ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20130918133247

ALGORITHMS FOR UNIVERSITY COURSE SCHEDULING PROBLEMS

Mehdi Yazdani, Bahman Naderi, Esmaeil Zeinali

Orginal scientific paper
This paper deals with the problem of course scheduling where we have a set of courses, lecturers and classrooms. Courses are assigned and scheduled in such a way that the total preference is maximized. We develop the mathematical model of the problem in form of a linear integer program. The small sized problem can be solved to optimality using commercial software. We then develop three different metaheuristics based on artificial immune, genetic and simulated annealing algorithms. These three solution methods are equipped with novel procedures such as move and crossing operators. The parameters of the proposed metaheuristics are first tuned, and then they are evaluated with optimal solutions found by the model. They are, furthermore, evaluated by comparing their performance. The experiments demonstrate that the artificial immune algorithm performs better than the other algorithms.

Keywords: *artificial immune algorithm; genetic algorithm; mathematical model; simulated annealing; university course scheduling*

Algoritmi za probleme planiranja fakultetskih predavanja

Izvorni znanstveni članak
Rad se bavi problemom planiranja predavanja gdje postoji niz kolegija, predavača i učionica. Kolegiji se dodjeljuju i planiraju tako da se maksimalno zadovolje preferencije. Razvijamo matematički model problema u obliku linearnog programa cijelih brojeva. Manji se problem može optimalno riješiti primjenom komercijalnog softvera. Zatim razvijamo tri različite metaheuristike na temelju umjetnih imunih, genetičkih i algoritama simuliranog kaljenje. Te tri metode rješenja opremljene su novim postupcima kao što su operatori kretanja i križanja. Parametri predložene metaheuristike najprije se usklađuju, a zatim procjenjuju optimalnim rješenjima koje je model pronašao. Nadalje se procjenjuju usporedbom njihovih performansi. Eksperimenti pokazuju da je umjetni imuni algoritam uspješniji od drugih algoritama.

Ključne riječi: *genetički algoritam; matematički model; planiranje fakultetskih predavanja; simulirano kaljenje; umjetni imuni algoritam*

1 Introduction

Scheduling is the problem of performing a set of given tasks by a set of limited resources. Among important scheduling problems, university scheduling (US) problem attracts great attention from both artificial intelligence and operations research [2]. In classical US problems, a set of events (such as courses and exams) is assigned to a number of *'classroom-time'* slots so as to satisfy a set of constraints [1].

The US problems can be either examination or course scheduling. In the examination scheduling, one objective is to spread several exams as evenly as possible, while in course scheduling students demand a schedule as compact as possible [3]. This paper considers a class of university scheduling problems, known as the university course scheduling (UCS) problem. It is always a complex job for academic offices at each semester to plan courses.

The UCS problem is NP-hard [4] and includes two decisions: instructor assignment and class scheduling. In the instructor assignment, we determine which instructor would present what courses. In the class scheduling, we specify in which class each course would be presented. The decisions have to be made in such a way to fulfil a set of constraints. It takes into consideration instructor's professional qualifications, preferences about courses and days, reasonable distribution of overtime among the instructors and availability of equipment and facilities. It, moreover, avoids any conflict in instructors' schedules and classrooms.

The constraints of UCS problem can be divided into two groups; hard and soft [5]. As long as all hard constraints are met and satisfied, a schedule is feasible. A typical example of hard constraints is that no instructor can teach at most one course at a time. On the other hand, soft constraints are those requirements that although essential, should be fulfilled to the extent possible. That is, they may be violated if necessary. As a result, soft constraints can be referred to as preferences and used to evaluate the quality of a solution. For instance, a typical soft constraint is to spread classes as evenly as possible. The main objective is to find a feasible schedule satisfying all hard constraints and maximizing the satisfaction of both instructors and classes based on their preferences. This is equal to minimizing the contravention of the soft constraint.

UCS problems are usually different from one university to another [2, 6]. It is very likely that each university has its own unique set of requirements to utilize its resources effectively, fulfil the requirements of its business, give a high level of satisfaction to its students and so forth. Consequently, a customized system for the course scheduling has to be developed to meet all these unique requirements.

The different aspects of the problem are first described. Mathematical formulation of the problem is then presented as an integer linear program. Using specified mathematical programming software, the model is solved. Recently, interest in metaheuristics for solving UCS problems is increasing. Such algorithms include tabu search by Lü and Hao [7], simulated annealing by Zhanget al. [8] and genetic algorithm by Wang [9]. We propose three metaheuristics based on artificial immune, genetic and simulated annealing algorithms in order to solve this problem in large instances. These algorithms utilize novel procedures such as move and crossing operators.

The remainder of this paper is organized as follows. In section 2, the problem is mathematically formulated. In section 3, solution algorithms are developed. In section 4, experiments are conducted. Finally, in section 5, the paper is concluded.

2 The problem formulation

The UCS problem consists of a set of l instructors, a set of c courses and a set of r classrooms. We have d days to schedule them and on each day, there exist some time periods. At each time period, one course can be presented at each classroom. Each lecturer has his unique preference to teach each course of his expertise. Each course can be presented only in a subset of classrooms and days. The objective is both maximizing the instructors' preference and minimizing the number of classrooms used.

UCS problems are usually modelled by integer linear programming formulation. The developed mathematical model is presented in the following section. The notations and parameters used in the model are as follows:

l - The number of instructors
c - The number of courses
r - The number of classrooms
d - The number of days
t - Index for working days $\{1, 2, \ldots, d\}$
i - Index for instructors where $\{1, 2, \ldots, l\}$
j - Index for courses where $\{1, 2, \ldots, c\}$
k - Index for classrooms $\{1, 2, \ldots, e\}$
h - Index for time period $\{1, 2, 3\}$
$u^1_{i,j}$ - The utility of instructor i for teaching course j
$u^2_{i,t}$ - The utility of instructor i for teaching course t
$u^3_{j,t}$ - The utility of instructor j for teaching course t
$g_{i,t}$ - 1 if instructor i can be invited on day t, and 0 otherwise
$s_{i,j}$ - 1 if instructor i can teach course j, and 0 otherwise
$v_{j,k}$ - 1 if course j can be presented in classroom k, and 0 otherwise

Decision variables:
$X_{i,j,t,j,h}$ - 1 if instructor i teaches course j on day t in classroom l in time period h, and 0 otherwise
$Y_{i,t}$ - Binary variable taking value 1 if instructor i is invited on day t, and 0 otherwise.

The model is as follows:

$$\text{Maximize } Z = \sum_{i=1}^{l}\sum_{j=1}^{c}\sum_{t=1}^{d}\sum_{k=1}^{e}\sum_{h=1}^{3} X_{i,j,t,k,h}\cdot u^1_{i,j} + \\ + \sum_{i=1}^{l}\sum_{t=1}^{d} Y_{i,t}\cdot u^2_{i,t} + \sum_{i=1}^{l}\sum_{j=1}^{c}\sum_{t=1}^{d}\sum_{k=1}^{e}\sum_{h=1}^{3} X_{i,j,t,k,h}\cdot u^3_{j,t} \quad (1)$$

Subject to:

$$\sum_{i=1}^{l}\sum_{t=1}^{d}\sum_{k=1}^{e}\sum_{h=1}^{3} X_{i,j,t,k,h} = 1 \quad \forall_j \quad (2)$$

$$\sum_{j=1}^{c}\sum_{k=1}^{e} X_{i,j,t,k,h} \le Y_{i,t} \quad \forall_{i,t,h} \quad (3)$$

$$Y_{i,t} \le g_{i,t} \quad \forall_{i,t} \quad (4)$$

$$\sum_{i=1}^{l}\sum_{j=1}^{c} X_{i,j,t,k,h} \le 1 \quad \forall_{t,k,h} \quad (5)$$

$$\sum_{t=1}^{d}\sum_{k=1}^{e}\sum_{h=1}^{3} X_{i,j,t,k,h} \le s_{i,j} \quad \forall_{i,j} \quad (6)$$

$$\sum_{i=1}^{l}\sum_{t=1}^{d}\sum_{h=1}^{3} X_{i,j,t,k,h} \le v_{j,k} \quad \forall_{j,k} \quad (7)$$

$$Y_{i,t} \in \{0,1\} \quad \forall_{i,t} \quad (8)$$

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

Eq. (1) calculates the total utility. Constraint set (2) assures that each course is taught. Constraint set (3) specifies days an instructor has courses to teach. Moreover, it ensures that each instructor presents at most one course at any time slot. Constraint set (4) ensures instructors are invited on days they prefer. Constraint set (5) prevents from cross assignment, that is, at most one course can be presented in each classroom at a time. Constraint set (6) is to make sure that each instructor is assigned to courses of expertise. Constraint set (7) specifies that each course is assigned to eligible classes. Constraint sets (8) and (9) define the decision binary variables.

3 Solution algorithms

Since the problem is NP-hard, the most effective algorithms to solve it are metaheuristics. Examples of these algorithms include graph colouring heuristics [2], Tabu Search [7], simulated annealing [8], evolutionary algorithms [10], case-based reasoning [11], two-stage heuristic algorithms [12], tabu search [13], ant colony [14] and so on. Interested readers are referred to [6] for a comprehensive survey of the automated approaches for university timetabling presented in recent years. This paper proposes three different metaheuristics, artificial immune, genetic and simulated annealing algorithms. 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 encoding scheme to represent solutions of the problem. 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.

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.

The encoding scheme is described by applying to an illustrative example. Consider a problem with eight courses, four lecturers, two working days and two classrooms with four time-slots on each day. An encoded solution is represented by Fig. 1. In this figure, Part *a* shows the first matrix and Part *b* the second matrix. In this solution, Courses 1 and 6 are assigned to Lecturer 1, Courses 3 and 7 to Lecturer 2 and Courses 2, 4, 5 and 8 to Lecturer 4. No course is assigned to Lecturer 3. Lecturers 1 and 4 are invited on the first day and Lecturers 2 and 4 on the second day. Therefore, Lecturer 4 is invited on both days.

Course \ *Lecturer*	1	2	3	4
1	1	-	-	0
2	-	0	-	1
3	-	1	0	-
4	-	-	-	1
5	0	-	0	1
6	1	-	0	-
7	0	1	-	-
8	-	-	0	1

a) Instructor-course assignment

Day \ *Lecturer*	1	2	3	4
1	1	0	-	1
2	-	1	0	1

b) Instructor-day assignment

Figure 1 An example of encoded solution.

3.2 Genetic algorithm

Genetic algorithm (GA) is designed to deal with some problems of industry that were difficult to solve with conventional methods. Todays, GA is well-known population based evolutionary algorithms tackling both discrete and continuous optimization problems. The idea behind GA comes from Darwin's "survival of the fittest" concept, meaning that good parents produce better offsprings.

3.2.1 General structure

GA searches a solution space with a population of chromosomes each of which represents an encoded solution. A fitness value is assigned to each chromosome according to its performance. The better the chromosome is, the higher this value becomes. The population evolves by a set of operators until some stopping criterion is visited. A typical iteration of a GA, generation, proceeds as follows. The best chromosomes of current population are directly copied to the next generation (reproduction). A selection mechanism chooses chromosomes of the current population so as to give higher chance to chromosomes with the higher fitness value. The selected chromosomes are crossed to generate new offspring. After crossing process, each offspring might mutate by another mechanism called mutation. Afterwards, the new population is evaluated again and the whole process is repeated. The outline of the proposed GA is shown in Fig. 2.

The procedure: the proposed GA
Initialization mechanism
While *the stopping criterion is not met* **do**
    Selection mechanism
    Crossover mechanism
    Mutation mechanism
End while

Figure 2 The outline of the proposed GA

3.2.1 The initialization and selection mechanisms

GA starts with a number of chromosomes each of which represents a possible solution. The number of chromosomes is the population size indicated by *pop*. The initial chromosomes are randomly generated from the feasible solutions.

After initializing the algorithms, each chromosome is evaluated and its fitness (i.e., objective function) is determined. The chance of chromosome k to be selected for crossover mechanism is as follows.

$$p_k = \frac{fit(k)}{\sum_{h=1}^{pop} fit(h)}, \tag{10}$$

where $fit(k)$ is the fitness of chromosome k.

3.2.2 Crossover and mutation mechanisms

New solutions are produced by crossing two other parents by an operator called crossover. Note that the crossover operators must avoid generating infeasible solutions. The purpose is to generate better offsprings. To take a solution towards better areas, we define a new solution that inherits from both parents. In fact, we combine two parents to form a new solution. This is done through an operator with the following steps.

A random number uniformly distributed between 0 and 1 is generated. If this number is less than 0.6, the two columns of that instructor from the parent 1 (i.e., one from the instructor-course assignment and one in the instructor-day assignment) are copied into the new solution. Other columns of the new solution are filled from parent 2. Note that the new solution probably needs modification to be feasible solution. In the instructor-course assignment, if a course is assign to two instructors, one of the instructors is randomly selected and the other one is crossed out. Moreover, if a course is not assigned to any instructor it is randomly assigned to a lecturer.

Regarding this new instructor-course assignment, instructor-day assignment is updated.

After crossover, each solution is changed by the mutation operator. The main purpose of applying mutation is to avoid convergence to a local optimum and diversify the population. We use the following mutation operator. One instructor 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 instructors at random without repetition. Therefore, it results in m new solutions generated by rescheduling each instructor. The best solution among these new solutions is selected.

3.3 Simulated annealing

The simulated annealing (SA) is a local search based metaheuristic simulating the annealing process [15]. SA includes a mechanism, called acceptance criterion, which enables it to escape from local optima. The acceptance criterion determines whether to accept or reject the new generated solution. In this mechanism, even inferior solutions may be accepted.

3.3.1 The general structure and acceptance criterion

Simulated annealing processes an initial solution and makesa series of moves until a stopping criterion is met. The idea is to generate a new solution s by an operator from the current solution x. Another random rule is used to accept or reject this new solution. The acceptance rule is controlled by a parameter t, called the temperature. The variation between objective values of two candidate solutions is calculated $\Delta = fit(s) - fit(x)$. If $\Delta \leq 0$, solution s is accepted. Otherwise, solution s is accepted with probability equal to $exp(\Delta/t_i)$. At each temperature t_i, the algorithm goes on by a fixed number of moves. While SA proceeds, the temperature is gradually decreased. The exponential cooling schedule is used, $t_i = \alpha \cdot t_{i-1}$ (where $\alpha \in (0, 1)$ is temperature decrease rate). The initial temperature is set to be 50 and $\alpha = 0.97$. Fig. 3 shows the general outline of the proposed SA.

```
The procedure: the proposed SA
Initialization mechanism
While the stopping criterion is not met do
    Move mechanism
    Acceptance mechanism
    Cooling schedule mechanism
End while
```

Figure 3 The outline of the proposed SA

3.3.2 The move operator

In this paper, we generate a new solution from the current solution by the following procedure. We select one course randomly and assign it to another qualified instructor. Note that course reassignment impacts two instructors, one with course reduction and the other with course addition. Hence, the number of days that these two instructors are invited may need to be updated. Considering this new assignment, instructor-day invitations of both instructors are updated.

3.4 Artificial immune algorithm

Artificial immune algorithm (AIA) is a population-based metaheuristic meaning that it conducts parallel search by several encoded solutions. AIA is known as an effective algorithm [16, 17]. The basic idea comes from the simulation of the physiological immune system of natural living organisms. The immune system is to defend the body from foreign virus (antigens). This is done by a set of antibodies. In case an antigen is detected, antibodies identifying this antigen proliferate by cloning. That is, whenever an antigen penetrates into the body, they first recognize those antibodies are more eligible to fight with the invasion of antigen. Then the system produces more variations of those antibodies in the next generation.

The quality of each antibody is shown by an affinity value. The antibody of larger affinity can better fight with antigens. The antigen refers to the problem under consideration. Each antibody is a feasible solution and the affinity is the objective function value obtained by an antibody.

3.4.1 The general structure

The procedure of AIA can be described as follows. AIA searches a problem space with a population of antibodies. An affinity value is assigned to each antibody according to its performance. The more desirable the antibody is, the higher this value becomes. Using immune operators such as cloning selection and affinity maturation, a new population is generated from the current population. The cloning selection refers to the proliferation of antibodies better at eliminating the antigens.

The mechanism of affinity maturation consists of hypermutation and receptor editing. Hypermutation corresponds to generating new solutions similar to their creators but not exactly the same. The inferior antibodies should be hypermutated at higher rate while the superior antibodies undergo lower rate. The receptor editing is the mechanism of determining hypermutation rate. AIA proceeds as follows. A user-defined affinity function calculates the number of the clones proliferated from each antibody. All the proliferated clones are collected in a mutating pool. The clones of mutating pool are hypermutated to generate new antibodies. Then, the new population is evaluated and the whole process repeats. The initial chromosomes are randomly generated from the feasible solutions.

3.4.2 Cloning selection procedure

Affinity measure gauges the quality of each antibody to eliminate antigens. The higher affinity value means the better antibody at fighting with antigens. We can assume our optimization problem as the antigen and an encoded solution as an antibody. Better antibodies are those that obtain better objective (i.e., they can fight against the optimization problem better). We set each antibody's affinity equal to objective function value.

The probability of cloning for each antibody to move into the mutating pool is a proportion of its affinity value. Thus, the antibodies with higher affinity are more likely

to have more clones. The mutating pool has a fixed size of clones. One of them is fulfilled by the best antibody of the current population. To fulfil the rest, we use a selection mechanism. The chance of an antibody is calculated using Eq. (10).

4.3 Affinity maturing procedure

The main function of affinity maturing procedure is to make a random change in all of the clones existing in the pool. Each clone undergoes different rate of change according to the affinity value. The inferior clones undergo high rate of change while better clones suffer a slight change. These changes are made by an operator called hypermutation. As a low rate hypermutation, we utilize the following operator.

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.

As high rate hypermutation, we use the mutation operator defined in GA. A criterion should be defined to determine the condition under which one of the operators is used. We use the low rate hypermutation if we have

$$\frac{affinity(x) - affinity(best\ antibody)}{affinity(best\ antibody)} < 0.1$$

Otherwise, the high rate hypermutation is applied.

Contrary to previous work [16] in which the offsprings are accepted only if they have lower makespan than their creators, we use simple simulated annealing-like acceptance criterion. Besides the acceptance of better offspring, inferior offspring might be accepted with probability of

$$\exp\left(\frac{affinity(x) - affinity(creator)}{20}\right) < 0.1$$

With this, we let the algorithm easily avoid getting stuck into local minima. We need to state that we apply the elite strategy meaning that the best clone is directly copied into the next generation.

4 Numerical experiment

In this section, we evaluate the performance of the model and the proposed algorithms. To this end, two sets of experimental instances are generated. The first set consists of small-sized instances and is used to assess the capability of the model to solve the problem and also general performance of the algorithms. We consider the following small sizes for (m, n):

(20, 5), (20, 7), (40, 10), (40, 15), (60, 10), (60, 15), (80, 15), (80, 20), (100, 20) and (100, 25)

The second set includes large-sized instances by which the performance of the proposed algorithms is compared. We consider the following 6 combinations for (m, n):

(100, 20), (100, 30), (200, 30), (200, 50), (300, 50) and (300, 70)

It is also assumed that there exist 0,2m classrooms, a week has 5 working days and 3 time-slots on each day. We generate 10 instances for each problem size by randomly generating the other parameters of the problem. Care must be taken when generating data. For example, to teach each course, there must be at least one lecturer.

The model and the algorithms are implemented in CPLEX and C++, respectively. They are run on a PC with 2.0 GHz Intel Core 2 Duo and 2 GB of RAM memory. The model is allowed a maximum of 600 seconds of computational time. The stopping criterion for the algorithms 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. Relative percentage deviation (RPD) is used as the performance measure [18]. RPD is calculated as follows.

$$RPD = \frac{ALG - Max}{Max} \times 100$$

where ALG and Max 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.

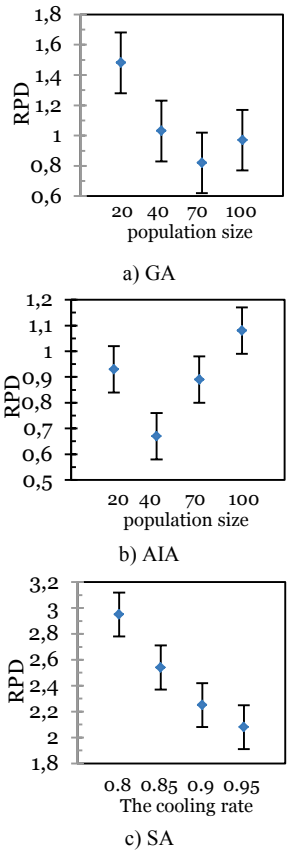

Figure 4 The average RPD and LSD intervals for the tested algorithms

4.1 Parameter setting

The parameter of GA and AIA is the population size and that of SA is cooling rate. The considered population sizes are {20, 40, 70, 100}. The considered levels for cooling rate are {0.95, 0.90, 0.85, 0.8}. 20 different instances are generated. Then, we solve them by the obtained algorithms. Fig. 4 shows the results. As it can be seen, for GA the best population size is 70 while this value for AIA is 40. The best cooling rate is also 0.95.

4.2 The experiment on small-sized instances

In this subsection, we solve the small-sized instances by the model and algorithms. Tab. 1 presents the results. The model is able to solve the instances up to $m = 60$ and $n = 15$ in less than only 60 seconds. The instances with $m = 80$ are required less than 500 seconds. The model cannot solve instances with $m = 100$ in less than 600 seconds.

Tab. 2 shows the average RPD obtained by the tested algorithms in each group size. As can be seen, AIA outperforms the other algorithms with average RPD of 0,87 %. GA is obtained the second rank with 1,47 %. The worst performing algorithm is SA with average RPD of 1,91 %. Analysing the performance versus different problem sizes, the proposed AIA performs well in all sizes.

Table 1 The model's results on small-sized instances

Instance			Optimality gap	Computational time (sec)
m	*n*	*e*		
20	5	4	0	0,6
20	7	4	0	1,4
40	10	5	0	42
40	15	5	0	5
60	10	6	0	92
60	15	6	0	55
80	15	7	0	414
80	20	7	0	79
100	20	8	4 %	600
100	25	8	3 %	600

Table 2 The average RPD of the algorithms on small-sized instances

Instance			GA	SA	AIA
c	*l*	*e*			
20	5	4	0,76	1,17	0,44
	7	4	1,18	1,57	0,68
40	10	5	1,03	1,32	0,71
	15	5	1,57	2,9	0,9
60	10	6	2,37	1,16	1,14
	15	6	1,2	2,12	0,67
80	15	7	2,56	3,42	1,64
	20	7	1,12	1,59	0,78
Average			1,47	1,91	0,87

4.3 The experiment on large-sized instances

In this section, the proposed algorithms (GA, SA and AIA) are compared on the set of 60 large-sized instances mentioned in section 4. Tab. 3 shows the results obtained by the algorithms. Fig. 5 shows the average RPD and least significant difference (LSD) intervals for the three tested algorithms. The best performing algorithm is AIA with the average RPD of 0,63 %. After AIA, GAyields the average RPD of 1,66 % while the worst performing algorithm is SA with average RPD of 2,89 %.

We assess the effect of problem size (the number of courses) on the performance of the tested algorithms. Fig. 6 shows the performance of algorithms versus the number of courses. Regarding the number of courses, AIA keeps its robust performance in different sizes while the performance of SA becomes worse in larger sizes.

Table 3 The average RPDs obtained by the algorithms

c	*l*	Algorithms		
		GA	SA	AIA
100	20	1,32	2,18	0,49
100	30	1,2	2,24	0,68
200	30	1,45	2,59	0,51
200	50	2,12	2,71	1,01
300	50	2,07	3,76	0,41
300	70	1,83	3,91	0,66
Average		1,66	2,89	0,63

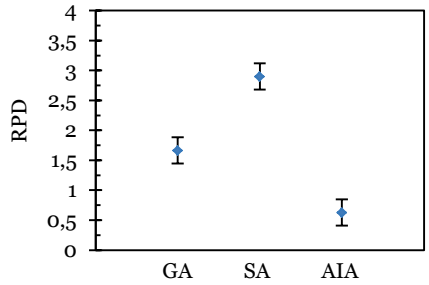

Figure 5 Means plot and LSD intervals for the different algorithms

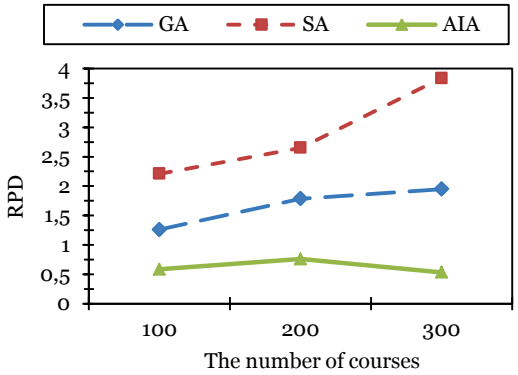

Figure 6 The average RPD of the algorithms versus the number of courses

5 Conclusion

As an important problem, we considered the problem of university course scheduling. The objective function was defined to schedule courses to maximize the total utility of instructor, courses and days. We first modelled the problem mathematically in form of an integer linear program. This model is able to solve problems up to 6 courses and 15 instructors. Moreover, we developed three advanced metaheuristics to solve the large-sized problems. The algorithms were based on artificial immune, genetic and simulated annealing algorithms. They utilized novel procedures such as move and crossing operators. After tuning the algorithms, we evaluated them by comparing with the optimal solutions obtained by the model. Then, the algorithms were compared on a set of larger problems. The results show that the proposed artificial immune algorithm outperforms the other algorithms.

6 References

[1] Turabieh, H.; Abdullah S. An integrated hybrid approach to the examination time tabling problem. // Omega, 39, (2011), pp. 598–607. https://doi.org/10.1016/j.omega.2010.12.005

[2] Burke, E. K.; McCollum, B.; Meisels, A.; Petrovic, S.; Qu R. A graph-based hyper-heuristic for educational timetabling problems. // European Journal of Operational Research, 176, (2007), pp. 177–192. https://doi.org/10.1016/j.ejor.2005.08.012

[3] Causmaecker, P. D.; Demeester, P.; Vanden. B.G.A decomposed metaheuristic approach for a real-world university timetabling problem. // European Journal of Operational Research, 195, (2009), pp. 307–318. https://doi.org/10.1016/j.ejor.2008.01.043

[4] Bardadym, V. A. 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, (1996), pp. 22–45. Berlin: Springer. https://doi.org/10.1007/3-540-61794-9_50

[5] Shiau, D. F. A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences. // Expert Systems with Applications, 38, (2011), pp. 235–248. https://doi.org/10.1016/j.eswa.2010.06.051

[6] Al-Yakoob, S. M.; Sherali, H. D. 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, (2007), pp. 1028–1044. https://doi.org/10.1016/j.ejor.2006.04.035

[7] Lü, Z.; Hao, J. K. Adaptive Tabu Search for course timetabling. // European Journal of Operational Research, 200, (2010), pp. 235–244. https://doi.org/10.1016/j.ejor.2008.12.007

[8] Zhang, D.; Liu, Y.; M'Hallah, R.; Leung, S. C. H. A simulated annealing with a new neighbourhood structure based algorithm for high school timetabling problems. // European Journal of Operational Research, 203, (2010), pp. 550–558. https://doi.org/10.1016/j.ejor.2009.09.014

[9] Wang, Y. Z. An application of genetic algorithm methods for teacher assignment problems. // Expert Systems with Applications, 22, (2002), pp. 295–302. https://doi.org/10.1016/S0957-4174(02)00017-9

[10] MirHassani, S. A. A computational approach to enhancing course timetabling with integer programming. // Applied Mathematics and Computation, 175, (2006), pp. 814–822. https://doi.org/10.1016/j.amc.2005.07.039

[11] Burke, E. K.; Eckersley, A. J.; McCollum, B.; Petrovic, S.; Qu, R. Hybrid variable neighbourhood approaches to university exam timetabling, // European Journal of Operational Research, 206, (2010), pp. 46–53. https://doi.org/10.1016/j.ejor.2010.01.044

[12] Daskalaki, S.; Birbas, T. Efficient solutions for a university timetabling problem through integer programming. // European Journal of Operational Research, 160, (2005), pp. 106–120. https://doi.org/10.1016/j.ejor.2003.06.023

[13] Abdullah, S.; Turabieh, H. On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems. //information Sciences, 191, (2012), pp. 146-168.

[14] Thepphakorn, T.; Pongcharoen, P.; Hicks, C. An ant colony based timetabling tool. // International Journal of Production Economics, 149, 3(2014), pp. 131-144. https://doi.org/10.1016/j.ijpe.2013.04.026

[15] Kolon, M. Some new results on simulated annealing applied to the job shop scheduling problem. // European Journal of Operational Research, 113, (1999), pp. 123–136. https://doi.org/10.1016/S0377-2217(97)00420-7

[16] Engin, O.; Döyen, A. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. // Future Generation Computer Systems, 20, (2004), pp. 1083–1095. https://doi.org/10.1016/j.future.2004.03.014

[17] Naderi, B.; Khalili, M.; Tavakkoli-Moghaddam, R. A hybrid artificial immune algorithm for a realistic variant of job shops to minimize the total completion time. // Computers and Industrial Engineering, 56, 4(2009), pp. 1494–1501. https://doi.org/10.1016/j.cie.2008.09.031

[18] Naderi, B.; Zandieh, M.; Khaleghi Ghoshe Balagh, A.; Roshanaei, V. 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, 6(2009), pp. 9625–9633. https://doi.org/10.1016/j.eswa.2008.09.063

Authors' address

Mehdi Yazdani *(corresponding author)*
Department of Industrial Engineering,
Qazvin Branch, Islamic Azad University,
Qazvin, Iran
mehdi_yazdani2007@yahoo.com

Bahman Naderi
Department of Industrial Engineering,
Kharazmi University,
Tehran, Iran
bahman.naderi@khu.ac.ir

Esmaeil Zeinali
Department of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University,
Qazvin, Iran