Yokohama Publishers
ISSN 1880-5221 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 2, No. 2, pp. 341–366, May 2006

INTRODUCTION TO ELECTROMAGNETISM ALGORITHM FOR THE EXAMINATION TIMETABLING PROBLEM AND COMPARISON OF IT WITH OTHER METAHEURISTICS

Majid Salari and Zahra Naji Azimi

Abstract: Electromagnetism (EM) has been recently introduced as a strong method for the optimization of unconstrained continuous functions based on an analogy with electromagnetism theory. Also Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA) and Ant Colony System (ACS) are five of the main algorithms for solving challenging problems of optimization and intelligent systems. In this paper, we represent a new heuristic method based on EM and apply these four techniques to a classical Examination Timetabling Problem (ETP), an NP complete problem. The EM method is applied on this problem for the first time in literature; and all of these methods are tested on ten different scenarios of the classical ETP. Statistical comparative analyses conclude that EM technique is significantly better than each metaheuristic, and furthermore provides the superior solution of all. Finally, we effort to obtain some improvements of EM via changing some parts of it.

Key words: *electromagnetism, timetabling, ant colony system, genetic algorithm, simulated annealing, tabu search*

Mathematics Subject Classification: *90B99, 90C99*

1 Introduction

Proper scheduling of exams is a common problem for all universities and institutions of higher education. Usually solving this problem involves taking the previous year's timetable and modifying it so that it would work for the new year. But changing the number of students, variety of the courses being offered and student's freedom in selecting them requires significant alteration of previous year's timetable. The Examination Timetabling problem regards the scheduling for the exams of a set of university courses, avoiding overlap 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 with no confliction has been shown to be equivalent to assigning colors to vertices in graph so that adjacent vertices always have different colors [37]. This in turn has been proved to lie in the set of NP complete problems [28] 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 coloring as well as many metaheuristic methods such as SA, TS, GA, and ACS. Each of these algorithms has been applied in different frames and there is not any specific method even in a special one in different papers. Also many authors have considered the problem of their university with its related conditions.

In this paper we consider the above metaheuristics and introduce Electromagnetism (EM) method for solving the ETP, and compare the results of them on 10 datasets. In the following sections, we first begin with describing the Examination Timetabling problem in Section 2. In Section 3, we briefly introduce the EM algorithm, and a common structure of the five metaheuristics is provided in Section 4. The five basic metaheuristics and their structure as used in this paper are discussed in Section 5. In Section 6, the structure of the input data set is discussed, and the results of application of the pure metaheuristics are analyzed in Section 7. Section 8 includes the final analysis of the EM method and its improvement and comparison with pure frame of it.

2 Problem Description

Given 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 [22]. Many different constraint types have been proposed in the literature. In this work, we consider the version proposed by Carter et al. [10], 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 modeled 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 [22], 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,

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

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

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

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. After comparing of metaheuristics we will discuss the improvement ways of EM in Section 8.

3 Electromagnetism

Birbil and Fang [9] have introduced the electromagnetism metaheuristic algorithm for special class of optimization problems with bounded variables in the form of:

$$\begin{aligned}&\text{Min } f(x)\\ &\text{s.t. } x \in [l,u]\end{aligned}$$

Where $[l,u] = \{x \in \Re^n | l_k \leq x_k \leq u_k, k = 1,...,n\}$ and n is the dimension of the solution space. Similar to electromagnetism theory, each x, representing a solution, is released to a space as a charged particle and the charge of each point is related to the objective function value. The charge also determines the magnitude of attraction or repulsion of the point over the sample population. To encourage the points to converge to the highly attractive valleys, solutions with the better objective function values give the higher magnitude of attraction.

After calculating charges, they are used to find a direction to move and this direction is determined by evaluating a combination force exerted on the point via other points. According to the Coulomb's Low the magnitude of the force between two points is proportional to the product of the charges and inversely proportional to the distance between the points [4].

In the electromagnetism that Birbil and Fang introduced, n represents the dimension of the problem and u_k, l_k and $f(x)$ represent upper and lower bound in the k^{th} dimension and objective function respectively. Although the pure EM algorithm is introduced for the special class of optimization problems, in Section 5.1 we input our modification of this algorithm for our problem.

4 Common Structure

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

4.1 Search Landscape

Since a good solution may be in the neighborhood 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 neighborhood. This results in a continuous but not smooth search space.

4.2 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.

4.3 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.

Table 1. The general Electromagnetism technique

```
Algorithm EM ($MAXITER$, $LSITER$)
  initialize
  $iteration$ :=1
   **While** $iteration < MAXITER$ do
    Local($LSITER$)
    calculate forces
    move
    $iteration$ := $iteration$+1;
  **End while**
```

$\boxed{4.4}$ 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.

$\boxed{5}$ Description of the Heuristics

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

$\boxed{5.1}$ Algorithm

The main frame of EM is as follows:

After producing the initial population in initialize, the function local explores the immediate (Euclidian) neighborhood of individual points [16]. To gather the local information for a point we use local search. We select a bit from solution randomly and change its value to a random number. If this new point observes a better objective function, the point is replaced by the new one and otherwise the procedure is repeated until a better point is found up to $LSITER$ times. This simple random search is applied point by point and does not require any gradient information to perform the local search.

The total force exerted on each point by all other points is calculated in the function $calculateforces$. According to the electromagnetism theory the force exerted on a point via other points is proportional to the product of their charges and inversely proportional to the distance between them [4].

The charges of the points are calculated according to their objective function value in each iteration. The charge of each point x^i is determined by its objective function value $f(x^i)$ in relation to the current best point x^{best} in the population as:

$$q^i = \exp\Big(-n\frac{f(x^i)-f(x^{best})}{\sum_{k=1}^{m}\left(f(x^k)-f(x^{best})\right)}\Big), \quad \forall i. \tag{1}$$

Where m represents the population size and n is the dimension of the solution space. In the EM algorithm no signs are attached to the charge of an individual point [9]. Subsequently, the total force exerted on each point is determined:

$$F^i = \sum_{j\neq i}^{m} \left\{ \begin{array}{lll} (x^j - x^i)\frac{q^i q^j}{\|x^j - x^i\|^2} & if & f(x^j) < f(x^i) \\ (x^i - x^j)\frac{q^i q^j}{\|x^j - x^i\|^2} & if & f(x^j) \geq f(x^i) \end{array} \right\}, \quad \forall i \tag{2}$$

Where in this paper we consider $||x^j - x^i||$ as:

$$\|x^j - x^i\| = \big(\sum_{k=1}^{n} (x_k^j - x_k^i)^2\big)^{1/2} \tag{3}$$

A point with a relatively good objective function value attracts the other ones; points with inferior objective value repel the other population members [15].

After evaluating the total force vector F^i, the point i is moved in the direction of the force by a random step length λ that is assumed to be uniformly distributed between 0 and 1 [9]. Also the force exerted on each particle is normalized and therefore we can calculate the new solution as a new point by this formula:

$$x^i = \left\{ \begin{array}{lll} x^i + round(\lambda\frac{F^i}{\|F^i\|}(np - x_k^i)) & if & F_k^i > 0 \\ x^i + round(\lambda\frac{F^i}{\|F^i\|}(x_k^i - 1)) & if & F_k^i \leq 0 \end{array} \right\}, \quad i = 1, 2, ..., m \tag{4}$$

Where np is the maximum period number that is available to select for scheduling. We have to round the value because our variables are discontinuous in this problem. When we round this amount, we must consider that the new point is different from previous one; otherwise we can use another step length or apply a mutation to move it to a near neighbor of this solution.

5.1.1 Parameters

In this algorithm, $MAXITER$ defines the number of iterations, and $LSITER$ defines the number of iterations in a local sub procedure local search. Because of full discussion of these parameters in Section 8, we do not discuss them here.

5.2 Simulated Annealing Metaheuristic

The principle of the SA metaheuristic is deduced from the physical annealing process of solids. Kirckpatrick et al. [27] and Cerny [11] proposed the use of SA for combinatorial problems. Their work is based on the research of Metropolis et al. [29] in the field of Statistical Mechanics. For an overview of the research and applications of SA, the reader is referred to Vanlaarhoven and Arts [36], Aarts and Karts [1], Collins et al. [12] and Eglese [19].

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:

$$P_a = e^{-\Delta C / T_{\text{begin}}} \tag{5}$$

and hence

$$T_{\text{begin}} = \frac{-\Delta C}{\ln P_a} \tag{6}$$

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

Table 2. 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 neighbor 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.
```

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:

$$T_t = R.T_{t-1} \tag{7}$$

where, R represents the cooling rate.

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

$\boxed{5.2.1}$ Algorithm

A general description of SA is given in Table 2.

$\boxed{5.2.2}$ 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

Cooling rate (R):

This is the fraction by which the temperature is reduced in the geometric temperature function (5.2.3).

Values assigned to this parameter are:

R	0.70	0.80	0.90	0.99

Table 3. SA parameters setting

Parameter	**Value**
Acceptance fraction	0.5
Cooling rate	0.99

Among above values, the best pair of parameters pair is reported in Table 3.

5.3 Tabu Search Metaheuristic

Tabu search was conceived by Glover [23]. 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 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, period of this exam can not change until the length of 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 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 [24, 32].

5.3.1 Algorithm

A general description of TS is given in Table 4.

5.3.2 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

Table 4. 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
```

Table 5. TS parameters setting

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

Among above combination of parameter settings, the best parameters setting achieved for TS is reported in Table 5.

$\boxed{5.4}$ Genetic Algorithm Metaheuristic

Genetic Algorithm was conceived by Holland [25]. 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

Table 6. 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.
```

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 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.

$\boxed{5.4.1}$ Algorithm

A general description of GA is given in Table 6.

Table 7. GA parameters setting

Parameter	**Value**
N	100
P_m	0.02
P_c	0.8

5.4.2 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 7.

5.5 Ant Colony System Metaheuristic

Ant Colony algorithms were conceived by Marco Dorigo, Vittotrio Maniezzo and Alberto Colorni [18]. 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.

1. Evaluate $Z(a_i), i = 1, \ldots, m$ and update $\tau_j, j = 1, \ldots, n$ accordingly.
2. 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:

$$p_{e_i,t}(\tau(A_{i-1}),\eta(A_{i-1})) = \frac{(\tau_{(e_i,t)}(A_{i-1}))^{\alpha}.(\eta_{(e_i,t)}(A_{i-1}))^{\beta}}{\sum\limits_{\theta\in T}(\tau_{(e_i,\theta)}(A_{i-1}))^{\alpha}.(\eta_{(e_i,\theta)}(A_{i-1}))^{\beta}} \tag{8}$$

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:

$$\eta_{(e,t)}(A_{i-1}) = \frac{1.0}{1.0 + V_{(e,t)}(A_{i-1})} \tag{9}$$

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:

$$\tau_{(e,t)} = \begin{cases} (1-\rho).\tau_{(e,t)} + 1 & \text{if } A_{global\ best}(e) = t \\ (1-\rho).\tau_{(e,t)} & \text{otherwise} \end{cases} \tag{10}$$

5.5.1 Algorithm

A general description of ACS is given in Table 8.

Table 8. The general Ant System algorithm

```
Input: A problem instance $I$
  $\tau_0 \leftarrow \frac{1}{\rho}$
  $\tau(e,t) \leftarrow \tau_0 \quad \forall(e,t) \in E \times T$
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
choose timeslot t randomly according to probabilities $p_{e_i,t}$ for exam $e_i$
    $A_i \leftarrow A_{i-1} \cup \{(e_i,t)\}$
    end for
    $C \leftarrow$solution
    $C_{iteration\ best} \leftarrow$ best of $C$ and $C_{iteration\ best}$
  end for
  $C_{iteration\ best} \leftarrow$ solution after applying local search to $C_{iteration\ best}$
  $C_{global\ best} \leftarrow$ best of $C_{iteration\ best}$ and $C_{global\ best}$
  Global pheromone update for $\tau$ using $C_{global\ best}$
End while
Output: An optimized candidate solution $C_{global\ best}$ for $I$
```

5.5.2 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

The best parameters setting achieved for ACS are reported in Table 9.

Table 9. ACS parameters setting

Parameter	**Value**
ρ	0.8
α	1
β	0.4
m	n

6 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 the Table 10.

7 Heuristic Analysis

Due to the fact that the stopping criterion of the metaheuristics is not similar, a simple comparison of only the final solution values of the five 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 five metaheuristics without taking into account the run time is not appropriate.

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. Because of probabilistic nature of these algorithms, through this paper we run the algorithms 10 times in each case, and finally record the average results.

Table 10. 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

In Table 11 The symbol '*' indicates which heuristic attains its minimal value after the given run time. The best solutions of the five 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 point of time.

$$\text{relative difference(solution)} = \frac{\text{cost of solution-cost of best solution}}{\text{cost of best solution}} \tag{11}$$

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 11, we gain the best solution from EM in all time points and therefore, EM makes the first grade and all of the others have second grade in opposite of EM. It is better to mention that all of these algorithms are considered in basic version and no special improvement is considered for EM or each of them. Although there are some amazing results of GA, TS, SA or ACS in literature and they can act very good on this problem, but we want to compare them in equal conditions and with basic version. It is obvious that each of them can be better with little improvement but we want to have meaningful comparison. Since the initial solutions of each of the five metaheuristics are different and this affects quality of the final solution, we calculate amount of cost reduction for all of them and in comparison of all, again EM has the highest reduction in the cost of solutions at the same time.

$\boxed{8}$ Improvements of EM

In this section we consider the effect of some components and parameters of EM algorithm and after that add some concept and details to it and examine their affect on algorithm improvements.

Table 11. Heuristic analysis of ACS, GA, SA, TS and EM

P		Initial	Min 1		Min 2		Min 3		Cost Reduction
S1	Time		50		140		180		
	SA	1126.4	604.4750*	1.9	604.4750	1.9	604.4750	1.9	521.925
	TS	1184.3	263.7825	0.3	243.2350	0.2	242.7550*	0.2	**941.5453**
	GA	705.0728	526.9488	1.6	519.7138*	1.5	519.7183	1.5	185.359
	ACS	1055.2	245.6113	0.2	224.4350**	0.1	224.4350	0.1	830.765
	EM	985.1850	**202.2413***		**202.2413**		**202.2413**		782.9437
S2	Time		30		120		480		
	SA	1375.1	961.9693	1.3	961.9693	1.4	897.6829*	1.2	477.4171
	TS	1223.9	634.1629	.5	466.5271	.2	432.9636*	0.1	**790.9364**
	GA	1001.7	888.3821	1.1	847.4529	1.1	833.5079*	1.1	168.1921
	ACS	1112.6	613.4729	0.5	505.5036	0.3	462.9214*	.2	649.6786
	EM	1181.5	**418.345**		**398.7993***		**398.7993**		782.7007
S3	Time		60		300		540		
	SA	1250.7	1011	1.1	979.3521	1.5	923.8658*	1.4	326.8342
	TS	1258	621.9537	0.3	456.2047	0.2	441.0632*	0.1	816.9368
	GA	1028.4	932.1637	1	883.8426	1.3	879.3311*	1.3	149.0689
	ACS	1242.1	977.6058	1.1	650.8363	.7	541.7026*	.4	700.3974
	EM	1321.4	**470.5547**		**383.8926***		**383.8926**		**937.5074**
S4	Time		60		300		540		
	SA	906.8011	584.9516*	1	584.9516	1.7	584.9516	1.9	321.8495
	TS	801.9888	398.4474	.3	304.5884	0.4	286.5772*	.4	515.4116
	GA	648.0547	557.6849	.9	514.2979	1.4	491.8446*	1.5	156.2101
	ACS	706.0677	655.2456	1.2	552.2968	1.5	483.93543*	1.4	221.1323
	EM	719.2172	**297.0874**		**215.4154**		**198.6611***		**520.5561**
S5	Time		100		300		540		
	SA	1644.8	1308.6	1.6	1284.8*	2	1284.8	2.2	360
	TS	946.4561	618.4233	.2	565.1306	.3	532.8622*	.3	413.5939
	GA	888.2081	842.5875	.7	826.4558	.9	798.6875*	1	89.5206
	ACS	945.0608	863.8731	0.7	794.3214	0.8	691.3533*	.7	253.7075
	EM	964.5678	**502.7128**		**429.4836**		**406.2528***		**558.315**
S6	Time		300		600		960		
	SA	1473.5	1239*	2	1239	2.8	1239	3.2	234.5
	TS	1827.2	639.0628	.6	541.3983	.6	518.5209*	.7	1308.6791
	GA	1295.6	1203.6	1.9	1168*	2.5	1168	3	127.6
	ACS	1393.3	1234.3	2	1162.1	2.5	1040.4*	2.5	352.6381
	EM	1458.7	**409.6305**		**328.5246**		**295.8062**		**1162.8938**
S7	Time		350		700		1020		
	SA	1491.3	1267.3*	1.8	1267.3	2.4	1267.3	2.9	224
	TS	1647.4	715.271	.6	621.9535	.7	572.389*	.8	1075.011
	GA	1320.3	1232.4	1.8	1221.1	2.3	11.76*	2.6	144.3
	ACS	1421.6	1314.4	2	1278.9	2.5	1259.1*	2.9	162.3
	EM	1510.6	**442.0465**		**368.4504**		**321.2431***		**1189.3569**

S8	Time		380		800		1200		
	SA	1214.2	1075.3	2.7	1068.1	4	1051.6*	4.5	162.6
	TS	1244.4	503.9279	.8	438.0178	1	390.8712*	1	853.5288
	GA	1063.1	975.125	2.4	975.125	3.5	960.5668∗	4	102.5332
	ACS	1135.3	1084.3	2.7	1015.1	3.7	984.50∗	4.1	150.80
	EM	1146.1	**286.03**		**214.0618**		**190.8651∗**		**955.2349**
S9	Time		470		670		1200		
	SA	1402.4	1250.5*	1.5	1250.5	1.9	1250.5	2.6	151.9
	TS	1513	702.3457	0.4	661.6787	.5	577.3558*	.7	935.6442
	GA	1406.5	1097.6	1.2	1065.7	1.5	960.5668∗	1.8	445.9332
	ACS	1381.9	1110.7	1.2	1041.6	1.4	927.7∗	1.4	454.2
	EM	1403.8	**489.5450**		**427.7425**		**344.5035∗**		**1059.2965**
S10	Time		300		640		1200		
	SA	1572.1	1422	1.4	1361∗	1.6	1361	2.1	211.1
	TS	1537.1	874.4039	.5	790.3174	.5	662.2463*	.5	874.8537
	GA	1373.4	1298.2	1.2	1262.4	1.4	1255.2*	1.9	118.2
	ACS	1250.9	1245.6*	1.09	1244.3	1.3	1242.6	1.8	8.3
	EM	1495.6	**593.4014**		**523.4703**		**436.0824∗**		**1059.5176**

8.1 Existence of Local Search

As a first step, we consider the necessity of Local Search procedure. We first run the algorithm without any local search and then we apply it on all of points in population. We test it because if we can exclude $local search$, we can obtain solution in a shorter time. Our results of these runs are as follows:

Table 13. Ability of EM (with and without local search) to find a better solution

Algorithm	**EM with Local Search**	**EM without Local Search**
Ability to find the best solution	%100	%0

Table 14. Ability of EM (with and without local search) to produce maximum cost reduction

Algorithm	**EM with Local Search**	**EM without Local Search**
Ability to produce maximum cost reduction	%100	%0

As it is shown in Table.13 and 14, applying EM with Local Search produce the better solution and the more cost reduction in all datasets although it takes more CPU time.

8.2 How to apply Local Search?

In the previous test we found that it is better to consider Local Search, but how do we apply this procedure? First we consider local search just for the best point. If we can use this version and gain the better solution, we can save our time. On the other hand we may think that it is better to improve the quality of the worst point in population in order to have the more even solutions. This version approximately is equal in CPU time to previous one.

Table 12. EM with local search and without Local Search

		Number of iterations	EM with Local Search			EM without Local Search		
			Best solution	Cost reduction	CPU Time	Best solution	Cost reduction	CPU Time
Simple	S1	100	**246.6400**	**616.2238**	3.5	862.8637	0	.4850
	S2	100	**566.5007**	**522.6129**	6.3750	1155.9	0.5271	0.9060
	S3	100	**609.8279**	**612.2911**	10.3440	1180.3	0	1.3600
Moderate	S4	200	**308.6712.6**	**419.7863**	33.8440	733.1053	0	4.0160
	S5	200	**548.8047**	**483.6297**	50.1250	1005.5	0	5.6090
	S6	200	**653.2841**	**751.8442**	68.1880	1383	87.3098	7.6720
	S7	200	**693.6721**	844.0179	75.2650	1527.8	0	8.7970
Difficult	S8	350	**48.9323**	**814.0251**	155.5310	1246.7	0	17.5310
	S9	350	**524.9735**	**846.9679**	217.0310	1459.5	0	22.0930
	S10	350	**631.9817**	**857.9903**	245.6710	1544.8	0	28.3290

Table 16. LS for the 40% of better and worse points and for all points

		Number of iterations	LS for the 40% of Worse points			LS for the 40% of better points			LS for all points		
			Best solution	Cost reduction	CPU	Best solution	Cost reduction	CPU	Best solution	Cost reduction	CPU
Simple	S1	100	862.8	0	1.5	247.1	**704.8**	1.3	**246.6**	616.2	3.5
	S2	100	1205.6	0	2.7	**536.9**	**613.4**	2.4	566.5	522.6	6.3
	S3	100	1091.9	0	5.1	598.1	**694.0**	3.9	609.8	612.2	10.3
Moderate	S4	200	681.9	0	15.5	314.5	**424.3**	13	308.6	419.7	33.8
	S5	200	469.4	0	21.8	**525.3**	470.2	18.9	548.8	**483.6**	50.1
	S6	200	1357.1	0	30.3	**628.7**	785.3	24.7	653.2	751.8	68.1
	S7	200	1380.1	0	34.7	667.1	722.2	28.3	693.6	**844**	75.2
Difficult	S8	350	1183.1	0.1	71.9	362.6	**844.7**	60	**48.9**	**814.7**	155.5
	S9	350	1421.7	0	87	544	**913.0**	83.8	**524.9**	846.9	217
	S10	350	1447.8	0	120.4	**630.6**	**891.9**	131.9	631.9	857.9	245.6

Also if we improve more points, obviously we can obtain better solutions and so we apply Local Search to 40% of the better and 40% of the worse points in two different versions as follows:

Table 15. LS for the best and for the worst point

		Number of	LS for the best point			LS for the worst point		
		iterations	Best solution	Cost reduction	CPU Time	Best solution	Cost reduction	CPU Time
Simple	S1	100	269.0	593.8	.7	502.6	396.8	.7
	S2	100	546.3	519.6	1.2	1173.1	0	1.6
	S3	100	597.9	572.5	6.4	**574.4**	607.6	.2
Moderate	S4	200	**303.4**	396.1	5.6	653.9	0	8.6
	S5	200	546.34	378.3	7.8	966.4	0	11.4
	S6	200	685.6	**864.4**	9.2	1366.1	0	15.7
	S7	200	**659.6**	705.1	11	399.1	113.7	17.1
Difficult	S8	350	348.6	874.6	25.2	1223	5.2	35.8
	S9	350	552.3	782.6	37.7	569.3	796.9	42.4
	S10	350	691.6	849.4	52.0	1548.8	0	76.6

As it is shown in Table.17 and 18, when we apply Local Search on 40 percent of better points, we can obtain the best solution and maximum cost reduction in most datasets.

Table 17. Ability of EM with LS (on 40% of better and worse points and on all points) to find the better solution

Algorithm	LS for the best point	LS for the worst point	LS for the 40% of Worse points	LS for the 40% of better points	LS for all points
Ability to find the best solution	%20	%10	%0	%40	%30

When we apply LS on all points, in fact we waste some time to improve the worse points and as it is shown in above tables, we can not obtain good solutions with improving these points.

$\boxed{8.3}$ Analyzing LSITER and Population Size

In the next step, we want to know when we apply Local Search, how much we can try to find the better solution with consider to have reasonable time. On the other hand population size is an important factor to reach a good solution in a given time, so the analyzing *LSITER* value without considering population size (M) is meaningless. Therefore we have the following table:

Table 18. Ability of EM with LS (on 40% of better and worse points and on all points) to produce maximum cost reduction

Algorithm	LS for the best point	LS for the worst point	LS for the 40% of Worse points	LS for the 40% of better points	LS for all points
Ability to produce maximum cost reduction	%10	%0	%0	%70	%20

Table 19. Analyzing of *LSITER* and Population Size (M)

			Number of iterations	*LSITER*=5			*LSITER*=30		
				Best solution	Cost reduction	CPU time	Best solution	Cost reduction	CPU time
Simple	S1	M=5	50	411.4	451.3	1	303.4	616.1	4.2
		M=10	50	405.0	**368.9**	1.6	258.8	482.7	8.4
		M=20	50	345.6	**325.3**	3.4	283.38	519.7	16.9
		M=30	50	382.4	458.7	5.0	290.0	**546.6**	26.2
		M=45	50	339.7	454.2	7.7	324.5	391.5	38.5
	S2	M=5	50	726.0	408.5	1.7	645.1	506.5	7.8
		M=10	50	685.9	374.0	3.3	624.9	561.0	18.4
		M=20	50	731.4	414.4	6.7	585.1	490.2	36.7
		M=30	50	700.2	365.8	9.9	568.16	468.7	51.1
		M=45	50	734.8	342.3	14.5	570.2	446.8	45.1
	S3	M=5	50	738.5	**504.5**	2.8	**681.9**	489.9	14.7
		M=10	50	880.8	326.6	5.9	**635.7**	**479.0**	33.1
		M=20	50	**704.0**	372.9	11.9	718.3	**445.4**	64.6
		M=30	50	783.3	286.9	17.7	**672.3**	422.5	98.1
		M=45	50	745.2	400.5	26.9	664.8	457.3	140.8
Moderate	S4	M=5	75	470.3	257.4	6.3	**385.5**	367.7	32.8
		M=10	75	475.2	224.2	14.1	**345.5**	310.3	79.3
		M=20	75	435.6	243.0	28.7	390.2	304.8	159.8
		M=30	75	434.2	234.8	43.4	383.8	287.6	241.6
		M=45	75	423.4	237.3	64.1	402.1	272.4	365.9
	S5	M=5	75	732.8	250.8	9.6	**662.3**	272.0	51.6
		M=10	75	740.1	156.6	20.0	692.3	273.9	11.9
		M=20	75	743.7	219.0	40.2	681.4	**304.2**	215.2
		M=30	75	699.8	233.6	58.2	**642.3**	**291.4**	326.8
		M=45	75	700.5	238.5	0.86	670.7	274.5	471.3
	S6	M=5	75	1008	420.2	13.2	960.8	482.2	70.4
		M=10	75	940.6	410.6	27.4	917.8	**335.6**	151.8
		M=20	75	1003.2	428.7	55.1	954.3	**479.5**	308.8
		M=30	75	929.1	364.9	82.5	**866.0**	**474.1**	464.1
		M=45	75	925.0	441.4	127.8	908.1	399.8	689.0
	S7	M=5	75	1111	406.3	15.2	1006.7	475.9	79.1
		M=10	75	1067	396.7	31.4	**929.7**	**569.9**	172.7
		M=20	75	1026	405.3	62.9	**918.2**	**517.1**	362
		M=30	75	978.5	407.9	95.1	**923.9**	453.3	539.2
		M=45	75	1001.8	387.6	139.3	958.3	447.0	797.4
Difficult	S8	M=5	100	709.0	475.0	22.3	736.1	541.1	116.6
		M=10	100	755.9	429.8	47.7	672.1	481.8	266.1
		M=20	100	758.6	438.9	99.0	615.8	**561.1**	562.5
		M=30	100	716.9	489.1	148.6	**582.0**	521.0	859.2
		M=45	100	705.5	393.6	222.0	636.9	**546.9**	1275.7
	S9	M=5	100	1001.5	442.7	29.2	983.2	**490.1**	152.4
		M=10	100	953.3	423.2	61.0	**905.7**	450.5	340.1
		M=20	100	976.8	349.4	124.6	886.2	500.4	713.8
		M=30	100	923.3	427.2	186.6	911.7	468.0	1071.5
		M=45	100	551.8	373.0	278.2	953.6	397.6	1585.5
	S10	M=5	100	1134.5	408.8	37.1	**1018.1**	468.1	186.62
		M=10	100	1094.3	468.2	75.9	1055.6	**505.4**	427.2
		M=20	100	1046.6	365.1	158.8	1021.8	446.0	904.3
		M=30	100	1077.1	442.5	233.1	998.5	457.0	1357.4
		M=45	100	1102.3	354.4	350.3	980.3	454.6	1968.7

			Number of iterations	*LSITER*=80			*LSITER*=150		
				Best solution	Cost reduction	CPU time	Best solution	Cost reduction	CPU time
Simple	S1	M=5	50	323.3	604.6	10.1	247.6	611.7	17.8
		M=10	50	**228.7**	**605.4**	22.0	256.2	555.5	40.3
		M=20	50	260.2	**520.2**	45.2	**236.2**	506.6	84.7
		M=30	50	**237.8**	443.6	67.0	269.1	518.4	128.2
		M=45	50	235.2	465.8	98.9	**220.2**	**526.6**	186.3
	S2	M=5	50	**639.9**	543.9	22.2	672.0	**582.7**	35.9
		M=10	50	620.2	540.9	48.3	**562.5**	**609.1**	88.7
		M=20	50	**548.6**	**574.9**	94.0	606.2	448.4	172.9
		M=30	50	**512.8**	557.3	121.9	553.6	**561.8**	237.4
		M=45	50	**530.9**	473.1	180.8	596.6	**527.6**	353.3
	S3	M=5	50	753.3	397.6	38.8	696.3	461.9	61.7
		M=5	50	323.3	604.6	10.1	247.6	611.7	17.8
		M=20	50	709.3	437.1	170.4	708.3	416.7	309.8
		M=30	50	676.9	**452.8**	258.1	695.1	405.2	478.5
		M=45	50	**643.6**	468.3	358.2	**647.8**	**481.9**	670.7
Moderate	S4	M=5	75	389.3	**389.6**	91.2	393.6	302.6	158.3
		M=10	75	404.4	263.4	205.2	396.4	**311.7**	369.0
		M=20	75	386.3	**315.3**	428.2	**354.2**	275.7	800.2
		M=30	75	**371.5**	**310.0**	638.3	378.3	306.6	1202.3
		M=45	75	385.5	256.4	945.6	**368.1**	**277.4**	1751.5
	S5	M=5	75	694.2	290.4	133.9	**638.3**	**332.5**	248.1
		M=10	75	683.6	248.1	285.7	**668.1**	**294.3**	533.2
		M=20	75	**648.6**	296.6	589.1	660.0	282.5	1118.8
		M=30	75	645.8	286.6	864.6	670.2	258.0	1563.8
		M=45	75	**623.5**	**316.4**	1303.3	659.2	291.8	2209.8
	S6	M=5	75	**921.0**	**520.9**	181.2	946.3	470.8	335.1
		M=10	75	937.8	484.0	384.3	**888.7**	518.0	739.7
		M=20	75	906.1	438.6	805.4	**901.5**	476.5	1356.9
		M=30	75	897.9	427.1	1190.8	918.1	426.1	2314.8
		M=45	75	920.2	**455.5**	1737.6	**870.8**	437.8	3139.2
	S7	M=5	75	**973.1**	**565.4**	201.3	1049.8	419.5	393.0
		M=10	75	100.4	491.2	459.1	1037.5	456.3.1	864.7
		M=20	75	950.6	478.9	970.4	954.8	431.3	1737.5
		M=30	75	945.4	**511.9**	1438	956.5	492.3	2574.9
		M=45	75	976.9	433.8	1980.1	**948.3**	**493.1**	3709.5
Difficult	S8	M=5	100	689.9	543.7	313.3	**647.7**	**627.6**	527.9
		M=10	100	677.3	541.6	717.4	**632.3**	**568.4**	1314.3
		M=20	100	**592.5**	496.3	1486.7	672.0	487.3	2726.4
		M=30	100	690.1	449.3	2264.8	637.9	469.9	4229.7
		M=45	100	652.6	519.5	3391.4	**629.4**	475.5	6335.1
	S9	M=5	100	**949.7**	472.7	410.9	960.5	455.2	760.6
		M=10	100	965.3	487.6	892.0	942.9	**537.9**	1697.6
		M=20	100	**835.7**	**549.1**	1852.8	840.9	531.9	3505.8
		M=30	100	**832.9**	489.3	2748.7	849.3	**511.9**	5311.3
		M=45	100	887.5	446.6	4112	**847.2**	**498.6**	7944.6
	S10	M=5	100	1042.2	419.1	511.8	1058.1	**493.7**	937.4
		M=10	100	1030.6	504.8	1121.1	**1009.5**	478.1	2113.8
		M=20	100	**1019.4**	464.3	2363.7	1051.6	**470.1**	4334.8
		M=30	100	**990.7**	449.1	3541.4	1053.6	**471.1**	6580
		M=45	100	1049.1	415.6	5259.5	**964.8**	**472.0**	9717.9

Table 20. Analyzing of *LSITER* value in all of datasets with any M in order to find the best solution

Algorithm	$LSITER$ =5	$LSITER$ =30	$LSITER$ =80	$LSITER$ =150
Ability to find the best solution	%2	%24	%36	%38

Table 21. Analyzing of *LSITER* value in all of datasets with any M in order to produce maximum cost reduction

Algorithm	$LSITER$ =5	$LSITER$ =30	$LSITER$ =80	$LSITER$ =150
Ability to produce maximum Cost reduction	%2	%32	%26	%40

In all of datasets and without distinction of M, 150 is the best amount of *LSITER* to obtain the best solution and to produce maximum cost reduction. In below table, we show results in each M against *LSITER* separately, and best solution and cost reduction columns show the number of times that algorithm has found the best result from S1 to S10 for each M and *LSITER*.

8.4 EM with Mutation Procedure

To improve overall results of EM, we add some components to it. We consider mutation sub procedure to increase diversity of population. For this aim, in each execution of main loop, we apply mutation procedure. In this procedure we select a solution randomly and change some of its elements, according to mutation rate, to another random number. We consider this improvement with some mutation rate to find the best rate as follows:

As we can see in below tables the rates 40% and 10% are competitive to reach to the best solution and to produce maximum cost reduction.

Table 24. Ability to find the best solution in respect of mutation rate.

Algorithm	$\mu = 10\%$	$\mu = 20\%$	$\mu = 30\%$	$\mu = 40\%$
Ability to find the best solution	%30	%20	%10	%40

Table 25. Ability to produce maximum cost reduction in respect of mutation rate.

Algorithm	$\mu = 10\%$	$\mu = 20\%$	$\mu = 30\%$	$\mu = 40\%$
Ability to produce maximum Cost reduction	%40	%30	%10	%20

8.5 EM with Reinitializing

In order to prevent fast convergence, when we come close to final solution and algorithm can not find any better solution in some consecutive runs, we save this solution before converging

Table 22. Analyzing of *LSITER* value against M in number of times to find the best result

	LSITER=5		*LSITER*=30		*LSITER*=80		*LSITER*=150	
	Best solution	Cost reduction	Best solution	Cost reduction	Best solution	Cost reduction	Best solution	Cost reduction
M=5	0	1	3	2	**4**	3	3	**5**
M=10	0	0	3	**4**	2	1	**5**	**4**
M=20	1	0	1	**5**	**5**	4	3	1
M=30	0	0	**5**	**4**	**5**	**4**	0	2
M=45	0	0	0	1	4	2	**6**	**7**
Total	1	1	12	16	20	14	17	19

Table 23. Mutation rate to apply mutation procedure

		Number of iterations	$\mu = 10\%$			$\mu = 20\%$			$\mu = 30\%$			$\mu = 40\%$		
			Best solution	Cost reduction	CPU	Best solution	Cost reduction	CPU	Best solution	Cost reduction	CPU	Best solution	Cost reduction	CPU
Simple	S1	250	240.3	7.8	676.7	166.4	7.6	684.1	358.5	8.0	233.1	226.4	628.5	711
	S2	300	379.8	710.6	443	375.7	681.5	452	383.8	875.7	344	377.2	864.7	362
	S3	300	372.0	862.6	272	322.2	872.0	275	461.7	740.6	191	365.3	769.5	229
Moderate	S4	400	222.6	492.1	204	224.7	465.4	210	357.8	443.7	168	195.7	485.1	223
	S5	400	513.6	437.4	148	550.5	401.7	127	680.9	220.2	119	483.8	474.9	162
	S6	400	872.8	658.8	94	839.4	644.5	97	1032	405.5	90	652.1	749.4	127
	S7	400	989.6	524.5	84	892.9	588.9	85	1112	344.2	80	783.7	687.0	108
Difficult	S8	550	684.5	522.1	99	627.3	600.3	117	893.7	361.7	92	544.2	718.9	141
	S9	550	985.4	484.5	89	945.3	422.0	87	1147.7	262.3	74	977.0	419.1	95
	S10	550	1191.5	383.4	75	1190.8	384.1	72	1284.3	181.4	60	1464.8	83.4	59

and reinitialize the population. The new population consists $M-1$ new points that are randomly produced and one high quality point from previous execution. So we increase the diversity of population when we may be near local optimum. We do this process until we reach to predefined CPU time. The results of this algorithm are shown in Table 26.

8.6 EM with Multipopulation

In order to search solution space in a wide range and to gain more freedom, in this method we produce multi random populations and then select the best one among each of them. We repeat this procedure until we construct whole population. In this way we use quality and diversion together in the initial population. The results of applying this algorithm are shown in Table 26.

8.7 Parallel EM

One of the improvement ways that we consider, is applying parallel runs of EM and before converging, we stop all of them and select the best solution of each. These new and high quality points construct the new population and algorithm continues with a collection of high quality initial points. The results of this algorithm are shown in Table 26.

Table 26. Comparison of different EM algorithms

		CPU	EM with reinitializing			EM with multi population			Parallel EM			EM with mutation			EM with local search		
			5 times			5 population			5 Parallel EM			Best rate			Best parameters		
			Best solution	Cost reduction	NIter NIter	Best solution	Cost reduction	NIter NIter	Best solution	Cost reduction	NIter NIter	Best solution	Cost reduction	NIter NIter	Best solution	Cost reduction	NIter NIter
	S1	300	199.8	6662.9	799	195.4	478.3	784	211.2	620.3	723	226.4	628.5	711	199.8	663.0	577
Simple	S2	300	379.8	710.6	443	375.7	681.5	452	383.8	875.7	344	377.2	864.7	362	429.3	780.3	304
	S3	300	372.0	862.6	272	322.2	872.0	275	461.7	740.6	191	365.3	769.5	229	339.4	864.2	199
	S4	400	222.6	492.1	204	224.7	465.4	210	357.8	443.7	168	195.7	485.1	223	251.4	457.0	177
Moderate	S5	400	513.6	437.4	148	550.5	401.7	127	680.9	220.2	119	483.8	474.9	162	555.6	480.1	138
	S6	400	872.8	658.8	94	839.4	644.5	97	1032	405.5	90	652.1	749.4	127	897.8	572.4	87
	S7	400	989.6	524.5	84	892.9	588.9	85	1112	344.2	80	783.7	687.0	108	985.1	509.3	76
	S8	550	684.5	522.1	99	627.3	600.3	117	893.7	361.7	92	544.2	718.9	141	691.4	492.9	99
	S9	550	985.4	484.5	89	945.3	422.0	87	1147.7	262.3	74	977.0	419.1	95	1041.1	331.7	72
Difficult	S10	550	1191.5	383.4	75	1190.8	384.1	72	1284.3	181.4	60	1464.8	83.4	59	1148.9	383.3	57

As it is shown in Tables 27 and 28, EM with mutation has the first grade in finding the best solution and also can produce maximum cost reduction.

Table 27: Comparison of different EM algorithms in order to find the best solution

Algorithm	EM with reinitializing	EM with multi population	Parallel EM	EM with mutation	EM with local search
Ability to find the best solution	%0	%40	%0	%50	%10

Table 28: Different EM algorithms in order to produce maximum cost reduction

Algorithm	EM with reinitializing	EM with multi population	Parallel EM	EM with mutation	EM with local search
Ability to produce maximum cost reduction	%20	%20	%10	%20	%20

9 Conclusions

In this paper we proposed a new method based on Electromagnetism and considered four metaheuristic methods (Simulated Annealing, Tabu search, Genetic Algorithm and Ant Colony System) for solving ETP. The application of EM algorithm is shown on this problem for the first time in literature. All the algorithms were performed in a common structure and pure framework on 10 datasets from simple to difficult ones. When comparing the results of these algorithms, EM showed better ability to find the best solutions and in terms of improving its performance over time (cost reduction). Also we examine Local Search part of this algorithm and the ways of applying it and after analyzing EM parameters, we introduced some new versions of EM to improve its performance. We consider EM with reinitializing, EM with multi population, Parallel EM and EM with mutation and finally conclude that EM with mutation gives the first grade to find the best solution and also to produce maximum cost reduction in a fixed given CPU time.

References

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

[2] A.S. Asratian and D. de Werra, A generalized class_teacher model for some timetabling problems, *European J. Ope. Res.* 143 (2002) 531–542.

[3] V.A. Bardadym, Computer-aided school and university timetabling: the new wave, in *The Practice and Theory of Automated Timetabling*, E. Burke and P. Ross (eds.), Lecture Notes in Comput. Sci., Vol. 1153, 1996, pp. 22–45.

[4] S.I. Birbil and S.C. Fang, An Electromagnetism-like mechanism for global optimization, *J. Global Optim.* 25 (2003) 263–282.

[5] 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.

[6] E.K. Burke and S. Petrovic, Recent research directions in automated timetabling, *European J. Ope. Res.* 140 (2002) 266–280.

[7] 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.

[8] M. Cangalovic and Jan A.M. Schreuder, Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths, *European J. Ope. Res.* 51 (1991) 248–258.

[9] M.W. Carter, A survey of practical applications of examination timetabling algorithms, *Oper. Res.* 34 (2) (1986) 193–202.

[10] M.W. Carter, G. Laporte and S.Y. Lee, Examination timetabling: algorithmic strategies and applications, *J. Oper. Res. Soc. Japan* 47 (1996) 373–383.

[11] V. Cerny, A Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, *J. Optim. Theory Appl.* 45 (1985) 41–51.

[12] N. Collins, R.W. Eglese and B. Golden, Simulated annealing an annotated bibliography, *Amer. J. Math. Management Sci.* 8 (1988) 209–307.

[13] A. Colorni, M. Dorigo and V. Maniezzo, Genetic algorithms: a new approach to the timetabling problem, in *The Practice and Theory of Automated Timetabling*, E. Burke and P. Ross (eds.), Lecture Notes in Comput. Sci, Springer, Vol. 1153, 1996, pp. 235–239.

[14] D. Corne, P. Ross and H.L Fang, Evolving timetabling, in *Practical Hand Book of Genetic Algorithms: Applications*, Lance Chambers (ed.), Vol. 1, CRC Press, 1999.

[15] D. Debels, B.D. Reyck, R. Leus and M. Vanhouke, A hybrid scatter search/electromagnetism meta_heuristic for project scheduling, *European J. Ope. Res.* (2004) Article in Press.

[16] D. Debels and M. Vanhoucke, An electromagnetism meta-heuristic for the resource-constrained project scheduling problem, Ghent University, working paper, 2004/251.

[17] S. Deris, S. Omatu, H. Ohta, Timetable planning using the constraint-based reasoning, *Compu. Oper. Res.* 27 (2000) 819–840.

[18] 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).

[19] R.W. Eglese, Simulated annealing: A tool for operational research, *European J. Ope. Res.* 46 (1990) 271–281.

[20] W. Erben and J. Keppler, A genetic algorithm solving a weekly course-timetabling problem, in *The Practice and Theory of Automated Timetabling*, E. Burke and P. Ross (eds.), Lecture Notes in Comput. Sci., Springer, Vol. 1153, 1996, pp. 198–211.

[21] L.R. Foulds and D.G. Johnson, SlotManager: A microcomputer-based decision support system for university course timetabling, *Decision Support Systems* 27 (2000) 367–381.

[22] L.D. Gaspero and A. Schaerf, Tabu search techniques for examination timetabling, in *Third International Conference Patat 2000*, E.K. Burke and W. Erben (eds.), Lecture Notes in Comput.Sci., Vol. 2079, 2000, pp. 104–117.

[23] F. Glover, The general employee scheduling problem: an integration of management science and artificial intelligence, *Comput. Oper. Res.* 15 (1986) 563–593.

[24] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997.

[25] R.L. Haupt and S.E. Haupt, *Practical Genetic Algorithms*, John Wiley and Sons, New York, 1997.

[26] A. Hertz, Tabu search for large scale timetabling problems, *European J. Ope. Res.* 54 (1991) 39–47.

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

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

[29] 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.

[30] L.F. Paquete and C.M. Fonseca, A study of examination timetabling with multiobjective evolutionary algorithms, in *MIC the Metaheuristic International Conference (2001-4)*, Porto, Portugal, 2001, pp. 16–20.

[31] D-C. Rich, A smart genetic algorithm for university timetabling, in *The Practice and Theory of Automated Timetabling*, E. Burke and P. Ross (eds.), Lecture Notes in Comput.Sci, Springer, Vol. 1153, 1996, pp. 181–197.

[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 J. Ope. Res.* 104 (1998) 93–99.

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

[34] K. Socha, J. Knowles and 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 Comput.Sci., Springer Verlag. Berlin, Germany, Vol. 2463, 2002, 1–13. (also Technical Report /TR / IRIDIA /2002-18).

[35] J.M. Thompson and K.A. Dowsland, A robust simulated annealing based examination timetabling system, *Compu. Oper. Res.* 25(7/8) (1998) 637–648.

[36] P. Van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Practice*, Kluwer Academic Publishers, Netherlands, Dordrecht, 1987.

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

[38] J. Wood and D. Whitaker, Student centered school timetabling, *J. Oper. Res. Soc. Japan* 49 (1998) 1146–1152.

Manuscript received 27 May 2005
revised 20 December 2005
accepted for publication 25 February 2006

Majid Salari
Department of Mathematics, Ferdowsi University of Mashhad, Mashhad, Iran
E-mail address: salarimajid@yahoo.com

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