization

I. INTRODUCTION

In the timetabling articles, there are various definitions for UCTP. UCTP is a multi dimensional problem in which teachers, students and courses are assigned to timeslots and located in suitable rooms [1]. Because human timetabling is a sophisticated job, and in many cases it is impossible to create proper timetables, it has been about five decades that researchers have been working on automated timetabling [2]. Since 1962 when Gotlieb presented the first article about this problem, many articles have been offered in different journals [3].

Since UCTP can be reduced to a graph coloring problem, it is an NP-complete problem [4, 5]. Consequently, it is reasonable that most of the attempts to solve this problem are affiliated with population-based approaches (e.g., ant colony optimization [6-9], bee colony optimization [10], and particle swarm optimization (PSO) [11, 12]), evolutionary algorithms (e.g., genetic algorithms (GAs) [13-17], memetic algorithms (MAs) [18-20]), local searches (e.g., hill climbing, variable neighborhood search (VNS) [21, 22]), metaheuristic methods (e.g., harmony search algorithm (HSA) [23], great deluge and simulated annealing [24]) and heuristic-based approaches (e.g. tabu search) [25], etc.

Among above methods, HEA and HGA are the most interesting approaches because these algorithms use both properties of global search and local search. In fact, HEA and HGA use both exploration and exploitation ability of global and local search to find proper timetables in search space [23]. Having taken these reasons into account, we used an HGA method to solve UCTP in this article.

In our HGA method in this paper, there are some innovations. A new formula has been designed for selection operator in HGA to prevent premature convergence. Red-black tree data structure has also been used to reduce the time complexity of our algorithm. Moreover, a new concept is introduced named group of timeslots (GOT) and with GOT a new crossover algorithm has been innovated.

In the last decade, three famous timetabling competitions have been held [23]. The first one was held in 2002 (TTComp 2002). The second one was held in 2007 (TTComp 2007), and the last one was held in 2011 and continues until 31st August 2012 [23]. Metaheuristics network (MN) institute, created 11 datasets defined as Socha dataset, with the same format as TTComp 2002 datasets. The Socha dataset, is divided into 5 small, 5 medium and one large dataset. Socha dataset has the most famous one among other datasets in UCTP [23]. Besides, it offers a clear and simple definition of UCTP. Many articles compare their results with this dataset (e.g. [3, 6, 13]). So for those reasons, we used the definition offered from Socha dataset.[1]

The paper is organized as follows: The next section describes the university course timetabling problem. Section III describes our HGA method. Experimental results and Comparison between other literatures are presented in Section IV. Section V devoted to conclusion and future works.

II. THE UNIVERSITY COURSE TIMETABLING PROBLEM

The problem definition considered in this paper is Socha's dataset. It is a reduction of a typical university course timetabling problem. It consists of a set of events to be scheduled in 45 timeslots (5 days of 9 hours each), a set of rooms in which events can take place, a set of students who attend the events, and a set of features satisfied by rooms and required by events. Each student attends a number of events and each room has a size. A feasible timetable is one in which all events have been assigned a timeslot and a room so that the following hard constraints are satisfied [23]:

[1] For details, see http://iridia.ulb.ac.be/~msampels/tt.data.

- No student attends more than one event at the same time.
- The room is big enough for all the attending students and satisfies all the features required by the event.
- Only one event is in each room at any timeslot.

In addition, a candidate timetable is penalized equally for each occurrence of the following soft constraint violations [23]:

- A student has a class in the last timeslot of the day.
- A student has more than two classes consecutively.
- A student has a single class on a day.

The goal of this problem is to create feasible timetables (remember that feasible timetable means there is not hard constraint violation in it), with the lowest penalty (that occurred from soft constraint violation).

III. A Hybrid Genetic Algorithm For The University Course Timetabling Problem

In our proposed HGA, timetables are created from hybridization of genetic algorithm and hill climbing. The details of each operator and section will be discussed in the following sections. The used structure to store timetables is red-black tree. This causes the reduction of time complexity and better performance of our algorithm.

A. Initial Population

The initial population is created from three heuristic functions and random shuffle order of timeslots. First, events are sorted according to the numbers of suitable rooms that can be located on them. If an event has less suitable room, this course must be scheduled first. Actually, the first heuristic is minimum remaining value (MRV). The second heuristic is degree heuristic. According to this heuristic, events that have more conflict with other events must be scheduled sooner; indeed, an event that has more conflict with other events causes much more constraint to the other events. Thus, these events must be scheduled sooner. Note that the priority of MRV heuristic is greater than degree heuristic.

After choosing proper order for events (according to MRV and degree heuristic), we must assign suitable rooms to events. For this purpose, least constraining variable heuristic (LCV) is used. This heuristic functions in the way that, in the process of assigning room to event, first a room is chosen that has the least constraint to initialize other events. In other words, first the possibility of each room to be used by several events is evaluated. It is crystal clear that the more events exist, the less constraint the mentioned room creates for the problem. Hence, in each event's room domain (i.e. the suitable rooms for this event), it's better that one room that creates less constraint on choosing other events, be allocated to the mentioned event. This causes the other event's domain to get empty later. It is worth mentioning that by using these heuristics, a feasible timetable for small and medium datasets was created.

One of the most important factors in the improvement of GA function is the diversity of initial population [26]. For this purpose, we use random shuffle order of timeslots to create feasible and diverse timetables. In this approach, permutation of timeslots while putting events into timetable is used. This causes much diverse timetables to be created.

Consequently, in this section by integrating random shuffle order of timeslots, MRV, degree and LCV heuristics a proper and diverse initial population for GA could be created.

B. Structure of Chromosome

In this design, each chromosome is a timetable that includes events, timeslots and rooms. In essence, in this structure, a matrix with rows that are rooms and columns that are timeslots exist. Events can be scheduled in the entries of this matrix. An illustration of chromosome structure is shown in Fig. 1.

For example in Fig. 1, the math event is located in timeslot#0 and room#0. The empty entries show that in the timeslot in that room no event is held. For instance, in Fig. 1, the entry with timeslot#0 and room#1 is an empty entry.

C. Selection Operator

In order to select proper parents for crossover, a rank based selection operator is used since in rank based selection method, selective pressure can be adjusted while the algorithm runs. Selective pressure is defined by dividing the probability of selecting the best chromosome into average probability of the other chromosomes [27]. By adjusting selective pressure, premature convergence of GA could be prevented. Rank based selection is divided into linear and nonlinear categories [28]. Nonlinear ranking operators function better in comparison to linear ranking operators because in nonlinear approaches the probability of chromosome selection for crossover is distributed in a nonlinear way. This causes selection probability of appropriate chromosomes to be higher than the selection probability of inappropriate chromosomes in a nonlinear way [29]. For this reason (1), which is a nonlinear formula, was innovated in order to select proper chromosomes for crossover.

$$P_i = \alpha(1-\alpha)^{rank_i} \tag{1}$$

In (1), the selective pressure is controlled by α parameter. $Rank_i$ is the rank of chromosome i, among other chromosomes based on fitness value. The best chromosome rank is considered zero and the worst chromosome rank equals to popsize-1 (popsize is the size of initial population). Also $0<\alpha<1$.

D. The Used Data Structure to Store Individuals

Using the appropriate data structure in most cases can reduce the time complexity of an algorithm. By considering the points mentioned in the selection operator section, chromosomes should be sorted in each iteration of GA. Sorting chromosomes in survivor operator is also needed.

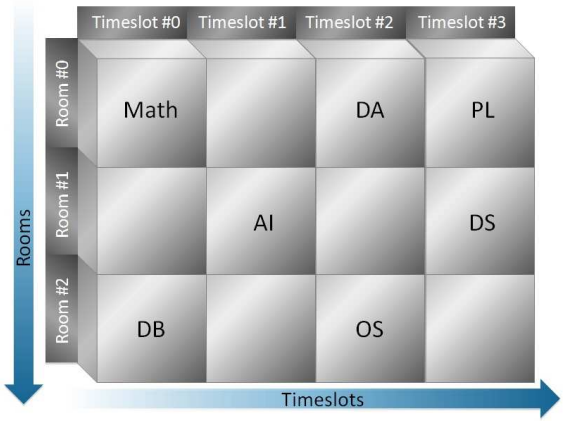

Figure 1. An illustration of chromosome structure

We came to the conclusion that red- black tree data structure is a good choice to store chromosomes since red-black tree is based on binary search tree (BST). As a result, it can hold data in a sorted manner. In addition, by considering the fact that the height of red- black tree is O(lgn) (n is the number of stored data), the insert and delete functions operate in logarithmic time [30]. Thus, using this data structure reduces the overhead of sorting chromosomes in each iteration of GA.

E. Crossover Operator

By using the two selected parents' data in selection operator, one chromosome is created as a child. In general, if the structure of chromosome in GA is block by block, then in crossover operator it could be used and each chromosome's properties could be used well. Therefore, in the proposed approach, the structure of chromosomes was smartly taken into consideration. For this, in each chromosome structure a concept named group of timeslots (GOT) was innovated. As a matter of fact, each GOT is a gene that is considered a block. Each GOT has 9 timeslots. In other words, each GOT is a working day. Because all soft constraints are related to a working day, we introduced GOT. In order to incorporate each event into timetable, two values must be assigned (timeslot and room). The presence of these two values made designing the structure of chromosome as a block a complicated matter. It must be noted that a block design of chromosome and the crossover operator, which will be discussed, is innovative. The idea of this crossover cannot be found in any articles related to UCTP. Before designing crossover operator based on GOT, a criterion for the superiority of GOTs must be considered. It is clear that the more events existing in a GOT and the less fitness value (violation of soft constraints that occurred into a GOT) of GOT, the better a GOT is. By using GOT courses and Fitness value of GOT, the below equation is designed.

$$GOT_{goodness}=|GOT\_Courses|\times(\alpha-(GOT\_Fitness\times\beta)) \quad (2)$$

In (2), GOT_Fitness and GOT_Courses represent the fitness value and the number of course that are placed in a GOT. α and β are two parameters that can be changed and this change indicates the importance of events that are in a GOT and its fitness. Experiment results show α=500 and β=50 are proper.

Now the operation of crossover is explained in details. At first, the best GOT between GOTs of parents, that were chosen before, are selected and put in the child. The best GOT is a GOT that its $GOT_{goodness}$ is more than other GOTs. Then, the events that were placed in this GOT are omitted from each parent and $GOT_{goodness}$ value of each parent's GOT is updated. The same procedure is repeated until a new child is born (until each GOT is valued; for example, if five working days are available, this procedure ends when all five days are valued, and the best working days are incorporated in the child's timetable). After the end of this procedure, if there is any event left in the parents that are not placed in the child, those would be placed in the appropriate place in the child's timetable.

The crossover that is designed for this problem uses the parent's data in the best way and creates a very good new child. This crossover guaranties that the new born child would not violate any of the hard constraints, and all existing events can be found in it. The pseudo code of the crossover operator is shown as Algorithm 1.

F. Mutation Operator

The purpose of this section is to create a mutation on chromosomes so that chromosomes would get away from local optimum and the algorithm converges to global optimum. The proposed mutation operator consists of three various parts, denoted as M1, M2 and M3. They are described as follows:

M1: in M1, one of the events that have caused violation in the timetable is randomly conveyed to one of the proper timeslots of that event. The proper timeslot is a timeslot in which the mentioned event has no conflict with other events. An illustration of M1 operator is shown in Fig. 2. In the hypothetical timetable present in Fig. 1 the PL event has caused penalty. In Fig. 2, there has been a change in the timeslot of this event.

M2: in M2, one event is randomly selected (an event that has not caused violation) and is conveyed to another timeslot. An illustration of M2 operator is shown in Fig. 3. In this section, it is assumed that in the timetable present in Fig. 1, the event AI has not caused penalty, but in M2 operator it has been randomly selected, and its timeslot has changed from timeslot#1 to timeslot#2.

Algorithm 1 Crossover operator

```
Crossover(Parent P1, Parent P2)
1  Let C as a new empty chromosome (New Child)
2  for i = 1 to D do  //D is the number of working days
3     sort GOT of P1 & P2 as GOT_goodness
4     choose the best GOT from above and put it to C
5     remove courses in best GOT, from other parent's GOTs
6     update GOT_goodness of parent's GOTs
7  end for
8  if remains any courses that has not put to C then
9     put them to proper timeslot in C
10 end if
11 return C
```

M3: in M3, an event is randomly selected and its room changes in the same timeslots in which it is located. An illustration of M3 operator is shown in Fig. 4. In this section, it is assumed that the event math has been randomly selected, and its room has changed from room#0 to room#1.

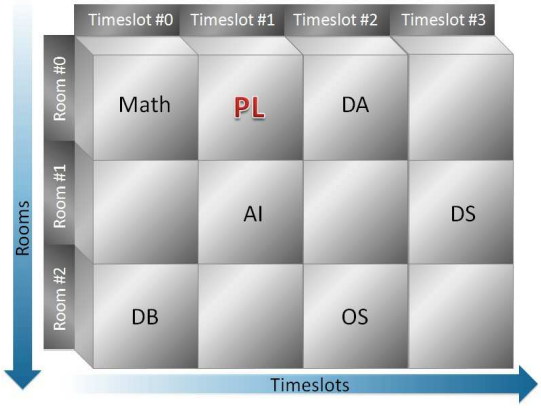

Figure 2. An illustration of M1 operator

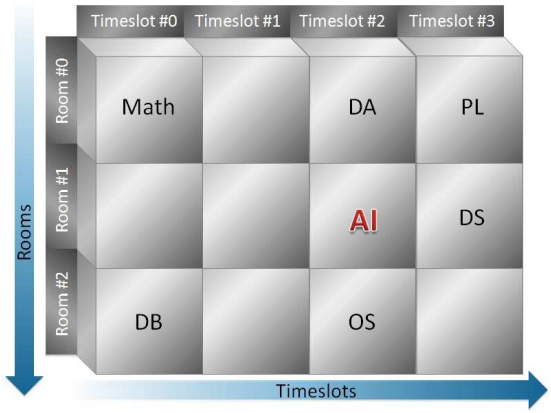

Figure 3. An illustration of M2 operator

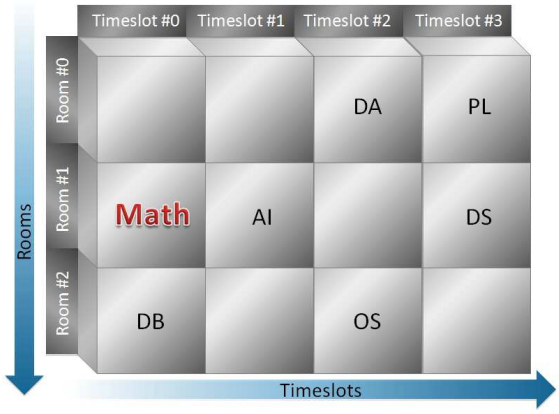

Figure 4. An illustration of M3 operator

G. *Hill Climbing Algorithm*

Up to this stage, the created child from the crossover section has also been mutated. In this stage, the hill climbing algorithm is exerted on the child with P_{HC} probability. The proposed hill climbing algorithm consists of three neighborhood structures.

Neighborhood structure1: in neighborhood structure1, each event existing in the timetable is conveyed to another appropriate timeslot (a timeslot in which no event conflicts with that event).

Neighborhood structure2: in Neighborhood structure2, events that are in conflict with each other, if possible, are swapped by one another.

Neighborhood structure3: in Neighborhood structure3, two timeslots are swapped by each other i.e. all existing events in those two timeslots are swapped by each other.

All the mentioned movements are only accepted on the condition that they reduce the penalty of timetable; otherwise, they are not accepted.

H. *Survivor Operator*

In the proposed HGA, the size of population in each stage is constant. The inappropriate chromosomes are removed from the population and then the new offsprings are added to the population. It must be noted that the number inappropriate chromosomes is equal to the number of the created offsprings. In addition, half of the chromosomes are always kept in the population in order not to lose the final result of the problem.

I. *Stop Criteria*

The mentioned algorithm continues up one of the stop criteria conditions in which a timetable with zero penalty is created or the number of iterations of HGA reaches the assigned limit. The final pseudo code of the proposed HGA is shown as Algorithm 2.

IV. EXPERIMENTAL RESULTS AND COMPARISON WITH OTHER ARTICLES

The proposed HGA method was programmed in GNU C++ under Windows 7 on an Intel core i7 processor with 6GB of RAM. Our HGA was tested on the renowned Socha's datasets described in section I.

Algorithm 2 The proposed hybrid genetic algorithm (HGA)

```
Hybrid genetic algorithm
Call Hill_Climbing algorithm to all of individuals
for i = 1 to number of iterations do
   if best chromosome in the population reached zero penalty then
      terminate HGA
   end if
   call Parent_Selection operator()
   C ⟵ call Crossover(P1, P2) //Note that P1 & P2 are parents that come from Parent_Selection operator
   MC ⟵ call Mutation(C)
   choose r~U(0,1) //r is a random number between zero & one
   if r < PHC then
      HCMC ⟵ call Hill_Climbing(MC)
   end if
   call Survivor_Operator()
end for
```

The characteristics of these datasets are show in Table I. Table I introduces the features of each dataset. Remember that in Socha's datasets there are 5 small instances, 5 medium instances and 1 large instance. The parameters set to our HGA are presented in Table II. The term "size of mating pool", in Table II means in each iteration of our algorithm, 10 individuals are selected to crossover and by these 10 individuals, 5 new offsprings are created.

We produce our best results over 3 runs, and it takes approximately 20 hours for each dataset. The best results of our algorithm are shown among 10 other article's results in Table III. In Table III, the term "%In" represents the percentage of runs that failed to obtain a feasible timetable. In addition the best results for a dataset were represented as bold font.

The results show that our HGA can produce feasible timetables with zero penalty for all small datasets, which is the simplest dataset, similar to many other articles. Also in sequence of medium1 to medium5 datasets, our HGA is placed in rank 2, 4, 1, 2 and 5.

Unfortunately, our HGA cannot produce a feasible timetable for large dataset. The best result for large dataset in HGA was a timetable that violates 5 hard constraints.

V. Conclusion and Future Works

On the whole, it could be concluded that the obtained results in comparison with the other articles results existing in this field are promising. Regarding the medium instances, our HGA's convergence rate is better than that of others.

As it can been seen in Table II, the results are achieved just by 200 times of iteration of proposed HGA. On the other hand, in HEA method presented in Table III, a method based on hybrid evolutionary algorithms, the number of iterations of HEA is 200000 [20]. It must be emphasized that our method considering its great convergence rate can be a suitable approach in order to solve the UCTP at a reasonable time.

Unfortunately, regarding the large instance, our HGA was not able to produce a feasible timetable. Our method like six other approaches in Table III cannot produce a feasible timetable for large instance. The inability to produce a feasible timetable in the large instance could be considered as the major problem of our algorithm.

Overall in this paper, new ideas such as the use of red-black tree to store individuals and a new crossover operator for this problem which cannot be found in the previous articles were presented. Hybridizing hill climbing algorithm and GA has also improved the efficiency of simple GA.

TABLE I. Characteristics of Socha's Datasets

Class	Small	Medium	Large
Number of events	100	400	400
Number of Rooms	5	10	10
Number of features	5	5	10
Number of Timeslots	45	45	45
Approximate features per room	3	3	5
Percentage of the feature used	70%	80%	90%
Number of students	80	200	400
Maximum events per student	20	20	20
Maximum students per event	20	50	100

TABLE II. Parameters of HGA For Test

Parameters of Hybrid Genetic Algorithm (HGA) for test		
Parameter	**Small**	**Medium**
Size of Population	50	10
Size of mating pool	10	10
Number of Iterations	50	200
P_{HC} *(in hill climbing algorithm)*	0.5	0.5

TABLE III. Comparison of 11 Different Algorithms on Small and Medium instances

Datasets	HGA	MA	RIIA	HEA	GBHH
	Best	Best	Best	Best	Best
Small1	**0**	**0**	**0**	**0**	6
Small2	**0**	**0**	**0**	**0**	7
Small3	**0**	**0**	**0**	**0**	3
Small4	**0**	**0**	**0**	**0**	3
Small5	**0**	**0**	**0**	**0**	4
Medium1	180	227	242	221	372
Medium2	176	180	161	**147**	419
Medium3	**219**	235	265	246	359
Medium4	150	**142**	181	165	348
Medium5	196	200	151	**130**	171

Datasets	VNS	THHS	LS	EA	AA	FA
	Best	Best	Median	Best	Median	Best
Small1	**0**	1	8	**0**	1	10
Small2	**0**	2	11	3	3	9
Small3	**0**	**0**	8	**0**	1	7
Small4	**0**	1	7	**0**	1	17
Small5	**0**	**0**	5	**0**	**0**	7
Medium1	317	**146**	199	280	195	243
Medium2	313	173	202.5	188	184	325
Medium3	357	267	77.5%In	249	248	249
Medium4	247	169	177.5	247	164.5	285
Medium5	292	303	100%In	232	219.5	132

Legend:
HGA: our hybrid genetic algorithm
MA: The memetic algorithm approach by Jat et al., 2008 [3]
RIIA: The randomized iterative improvement method by Abdullah et al., 2005 [31]
HEA: The hybrid evolutionary algorithm by Abdullah et al., 2007 [20]
GBHH: The graph based hyper heuristic by Burke et al., 2007 [32]
VNS: The variable neighborhood search by Abdullah et al., 2005 [33]
THHS: The tabu based hyper heuristic search by Burke et al., 2003 [25]
LS: The local search method by Socha et al., 2002 [6]
EA: The evolutionary algorithm by Rossi-Doria et al., 2002 [34]
AA: The ant algorithm by Socha et al., 2002 [6]
FA: The fuzzy algorithm by Asmuni et al., 2005 [35]

In the future, the combination of backtracking algorithm and the initial population creator algorithm described in this paper could be used to improve the results of algorithm.

REFERENCES

[1] M. Carter and G. Laporte, "Recent developments in practical course timetabling," Practice and Theory of Automated Timetabling II Springer, pp. 3-19, 1998.

[2] P. Rattadilok, A. Gaw and R. Kwan, "Distributed choice function hyper-heuristics for timetabling and scheduling," Practice and Theory of Automated Timetabling – Springer, pp. 51-67, 2005.

[3] S. N. Jat and S. Yang, "A memetic algorithm for the university course timetabling problem," 20th IEEE International Conference on Tools with Artificial Intelligence, pp. 427-433, 2008.

[4] N. Srndic, E. Pandzo, M. Dervisevic and S. Konjicija, "The application of a parallel genetic algorithm to timetabling of elementary school classes: a coarse grained approach," International Symposium on Information, Communication and Automation Technologies (ICAT), pp. 1-5, 2009.

[5] R. Lewis and B. Paechter, "Finding feasible timetables using group-based operators," IEEE Transactions on Evolutionary Computation, VOL. 11, NO. 3, pp. 397-413, June 2007.

[6] K. Socha, J. Knowles and M. Sampels, "A max-min ant system for the university course timetabling problem," Springer, pp. 63-77, 2002.

[7] M. Ayob and G. Jaradat, "Hybrid ant colony systems for course timetabling problems," 2nd IEEE Conference on Data Mining and Optimization, pp. 120 - 126, 2009.

[8] D. Djamarus and K. R. Ku-Mahamud, "Heuristic factors in ant system algorithm for course timetabling problem," 9th IEEE International Conference on Intelligent Systems Design and Applications (ISDA'09), pp. 232-236, Dec, 2009.

[9] T. Lutuksin and P. Pongcharoen, "Best-worst ant colony system parameter investigation by using experimental design and analysis for course timetabling problem," IEEE International Conference on Computer and Network Technology (ICCNT), pp. 467-471, 2010.

[10] N. T. T. M. Khang, N. B. Phuc and T. T. H. Nuong, "The bees algorithm for a practical university timetabling problem in vietnam," IEEE International Conference on Computer Science and Automation Engineering (CSAE), pp. 42-47, June, 2011.

[11] I. S. F. Ho, D. Safaai and M. Zaiton, "A combination of PSO and local search in university course timetabling problem," IEEE International Conference on Computer Engineering and Technology (ICCET), pp. 492-495, Jan, 2009.

[12] I. Sheau Fen Ho, D. Safaai and S. Hashim, "A study on PSO-based university course timetabling problem," IEEE International Conference on Advanced Computer Control (ICACC'09), pp. 648-651, Jan, 2009.

[13] S. Yang and S. N. Jat, "Genetic algorithms with guided and local search strategies for university course timetabling," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, pp. 93-106, 2011.

[14] S. Abdullah and H. Turabieh, "Generating university course timetable using genetic algorithms and local search," IEEE International Conference on Convergence and Hybrid Information Technology (ICCIT'08), pp. 254-260, Nov, 2008.

[15] N. Najdpour and M. R. Feizi-Derakhshi, "A two-phase evolutionary algorithm for the university course timetabling problem," IEEE International Conference on Software Technology and Engineering (ICSTE), Vol 2. pp. 266- 271, Oct, 2010.

[16] R. Els and N. Pillay, "An evolutionary algorithm hyper-heuristic for producing feasible timetables for the curriculum based university course timetabling problem," IEEE International Conference on Nature and Biologically Inspired Computing (NaBIC), pp. 460-466, Dec, 2010.

[17] O. Mk. Alsmadi, Z. S. Abo-Hammour, D. I. Abu-Nadi, A. Algsoon, "A novel genetic algorithm technique for solving university course timetabling problems," 7th IEEE International Workshop on Systems, Signal Processing and their Applications (WOSSPA), pp. 195-198, May, 2011.

[18] Z. Wang and J. Liu, "Hybrid memetic algorithm for uniting classes of university timetabling problem," IEEE International Conference on Computational Intelligence and Security (CIS'09), pp. 97-101, Dec, 2009.

[19] A. Alkan and E. Ozcan, "Memetic algorithms for timetabling," IEEE Evolutionary Computation, VOL. 3, pp. 1796 - 1802, May, 2004.

[20] S. Abdullah, E.K. Burke and B. McCollum, "A hybrid evolutionary approach to the university course timetabling problem," IEEE Congress on Evolutionary Computation 2007 (CEC 2007), pp. 1764 - 1768, 2007.

[21] D. T. Anh, V. H. Tam and N. Q. V. Hung, "Generating complete university course timetables by using local search methods," Proceedings of the 4th IEEE International Conference on Research, Innovation and Vision for the Future (RIVF'06), 2006.

[22] A. Abuhamdah and M. Ayob, "Multi-neighbourhood particle collision algorithm for solving course timetabling problems," 2nd IEEE Conference on Data Mining and Optimization, pp. 21-27, Oct, 2009.

[23] M. A. AL-Betar, A. T. Khader and M. Zaman, "University course timetabling using a hybrid harmony search metaheuristic algorithm," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, pp. 1-18, 2012.

[24] P. Mcmullan, "An extended implementation of the great deluge algorithm for course timetabling," Springer International Conference on Computational Science (ICCS), pp. 538-545, 2007.

[25] E. K. Burke, G. Kendall and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," Springer Journal of Heuristics, pp. 451-470, 2003.

[26] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, The Springer Press, 2007.

[27] S. Sumathi, T. Hamsapriya, P. Surekha, Evolutionary Intelligence: An Introduction to Theory and Applications With Matlab, The Springer Press, 2008.

[28] A. P. Engelbrecht. Computational Intelligence, Second Edition. Wiley Press, 2007.

[29] X. Yu, M. Gen. Introduction to Evolutionary Algorithms, The Springer Press, 2010.

[30] T.H. Cormen, C.E Leiserson, R.L Rivest and C. Stein, Introduction to Algorithms, Second Edition, The MIT Press, 2002.

[31] S. Abdullah, E. K. Burke, B. McCollum, "Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem," Metaheuristics International Conference (MIC'2005), Aug, 2007.

[32] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," European Journal of Operational Research, pp. 177-192, 2007.

[33] S. Abdullah, E. K. Burke, B. McCollum, "An investigation of a variable neighbourhood search approach for course timetabling," In: The Proceeding of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005), New York, USA, pp. 413-427, July 18th -21st, 2005.

[34] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T.Stutzle. "A comparison of the performance of different metaheuristics on the timetabling problem," Lecture Notes in Computer Science, pp. 329–351, 2002.

[35] H. Asmuni, E. K. Burke, and J. M. Garibaldi. "Fuzzy multiple heuristic ordering for course timetabling," Proc of the 5th UK Workshop on Comput Intell, pp. 302-309, 2005.