J. Appl. Math. & Computing Vol. **16***(2004), No. 1 - 2, pp. 337 - 354*

COMPARISON OF METAHEURISTIC ALGORITHMS FOR EXAMINATION TIMETABLING PROBLEM

ZAHRA NAJI AZIMI

Abstract. SA, TS, GA and ACS are four of the main algorithms for solving challenging problems of intelligent systems. In this paper we consider Examination Timetabling Problem that is a common problem for all universities and institutions of higher education. There are many methods to solve this problem, In this paper we use Simulated Annealing, Tabu Search, Genetic Algorithm and Ant Colony System in their basic frameworks for solving this problem and compare results of them with each other.

AMS Mathematics Subject Classification : 90B99, 90C99.
Key words and phrases : Timetabling, simulated annealing, tabu search, genetic algorithm, ant colony system.

1. Introduction

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 colors to vertices in graph so that adjacent vertices always have different colors [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 so many heuristic methods which has been offered for solving this problem based on graph coloring as well as many metaheuristic methods such as SA, TS, GA, and ACS [3-22].

In this paper we use metaheuristics for solving this problem and compare results of them with each other.

To continue we explain Examination Timetabling problem.

Received July 30, 2003. Revised June 24, 2004.

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 $soft$ constraint 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: $w(1) = 16, w(2) = 8, w(3) = 4, w(5) = 1$ 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,

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

$$f_1 = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} C_{ij}.w(i,j), \quad f_2 = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} C_{ij}.w'(i,j)$$

where

$$w(i,j) = \begin{cases} 2^{5-|t_i - t_j|} & \text{if } 1 \leq |t_i = t_j| \leq 5 \\ 0 & \text{otherwise} \end{cases}$$

$$w'(i,j) = \begin{cases} 1000 & \text{if } t_i = t_j \\ 0 & \text{otherwise.} \end{cases}$$

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 the period of exam i (for i=1,...,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 common framework for all metaheuristics that are used for solving this problem.

3.1. Search landscape

Since a good solution may be in neighborhood of a bad solution, we try not to mit infeasible ones. But because of low quality of these solution we assign 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 neighborhood. 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 and each elements of this vector shows the assigned period for each exam.

3.4. Neighbour solution

A neighbor 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 metaheuristic

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 Vanlaarhoven 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:

$$P_a = e^{-\Delta C/T_{\text{begin}}} \tag{4.1.1}$$

and hence

$$T_{\text{begin}} = \frac{-\Delta C}{\ln P_a} \tag{4.1.2}$$

Here ΔC represents the average deterioration value, which is computed as the cumulative value of the values of all worsening moves possibe from the initial solution, divided by the number of moves have caused a deterioration of the objective function value. Parameter P_a represents the acceptane 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 is a simple geometric function. The temperature at iteration t, T_t, is obtained from the temperature of the previous iteration as follows:

$$T_t = R.T_{t-1} \tag{4.1.3}$$

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 ∈ S
Select an initial temperature T > 0;
Set temperature change counter t = 0;
Repeat
    Set repetition Counter n = 0;
    Repeat
        Generate state j, a neighbor of i;
        Calculate δ = f(j) − f(i);
        if δ < 0 then i := j;
        else if random(0,1)< exp(−δ/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.

Table 2. SA parameters setting

Parameter	Value
Acceptance fraction	0.5
Cooling rate	0.99

4.2. Tabu search metaheuristic

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 it to one of its current neighbors. Each time a move is performed and linked, the pairs (exam, period) are added to the tabu list that includes inhibited moves. It means that period of this exam can't change until |tabu list|. From a given solution, not all neighbors can usually be reached. A new candidate move in fact brings the solution to its best neighbor, 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 iteration 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.

Table 3. The general Tabu search technique

```
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
```

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]

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 metaheuristic

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 (f.f.)*, which is related to the objective function to optimize. The

f.f. quantities how good a solution is for the problem faced. In GAs individuals are sometimes also called *chromosome*, 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.

Then 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 chromosomes 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:

N	10	50	100	200

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

P_m:

P_m	0.02	0.1	0.5

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

P_c:

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 f.f.  for each individual;
  apply reproduction;
  apply parent selection;
  apply crossover with a probability p_c;
  apply mutation with a probability p_m;
end.
```

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 6. GA parameters setting

Parameter	Value
N	100
P_m	0.02
P_c	0.8

4.4. Ant colony system metaheuristic

Ant Colony algorithms were conceived by Marco Dorigo, Vittotrio Maniezzo and Alberto Colorni [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 (ASC) 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 n
   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 $Z(a_i), i = 1, \ldots, m$ and update $\tau_j, j = 1, \ldots, n$ 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 $i = 0, \ldots, |E|$, where $E_i = \{e_1, \ldots, e_i\}$. An ant starts with the empty assignment $A_0 = \emptyset$. 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:

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

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

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

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

Table 7. The general Ant system algorithm

```
INPUT: A problem instance I
   τ_0 ← 1/ρ
   τ(e,t) ← τ_0 ,   ∀(e,t) ∈ E × T
 WHILE time limit not reached  DO
   FOR a=1 to m  DO
    { construction process of ant a}
    A_0 ← ∅
     FOR i=1 TO |E|  DO
     choose timeslot t randomly according to
     probabilities p_{e_i,t} for exam e_i
     A_i ← A_{i−1} ∪ {(e_i, t)}
     END FOR
    C ←solution
    C_{iteration best} ← best of C and C_{iteration best}
  END FOR
  C_{iteration best} ← solution after applying local search to
  C_{iteration best}
  C_{global best} ← best of C_{iteration best} and C_{global best}
  Global pheromone update for τ using C_{global best}
END WHILE
OUTPUT: An optimized candidate solution C_{global best} for I
```

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

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

Table 8. ACS parameters setting

Parameter	Value
ρ	0.8
α	1
β	0.4
m	n

5. 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 the Table 9.

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

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 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. Example are given in Figure 1.

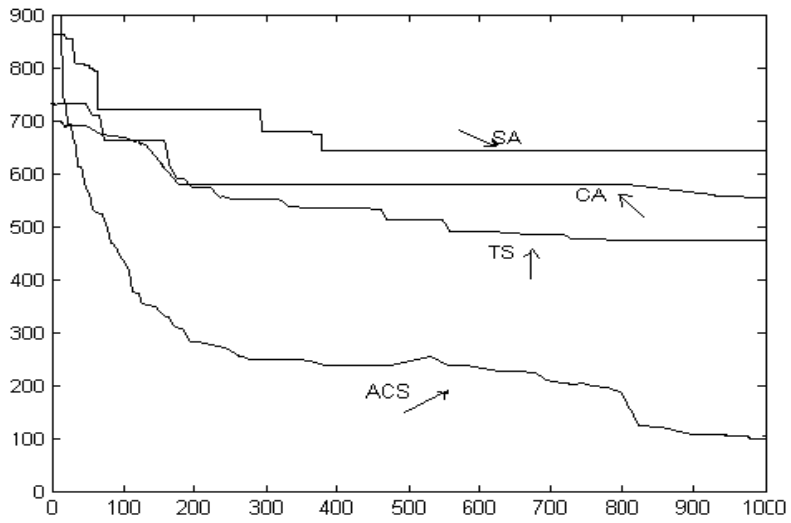

Figure 1

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

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

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

P		Initial	Min 1		Min 2		Min 3		Cost Reduction
S1	Time		50	140		180			
	SA	938.5	881.9	0.47	813.7	1.03	765.5*	1.16	173.0
	TS	877.0	732.1	0.22	532.6*	0.33	532.6	0.50	344.3
	GA	771.8	668.2	0.12	627.4*	0.57	627.4	0.77	144.3
	ACS	811.0	**597.8**		**399.8**		**354.4***		**456.5**
S2	Time		30	120		480			
	SA	1116.4	1100.2	0.05	1038.1	0.14	976.0*	0.25	140.3
	TS	1197.5	1066.8	0.01	**912.3**		835.0*	0.07	362.4
	GA	1226.2	**1051.6**		1016.5	0.1	975.7*	0.25	250.4
	ACS	1222.7	1171.2	0.11	1073.6	0.18	**780.3***		**442.3**
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	956.0	0.03	905.3*	0.02	**459.6**
	GA	1149.3	1072.3	0.10	1072.3	0.16	965.5*	0.09	183.7
	ACS	1139.4	**977.4**		**927.8**		**887.1***		252.2
S4	Time		60	300		540			
	SA	940.2	723.8	0.14	697.2	0.09	676.2*	0.15	**263.9**
	TS	750.8	697.1	0.09	646.0	0.01	**585.7***		165.0
	GA	727.6	**641.2***		**641.2**		641.2	0.09	86.3
	ACS	683.0	667.4	0.04	656.3	0.02	630.3*	0.08	52.7
S5	Time		100	300		540			
	SA	1015.2	994.5	0.10	979.9*	0.09	979.9	0.09	35.2
	TS	1125.5	1001.8	0.11	952.2	0.05	915.0*	0.01	**210.4**
	GA	986.6	**902.4***		**902.4**		**902.4**		84.2
	ACS	1034.3	945.5	0.05	945.5	0.05	919.4*	0.02	114.8
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
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

In Table 10 the symbol '*' indicates which heuristic attains its minimal value after the given run time. The best solution 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.

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

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

As it is shown in the 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. Ability of metaheuristics to find the best solution

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

The values that have been shown in above table achieved from following formula:

$$\frac{\text{The Number os points that algorithm has been found the best solution}}{\text{The Number of all points}} \times 100 \tag{6.2}$$

Since the initial solution of each of the four metaheuristics are different and this effects 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.

Table 12. Ability of metaheuristics to produce maximum cost reduction

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

9. Conclusions

In this paper we considered four metaheuristic (Simulated Annealing, Tabu search, Genetic Algorithm and Ant Colony System) for solving Examination Timetabling Problem and compared the results of them on 10 datasets.

All the algorithms used the same direct representation and were implemented in their basic components in a straightforward manner using a common search landscape for a fair and meaningful analysis.

Since simple comparison of the final solution of the four metaheuristics without taking into account the run time is not appropriate, we used dynamic comparison.

The result showed that ACS and then TS algorithm worked better in compare with others (See Table. 10-12 and Figure 1).

The initial solution of SA, TS and GA are completely random. But initial solution for ACS is constructed based on initial pheromone and heuristic information. Since the amount of initial pheromone is same in all paths, 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 12, TS had the most ability to produce maximum cost reduction in comparison of other metaheuristics. But we still say that ACS works better on these problems. Because as mentioned before, ACS starts with better initial solution than TS and regards to it, it has less reduction in cost of solution. Therefore we can say ACS works better on these problems.

References

1. D. J. A. Welsh and M. B. Powel, *An Upper bound for the Charomatic Number of a Graph and Its Application to Timetabling Problems*, Comp. Jrnl **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 and S. Y. Lee, *Examination Timetabling: Algorithmic Strategies and Applications*, Journal of the Operational Research Society **47** (1996), 373-383.
5. M. Cangalovic and 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, The practic and theory of automatedtimetabling*, Lecture Notes in Computer Science, eds. E. Burke and P. Ross. Springer **1153** (1996), 181-196.
7. A. Colorni, M. Dorigo and V. Maniezzo, *Genetic Algorithms: A New Approach to the Timetabling Problem, The practic and theory of automated timetabling*, Lecture Notes in Computer Science, eds. E. Burke and P. Ross. Springer **1153** (1996), 235-239.
8. W. Erben and J. Keppler, *A genetic Algorithm Solving a Weekly Course-Timetabling Problem, The practic and theory of automated timetabling, Lecture Notes in Computer Science*, eds. E. Burke and P. Ross. Springer **1153** (1996), 198-211.
9. L. Chambers, *Practical hand book of Genetic Algorithms: Applications, CRC*, Press 1999, Vol. 1, Chapter 8. Dave Corne, Peter Ross, Hsiao-Lan Fang, 219-274.
10. L. F. Paquete and C. M. Fonseca, *A study of Examination Timetabling with Multiobjective Evolutionary Algorithms*, MIC 2001-4th mataheuristic International Conference. Porto, Portugal, July 16-20, 2001.
11. A. S. Asratian and D. de Werra, *A generalized class-teacher model fo some timetabling problems*, European Journal of Operational Research **143** (2002), 531-542.
12. O. Rossi-Doria, Ch. Blum, J. Knowels, M. Sampels, K. Socha and 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. Newall and 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 and 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 and K. A. Dowsland, *A Robust Simulated Annealing Based Examination Timetabling System*, Computers and Operations Researchs **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. Di Gaspero and A. Schaerf, *Tabu Search Techniques for Examination Timetabling*, Third International Conference Patat 2000, Lecture Notes in Computer Science **2079** (2000), 104ff.
18. K. Socha, J. Knowles and M. Sampels, *A Max-Min Ant System for the University Course Timetabling Problem*, Proceedings of ANTS 2002-Third International workshop on Ant Algorithms, Lecture Notes in Computer Science, Springer Verlag. Berlin, Germany **2463** (2002), 1-13 (Also Technical Report/TR/IRIDIA/2002-18).

19. J. Wood and D. Whitaker, *Student centred school timetabling*, Journal of the Operational Research Society **49** (1998), 1146-1152.
20. V. 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 and H. Ohta, *Timetable planning using the constraint-based reasoning*, Computer & Operations Research **27** (2000), 819-840.
22. E. K. Burke and S. Petrovic, *Recent research directions in automated timetabling*, European Journal of Operational Research **140** (2002), 266-280.
23. S. Kirckpatric, Jr. C. Gellat and M. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), 671-680.
24. V. Gerny, *A Termodynamical approach to the traveling salseman problem: an efficient simulation algorithm*, Journal of Optimization Theory Application **45** (1985), 41-51.
25. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, *Equation of state calculations by fast computing machines*, Journal of Chemical Physics **21** (1953), 1087-1092.
26. V. P. Laarhoven and E. Aarts, *Simulated Annealing: Theory and Practic*, Dordrecht: Kluwer Academic Publishers, The Netherlands, 1987.
27. E. Aarts and J. Korst, *Simulated annealing and Boltamann machines*, Chichester: Wiley, 1989.
28. N. Collins, R. Eglese and B. Golden, *Simulated annealing an annotated bibliography*, American Journal of Mathematical and Management Science **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 emploee scheduling problem: an integration of management science and artifical intelligence*, Computers and Operation Research **15** (1986), 563-593.
31. F. Glover and M. Laguna, *Tabu search*, Kluwer academic Publishers, 1997.
32. K. E. Rosing, C. S. ReVelle, E. Rolland, D. A. Schilling and 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 and S. E. Haupt, *Practical Genetic Algorithms*, A Wiley-interscience publication, John Wiley and sons, INC., NEW YORK, 1997.
34. M. Dorigo, V. Maniezzo and A. Colorni, *Positive feedback as a search strategy*, Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, IT, 1991.

Zahra Naji Azimi received her BS in Applied Mathematics from Ferdowsi University of Mashhad, Iran, She is a member of Young Research Club of Iran, She works on Optimization problems and Simulation. Her research interests focus on new metaheuristic and heuristic methods in Operations Research, Simulation, Computer Science and Management.

Zahra Naji Azimi, Department of Mathematical Sciences, Ferdowsi University, Mashhad, Iran
E-mail: najiazimi@@ yahoo.com