Int. J. Res. Ind. Eng. Vol. 5, No. 1-4 (2016) 1-15

International Journal of Research in Industrial Engineering
www.riejournal.com

Modeling and Scheduling University Course Timetabling Problems

B. Naderi*

Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran.

A B S T R A C T
This paper considers the problem of university course timetabling. In this problem, there are a set of courses, lecturers and classrooms. The objective is to assign schedule courses so as to maximize the total preference of lecturer-course, lecturer-day and course-day. The paper first formulates the problem in form of linear integer programming model. Using the model and commercial software, the small sized instances are optimally solved. Then, the paper proposes three different algorithms based on imperialist competitive algorithm, simulated annealing and variable neighborhood search. The algorithms employ several novel procedures such as encoding scheme, move operator, crossing operators. The algorithms are tuned and evaluated with optimal solutions found by the model. Then, they are evaluated by comparing their performance. The results show that imperialist competitive algorithm outperforms the other algorithms.

Keywords: *University course scheduling, mathematical model, Metaheuristics.*

Article history:
Received: 31 July 2016 Revised: 18 August 2016 Accepted: 07 September 2016.

1. Introduction

A scheduling problem consists of utilizing a set of limited resources to perform a set of tasks. One of the most important scheduling problems is educational timetabling [17]. In classical educational timetabling problems, we need to assign a set of events (such as courses and exams) into a certain number of 'classroom-time' slots subject to a set of different constraints [19]. Timetabling problems have attracted significant research activities from operations research and artificial intelligence. It is a very well-known problem since many academic researchers are daily confronted with university timetabling [6].
The general educational timetabling problems commonly refer to both course and examination timetabling which are different in nature. In the examination timetabling, one goal is to spread the different exams for each student as evenly as possible, while in course timetabling the students want an as compact timetable as possible [7]. This paper studies a

*Corresponding author
E-mail address: *bahman.naderi@aut.ac.ir*
DOI: 10.22105/riej.2017.49167

class of educational timetabling problems, known as the university course timetabling problem (UCTP). It has always been a difficult task for academic offices at universities each semester to manage this problem. The job of course arrangement is a complex system in which various issues must be considered.

The UCTP has been proved to be an NP-hard [3]. It includes two decision dimensions: teacher assignment and class scheduling. The teacher assignment is to specify which teachers will teach what courses. The class scheduling determines each course will present in which class. The decisions must be taken so as to satisfy a given set of constraints. It should consider teacher's professional knowledge and qualifications, teacher preferences (such as their preferred courses, days and time periods), fair distribution of overtime among the teachers, student requests, expectations in class offerings, curriculum planning policies of the school and school's available equipment and facilities. Any conflict with teacher schedules and class rooms is also avoided. All these issues make the problem very hard to solve in real world circumstances [16].
The constraints that contribute to the complexity of UCSP can be divided into two categories; hard and soft [18]. For a timetable to be feasible, all hard constraints must be satisfied. An example of a hard constraint is that no student should be required to simultaneously sit two classes. On the other hand, soft constraints are requirements that are not essential but should be held as far as possible (soft constraints can be contravened if necessary). Therefore, soft constraints are sometimes referred as preferences; and they are to evaluate the quality of the solutions. For an example, a common soft constraint is to evenly spread classes to the best possible extent. The overall objective is to find a feasible timetable that satisfies all the hard constraints to maximize the satisfaction of both teachers and classes based on their preferences (or minimize the violation of the soft constraint).
The UCTP can totally differ from one university to another [2, 6, 9, 30]. Every university has a unique set of requirements in order to effectively utilize their resources, meet the requirements of their business, provide a high level of satisfaction to their students etc. Therefore, a customized procedure for the course scheduling system must be developed to fulfill all these unique requirements.
The rest of the paper is organized as follows. Section 2 formulates the problem. Section 3 develops solution algorithms. Section 4 presents the results of experiments. Finally, Section 5 concludes the paper.

2. Problem formulation

The first step to study optimization problems is building a mathematical model. For a long time, mathematical models were only used to express all the characteristics of a problem. If the problem is NP-hard, solving the mathematical model is not an effective solution method. Moreover, the mathematical model is a starting point in other solution algorithms such as branch and bound, and their performance highly depends on the effectiveness of the model. All these reasons together show the importance of the model. Timetabling problems are commonly formulated by integer linear programming models. The proposed mathematical model is described in the following sections. The notations and parameters used in both models are as follows.

The UCTP consists of a set of m courses, a set of n lecturers and a set of e classrooms. They should be scheduled in d days and on each day, there are time periods. In each time period, one course can be executed at each classroom. Each lecturer has his own preference to teach each course among the courses of his expertise. Each course can be executed only in a subset of classrooms and a subset of days. The objective is to maximize the preference of lecturers and to minimize the number of used classrooms. The following parameters and indices are established.

n	The number of lecturers
m	The number of courses
e	The number of classrooms
d	The number of days
k	Index for working days $\{1,2,\dots,d\}$
i	Index for lecturers where $\{1,2,\dots,n\}$
j	Index for courses where $\{1,2,\dots,m\}$
l	Index for classrooms $l=\{1,2,\dots,e\}$
t	Index for time period $t=\{1,2,\dots,3\}$
$p^1_{i,j}$	The preference of lecturer i for teaching course j.
$p^2_{i,k}$	The preference of lecturer i for being invited on day k.
$p^3_{j,k}$	The preference of course j for being presented on day k.
$a_{i,k}$	Parameter taking value 1 if lecturer i can be invited on day k, and 0 otherwise.
$b_{i,j}$	Parameter taking value 1 if lecturer i can teach course j, and 0 otherwise.
$c_{j,l}$	Parameter taking value 1 if course j can be presented in classroom l, and 0 otherwise.

Decision Variables:

$Z_{i,j,k,l,t}$	Binary variable taking value 1 if lecturer i teaches course j on day k in classroom l in time period t, and 0 otherwise.
$X_{i,k}$	Binary variable taking value 1 if lecturer i is invited on day k, and 0 otherwise.

$$\text{Maximize } Z=\sum_{i=1}^{n}\sum_{k=1}^{d}X_{i,k}\cdot p^1_{i,k}+\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{d}\sum_{l=1}^{e}\sum_{t=1}^{3}Z_{i,j,k,l,t}\cdot p^2_{i,j}+\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{d}\sum_{l=1}^{e}\sum_{t=1}^{3}Z_{i,j,k,l,t}\cdot p^3_{j,k} \quad (1)$$

Subject to:

$$\sum_{i=1}^{n}\sum_{k=1}^{d}\sum_{l=1}^{e}\sum_{t=1}^{3} Z_{i,j,k,l,t} = 1 \qquad \forall_{j} \qquad (2)$$

$$\sum_{j=1}^{m}\sum_{l=1}^{e} Z_{i,j,k,l,t} \leq X_{i,k} \qquad \forall_{i,k,t} \qquad (3)$$

$$X_{i,k} \leq a_{i,k} \qquad \forall_{i,k} \qquad (4)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} Z_{i,j,k,l,t} \leq 1 \qquad \forall_{k,l,t} \qquad (5)$$

$$\sum_{k=1}^{d}\sum_{l=1}^{e}\sum_{t=1}^{3} Z_{i,j,k,l,t} \leq b_{i,j} \qquad \forall_{i,j} \qquad (6)$$

$$\sum_{i=1}^{n}\sum_{k=1}^{d}\sum_{t=1}^{3} Z_{i,j,k,l,t} \leq c_{j,l} \qquad \forall_{j,l} \qquad (7)$$

$$X_{i,k} \in \{0,1\} \qquad \forall_{i,k} \qquad (8)$$

$$Z_{i,j,k,l,t} \in \{0,1\} \qquad \forall_{i,j,k,l,t} \qquad (9)$$

Equation (1) is the objective function which calculates the total utility. Constraint set (2) ensures that each course is presented. Constraint set (3) determines the days a lecturer has courses to teach. Besides, it assures that each lecturer teaches at most one course at any time slot. Constraint set (4) limits lecturers' course to the days they prefer. onstraint set (5) avoids cross assignment, i.e., at each classroom, at most one course can be presented at a time. Constraint set (6) specifies that each lecturer is assigned to courses that he/she can teach. Constraint set (7) ensures that each course is assigned to classes having necessary equipment. Constraint sets (8) and (9) define the decision variables.

3. Solution algorithms

The problem of university course scheduling is known to be a hard problem to solve. Therefore, the most effective algorithms to solve it are metaheuristics [14]. Examples of these algorithms include graph coloring heuristics [6], Tabu Search [31], simulated annealing [29], evolutionary algorithms [27], case-based reasoning [5], two-stage heuristic algorithms [8, 10] and so on. Interested readers are referred to [18] for a comprehensive survey of the automated approaches for university timetabling presented in recent years.
This paper proposes three different metaheuristics, imperialist competitive algorithm, simulated annealing and variable neighborhood search. We first explain the encoding scheme used in the algorithms and then describe the algorithms.

3.1. The encoding scheme

The first step to develop an algorithm is to design an encode scheme to represent solutions of the problem under consideration to the algorithm. The encoding scheme could be either direct or indirect encoding. The indirect encoding schemes are those we need to decode the solutions to calculate the objective functions while the direct encoding schemes are those in which the objective function of the corresponding solution is directly calculated from the encoded solution. In another sense, the encoding scheme could be either complete or incomplete. The complete encoding schemes are those can represent all possible solution for the problem while incomplete schemes are those by which all solutions of the problem cannot be represented.

We represent the solution space by two binary matrixes and a dispatching rule. The first binary matrix shows the course-lecturer assignment; that is, which course is taught by which lecturer. In this matrix, rows and columns represent courses and lecturers, respectively where ”1” means assignment while “0” means non-assignment and “-” means the corresponding lecturer cannot teach the course. The second matrix represents the lecturer-day invitation; that is, each course-lecturer is presented on which day. In this matrix, row and columns represent days and lecturers, respectively where “1” means invitation, “0” means non-invitation and “-” means the lecturer cannot be invited on that day. Notice that a lecturer at each day can have at most 3 courses. Therefore, the number of days that a lecturer is invited depends on the number of courses assigned to.

The dispatching rule applied here is to assign courses to classroom-time slots. Once the two decisions of course-lecturer assignment and lecturer-day are specified, the classroom-time slot decision is remaining; that is, which course is presented in what classroom and which time slots regarding the hard constraints of the problem. We propose the following rule to do so. Each course is assigned to the first available classroom that is qualified for the course when the lecturer is also available. If a lecturer is invited on more than one day, each course is presented on the day with the highest preference and available classrooms. Notice that this representation is complete and indirect. It is indirect since we need to decode the solution in order to calculate the objective functions and it is complete because all possible solutions for the problem can be represented.

3.2. Imperialist competitive algorithm

The imperialist competitive algorithm (ICA) is a novel population based evolutionary algorithm to solve various optimization problems. This algorithm contains a population of agents, known as countries where they are classified as imperialists and colonies. A collection of one imperialist and several colonies is called an empire. The basis of ICA is to simulate three sociopolitical processes among the empires: imperialistic behavior, imperialistic competition and independence. The idea behind the imperialistic behavior is that the imperialist attempts to penetrate the colony by attracting the culture and the social structure of each colony toward itself. During the imperialist competition, weak empires collapse and powerful ones take possession of their colonies. There is always a probability for some colonies to jointly separate from their empires and constitute a new empire.

3.2.1. *Initialization*

ICA starts with a number of countries each of which represents a possible solution for the problem. It selects those with relative high fitness to be the imperialist, and the remaining becomes the colonies of these imperialists. The number of the colonies in each empire depends on the power of its imperialist. Hence, powerful imperialists have greater number of colonies while weaker ones have less.

The number of countries is the population size indicated by pop. The initial countries are randomly generated from the feasible solutions. To define the initial imperialists, the first I best countries of the population are selected as the imperialist and the rest as colonies. Therefore, there are I empires. To rank the countries, we need to calculate the fitness (i.e., the value of objective function). To assign colonies to imperialist, a stochastic procedure in which more chance is given to more powerful imperialists. To chance of empire k to hold each colony is as follows.

$$p_k = \frac{fit(k)}{\sum_{h=1}^{pop} fit(h)}$$

3.2.2. *Imperialist behavior mechanism*

After forming initial empires, the imperialist behavior mechanism commences and the colonies of an empire move towards their imperialist. While a colony approaches its imperialist, it might become more powerful (better fitness) than its imperialist. In this case, the colony overcomes the imperialist and takes the control of the whole empire. In fact, the colony and the imperialist swap their positions. Then, the procedure continues by the new imperialist and colonies change their path and start moving toward this new imperialist. After the exchanging step, the total power of each empire is recalculated which depends on both the power of the imperialist and its colonies.

To take a colony towards its imperialist, we define a new country that inherits from both the colony and imperialist. In fact, we combine the colony and imperialist to form a new country. This is done through an operator with the following steps.

For each lecture a random number between 0 and 1 is generated. If it is less than 0.6, the two columns of that lecturer from the imperialist (i.e., one from the lecturer-course assignment and one in the lecturer-day assignment) are copied into the new solution. Other columns of the new solution are filled from the colony. It is necessary to indicate that the new solution probably needs modification to be feasible solution. In the lecturer-course assignment, if a course is assign to two lecturers, one of the lecturers is randomly selected and the other one is crossed out. Moreover, if a course is not assigned to any lecturer, it is randomly assigned to a lecturer. Regarding this new lecturer-course assignment, lecturer-day assignment is updated.

After colonies are taken towards the current imperialist, the imperialist of the empire is updated. In other words, it is checked whether any of the new countries can beat the imperialist or not. If this is the case, the imperialist is replaced with that new country.

Then, the total power of empire is reevaluated. It is recommended to use both power of imperialist and colonies to calculate the total power. We use the following formula to obtain the total power (tp) of empire k.

$$tp_k = fit_{imperialist}\left(1 + \frac{\sum_{h=1}^{s_k} fit_h}{\sum_{h=1}^{pop} fit_h}\right)$$

where s_k is the number of countries in empire k. Figure 1 shows the general procedure of the imperialist behavior mechanism.

```
The procedure: The imperialist behavior mechanism
For $k = 1$ to $I$ do
Take imperialist of empire $k$
    For $x = 1$ to $s_k$ do
        Generate new countries by combing imperialist and $x$th colony of empire
        $k$ using cyclic operator
    Endfor
    Evaluate the new countries.
    Update the imperialist and colonies of empire $k$
Endfor
Calculate the total power of all the empires
```

Fig 1. The general procedure of imperialist behavior mechanism

3.2.3. *Imperialist competition mechanism*

In the imperialistic competition process, empires endeavor to conquer colonies of other empires and control them. When an imperialist broadens its empire by conquering more colonies, it becomes more enhanced. On the other hand, the imperialist losing its colonies becomes weaker. Once an empire loses all of its colonies, it is collapsed. After a while, all the empires, one by one with exception of the most powerful one, will vanish. When all the colonies of the single remaining empire have the same position with their imperialist, consequently the same fitness, the algorithm converges to the best solution. To implement this concept, at each iteration, the weakest empire is selected and its weakest colony is given to the most powerful empire.

3.3. Simulated annealing

The simulated annealing (SA) is a local search based metaheuristic simulating the annealing process [28]. SA includes a mechanism, called acceptance criterion, which enables it to partially avoid getting trapped in local optima. The acceptance criterion decides if the new generated solution is accepted or not. In this mechanism, even inferior solutions might be accepted.

3.3.1. *The structure and acceptance criterion*

Simulated annealing starts from an initial solution, and a series of moves are made until a stopping criterion is met. The basic idea of SAs is to generate a new permutation s by an operator from the neighborhood of the current permutation x. This new sequence is accepted or rejected by another random rule. A parameter t, called the temperature, controls the acceptance rule. The variation between objective values of two candidate solutions is computed $\Delta = fit(s) - fit(x)$. If $\Delta \leq 0$, permutation s is accepted. Otherwise, permutation s is accepted with probability equal to $exp(\Delta/t_i)$. The algorithm proceeds by trying a fixed number of neighborhood moves at each temperature t_i, while temperature is gradually decreased. We use exponential cooling schedule, $t_i = \alpha . t_{i-1}$ (where $\alpha \in (0, 1)$ is temperature decrease rate). The initial temperature is set to be 50 and $\alpha = 0.97$.

3.3.2. *Move operator*

In this research, to generate a new solution from the current solution the following procedure is used. One randomly selected course is randomly assigned to another qualified lecturer. Notice that course reassignment affects two lecturers, one with course reduction and the other with course addition. Thus, the number of days that each of these two lecturers is invited might change. Regarding this new assignment, lecturer-day invitations of both lecturers are updated.

3.4. Variable neighborhood search

The general timetabling problem is known to be complex and difficult. In this context, exact solutions would be only possible for problems of limited sizes. Instead, solution algorithms based on metaheuristics have shown to be highly effective. Examples of these algorithms include genetic algorithm [20, 21], Tabu Search [1], simulated annealing [22], variable neighborhood search [4] and so on.

Variable neighborhood search (VNS) is a simple but effective local search based metaheuristic proposed by Mladenovic and Hansen [15]. Local search based methods have been applied in the optimization literature with very good results, like simulated annealing (SA), tabu search (TS) and the iterated local search (ILS). However, all these methods are based on the exploration of a single neighborhood structure. Hence, there exists high probability for them to get trapped in local optima after a certain number of iterations and the move required to separate the algorithms from the local optima cannot be performed. Therefore, they need mechanisms to have sufficient potentiality to escape from local optima.

The reasons why VNS has obtained its acceptability and popularity among researches are due to the utilization of several neighborhood structures, easy to implement and high flexibility and brilliant adaptability of VNS to different problems. VNS has been applied with success to other problems including [10, 12]. In the following sections we further detail the proposed VNS methods.

3.4.1. *General structure*

Instead of iterating over one constant type of neighborhood structure and relying on mechanisms such as random perturbations of ILS or memory structures of TS or metropolis mechanism of SA, VNS proceeds in this case by using a different type of neighborhood structure, which might contain the required improving moves. The term “VNS” is referred to all local search based approaches that are centered on the principle of systematically exploring more than one type of neighborhood structure during the search. VNS is based on two important facts: (1) a local optimum with respect to one type of neighborhood is not necessarily so with respect to another type, and (2) a global optimum is a local optimum with respect to all types of neighborhoods [11].

Generally, VNS iterates over some neighborhood structures until some stopping criterion is met. Our proposed VNS algorithm incorporate two different local search types, one local search to improve the course-lecturer assignment, and another one to explore the lecturer-day invitation.

3.4.2. *Neighborhood search structures*

In the first local search, one course is randomly selected and reassigned to another qualified lecturer who has the highest preference. Notice that course reassignment affects two lecturers, one with course reduction and the other with course addition. Thus, the number of days that each of these two lecturers is invited might change. Regarding this new assignment, lecturer-day invitations of both lecturers are updated. The local search repeats for all courses at random without repetition. Therefore, n new solutions are generated by reassigning each course. The best solution among these new solutions is selected. The current solution is replaced with the best solution if it is better than the current solution. In this case, the local search restarts; otherwise, the search proceeds with the second local search. Figure 2 shows the outline of the proposed VNS.

```
Procedure: The_local_search_type 1
$\theta$ = Take the current solution
For $i$ = 1 to $n$ do
    Reassign course $i$ at random without repetition
Endfor
$\theta$ = The best solution of reassigning courses
```

Fig 2. General outline of local search type 1.

In the second local search, one lecturer is randomly selected, and then, the day of invitation changes from the day with the most course load to the day with the least course load. The second local search is applied on all lecturers at random without repetition. Therefore, it results in m new solutions generated by rescheduling each lecturer. The best solution among these new solutions is selected. The current solution is replaced with the best solution if it is better than the current solution. If this is the case, the second local search restarts. Otherwise, the algorithm continues with the first local search. Figure 3 shows the outline of the local search type 2. The algorithm repeats until the stopping criterion is met.

```
Procedure: The_local_search_type 2
$\theta$ = Take the current solution
For $i$ = 1 to $m$ do
    Reschedule lecturer $i$ at random without repetition
Endfor
$\theta$ = The best solution of rescheduling courses
```

Fig 3. General outline of local search type 2.

4. Numerical experiments

This section evaluates the performance of the model and the proposed algorithms. To evaluate their performance, we generate two sets of experimental instances. The first set includes small-sized instances and is used to analyze the model's capability of solving the problem and general performance of the algorithms. The model is implemented in CPLEX and run on a PC with 2.0 GHz Intel Core 2 Duo and 2 GB of RAM memory. It is allowed a maximum of 600 seconds of computational time. The algorithms are also implemented in C++ and ran on the same PC. The stopping criterion is set at a limit CPU time fixed to nm seconds. This stopping criterion allows for more time as the number of courses or lecturers increases.

To gauge the performance, we use the relative percentage deviation (RPD) as the performance measure [29]. RPD is calculated as follows.

$$\text{RPD} = \frac{\text{SOL} - \text{UB}}{\text{UB}} \times 100$$

where SOL and UB is the solution of the algorithm and the upper bound of a given instance which is equal to the best objective found for the given instance.

4.1. Parameter setting

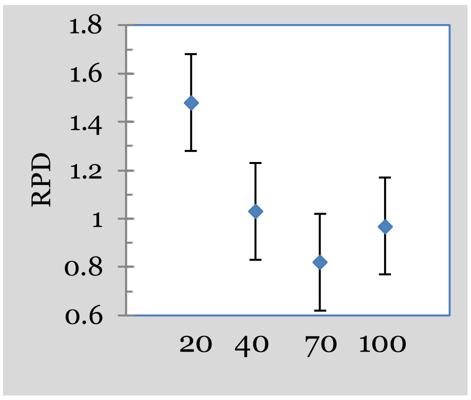
The parameter of ICA is the population size and that of SA is cooling rate. VNS is also has no parameter. The considered population sizes are {20, 40, 70, 100}. The considered levels for cooling rate are {0.95, 0.90, 0.85, 0.8}. We generate 20 different instances. Then we solve them by the obtained algorithms. Figure 4 shows the results. As it can be seen, for ICA the best population size is 70 while this value for GA is 40. The best cooling rate is also 0.9.

4.2. The experiment on small-sized instances

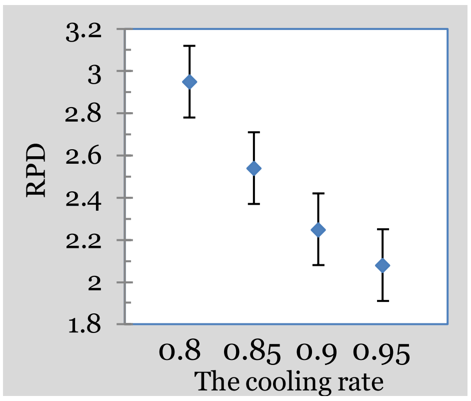
In this subsection, the model and algorithms solve the small-sized instances. The model is capable of solving the instances up to m = 60 and n = 15 in less than 60 seconds where the instances with m = 80 are solved within 500 seconds. The model could not solve instances with m = 100 in less than 600 seconds.

Table 1 shows the average RPD obtained by the tested algorithms in each group size. As it could be seen, ICA generally outperforms the other algorithms with average RPD of 0.96%.

The worst performing algorithm is SA with average RPD of 2.83%. Considering the performance versus different problem sizes, the proposed ICA performs well in all sizes.



a) ICA



b) SA

Fig 4. The average RPD and LSD intervals for the tested algorithms

Table 1. The average RPD of the algorithms on small-sized instances

Instance			ICA	SA	VNS
m	*n*	*e*			
20	5	4	0.44	2.01	1.17
	7	4	0.68	1.50	1.57
40	10	5	1.48	1.13	1.32
	15	5	1.13	1.86	2.90
60	10	6	1.63	2.06	1.16
	15	6	0.41	3.70	2.12
80	15	7	1.14	5.65	3.42
	20	7	0.78	4.74	1.59
Average			0.96	2.83	1.91

4.3. The experiment on large-sized instances

This section the proposed algorithms are evaluated and compared on the set of 60 large-sized instances mentioned earlier. Table 2 shows the results obtained by the algorithms, averaged by each combination n and m. Figure 5 shows the average RPD and least significant difference (LSD) intervals for the three tested algorithms. The best performing algorithm is ICA with the average RPD of 1.56%. VNS obtains the second rank with the average RPD of 2.11% while the worst performing algorithm is SA with average RPD of 2.79%.

Table 2. The average RPDs obtained by the algorithms

m	*n*	Algorithms		
		ICA	SA	VNS
100	20	1.21	2.67	1.66
	30	1.40	2.17	1.56
200	30	1.23	3.40	2.68
	50	1.54	2.74	1.71
300	50	1.39	2.40	2.41
	70	1.67	3.35	2.64
Average		1.41	2.79	2.11

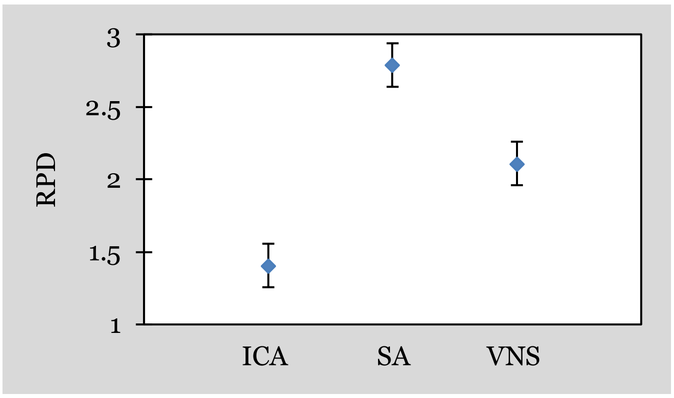


Fig 5. Means plot and LSD intervals for the different algorithms

5. Conclusion

This paper studied the university course timetabling problem. The objective was to schedule courses to maximize the total preference of lecturer-course, lecturer-day and course-day. The mathematical model of the problem was built. This model was an integer linear program and capable of solving problems up to 6 courses and 15 lecturers. Then, three advanced metaheuristics were designed to solve the large-sized problems. The algorithms were based on imperialist competitive algorithm, simulated annealing and variable neighborhood search. They employed novel procedures such as encoding scheme, move operator and crossing operators. The algorithms were first tuned and then evaluated by comparing with the optimal solutions obtained by the model. The algorithms were further evaluated on a set of larger problems. The results show that the proposed imperialist competitive algorithm outperforms the other algorithms.

Acknowledgement

The author thanks the research deputy of Kharazmi University for supporting this study as a research project.

References

[1] Aladag, C.H., Hocaoglu, G., and Basaran M.A. (2009). “The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem”, Expert Systems with Applications, 36, pp. 12349–12356.

[2] Al-Yakoob, S.M., and Sherali, H.D. (2007). “A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations”, European Journal of Operational Research, 180, pp. 1028–1044.

[3] Bardadym, V.A. (1996). “Computer-aided school and university timetabling: The new wave”. In E. Burke and P. Ross (Eds.), Practice and theory of automated timetabling. Lecture notes in computer science, 1153, pp. 22–45. Berlin: Springer.

[4] Boland, N., Hughes, B.D., Merlot, L.T.G., and Stuckey P.J. (2008). “New integer linear programming approaches for course timetabling”, Computers and Operations Research, 35, pp. 2209–2233.

[5] Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S., and Qu R. (2010). “Hybrid variable neighbourhood approaches to university exam timetabling”, European Journal of Operational Research, 206, pp. 46–53.

[6] Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). “A graph-based hyper-heuristic for educational timetabling problems”, European Journal of Operational Research, 176, pp. 177–192.

[7] Causmaecker, P.D., Demeester P., and Vanden Berghe, G. (2009). “A decomposed metaheuristic approach for a real-world university timetabling problem”, European Journal of Operational Research, 195, pp. 307–318.

[8] Daskalaki, S., and Birbas, T. (2005). “Efficient solutions for a university timetabling problem through integer programming”, European Journal of Operational Research, 160, pp. 106–120.

[9] Dimopoulou, M., and Miliotis, P. (2004). “An automated university course timetabling system developed in a distributed environment: A case study”, European Journal of Operational Research, 153, pp. 136–147.

[10] Flesza, K., and Hindi, K.S. (2004). “Solving the resource-constrained project scheduling problem by a variable neighborhood search”, European Journal of Operational Research, 155, pp. 402–413.

[11] Hansen, P., and Mladenovic, N. (2001). “Variable neighborhood search: principles and applications”, European Journal of Operational Research, 130, pp. 449–467.

[12] Liao, C.J., and Cheng, C.C. (2007). “A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date”, Computers and Industrial Engineering, 52, pp. 404–413.

[13] Lü, Z., and Hao, J.K. (2010). “Adaptive Tabu Search for course timetabling”, European Journal of Operational Research, 200, pp. 235–244.

[14] Teoh, C.K., Wibowo, A., and Ngadiman, M.S. (2013). Review of state of the art for metaheuristic techniques in Academic Scheduling Problems, Artificial Intelligence Review, 10.1007/s10462-013-9399-6.

[15] Mladenovic, N., and Hansen, P. (1997). "Variable neighborhood search", Computers and Operations Research, 24, pp. 1097–1100.

[16] MirHassani, S.A. (2006). "A computational approach to enhancing course timetabling with integer programming", Applied Mathematics and Computation, 175, pp. 814–822.

[17] MirHassani, S.A., and Habibi, F., (2013). Solution approaches to the course timetabling problem, Artificial Intelligence Review, 39, pp. 133-149.

[18] Shiau, D.F. (2011). "A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences", Expert Systems with Applications, 38, pp. 235–248.

[19] Turabieh, H., Abdullah, S. (2011). "An integrated hybrid approach to the examination time tabling problem", Omega, 39, pp. 598–607.

[20] Wang, Y.Z. (2002). "An application of genetic algorithm methods for teacher assignment problems", Expert Systems with Applications, 22, pp. 295–302.

[21] Wang, Y.Z. (2003). "Using genetic algorithm methods to solve course scheduling problems", Expert Systems with Applications, 25, pp. 39-50.

[22] Zhang, D., Liu, Y., M'Hallah, R., and Leung, S.C.H. (2010). "A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems", European Journal of Operational Research, 203, pp. 550–558.

[23] Atashpaz-Gargari, E., and Lucas, C., (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. IEEE Congress Evolutionary Computers, Singapore, pp. 4661–4667.

[24] Atashpaz-Gargari, E., Hashemzadeh, F., Rajabioun, R., and Lucas, C., (2008). Colonial competitive algorithm, a novel approach for PID controller design in MIMO distillation column process. International Journal of Intelligent Computation and Cyberntic, 1, pp. 337–355.

[25] Bagher, M., Zandieh, M., and Farsijani, H., (2010). Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. International Journal of Advanced Manufacturing Technology, 54, pp. 271–285.

[26] Banisadr, A.H., Zandieh, M., and Mahdavi, I., (2013). A hybrid imperialist competitive algorithm for single-machine scheduling problem with linear earliness and quadratic tardiness penalties. International Journal of Advanced Manufacturing Technology, pp. 981-989.

[27] Zhou, W., Yan, J., Li, Y., Xia, C., and Zheng, J., (2013). Imperialist competitive algorithm for assembly sequence planning. International Journal of Advanced Manufacturing Technology, 67, pp. 2207-2216.

[28] Kolon, M., (1999). Some new results on simulated annealing applied to the job shop scheduling problem. European Journal of Operational Research, 113, pp. 123–136.

[29] Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A., and Roshanaei, V., (2009). "An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness", Expert Systems with Applications, 36, pp. 9625–9633.

[30] Kahar, M.N.M., and Kendall, G. (2010). "The examination timetabling problem at Universiti Malaysia Pahang: Comparison of a constructive heuristic with an existing software solution", European Journal of Operational Research, 207, pp. 557–565.