Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER Applied Mathematics and Computation 163 (2005) 705–733

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

Hybrid heuristics for Examination Timetabling problem

Zahra Naji Azimi

Department of Mathematical Sciences, Ferdowsi University of Mashhad, Mashhad, Iran

Abstract

Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), and Ant Colony System (ACS) are four of the main algorithms for solving challenging problems of intelligent systems. In this paper, we apply these four techniques and three novel hybrid combinations of them to a classical Examination Timetabling problem (ETP), an NP complete problem. The novel hybrid algorithms consist of a Sequential TS–ACS, a Hybrid ACS/TS, and a Sequential ACS–TS algorithms. These various hybrid combinations are then tested on 10 different scenarios of the classical ETP. Statistical comparative analysis conclude that all of the three proposed novel techniques are significantly better than each of their non-hybrid competitors, and furthermore the Sequential ACS–TS provides the superior solution of all.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Timetabling; Simulated Annealing; Tabu Search; Genetic Algorithm; Ant Colony System; Hybrid methods

1. Introduction

Proper scheduling of exams is a common problem for all universities and institutions of higher education. Quite often it is done by hand or with the limited help of a simple administration system and usually involves taking the previous year's timetable and modifying it so it will work for the new year. But changing the number of students, variety of the courses which are offered and student's freedom in selecting them requires significant alteration of previous

E-mail address: najiazimi@yahoo.com (Z. Naji Azimi).

0096-3003/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2003.10.061

year's timetable. The Examination Timetabling problem regards the scheduling for the exams of a set of university courses, avoiding overlap of exams of courses having common students and spreading the exams for the students as much as possible.

The process of finding a period for each exam so that no two conflict has been shown to be equivalent to assigning colours to vertices in graph so that adjacent vertices always have different colours [1]. This in turn has been proved to lie in the set of NP complete problems [2] which means that carrying out an exhaustive search for the timetable is not possible in a reasonable time. There are many heuristic methods which have been offered for solving this problem based on graph colouring as well as many metaheuristic methods such as SA, TS, GA, and ACS [3–22].

In this paper we consider the above metaheuristics and introduce three hybrid heuristic methods for solving the ETP, and compare their results. In the following sections, we first begin with describing the Examination Timetabling problem. In Section 3, a common structure of the four metaheuristics are provided. The four basic metaheuristics and their structure as used in this paper is discussed in Section 4. In Section 5, the structure of the input data set is discussed, and the results of application of the pure metaheuristics is analysed in Section 6. The proposed hybrid methods are explained in Section 7. Section 8 includes the final analysis of the hybrid methods and comparison with pure methods.

2. Problem description

Given is a set of examinations, a set of (contiguous) time slots, a set of students, and a set of student enrollments to examinations. The problem is to assign examinations to time slots satisfying a set of constraints [17]. Many different constraint types have been proposed in the literature. In this work, we consider the version proposed by Carter et al. [4], which is based on the so-called *first-order* and *second-order* conflicts.

First-order conflicts arise when a student has to take two exams scheduled in the same time slot, while second order ones emerge when a student has to take two exams in time slots "close" to each other. Second-order conflicts are treated as *soft* constraints and first-order conflicts are modelled as *hard* constraints.

Assuming that consecutive time slots lie one unit apart, we define f assigning a proximity cost $w(i)$ whenever a student has to attend two exams scheduled within i time slots. The cost of each conflict is thus multiplied by the number of students involved in both examinations.

As in [17], the cost decreases logarithmically here from 16 to 1 for soft constraints as follows: $w(1) = 16$, $w(2) = 8$, $w(3) = 4$, $w(4) = 2$, $w(5) = 1$ and the cost for hard constraint is 1000. There are other constraints like room capacity that we do not consider for simplicity.

The cost function is then normalized based on the total number of students. This way we obtain a measure of the number of violations "per student", which allows us to compare results for instances of different size. So we have the below cost function,

$$F = (f_1 + f_2)/M$$

$$f_1 = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} C_{ij} \cdot w(i,j) \quad \text{where } w(i,j) = \begin{cases} 2^{5-|t_i - t_j|} & \text{if } 1 \leqslant |t_i - t_j| \leqslant 5 \\ 0 & \text{otherwise} \end{cases}$$

$$f_2 = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} C_{ij} \cdot w'(i,j) \quad \text{where } w'(i,j) = \begin{cases} 1000 & \text{if } t_i = t_j \\ 0 & \text{otherwise} \end{cases}$$

where N and M indicate the number of exams and students consecutively and $C(i,j)$ shows the number of common students between both exam i and exam j, also t_i is the period of exam i (for $i = 1, \dots, N$).

We attempt here an unbiased comparison of performance of basic versions of different metaheuristics on the Examination Timetabling problem using a common search landscape for a fair and meaningful analysis. All the algorithms use the same direct representation and are implemented in their basic components in a straightforward manner. The stress here is in the comparison of the different methods under a common framework, rather than in high performance in solving the problem. More freedom in the use of more efficient representations and more heuristic information may give different result.

3. Common structure

In this section we define a common framework for all non-hybrid metaheuristics that are used for solving the ETP here.

3.1. Search landscape

Since a good solution may be in neighbourhood of a bad solution, we try not to omit infeasible ones. But because of low quality of these solutions we assign a high cost to them ($w = 1000$). This helps the search process, to move away from these points while statistically we have not removed the possibility of a good solution in their neighbourhood. This results in a contiguous but not smooth search space.

3.2. Initial solution

It is reasonable to expect that the quality of initial solution would affect final solution, but we assume a random initial solution, and have not used heuristic methods to produce it. This will help evaluate the methods under study here based on their merits alone, and independent of initial solution.

3.3. Solution representation

We show every solution with one vector, with the length of the vector equal to the number of exams. Each elements of this vector shows the assigned period for each exam. Periods and exams are numbered sequentially.

1	2	3	4				$N-2$	$N-1$	N
20	3	18	5	$\cdots$	$\cdots$	$\cdots$	7	11	6

3.4. Neighbour solution

A neighbour solution may not be a feasible solution, and it is obtained by random alteration of one element of the solution vector.

It is better to mention that the above problem definition, datasets and cost function (as defined in previous section) are common to all the algorithms. Furthermore, all algorithms are executed on one computer.

4. Description of the heuristics

In this section, the four basic heuristic methods are separately but briefly described.

4.1. Simulated annealing

The principle of the SA metaheuristic is deduced from the physical annealing process of solids. Kirckpatrick et al. [23] and Cerny [24] proposed the use of SA for combinatorial problems. Their work is based on the research of Metropolis et al. [25] in the field of Statistical Mechanics. For an overview of the research and applications of SA, the reader is referred to Van Laarhoven and Aarts [26], Aarts and Korst [27], Collins et al. [28] and Eglese [29].

As far as our implementation is concerned, the following choices have been made. In order to determine the value of the *initial temperature*, T_{begin} is computed by solving the expression:

$$P_{\text{a}} = \mathrm{e}^{-\Delta C/T_{\text{begin}}}, \tag{4.1.1}$$

and hence

$$T_{\text{begin}} = \frac{-\Delta C}{\ln P_{\text{a}}}. \tag{4.1.2}$$

Here ΔC represents the average deterioration value, which is computed as the cumulative value of the values of all worsening moves possible from the initial solution divided by the number of moves which have caused a deterioration of the objective function value. Parameter P_{a} represents the acceptance fraction, i.e. the ratio of the accepted to the total number of generated moves.

The *cooling function* we use for the reduction of the temperature in a simple geometric function. The temperature at iteration t, T_t, is obtained from the temperature of the previous iteration as follows:

$$T_t = RT_{t-1} \tag{4.1.3}$$

where R represents the cooling rate.

The stopping criterion is satisfied if the temperature value is near zero.

4.1.1. Algorithm

A general description of SA is given in Table 1.

4.1.2. Parameters

Acceptance fraction (A). This is the percentage of accepted moves obtained when performing 1000 move cycles on the initial solution. This parameter is used to fix the initial temperature. Values assigned to this parameter are:

A	0.30	0.50	0.70

Table 1
The general Simulated Annealing technique

```
Select an initial state $i \in S$
Select an initial temperature $T > 0$;
Set temperature change counter $t = 0$;
Repeat
    Set repetition Counter $n = 0$;
    Repeat
        Generate state $j$, a neighbour of $i$;
        Calculate $\delta = f(j) - f(i)$;
        if $\delta < 0$ Then $i := j$;
        else if random $(0,1) < \exp(-\delta/T)$ Then $i := j$;
        $n := n + 1$;
    Until $n = N(t)$;
    $t := t + 1$;
    $T := T(t)$;
Until Stopping Criterion true.
```

Cooling rate (R). This is the fraction by which the temperature is reduced in the geometric temperature function (4.1.3). Values assigned to this parameter are:

R	0.70	0.80	0.90	0.99

Among above values, the best pair of parameters pair is reported in Table 2.

4.2. Tabu search

Tabu search was conceived by Glover [30]. TS is based on the principles of intelligent problem solving. The idea behind TS is to start from a random solution and successively move to one of its current neighbours. Each time a *move* is performed and linked, the pairs (exam, period) are added to the *tabu list* that includes inhibited moves. It means, period of this exam cannot change until $|tabu\ list|$. From a given solution, not all neighbours can usually be reached. A new candidate move in fact brings the solution to its best neighbour, but if the move is present in the tabu list, it is accepted only if it decreases the objective function value below the minimal level so far achieved (*aspiration level*). This process is repeated until a stopping criterion is reached. The stopping criterion of this algorithm is reaching to the limited number of iterations between current iteration and iteration that best solution is reached.

A good overview of TS and its applications is provided by Glover and Laguna [31,32].

4.2.1. Algorithm

A general description of TS is given in Table 3.

4.2.2. Parameters

Length of tabu list (L). This parameter indicates the size of tabu list and is considered as a fixed number. Values assigned to this parameter are:

L	10	20	30	$[N/3]$	$[N/2]$

Table 2
SA parameters setting

Parameter	Value
Acceptance fraction	0.5
Cooling rate	0.99

Table 3
The general Tabu Search technique

```
Initialization
  $s :=$ initial solution in $X$;
  $nbiter := 0$;
    (* current iteration *)
  $bestiter := 0$;
    (* iteration when the best solution has been found *)
  $bestsol := s$;
    (* best solution *)
  $T := 0$;
    Initialize the aspiration function $A$;
While $(f(s) > f^*)$ and $(nbiter - bestiter < nbmax)$ do
  $nbiter := nbiter + 1$;
  Generate a set $V^*$ of solutions $s_i$ in $N(s)$ which are either
  not tabu or such that $A(f(s)) >= f(s_i)$;
  Choose a solution $s^*$ minimizing $f$ over $V^*$;
  Update the aspiration function $A$ and the tabu list $T$;
  If $f(s^*) < f(bestsol)$ then
  $bestsol := s^*$; $bestiter := nbiter$;
   $s := s^*$;
End while
```

Long-term memory (G). This parameter determines whether or not a long-term memory is used. Values assigned to this parameter are:

1. Implementation without long-term memory
2. Implementation with long-term memory

Max cycles without improvement. This parameter sets the number of iteration between current iteration and the iteration where the best solution is reached. Values assigned to this parameter are:

Max cycles without improvement	5	10	20	30

Among above combination of parameter settings, the best parameters setting achieved for TS is reported in Table 4.

Table 4
TS parameters setting

Parameter	Value
Length of tabu list	$[N/3]$
Long-term memory	No
Max cycles without improvement	30

4.3. Genetic Algorithm

Genetic Algorithm was conceived by Holland [33]. GA is a population-based evolutionary heuristic, where every possible solution is represented by a specific encoding, often called an *individual*. Usually GA is initialized by a set of randomly generated feasible solutions (a *population*), and then individuals are randomly mated allowing the recombination of part of their encoding. The resulting individuals can then be mutated with a specific mutation probability. The new population so obtained undergoes a process of selection which probabilistically removes the worse solutions and provides the basis for a new evolutionary cycle. The fitness of the individuals is made explicit by means of a function, called the *fitness function*, which is related to the objective function to optimize. The *fitness function* quantifies how good a solution is for the problem faced. In GAs individuals are sometimes also called *chromosomes*, and the position in the chromosome are called genes. The value a gene actually takes is called an *allele* (or *allelic value*). Allelic values may vary on a predefined set, that is called *allelic alphabet*.

Let P be a population of N chromosomes (*individuals* of P). Let $P(0)$ be the initial population, randomly generated, and $P(t)$ the population at time t. The GA generates a new population $P(t+1)$ from the old population $P(t)$ applying some *genetic operators*. The four basic genetic operators are:

1. *Reproduction*. An operator which allocates in the population $P(t+1)$ an increasing number of copies of those individuals with a higher *fitness value than* the population $P(t)$ average.
2. *Parent selection*. The parent chromosome are selected according to their fitness ratio. This method is similar to roulette wheel selection and can be expressed as follows:

```
Order chromosomes by decreasing fitness ratio
Get a random number between 0 and 1
For $i = 0$ through (population size − 1)
Sum the fitness ratios of all chromosomes number 0 through I
If (1 − sum from above) is less than or equal to the random number
Then use chromosome $i$ and exit the loop
Otherwise, increment $i$ to the next chromosome and continue
```

3. *Crossover*. A genetic operator activated with a probability p_c. It takes as input two chosen individuals (parents) and combined them to generate two offspring. In this approach we use either one or two point crossovers based on a random process.
4. *Mutation*. An operator that causes, with probability p_m, the change of an allelic value of a randomly chosen gene. In this approach we randomly select an exam and change its timeslot to a random period.

4.3.1. Algorithm

A general description of GA is given in Table 5.

4.3.2. Parameters

N: This parameter indicates the size of population and values assigned to this parameter are:

N	10	50	100	200

P_m: This parameter indicates the mutation rate probability and values assigned to this parameter are:

P_m	0.02	0.1	0.5

P_c: This parameter indicates the crossover rate probability and values assigned to this parameter are:

P_c	0.5	0.8	1

The best parameters setting achieved for GA is reported in Table 6.

Table 5
The general Genetic Algorithm technique

```
  Initialization
      {this routine creates a population of N random individuals}
 while (NOT_VERIFIED_END_TEST) do
      {the end test is on the number of iterations performed}
  begin
      calculate the fitness value for each individual;
      apply reproduction;
      apply parent selection;
      apply crossover with a probability $p_c$;
      apply mutation with a probability $p_m$;
  end.
```

Table 6
GA parameters setting

Parameter	Value
N	100
P_m	0.02
P_c	0.8

4.4. Ant Colony System

Ant Colony algorithms were conceived by Dorigo et al. [34]. Ant Colony Optimization (ACO) algorithms take inspiration from the foraging behavior of real ants. The basic ingredient of ACO is use of a probabilistic solution construction mechanism based on stigmergy. The algorithm presented here is Ant Colony System (ACS) that is a first frame of an ACO algorithm.

The general framework is as follows:

1. Initialize a set A of partial solutions a_i.
2. For $i = 1$ to m
   Choose a component c_j to append to solution a_i with probability given as a function of a_i, η_j, τ_j.
3. If a solution in A are not complete solutions, go to step 2.
4. Evaluate $Z(a_i)$, $i = 1, \dots, m$ and update τ_j, $j = 1, \dots, n$ accordingly.
5. If not (end condition) go to step 1.

In this method each ant follows a list of exams, and for each exam $e \in E$, an ant chooses a timeslot $t \in T$. The ants construct partial assignments $A_i : E_i \to T$ for $i = 0, \dots, |E|$, where $E_i = \{e_1, \dots, e_i\}$. An ant starts with the empty assignment $A_0 = \phi$. After the construction of A_{i-1}, the assignment A_i is built probabilistically as $A_i = A_{i-1} \cup \{(e_i, t)\}$. The timeslot t is chosen randomly out of T according to probabilities $p_{e_i,t}$ that depend on the pheromone matrix $\tau(A_{i-1})$ and heuristic information $\eta(A_{i-1})$ given by

$$p_{e_i,t}(\tau(A_{i-1}), \eta(A_{i-1})) = \frac{(\tau_{(e_i,t)}(A_{i-1}))^{\alpha} \cdot (\eta_{(e_i,t)}(A_{i-1}))^{\beta}}{\sum_{\theta \in T}(\tau_{(e_i,\theta)}(A_{i-1}))^{\alpha} \cdot (\eta_{(e_i,\theta)}(A_{i-1}))^{\beta}} \tag{4.4.1}$$

The impact of the pheromone and the heuristic information can be weighted by parameters α and β and the pheromone matrix is given by $\tau(A_i) = \tau_0$, $i = 1, \dots, |E|$. A simple method for computing the heuristic information is the following:

$$\eta_{(e,t)}(A_{i-1}) = \frac{1.0}{1.0 + V_{(e,t)}(A_{i-1})} \tag{4.4.2}$$

where $V_{(e,t)}(A_{i-1})$ counts the additional number of violations caused by adding (e, t) to the partial assignment A_{i-1}. The function V may be a weighted sum of several soft and hard constraints.

Let $A_{\text{global best}}$ be the assignment of the best solution $C_{\text{global best}}$ found since the beginning. The following update rule is used:

$$\tau_{(e,t)} = \begin{cases} (1-\rho) \cdot \tau_{(e,t)} + 1 & \text{if } A_{\text{global best}}(e) = t \\ (1-\rho) \cdot \tau_{(e,t)} & \text{otherwise} \end{cases} \tag{4.4.3}$$

4.4.1. Algorithm

A general description of ACS is given in Table 7.

4.4.2. Parameters

ρ: This parameter is the evaporation rate and lies in interval $[0, 1]$. Values assigned to this parameter are:

ρ	0.2	0.4	0.6	0.8	0.99

α: This parameter indicates the importance of pheromone trace and values assigned to this parameter are:

α	0.2	0.4	0.6	0.8	1

β: This parameter indicates the importance of heuristic information and values assigned to this parameter are:

β	0.2	0.4	0.6	0.8	1

m: This parameter indicates number of ants and values assigned to this parameter are:

m	10	20	$[n/3]$	$[n/2]$	n

Table 7
The general Ant System algorithm

```
Input: A problem instance $I$
  $\tau_0 \leftarrow \frac{1}{\rho}$
  $\tau(e,t) \leftarrow \tau_0 \forall (e,t) \in E \times T$
  while time limit not reached do
    for $a = 1$ to $m$ do
      {construction process of ant $a$}
      $A_0 \leftarrow \emptyset$
      for $i = 1$ to $|E|$ do
        $e_i$ for exam $p_{e_i,t}$ choose timeslot $t$ randomly according to probabilities
        $A_i \leftarrow A_{i-1} \cup \{(e_i, t)\}$
      end for
      solution $C \leftarrow$
      best of $C$ and $C_{\text{iteration best}} \leftarrow \quad C_{\text{iteration best}}$
    end for
  solution after applying local search to $C_{\text{iteration best}} \leftarrow \quad C_{\text{iteration best}}$
  best of $C_{\text{iteration best}}$ and $C_{\text{global best}} \leftarrow \quad C_{\text{global best}}$
  using $C_{\text{global best}}$ $\tau$ Global pheromone update for
  End while
Output: An optimized candidate solution $C_{\text{global best}}$ for $I$
```

Table 8
ACS parameters setting

Parameter	Value
ρ	0.8
α	1
β	0.4
m	n

The best parameters setting achieved for ACS is reported in Table 8.

5. Problem datasets

We produce several problems in different size in order to apply these algorithms for different ones. In these problems the number of exams varies from 40 to 200 in line with the number of students and number of periods. The elements of conflict matrix of student A_{ij} (that shows the common students in both i and j exams) has been produced randomly. You can see the information about these problems in Table 9.

6. Heuristic analysis

Due to the fact that the stopping criterion of the metaheuristics are not similar, a simple comparison of only the final solution values of the four metaheuristics is not appropriate. Furthermore, the computing time of heuristics highly depends on the value assigned to the parameters. Also it is difficult to estimate the processing time of heuristics. Moreover, the probability of finding a better final solution increases with the run time. Therefore a simple

Table 9
Characteristics of data sets

Data set	Exams	Timeslots	Students
1	40	15	800
2	60	15	1400
3	80	20	1900
4	100	24	2850
5	120	20	3600
6	140	24	4552
7	150	25	4800
8	160	32	5226
9	180	28	6540
10	200	30	7000

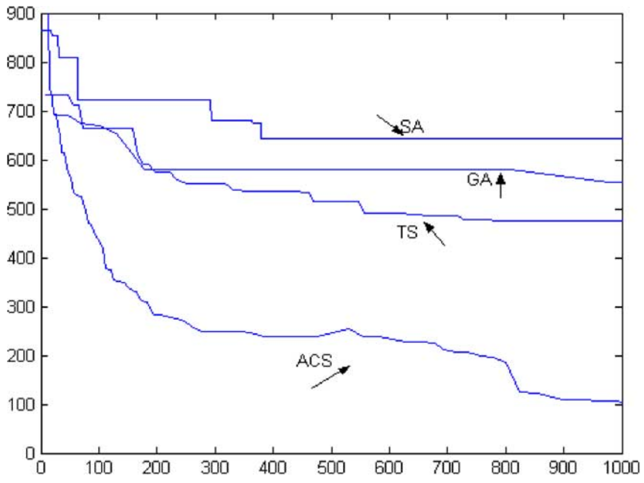

Fig. 1. Example of the path of the objective function value versus computing time for Simulated Annealing, Tabu Search, Genetic Algorithm and Ant Colony System.

comparison of the final solution of the four metaheuristics without taking into account the run time is not appropriate.

An important analysis tool for the dynamic heuristic analysis is the graphical representation of the path of the objective function value of each heuristic versus computing time. Examples are given in Fig. 1.

An alternative strategy for dynamic comparison of the heuristic algorithms is required. The specific feature of the dynamic analysis is that intermediary solutions of metaheuristics at various time points are compared and three time points are considered.

In Table 10 the symbol '*' indicates which heuristic attains its minimal value after the given run time. The best solutions of the four metaheuristics at each time point and maximum reduction of cost are printed in bold face. The column at the right of each cell contains the relative difference with respect to the best solution at that time point.

$$\text{relative difference (solution)} = \frac{\text{cost of solution} - \text{cost of best solution}}{\text{cost of best solution}} \tag{6.1}$$

The same computer has been used for all experiments and programs are written in Matlab software Vr.6.5.

The initial solution of SA, TS and GA are completely random. But initial condition for ACS is constructed based on initial pheromone and heuristic information. Since the amount of initial pheromone is same in all paths,

Table 10
Heuristic analysis of ACS, GA, SA and TS

Problem		Initial	Mini-mum 1		Mini-mum 2		Mini-mum 3		Cost reduction
S1	*Time*		50		140		180		
	SA	938.5788	881.9550	0.47	813.7112	1.03	765.5462*	1.16	173.0326
	TS	877.0063	732.1413	0.22	532.6425*	0.33	532.6425	0.50	344.3638
	GA	771.8504	668.2213	0.12	627.4588*	0.57	627.4588	0.77	144.3916
	ACS	811.0500	**597.8813**		**399.8650**		**354.4787***		**456.5713**
S2	*Time*		30		120		480		
	SA	1116.4	1100.2	0.05	1038.1	0.14	976.0064*	0.25	140.3936
	TS	1197.5	1066.8	0.01	**912.3136**		835.0114*	0.07	362.4886
	GA	1226.2	**1051.6**		1016.5	0.11	975.7036*	0.25	250.4964
	ACS	1222.7	1171.2	0.11	1073.6	0.18	**780.3814***		**442.3186**
S3	*Time*		60		300		540		
	SA	1217.3	1217.3	0.24	1150.3	0.24	1059.2*	0.19	158.1
	TS	1365.0	1802.8	0.84	956.0505	0.03	905.3221*	0.02	**459.6779**
	GA	1149.3	1072.3	0.10	1072.3	0.16	965.5547*	0.09	183.7453
	ACS	1139.4	**977.4874**		**927.8768**		**887.1479***		252.2521
S4	*Time*		60		300		540		
	SA	940.2081	723.8502	0.14	697.2733	0.09	676.2716*	0.15	**263.9365**
	TS	750.8274	697.1719	0.09	646.0586	0.01	**585.7909***		165.0365
	GA	727.6027	**641.2635***		**641.2635**		641.2635	0.09	86.3322
	ACS	683.0372	667.4112	0.04	656.3277	0.02	630.3284*	0.08	52.7088
S5	*Time*		100		300		540		
	SA	1015.2	994.5783	0.10	979.9100*	0.09	979.9100	0.09	35.29
	TS	1125.5	1001.8	0.11	952.2086	0.05	915.0758*	0.01	**210.4242**
	GA	986.6955	**902.4239***		**902.4239**		**902.4239**		84.2716
	ACS	1034.3	945.5294	0.05	945.5294	0.05	919.4200*	0.02	114.88
S6	*Time*		300		600		960		
	SA	1456.3	1417.6	0.07	1417.6	0.17	1379.2*	0.14	77.1
	TS	1468.8	1390.4	0.04	**1207.8***		**1207.8**		**261.0**
	GA	1464.0	**1330.1***		1330.1	0.10	1330.1	0.10	134.0
	ACS	1358.0	1348.5	0.01	1343.2	0.11	1299.8*	0.08	58.2
S7	*Time*		350		700		1020		
	SA	1739.3	1494.2	0.08	1494.2	0.12	1355.9*	0.06	383.4
	TS	1736.4	1434.1	0.04	**1335.0**		**1273.5***		**462.9**
	GA	1492.9	1404.8	0.02	1398.0	0.05	1362.0*	0.07	130.9
	ACS	1537.9	**1383.0**		1368.1	0.02	1341.8*	0.05	196.1
S8	*Time*		380		800		1200		
	SA	1266.3	**1147.4**		1143.9	0.01	1124.9*	0.05	141.4
	TS	1304.3	1258.5	0.10	1147.4	0.01	**1068.3***		**236.0**
	GA	1242.8	1163.8	0.01	**1130.4**		1094.1*	0.02	148.7
	ACS	1197.3	1159.3	0.01	1135.1	0.00	1134.5*	0.06	62.8

Table 10 (*continued*)

Problem		Initial	Minimum 1		Minimum 2		Minimum 3		Cost reduction
S9	*Time*		470		670		1200		
	SA	1507.6	1389.5*	0.06	1389.5	0.11	1389.5	0.13	118.1
	TS	1546.0	1376.0	0.04	1347.4	0.08	1299.3*	0.06	246.7
	GA	1634.8	1312.6*	0.00	1312.6	0.05	1312.6	0.07	**322.2**
	ACS	1491.9	**1310.7**		**1247.6**		**1227.7***		264.2
S10	*Time*		300		640		1200		
	SA	1573.2	1557.7	0.10	1557.7	0.10	1441.4*	0.08	131.8
	TS	1624.6	1530.6	0.08	1427.4	0.01	**1334.7***		**289.9**
	GA	1681.1	1561.0	0.11	1426.4*	0.01	1426.4	0.07	254.7
	ACS	1410.9	**1410.6***		**1410.6**		1410.6	0.06	0.3

Table 11
Ability of metaheuristics to find the best solution

Algorithm	SA	TS	GA	ACS
Ability of Alg. to find the best solution	%3.3	%26.6	%26.6	%43.3

heuristic information has the main effect in construction of initial solution. In %100 of problems ACS had a better initial solution.

As it is shown in Table 10, we gain the best solution from ACS at most time points. Therefore, ACS makes the first grade and then TS after that GA and SA has the fourth grade (Table 11).

The values that have been shown in Table 11 achieved from following formula:

$$\frac{\text{The number of trials that each algorithm has found the best solution}}{\text{Total number of all trials}} \times 100 \tag{6.2}$$

Since the initial solution of each of the four metaheuristics are different and this affects quality of the final solution, we calculate amount of cost reduction for all of them. As it is shown in Table 12, TS had the highest reduction in cost of solutions in same time.

$$\frac{\text{The number of test problems that each algorithm has found the highest cost reduction}}{\text{Total number of all test problems}} \times 100 \tag{6.3}$$

Table 12
Ability to produce maximum cost deterioration in four metaheuristics

Algorithm	SA	TS	GA	ACS
Ability to produce maximum cost deterioration	%10	%60	%10	%20

Table 13
The algorithm of Sequential TS–ACS

```
**Initialization**
  $s :=$ initial solution in $X$;
  $nbiter := 0$;
    (* current iteration *)
  $bestiter := 0$;
    (* iteration when the best solution has been found *)
  $bestsol := s$;
    (* best solution *)
  $T := 0$;
  Initialize the aspiration function $A$;
**While** $(f(s) > f^*)$ **and** $(nbiter\ bestiter < nbmax)$ **do**
  $nbiter := nbiter + 1$;
  Generate a set $V^*$ of solutions $s_i$ in $N(s)$ which are either
  not tabu or such that $A(f(s)) >= f(s_i)$;
  Choose a solution $s^*$ minimizing $f$ over $V^*$;
  Update the aspiration function $A$ and the tabu list $T$;
  If $f(s^*) < f(bestsol)$ then
  $bestsol := s^*$; $bestiter := nbiter$;
    $s := s^*$;
**End while**
{beginning of ACS}
**Input:** *best solution* of TS
using *best sol* of TS$\tau$ Global pheromone update for
  **while** time limit not reached **do**
  **for** $a = 1$ **to** m **do**
    {construction process of ant $a$}
  $A_0 \leftarrow \emptyset$
    **for** $i = 1$ **to** $|E|$ **do**
$e_i$ for exam $p_{e_i,t}$ choose timeslot $t$ randomly according to probabilities
$A_i \leftarrow A_{i-1} \cup \{(e_i, t)\}$
    **end for**
      solution $C \leftarrow$
best of $C$ and $C_{\text{iteration best}} \leftarrow$   $C_{\text{iteration best}}$
   **end for**
solution after applying local search to $C_{\text{iteration best}} \leftarrow$   $C_{\text{iteration best}}$
best of $C_{\text{iteration best}}$ and $C_{\text{global best}} \leftarrow$   $C_{\text{global best}}$
using $C_{\text{global best}}$ $\tau$ Global pheromone update for
  **End while**
  **Output:** An optimized candidate solution $C_{\text{global best}}$
```

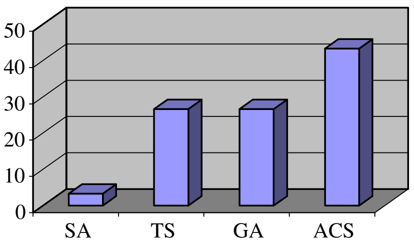

Chart 1. Ability of algorithm to find the best solution.

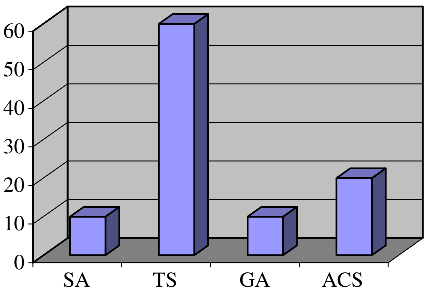

Chart 2. Ability of algorithm to produce maximum cost deterioration.

Results of Tables 12 and 13 are shown in Charts 1 and 2.

7. Proposed hybrid methods

By studying the behavior of each of the above pure methods, a properly chosen combination may be expected to yield improved performance. Here, we introduce three hybrid methods that combine the best metaheuristics of above-discussed methods, ACS and TS, in various ways.

7.1. Sequential TS–ACS (STA)

In this method, we fist use TS algorithm and, based on the best solution reached by TS, justify amount of initial pheromones in ACS path. This causes increasing initial pheromones in the path of better solutions and so guides the ants to fine tune this path faster.

To achieve above strategy, TS is first applied and then pheromone is updated globally before applying ACS and we use final solution of TS as a good initial solution in this part. Then ACS is used in the main framework but with

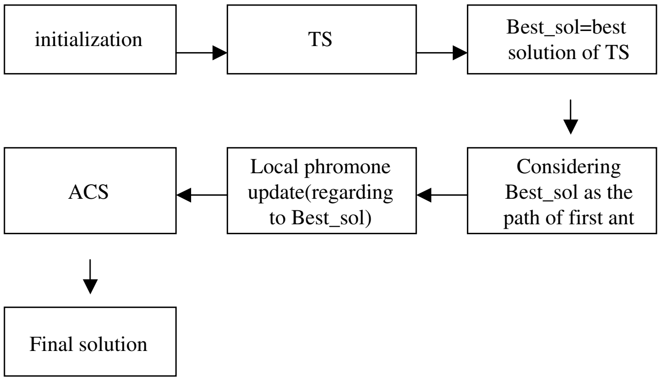
initialization
TS
Best_sol=best solution of TS
Considering Best_sol as the path of first ant
Local phromone update(regarding to Best_sol)
ACS
Final solution

Fig. 2. The block diagram of Sequential TS–ACS.

modified amount of initial pheromone. In fact we guide the ant in the beginning of their efforts in finding the best path.

Hence we can investigate that how the amount of initial pheromone affects the quality of initial as well as final solutions. The block diagram and algorithm of this method are given in Fig. 2 and Table 13.

7.2. Hybrid ACS/TS (HAT)

In Section 6, it was shown that TS had the highest cost reduction. So in this method ACS is the main algorithm and TS is used in local search part of it. TS is terminated before it converges and the process continues for the remaining steps of ACS. In other words, this hybrid method has another strong meta-heuristic searching engine in the local search which is a strategic and important part.

The block diagram and algorithm of this hybrid are given in Fig. 3 and Table 14.

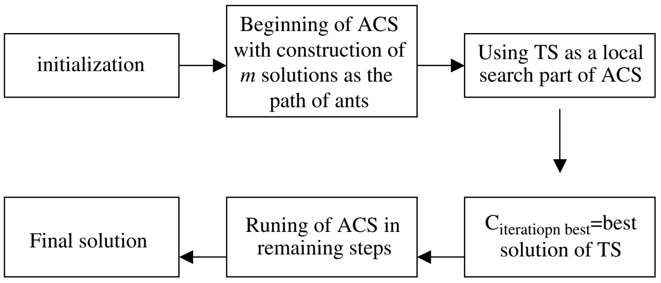
initialization
Beginning of ACS with construction of m solutions as the path of ants
Using TS as a local search part of ACS
C_iteratiopn best=best solution of TS
Runing of ACS in remaining steps
Final solution

Fig. 3. The block diagram of Hybrid ACS/TS.

Table 14
The algorithm of Hybrid ACS/TS

```
Input : A problem instance $I$
  $\tau_0 \leftarrow \frac{1}{\rho}$
  $\tau(e,t) \leftarrow \tau_0 \forall (e,t) \in E \times T$
while time limit not reached do
  for $a = 1$ to m do
    {construction process of ant $a$}
$A_0 \leftarrow \emptyset$
    for $i = 1$ to $|E|$ do
$e_i$ for exam $p_{e_i,t}$ choose timeslot $t$ randomly according to probabilities
$A_i \leftarrow A_{i-1} \cup \{(e_i, t)\}$
    end for
      solution $C \leftarrow$
best of $C$ and $C_{\text{iteration best}} \leftarrow$   $C_{\text{iterationbest}}$
  end for
  {applying local search with TS method}
Initialization
  $s := C_{\text{iteration best}}$ of ACS;
  $nbiter := 0$;
    (* current iteration *)
  $bestiter := 0$;
    (* iteration when the best solution has been found *)
  $bestsol := s$;
    (* best solution *)
  $T := 0$;
  Initialize the aspiration function $A$;
While $(f(s) > f^*)$ and $(nbiter\ bestiter < nbmax)$ do
  $nbiter := nbiter + 1$;
  Generate a set $V^*$ of solutions $s_i$ in $N(s)$ which are either
  not tabu or such that $A(f(s)) >= f(s_i)$;
  Choose a solution $s^*$ minimizing $f$ over $V^*$;
  Update the aspiration function $A$ and the tabu list $T$;
  If $f(s^*) < f(bestsol)$ then
  $bestsol := s^*$; $bestiter := nbiter$;
   $s := s^*$;
  End while
$bes\ sol$ from TS $\leftarrow C_{\text{iteration best}}$
best of $C_{\text{iteration best}}$ and $C_{\text{global best}} \leftarrow$   $C_{\text{global best}}$
using $C_{\text{global best}}$ $\tau$ Global pheromone update for
  End while
  Output: An optimized candidate solution $C_{\text{global best}}$ for $I$
```

7.3. Sequential ACS–TS (SAT)

Since ACS can produce better initial solutions, and at the same time, TS in comparison to ACS has highest cost reduction, in this hybrid method, at first we use ACS algorithm and then use the solution of this as a good initial solution in the TS algorithm.

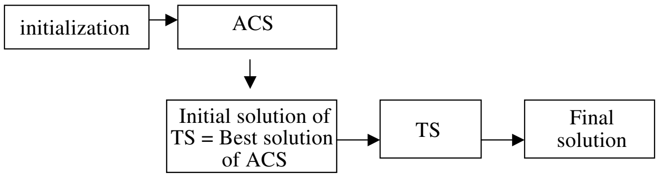

Fig. 4. The block diagram of Sequential ACS–TS.

ACS is terminated before it converged and TS continues the searching process until finding the best solution or some stop conditions are reached.

The block diagram and algorithm of this method are given in Fig. 4 and Table 15.

8. Hybrid heuristic analysis

Note that in all of these methods, the parameters have been used that have provided the best performance in previous simulations when we introduced them separately.

The path of the objective function versus computing time for one of the runs is shown in Fig. 5.

The cumulative result of these three hybrid methods is shown in Table 16. The framework of this table is as same as Table 10 and the notation for bold faces and symbol '*' are same. Also the same computer has been used for all experiments.

As it is shown in Tables 16–18, SAT is in first grade and then STA after that HAT and ACS has the fourth grade.

The elements of Table 17 are achieved from formula (6.2).

The chart of this results is shown in Charts 3 and 4.

The charts of final results of comparison of Simulated Annealing, Tabu Search, Genetic Algorithm, Ant Colony System, Sequential TS–ACS, Hybrid ACS/TS and Sequential ACS–TS are shown in Charts 5 and 6.

9. Conclusions

In this paper we considered four conventional metaheuristic methods (Simulated Annealing, Tabu Search, Genetic Algorithm and Ant Colony System) and proposed three new hybrid methods for solving ETP and compared the results of them on 10 datasets. When comparing the results of the

Table 15
The algorithm of Sequential ACS–TS

```
Input: A problem instance $I$
  $\tau_0 \leftarrow \frac{1}{\rho}$
  $\tau(e,t) \leftarrow \tau_0 \forall (e,t) \in E \times T$
  while time limit not reached do
  for $a = 1$ to m do
    {construction process of ant $a$}
  $A_0 \leftarrow \emptyset$
    for $i = 1$ to $|E|$ do
$e_i$ for exam $p_{e_i,t}$ choose timeslot $t$ randomly according to probabilities
$A_i \leftarrow A_{i-1} \cup \{(e_i,t)\}$
    end for
    solution $C \leftarrow$
best of $C$ and $C_{\text{iteration best}} \leftarrow$    $C_{\text{iteration best}}$
  end for
solution after applying local search to $C_{\text{iteration best}} \leftarrow$    $C_{\text{iteration best}}$
best of $C_{\text{iteration best}}$ and $C_{\text{global best}} \leftarrow$    $C_{\text{global best}}$
using $C_{\text{global best}}$ $\tau$ Global pheromone update for
  End while
  Output: An optimized candidate solution $C_{\text{global best}}$ for $I$
{beginning of TS}
Initialization
  $s := C_{\text{global best}}$ of ACS;
  $nbiter := 0$;
    (* current iteration *)
  $bestiter := 0$;
    (* iteration when the best solution has been found *)
  $bestsol := s$;
    (* best solution *)
  $T := 0$;
  Initialize the aspiration function $A$;
While $(f(s) > f^*)$ and $(nbiter\ bestiter < nbmax)$ do
  $nbiter := nbiter + 1$;
  Generate a set $V^*$ of solutions $s_i$ in $N(s)$ which are either
  not tabu or such that $A(f(s)) >= f(s_i)$;
  Choose a solution $s^*$ minimizing $f$ over $V^*$;
  Update the aspiration function $A$ and the tabu list $T$;
  If $f(s^*) < f(bestsol)$ then
  $bestsol := s^*$; $bestiter := nbiter$;
   $s := s^*$;
End while
```

conventional methods, ACS showed better ability to find best solutions while TS algorithm worked better terms of improving its performance over time.

In comparison, all of the three hybrid methods demonstrate better results than all metaheuristic methods. Among the hybrids, the Sequential ACS–TS is better than others. But why does this happen?

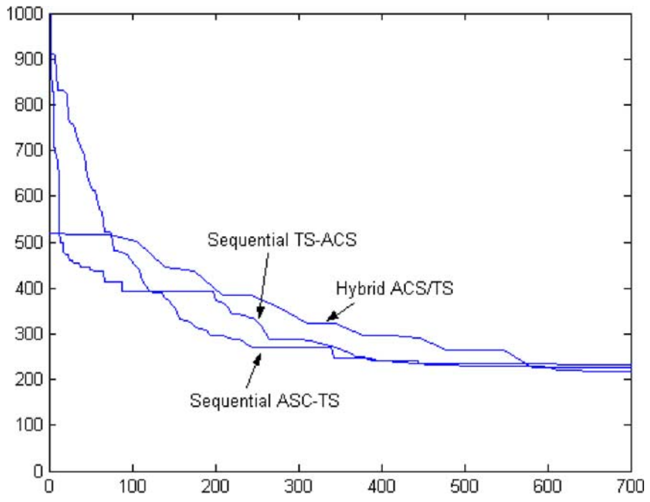

Fig. 5. Example of the path of the objective function value versus computing time for STA, HAT and SAT.

In the Sequential TS–ACS, we modify the amount of pheromones by applying TS before ACS. In this way, ants select the paths which have more pheromone on them. In the other word the pairs (exam, period) with more initial pheromones have more chances to be selected and others have less. So we are doing biased search in the solution space and the algorithm may converge faster in a local minima.

In the Hybrid ACS/TS, using TS is time-consuming for each ant and therefore one iteration of ACS takes a substantially longer duration, and so for a complete run of it. Therefore only a few iterations of the main algorithm can be applied in the indicated points.

The Sequential ACS–TS method at first produces a good solution by using ACS but in final iterations of ACS, solutions are converged, because of a lots of pheromone in good regions and a poor amount of it in bad ones.

In that time ACS causes poor improvement in the final solution and it is useless for us to continue it. Therefore we interrupt ACS before its convergence and then use TS to search the neighbourhood region. TS searches the neighbourhood of solution and can find the minimum point in the close points. We choose the TS because, considering results of previous section, it worked better than other metaheuristic methods in cost reduction in the same time duration.

The characteristics of Sequential ACS–TS method allows it to work better than others. Also the final results which have been shown in Table 16–18 shows that the Sequential ACS–TS works better than Sequential TS–ACS and Hybrid ACS/TS.

Table 16
Heuristic analysis of Sequential TS–ACS (STA), Hybrid ACS/TS (HAT) and Sequential ACS–TS (SAT)

Problem		Initial	Minimum 1		Minimum 2		Minimum 3		Cost reduction
S1	*Time*		50		140		180		
	Best solutions		597.8813 (ACS)	0.31	399.8650 (ACS)	0.26	354.4787 (ACS)	0.18	
	STA	844.8687	458.8850	0.01	340.1125	0.07	**301.2962***		543.5725
	HAT	951.7188	**456.1350**		405.7150	0.28	396.2275*	0.31	**555.4913**
	SAT	782.6875	491.2788	0.08	**316.4975**		311.7163*	0.03	470.9712
S2	*Time*		30		120		480		
	Best solutions		1051.6 (GA)	0.07	912.3136 (TS)	0.19	780.3814 (ACS)	0.42	
	STA	1463.3	**980.4007**		980.4007	0.28	667.8714*	0.21	**795.4286**
	HAT	1351.0	982.9750	0.00	776.0479	0.02	634.0179*	0.15	716.9821
	SAT	1104.8	1039.6	0.06	**763.6843**		**549.9636***		554.8364
S3	*Time*		60		300		540		
	Best solutions		977.4874 (ACS)	0.03	927.8768 (ACS)	0.17	887.1479 (ACS)	0.50	
	STA	1323.0	**951.6329**		863.3350	0.09	**591.3121***		**731.6879**
	HAT	1155.8	984.2968	0.03	879.9663	0.11	747.0868*	0.26	408.7132
	SAT	1188.6	1172.5	0.21	**794.2189**		683.5447*	0.16	505.0553
S4	*Time*		60		300		540		
	Best solutions		641.2635 (GA)	0.05	641.2635 (GA)	0.39	585.7909 (TS)	0.36	
	STA	730.5260	**608.5807**		598.0639*	0.30	598.0639	0.39	132.4621
	HAT	626.9172	611.1174	0.00	547.5333	0.19	518.1196*	0.21	108.7976
	SAT	645.5154	641.8604	0.05	**459.9646**		**429.3235***		**216.1919**
S5	*Time*		100		300		540		
	Best solutions		902.4239 (GA)	0.02	902.4932 (GA)	0.13	902.4932 (GA)	0.24	

Table 16 (*continued*)

Problem		Initial	Minimum 1		Minimum 2		Minimum 3		Cost reduction
	STA	1163.2	958.1642*	0.84	958.1642	0.20	958.1642	0.32	205.0358
	HAT	995.7794	906.9381	0.03	880.2150	0.10	793.4969*	0.09	202.2825
	SAT	959.6219	**884.2408**		**801.0167**		**726.9256***		**232.6963**
S6	*Time*		300		600		960		
	Best solutions		1330.1 (GA)	0.13	1207.8 (TS)	0.18	1207.8 (TS)	0.25	
	STA	1485.3	1302.1	0.10	1254.9	0.22	1222.3*	0.27	263.0
	HAT	1358.3	1285.3	0.09	1263.3	0.23	1185.4*	0.23	172.9
	SAT	1328.9	**1177.6**		**1026.1**		**964.5477***		**364.3523**
S7	*Time*		350		700		1020		
	Best solutions		1383.0 (ACS)	0.16	1335.0 (TS)	0.24	1273.5 (TS)	0.30	
	STA	1567.3	1333.5	0.12	1267.5	0.19	1255.0*	0.28	312.3
	HAT	1400.9	1354.6	0.13	1353.6	0.26	1295.2*	0.32	105.7
	SAT	1383.0	**1194.7**		**1073.5**		**981.2917***		**401.7083**

S8	*Time*		380		800		1200		
	Best solutions		1147.4	0.16	1130.4	0.36	1068.3	0.37	
			(SA)		(GA)		(TS)		
	STA	1269.1	1138.0	0.15	1026.3	0.24	1025.0*	0.31	244.1
	HAT	1100.3	1052.4	0.07	1008.6	0.22	938.3370*	0.20	161.963
	SAT	1100.3	**985.2937**		**829.4323**		**781.1***		**319.2**
S9	*Time*		470		670		1200		
	Best solutions		1310.7	0.10	1247.6	0.11	1227.7	0.20	
			(ACS)		(ACS)		(ACS)		
	STA	1485.7	1447.7	0.21	1370.8	0.22	1349.5	0.31	136.2
	HAT	1329.0	1255.7	0.05	1255.7	0.12	1232.2	0.20	96.8
	SAT	1322.2	**1192.6**		**1123.6**		**1025.6**		**296.6**
S10	*Time*		300		640		1200		
	Best solutions		1410.6	0.08	1410.6	0.17	1334.7	0.14	
			(ACS)		(ACS)		(TS)		
	STA	1703.5	1465.7	0.12	1465.7	0.21	1439.1*	0.23	**264.4**
	HAT	1508.9	1472.4	0.12	1375.2	0.14	1342.9*	0.14	166.0
	SAT	1421.5	**1310.4**		**1206.5**		**1173.2***		248.3

Table 17
Ability of STA, HAT and SAT to find the best solution as in Eq. (6.2)

Algorithm	STA	HAT	SAT
Ability of Alg. to find the best solution	%16.6	%3.3	%80

Table 18
Ability of algorithm to produce maximum cost deterioration in three methods as in Eq. (6.3)

Algorithm	STA	HAT	SAT
Ability to produce maximum cost deterioration	%30	%10	%60

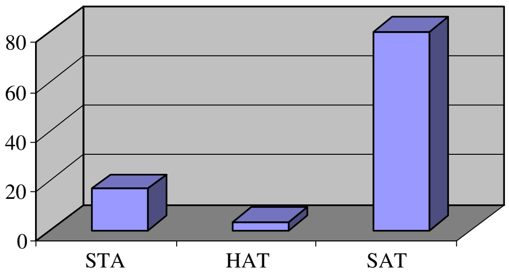

Chart 3. Ability of algorithm to find the best solution.

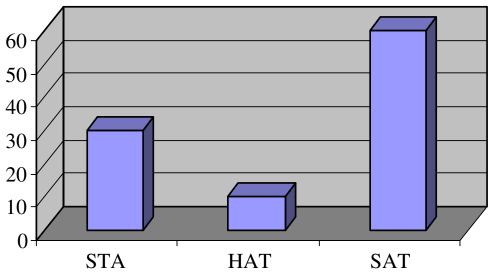

Chart 4. Ability of algorithm to produce maximum cost reduction.

Acknowledgements

I thank Dr. Mohamad-Reza Akbarzadeh-Totonchi for his review on this paper and editing it and his good comments.

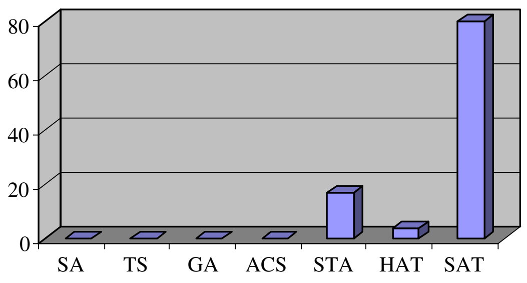

Chart 5. Final result of ability of algorithms to find the best solution.

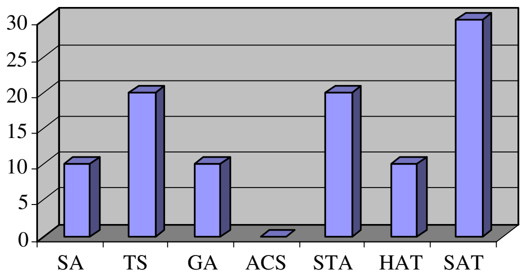

Chart 6. Final result of ability of algorithms to produce maximum cost reduction.

References

[1] D.J.A. Welsh, M.B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, The Computer Journal 10 (1967) 85–86.

[2] R.M. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, Plenum Press, New York, 1972.

[3] M.W. Carter, A survey of practical applications of Examination Timetabling algorithms, Operation Research 34 (2) (1986) 193–202.

[4] M.W. Carter, G. Laporte, S.Y. Lee, Examination Timetabling: algorithmic strategies and applications, Journal of the Operational Research Society 47 (1996) 373–383.

[5] M. Cangalovic, J.A.M. Schreuder, Exact Colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths, European Journal of Operational Research 51 (1991) 248–258.

[6] D.C. Rich, A Smart Genetic Algorithm for university timetabling, in: E. Burke, P. Ross (Eds.), The Practic and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, 1996, pp. 181–197.

[7] A. Colorni, M. Dorigo, V. Maniezzo, Genetic Algorithms: a new approach to the timetabling problem, in: E. Burke, P. Ross (Eds.), The Practic and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, 1996, pp. 235–239.

[8] W. Erben, J. Keppler, A Genetic Algorithm solving a weekly course-timetabling problem, in: E. Burke, P. Ross (Eds.), The Practic and Theory of Automated Timetabling, Lecture Notes in Computer Science, vol. 1153, Springer, 1996, pp. 198–211.

[9] D. Corne, P. Ross, H.-L. Fang, Evolving timetabling, in: L. Chambers (Ed.), Practical Hand Book of Genetic Algorithms: Applications, vol. 1, CRC Press, 1999.

[10] L.F. Paquete, C.M. Fonseca, A study of Examination Timetabling with Multiobjective Evolutionary algorithms, in: MIC 2001—4th Metaheuristic International Conference, Porto, Portugal, July 16–20, 2001.

[11] A.S. Asratian, D. de Werra, A generalized class-teacher model for some timetabling problems, European Journal of Operational Research 143 (2002) 531–542.

[12] O. Rossi-Doria, Ch. Blum, J. Knowles, M. Sampels, K. Socha, B. Paechter, A local search for the timetabling problem, Technical Report No. TR/IRIDIA/2002-16, July 2002.

[13] E.K. Burke, Y. Bykov, J.P. Newall, S. Petrovic, A time-predefined local search approach to exam timetabling problems, Computer Science Technical Report No. NOTTCS-TR-2001-6.

[14] E.K. Burke, Y. Bykov, J.P. Newall, S. Petrovic, A new local search approach with execution time as an input parameter, Computer Science Technical Report No. NOTTCS-TR-2002-3.

[15] J.M. Thompson, K.A. Dowsland, A robust simulated annealing based Examination Timetabling system, Computers and Operations Researches 25 (7/8) (1998) 637–648.

[16] A. Hertz, Tabu search for large scale timetabling problems, European Journal of Operational Research 54 (1991) 39–47.

[17] L.D. Gaspero, A. Schaerf, Tabu Search techniques for Examination Timetabling, in: Third International Conference Patat 2000, Lecture Notes in Computer Science, vol. 2079, 2000, p. 104ff.

[18] K. Socha, J. Knowles, M. Sampels, A max–min ant system for the university course timetabling problem, in: Proceedings of ANTS 2002—Third International Workshop on Ant Algorithms, Lecture Notes in Computer Science, vol. 2463, Springer Verlag, Berlin, Germany, 2002, pp. 1–13 (also Technical Report/TR/IRIDIA/2002-18).

[19] J. Wood, D. Whitaker, Student centred school timetabling, Journal of the Operational Research Society 49 (1998) 1146–1152.

[20] Victor A. Bardadym, Computer-aided school and university timetabling: the new wave, Lecture Notes in Computer Science 1153 (1996) 22–45.

[21] S. Deris, S. Omatu, H. Ohta, Timetable planning using the constraint-based reasoning, Computer & Operations Research 27 (2000) 819–840.

[22] E.K. Burke, S. Petrovic, Recent research directions in automated timetabling, European Journal of Operational Research 140 (2002) 266–280.

[23] S. Kirckpatric, C. Gellat Jr., M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[24] V. Cerny, A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory Application 45 (1985) 41–51.

[25] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, Equation of state calculations by fast computing machines, Journal of Chemical Physics 21 (1953) 1087–1092.

[26] P. Van Laarhoven, E. Aarts, Simulated Annealing: Theory and Practic, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1987.

[27] E. Aarts, J. Korst, Simulated Annealing and Boltzmann Machines, Wiley, Chichester, 1989.

[28] N. Collins, R. Eglese, B. Golden, Simulated annealing an annotated bibliography, American Journal of Mathematical and Management Sciences 8 (1988) 209–307.

[29] R.W. Eglese, Simulated annealing: a tool for operational research, European Journal of Operational Research 46 (1990) 271–281.

[30] F. Glover, The general employee scheduling problem: an integration of management science and artificial intelligence, Computers and Operations Research 15 (1986) 563–593.

[31] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.

[32] K.E. Rosing, C.S. ReVelle, E. Rolland, D.A. Schilling, J.R. Current, Heuristic concentration and Tabu Search: a head to head comparison, European Journal of Operational Research 104 (1998) 93–99.

[33] R.L. Haupt, S.E. Haupt, Practical Genetic Algorithms, Wiley–Interscience Publication, John Wiley, New York, 1997.
[34] M. Dorigo, V. Maniezzo, A. Colorni, Positive feedback as a search strategy, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, IT, 1991.