A Comparison of Metaheuristics for the Allocation of Elevators to Calls in Buildings

Berna BOLAT[1,*], Oğuz ALTUN[2], Pablo CORTES[3], Yunus Emre YILDIZ[4], Ali Osman TOPAL[5]

[1] Yildiz Technical University, Mechanical Engineering Department, Istanbul, Turkey

[2] Yildiz Technical University, Computer Engineering Department, Istanbul, Turkey

[3] Escuela Tecnica Superior de Ingeniería , Ingeniería Organizacion, University of Seville , Camino de los Descubrimientos, s/n Sevilla, Spain

[4]Epoka University, Faculty of Architecture and Engineering, Computer Engineering Department, Tirana, Albania

[5]Epoka University, Faculty of Architecture and Engineering, Computer Engineering Department, Tirana, Albania

(Received : 11.07.2016 ; Accepted : 12.07.2016)

ABSTRACT

This paper deals with the car-call allocation problem in vertical transportation in buildings. We have made a wide comparison of different metaheuristic optimization algorithms to identify those with a better performance dealing with the problem. The tested approaches are Differential Evolution (DE), Simulated Annealing with Random Starts (SAR), Artificial Bee Colony (ABC), Bat Algorithm (BA), Bacterial Foraging Optimization Algorithm (BF), Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and Tabu Search (TS). Each algorithm was tested in high-rise building simulations of 10 to 24 floors, with car configurations of 2 to 6 cars. Results proved that the ABC and TS algorithms generally result in better average journey times compared to other methods. It has to be noted that we introduced a new version of the Simulated Annealing, Simulated Annealing with Restarts (SAR), which ranked as the third best algorithm.

Keywords : Elevator group control system, evolutionary algorithms, optimization.

ÖZ

Bu çalışma binalarda düşey taşımacılıkta kullanılan asansörler için çağrıların kabinlere dağıtılması problemi üzerinedir. Geniş bir spektrumda farklı sezgisel optimizasyon algoritmaları problem üzerinde performans yönünden karşılaştırılmış ve başarılıları belirlenmiştir. Test edilen algoritmalar Çıkarımsal Evrim (Differential Evolution, DE), Rastgele Yeniden Başlatmalı Benzetimli Tavlama (Simulated Annealing with Random Starts, SAR), Yapay Arı Kolonisi (Artificial Bee Colony, ABC), Yarasa Algoritması (Bat Algoritması, BA), Bakteri Otlama Optimizasyon Algoritması (Bacterial Foraging Optimization Algorithm, BF), Parçacık Sürü Optimizasyonu (Particle Swarm Optimization, PSO), Genetic Algoritma (Genetic Algorithm, GA) ve Tabu Araştırmasıdır (Tabu Search, TS). Her algoritma simülasyon ile 10 ila 24 katlı binalar ve 2 ila 6 kabin ile test edilmiştir. Sonuçlar ABC ve TS algoritmalarının daha iyi bir ortalama yolculuk zamanı verdiğini göstermiştir. Ayrıca Benzetimli Tavlama algoritmasının yeni bir versiyonu olan Rastgele Yeniden Başlatmalı Benzetimli Tavlama (SAR) algoritması geliştirilmiştir. SAR deney sonuçlarında en iyi 3. algoritma olarak çıkmaktadır.

1. INTRODUCTION

The main problem in a vertical transportation system operated by an elevator group control system (EGCS) appears when a passenger in a floor makes a call and waits for a car to arrive his/her floor of destination in a quick and safe manner [1]. Therefore, the primary task an EGCS needs to solve efficiently is this landing call assignation problem. The major difficulty arises when such an EGCS needs to manage multiple elevators in a building in order to efficiently transport all the passengers in the building. Obviously, high-rise buildings with multiple coordinated cars increase the complexity of the problem. The generic problem to be solved consists of a passenger wanting to travel from one floor to another and therefore pressing the landing call button at a floor, generating what is called a landing call. The duty of the EGCS is to satisfy all the demands by assigning an elevator to each landing call (call made at a floor) in a way that some criteria are optimized. The most usual criterion is to minimize the passengers' waiting times. This so characterized problem is known to be a complex NP-hard problem [2,3,4], as such, an efficient deterministic solution is not known.

This problem recently attracted the increasing interest of the scientific community and several contributions have been made. Contributions based on metaheuristics have showed an adequate performance to deal with such complex problem. Examples of such techniques include Genetic Algorithm [5,6,7], Tabu Search [8], Particle Swarm Optimization [9,10,11], Immune Systems [12], Ant Colony Optimization [13], Viral Systems [14], and Fuzzy Logic approaches [15,16,17,18,19].

In this paper we present a comparison of diverse metaheuristics that can provide an interesting analysis about their performance for allocating cars to landing calls in buildings managed by an EGCS. This whole comparison has not been done previously and allows

*Corresponding Author
e-mail: balpan@yildiz.edu.tr
Digital Object Identifier (DOI) :

comparing different alternatives under the same basis of analysis. Tested techniques include Differential Evolution (DE), Simulated Annealing with Random Restarts (SAR), Artificial Bee Colony (ABC), Bat Algorithm (BA), Bacterial Foraging Optimization Algorithm (BF). All of them make use of the same solution encoding, which is based on a hall call allocation strategy to define the solution encoding (representation), together with a quick-to-evaluate car-call allocation quality estimation.

The rest of the paper deals with the specification of this solution encoding (in Section 2), the procedure to evaluate the quality of the different solutions (in Section 3), and the description of the different algorithms applied to the problem (in Section 4). The experimental results are described in Section 5.

2. REPRESENTATION

We start by the approach that was first proposed by [5]: Let l be number of elevators in an elevator system installed in a building with $\boldsymbol{n}$ floors. For the sake of giving an example let us assume $\boldsymbol{l=5}$ and $\boldsymbol{n=10}$. The requests for going to an upper floor can be represented by a $\boldsymbol{n-1}$ element binary vector as in (1), where a $\mathbf{1}$ represents a request (e.g. a push to the "up" button), and 0 represents no request. Notice there cannot be an up request in the highest floor.

$$\boldsymbol{B^u} = <\mathbf{0,0,1,1,0,1,1,1,1,}> \tag{1}$$

Down requests can also be represented with a similar $n-1$ element (there cannot be a down request in the lowest floor) binary vector B^d.

We then convert the representation of up requests to a more compact representation than proposed by [5], using an ordered set of integers where each element of the set represents a floor with an up request as in (2), where floor indices start from $\mathbf{0}$.

$$\boldsymbol{R^u} = \{\mathbf{2,3,5,6,7,8}\} \tag{2}$$

Down requests can be represented with a similar set of integers R^d. Then a solution x can be represented by a vector of elevator assignments to the requests in R^u and R^d as in (3), where the first 6 assignments are for R^u, and the rest are for R^d.

$$X = <\mathbf{3,2,2,1,0,2,1,1,3,3}> \tag{3}$$

Let $s(A)$ be the number of elements in collection$A$. Then $s(X) = s(R^u) + s(R^d)$.

Elevator indices start from $\mathbf{0}$ and go to $\boldsymbol{l-1}$, hence upper bounds for each dimension can be shown in a vector $\boldsymbol{U}$ as in (4), and lower bounds for each dimension can be shown in a vector $\boldsymbol{L}$ as in (5). In this representation changing the value of an element of $\boldsymbol{X}$ corresponds to changing the assigned elevator for that request.

$$U = <\mathbf{4,4,4,4,4,4,4,4,4,4}> \tag{4}$$

$$L = <\mathbf{0,0},0,0,0,0,0,0,0,0> \tag{5}$$

3. COST EVALUATION

To calculate the cost of the whole elevator system s, we start by calculating the personal cost c for each elevator as in [9], where Case I is when there are no requests, Case II is when there are only down requests, Case III is when there are only up requests, Case IV is when there are both up and down requests, ψ_1 is the ground floor level, ψ_2 is the highest down request level, ψ_3 is the number of down requests between ψ_1 and ψ_2, ψ_4 is the highest up request, ψ_5 is the number of up requests between ψ_1 and ψ_4, ψ_6 is the lowest down request, t is the door opening and closing time, and t_p is the passenger transfer time.

$$c = \begin{cases} 0, & \text{Case I} \\ \mathrm{t}(\psi_2-\psi_1) + t_\mathrm{p}(\psi_3-\psi_1), & \text{Case II} \\ \mathrm{t}(\psi_4-\psi_1) + t_\mathrm{p}(\psi_5-\psi_1), & \text{Case III} \\ \mathrm{t}(\psi_4-\psi_1) + t|\psi_2-\psi_4| + t|\psi_2-\psi_6| + t_\mathrm{p}|\psi_3+\psi_5-\psi_1|, & \text{Case IV} \end{cases} \tag{6}$$

After we calculate personal cost c_i for each elevator i, we get the average group cost g as in (7), where l is the number of elevators.

$$g = \frac{\mathbf{1}}{\mathbf{l}}\sum_{\boldsymbol{i}} \boldsymbol{f_i} \tag{7}$$

Finally, we get the solution cost value c for solutions s as in (8), where t^+ is the highest elevator trip time, t^- is the lowest elevator trip time, and k_1 and k_2 are constant weights.

$$c = \boldsymbol{k_1 g + k_2(t^+ - t^-)} \tag{8}$$

See [9] for a more in depth analysis of this cost function. We define cost mapper function $f(S)$ that takes a list of elevator systems S and return a list of cost values for each elevator system in S.

4. ALGORITHMS

This section presents the different approaches we have tested to deal with the car-call allocation problem in buildings managed by EGCS. All of them make use of the same representation described in Section 2, and evaluates the quality of the solutions by the procedure described in Section 3.

4.1. Global Best and Fixing Solutions

In all the algorithms it makes sense to keep a best solution found so far (e.g. global best) x^*. Hence we augment the objective function f to also update the global best when necessary as seen in Algorithm 1. In addition f checks for the validity of the input solution, and fixes it using the *fix_solution* function supplied by the user. f also takes care to return costs of all solution when there are more than one solutions.

Algorithm 1 In addition to returning the cost, objective function f also a) updates the global best x^* if necessary,

b) fixes out of range and similar problems with the solution (Line 6), c) works for single or multiple dimensions.

```
1:  function f
Inputs:
2:      $X \leftarrow< x_1, ..., x_n >$ // solution list
3:      $c \leftarrow$ cost function
4:      $s \leftarrow$ function for fixing invalid solutions
Body:
5:      if $X$ is a single solution then
6:          $x_f \leftarrow s(X)$
7:          $v \leftarrow c(x_f)$
8:          if isbetter$(v, v^*)$ then
9:              $x^* \leftarrow x_f$
10:             $v^* \leftarrow v$
11:         end if
12:         return $x_f, v$
13:     else
14:         $T \leftarrow< t_1, ..., t_n >$
15:         $v \leftarrow< v_1, ..., v_n >$
16:         for each $i \in$ indices$(X)$ do
17:             $t_i, v_i \leftarrow f(x_i)$
18:         end for
19:         return $T, v$
20:     end if
21: end function
```

All algorithms described in this section use the objective function f described in Algorithm 1.

4.2. Particle Swarm Optimization

Particle Swarm Optimization is a population based stochastic optimization method developed by Eberthart and Kennedy [20] in 1995. The algorithm, which is based on a metaphor of social interaction, searches a space by adjusting the trajectories of individual vectors. These vectors are called particles, as they are conceptualized as moving points in multidimensional space. The individual particles are drawn stochastically toward the positions of their own previous best performance and the best previous performance of their neighbors [21].

PSO algorithm is depicted in Figure 1. The algorithm starts with Node 1. In Node 2, we randomly initialize particle positions x_i and particle velocities v_i. In addition particle "personal best positions" p_i are initialized as starting positions x_i and the global best position x^* is initialized as the best of personal bests. In Node 3 if we decide to continue, because e.g. we have more time, we

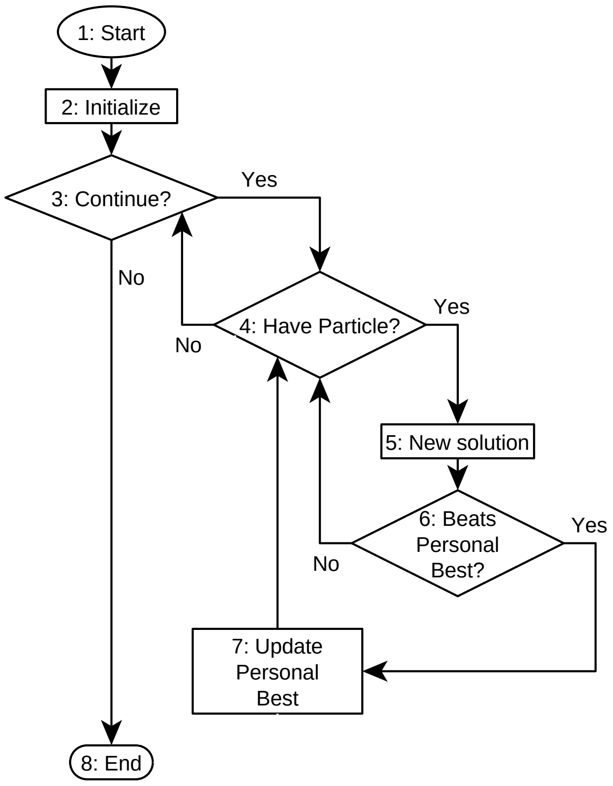

Figure 1. PSO algorithm flowchart.

start processing each particle i in the next generation t one by one. While we have unprocessed particles (Node 4), we get a new solution (Node5) using (9) and (10).

$$\boldsymbol{v_i(t+1) = wv_i(t) + c_1r_1(p_i - x_i(t)) + c_2r_2(x^* - x_i(t))} \tag{9}$$

$$\boldsymbol{x_i\,(t+1) = x_i\,(t) + v_i(t+1)} \tag{10}$$

Where w, c_1, and c_2 are constants and r_1 and r_2 are random values between [0,1] drawn from the uniform distribution. If the new solution is better than personal best $(f(xi(t+1)) > f(pi))$ as in Node 6, we update personal best in Node 7: $p_i = x_i(t+1)$. When we have no more particles to process (Node 4), we get out of particle loop. When the termination criteria is met (e.g. "No" edge in Node 3) the algorithm ends.

4.3. Artificial Bee Colony

Artificial Bee Colony (ABC) [22,23], simulates behavior of bees in a bee hive in search of food sources. As depicted in Figure 2, the algorithm starts with initializing random initial solutions x (Node 2). Note that in ABC metaphor each solution is a food source.

If there is enough time (Node 3), the algorithm moves to updating food sources. In each iteration, each food source is "visited" once (Node4). Visiting a solution x_i entails making a recombination of it with a random other solution x_j as in (11), where d is a random dimension and r is a uniform random number in the range $[-1,1]$.

$$x_i\,[d] \leftarrow x_i[d] + r(x_i[d] - x_j[d]) \tag{11}$$

In Node 5 each food source is assigned a probability of being re-visited, e.g. "luck", in the same iteration, based on the value of f. The food source with better f value has a higher probability p of being re-visited [15].

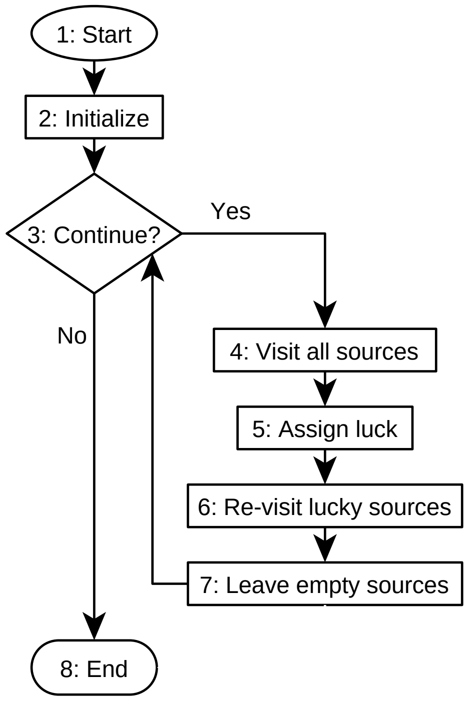

Figure 2. ABC algorithm flowchart.

$$p_i = \begin{cases} f(x_i) + 1, & f(x_i) \geq 0 \\ \dfrac{1}{(1 - f(x_i))}, & otherwise \end{cases} \tag{12}$$

In Node 6 if the food source is lucky, e.g. if $p_i > rand$ $(0,1)$, where $rand(0,1)$ is a random number in the range $[0,1]$ from the uniform distribution, the food source is re-visited according to probability determined in (12). This ensures that the neighborhoods of better solutions are visited more, hence makes the algorithm more elitist. In Node 7 the algorithm checks whether any of the current solution neighborhoods failed to produce any improvement for the last *limit* iterations. Such neighborhoods are abandoned for a random new neighborhood.

4.4. Genetic Algorithm

Genetic Algorithm (GA) [24], simulates evolution of solutions. As seen in Figure 3, GA starts with initialization of population x (Node2) randomly. If there is time to generate another generation of solutions (Node 3), the algorithm proceeds to produce a new generation. If still a new child is needed to complete the new generation (Node 4), two new children are generated in Node 5. In this step (Node 5), first two new parents are selected. From the two parents, two children are populated by crossover. Each child is mutated, and then added to the list of new generation solutions. In Node 6, the old generation is completely overwritten with the new generation. If no new generation is needed, the algorithm ends (Node 7).

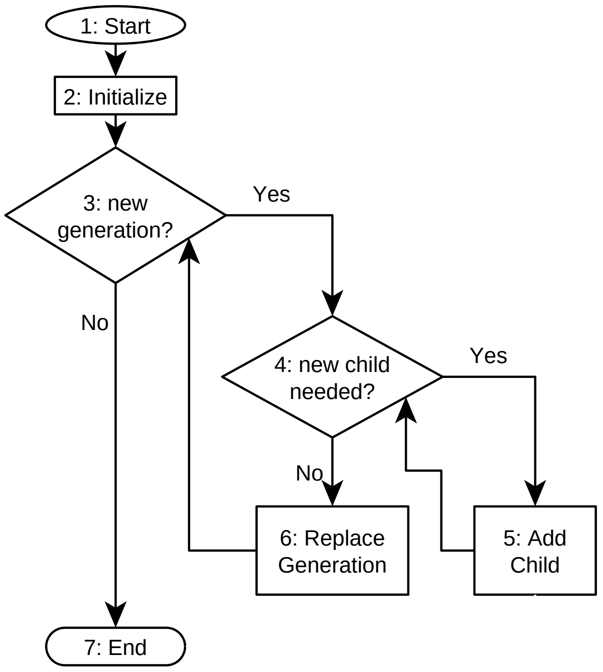

Figure 3. Genetic Algorithm flowchart.

4.5. Differential Evolution

Differential Evolution [24] is another algorithm that simulates the evolutionary behavior. As seen in Figure 4, the algorithm starts with initializing the populations of individuals X randomly. While we have time to work on another generation (Node 3), the algorithm takes a copy of the current population into parents Q (Node4), and iterates through each parent q (Node 5). In each iteration a new child e is made through (13) and (14), where a, b, and c are random parents that are different from q and each other, m is the constant mutation rate, and $\otimes$ represents the genetic crossover operator.

$$\boldsymbol{d} = \boldsymbol{a} + \boldsymbol{m}(\boldsymbol{b} - \boldsymbol{c}) \tag{13}$$

$$e = \boldsymbol{q} \otimes \boldsymbol{d} \tag{14}$$

If the child e is better than the parent q, e replaces corresponding element in X.

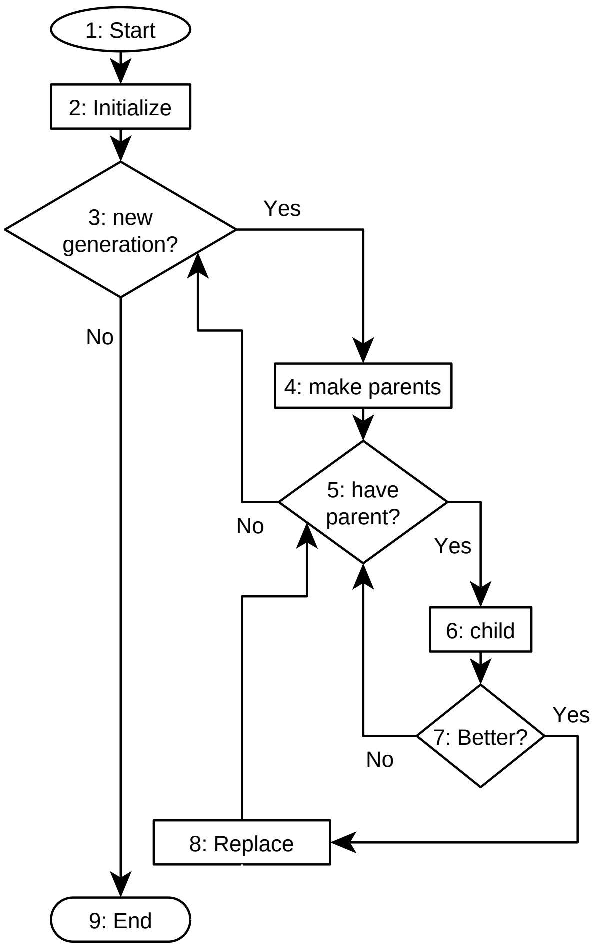

Figure 4. Differential Evolution flowchart

4.6. Tabu Search

Tabu Search [24] algorithm keeps a list of solutions that are already evaluated (tabu), and avoids re-evaluating those solutions. As depicted in Figure 5 the algorithm starts by initializing (Node 2) an empty tabu list L, getting a random current solution s, and adding s to L. While there is enough time (Node 3), we proceed to build a new solution r by tweaking the existing solution s (Node 4). If r is not in the tabu list (Node 5), current solution is updated (Node 5) $s \leftarrow r$, and r is added to the tabu list L.

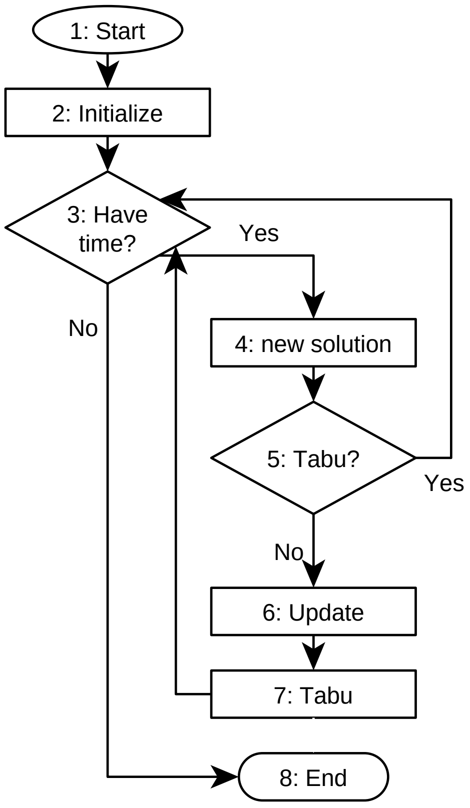

Figure 5. Tabu Search flowchart.

4.7. Simulated Annealing with Random Restarts

Simulated Annealing [24] algorithm simulates annealing process. The algorithm decreases the chance of getting stuck in local optima by occasionally accepting solutions that are worse than the current solution. This version with random restarts, restarts search from a random position after local search fails to improve a given time (limit l). As seen in Figure 6, the algorithm starts by initializing (Node 2) constants limit l, initial temperature t_0, and cooling scheduler c. If there is time to another annealing (Node 3), algorithm resets (Node 4) current temperature with $t \leftarrow 0$, and current solution with $s \leftarrow random\_solution$. If number of bad trials n is less than limit l, we continue (Node 5) improving current solution by producing a tweak of it (Node 6). If the tweak r is better than s (Nodes 7-8), we directly update current solution: $s \leftarrow r$. The interesting thing about Simulated Annealing is that we still update when r is worse than s if r is “lucky”, e.g. if (15) holds, where $rand(0,1)$ is a random value between 0 and 1, and $|.|$ denotes the absolute value.

$$rand(0,1) \leq \exp\left(-\frac{|f(s)-f(r)|}{t}\right) \qquad (15)$$

This allows the algorithm to escape from local optimums. When there is an update, the number of bad trials is reset: $n \leftarrow 0$. When an update was not done, the number of bat trials is incremented: $n \leftarrow n+1$ (Node 9). In each iteration the temperature is cooled down (Node 10) using schedule constant c that has a value less than 1: $t \leftarrow ct$.

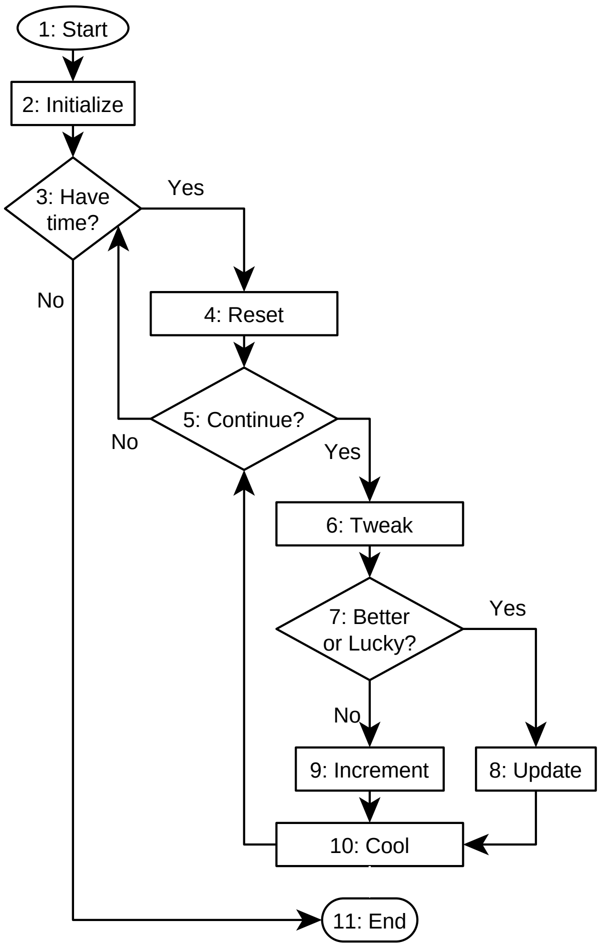

Figure 6. Simulated Annealing with Restarts flowchart.

4.8. Bat Algorithm

Bath algorithm is a nature inspired algorithm that simulates bat's echolocation ability to get optimum solution for tough optimization problems. Yang [25] has used advantages of other well-known algorithms (Particle Swarm Optimization (PSO)) and Harmony Search (HS) in the bat algorithm and proposed a powerful algorithm.

As depicted in Fig 7. the Bat Algorithm starts with n bats being placed randomly in the search space. Velocity v_i , frequency f_i , pulse rate r_i , and loudness A_i are initialized (Node 2) for each bat i at the beginning. Bat's position and velocity are updated using (16), (17), and (18),

$$\boldsymbol{f_i} = \boldsymbol{f_{max}} + (\boldsymbol{f_{max}} - \boldsymbol{f_{min}})\boldsymbol{\beta} \quad (16)$$

$$\boldsymbol{V_i^t} = \boldsymbol{V_i^{t-1}} + (\boldsymbol{x_i^t} - \boldsymbol{x^*}) \quad (17)$$

$$\boldsymbol{x_i^t} = \boldsymbol{x_i^{t-1}} + \boldsymbol{V_i^t} \quad (18)$$

where β is a random vector generated from a uniform distribution in the range [0,1]. Then the algorithm evaluates the fitness (solutions) and chooses the current best position x^* (Node 4). After these updates, in Node 5, if the bat's pulse rate is low (which means bat is far away from the prey), with a high probability it will fly near the current best bat (Node 6) and make a random short fly there. If its pulse rate is high then it should be near prey and with a high probability it will make a random fly around its current position (Node 7). After this fly if the bat's position is better than the current global best and its loudness is loud enough to be greater than a random number (Node 8), the bat will fly to this position and current global best will be updated with the new one. The bat's pulse rate r_i will be increased and loudness A_i will be decreased (Node 9). Then in Node 10, again fitness will be evaluated and the current best x^* will be found.

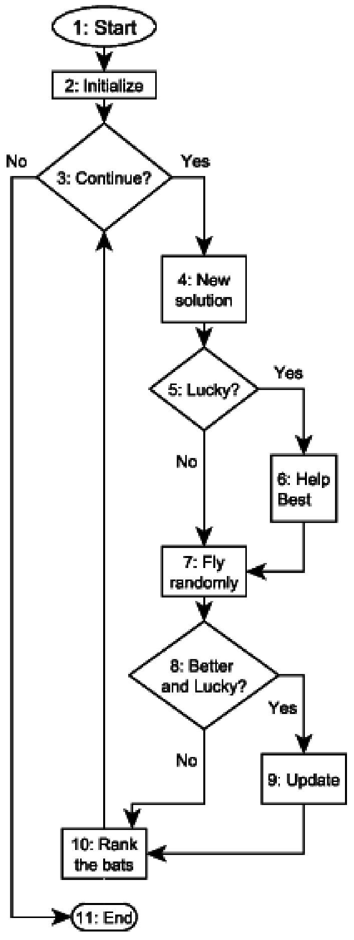

Figure 7. Bat Algorithm flowchart.

4.9. Bacterial Foraging Optimization Algorithm

The bacterial foraging system consists of three principal mechanisms, namely chemotaxis, reproduction and elimination-dispersal [26].

As seen in Figure 8, the algorithm starts with initializing the bacterium position x_i randomly. Suppose $\theta(i, j+1, k, l)$ represents bacterium i (Node 12), chemotactic step j (Node 9), reproductive step k (Node 5) and elimination – dispersal step l (Node 4). $C(i)$ is the size of the step taken in the random direction specified by the tumble (Node 14). The chemotaxis movement of the bacterium may be represented by (19), where $\Delta(i)$ is the random vector whose elements lie in range $[-1,1]$.

$$\theta(i, j+1, k, 1) = \theta(i, j, k, l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \tag{19}$$

$J(i, j, k, l)$ is the current bacterium's cost function value (Node 15). It only changes its position if the $J(i, j+1, k, l)$ is better than $J(i, j, k, l)$. We let bacterium i take a step size (Node 14 and 20) along the direction of tumble vector $\Delta(i)$. Elimination and dispersal events (Node7) may occur in the local environment when the bacteria are exposed to gradual or sudden changes such as significant rise of temperature or sudden flow of water. In order to simulate these events in BF, some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space. The health of each bacterium is computed as the sum of the objective function value during its lifetime. After each chemotaxis step, all bacteria are sorted in decreasing order of health. Only the first half of the bacteria (healtiest ones) asexually split into two bacteria which are then placed at the same locations. In this way, the size of the population is kept constant (Node 10).

5. EXPERIMENTAL RESULTS

We devised 6 different problems by changing the number of floors n and number of elevators l as in Table 1, and compared each algorithm on each of the problems.

Table 1. Information on the problems p is problem number, n is number of floors, and l is number of elevators.

P	N	l
1	25	5
2	25	10
3	100	5
4	100	10
5	200	5
6	200	10

For comparison of the algorithms on a problem p, we follow the procedure detailed in Algorithm 2. In

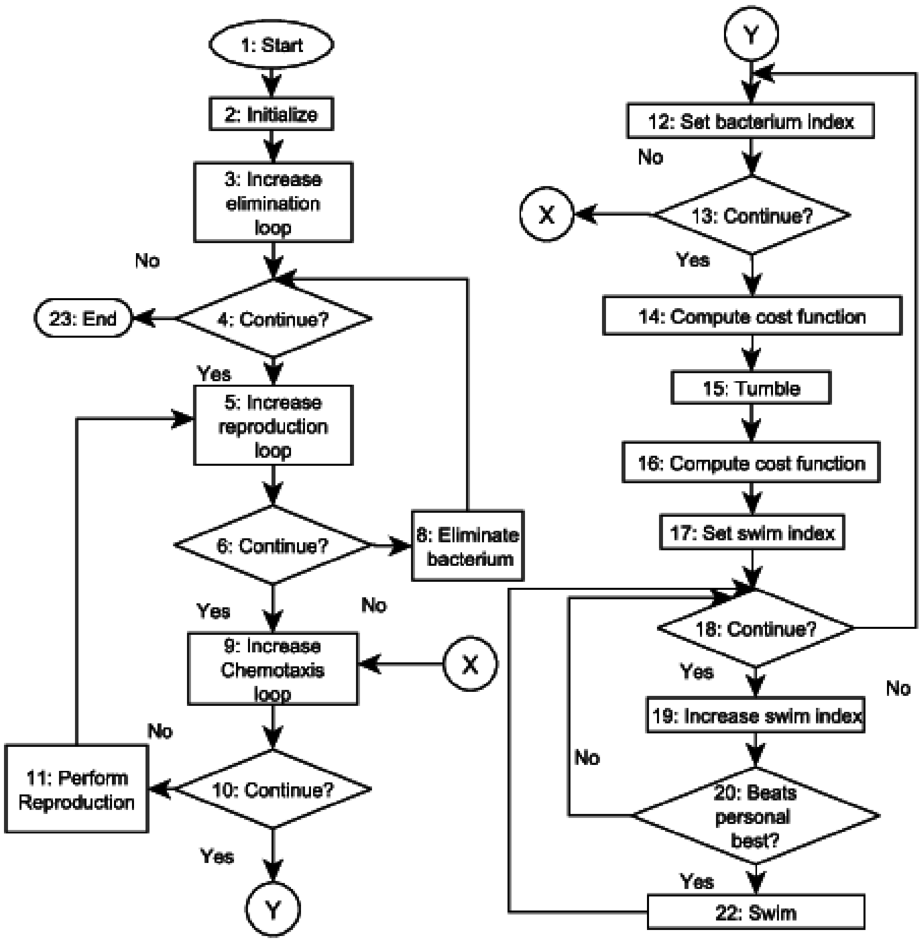

Figure 8. Classical Bacterial Foraging Algorithm (BF) flowchart.

Algorithm 2, we first define inputs. Notice that E and d are functions that are used in the procedure. Inside the procedure we loop over problems P, and algorithms A. We do 30 trials for each $< a, p >$ pair and get an average convergence graph. On each trial, the algorithm terminates after 10000 function evaluations. For each problem p we draw convergence comparison graphs.

Algorithm 2 The procedure that compares algorithms on problems. Line 10: Run the algorithm a on problem p to get list of <function evaluation, global best value> pairs Y. Algorithm a terminates after calling F function evaluations. Line 12: Get a mean Y_{ap} list of <function evaluation, global best value> pairs. Line 14: Draw the convergence comparison graph for A on p.

```
1:   procedure compare

Inputs:
2:        $P \leftarrow$ set of problems
3:        $A \leftarrow$ set of algorithms
4:        $I \leftarrow$ set of trials to carry out
5:        $F \leftarrow$ number of function evaluations allowed
6:        $E \leftarrow$ function to take a set of list of <function
          evaluation, global best value> pairs and return
          a mean list

Body:
7:        for each $p \in P$ do
8:            for each $a \in A$ do
9:                for $i \leftarrow 1$ to $I$ do
10:                   $Y_i \leftarrow a(p, F)$
11:               end for
12:               $Y_{ap} \leftarrow E(\{Y_i\})$
13:           end for
14:           draw($\{Y_{ap}\}$)
15:       end for
16:  end procedure
```

5.1. Convergence Graphs

As described in Algorithm 2, for each problem defined in Section 5 we draw a convergence graph.

A convergence graph shows the mean of the values of the best position found (f^*) with respect to the number of cost function evaluations (FES). In our work f represents travel time, so lower curves are better.

In Figure 9a we see that ABC and TS are the best performing algorithms. ABC did not converge when the maximum number of FES was reached, so it could still lower the result. The objective function f was reduced to less than 100.

In Figure 9b we see that ABC and SAR lead again. ABC is not still converged. f was decreased to less than 100.

In Figure 9c we see that ABC and SAR are the two best performing algorithms. SAR seems to have converged, and ABC seems to be still improving. Hence we expect ABC to further improve. The f was reduced to less than 625.

In Figure 9d we see that ABC and SAR lead again. ABC is still not converged. f was decreased to less than 650.

In Figure 9e we see that SAR and ABC are the best performing algorithms. ABC seems to be still improving. The f was reduced to less than 1250.

In Figure 9f we see that SAR and ABC are performing best, and neither of them was converged. f was reduced to less than 1200 by SAR.

We see that ABC and TS are always in the top two. We also see that as number of floors n change, the successful algorithm change: for $n = 25$ ABC is the best algorithm. When n increases to 100, TS becomes on par with ABC, and when n is increased furthermore to 200 TS beats ABC. SAR emerges as the third best algorithm for higher number of floors.

We also see that the number of elevators l does not affect the ordering of the algorithms, whereas number of floors n affects greatly. As the number of floors n increases, the probability of problem having a high dimensionality increases, since in the representation we introduced in Section 2, the dimensionality of the problem changes with the number of requests coming from the floors. E.g. when $n = 200$, the maximum possible dimensionality of the problem becomes $2n - 2 = 398$. Number of elevators l effects the upper bounds of the search space, and what we see from our experiments is that it does not affect the success order of different algorithms.

It needs to be noted that ABC did not seem to be converged in any of trails. When the convergence graphs of Problem 1, Problem 3 and Problem 5 are inspected consecutively, it can be hypothesized that ABC can beat TS in the long run, even for the larger number of floors n. Performance of ABC over much longer runs is to be seen.

The elevator system optimization obviously benefits from using soft computing techniques, as seen from the fact that the solution was improved vastly for all problems.

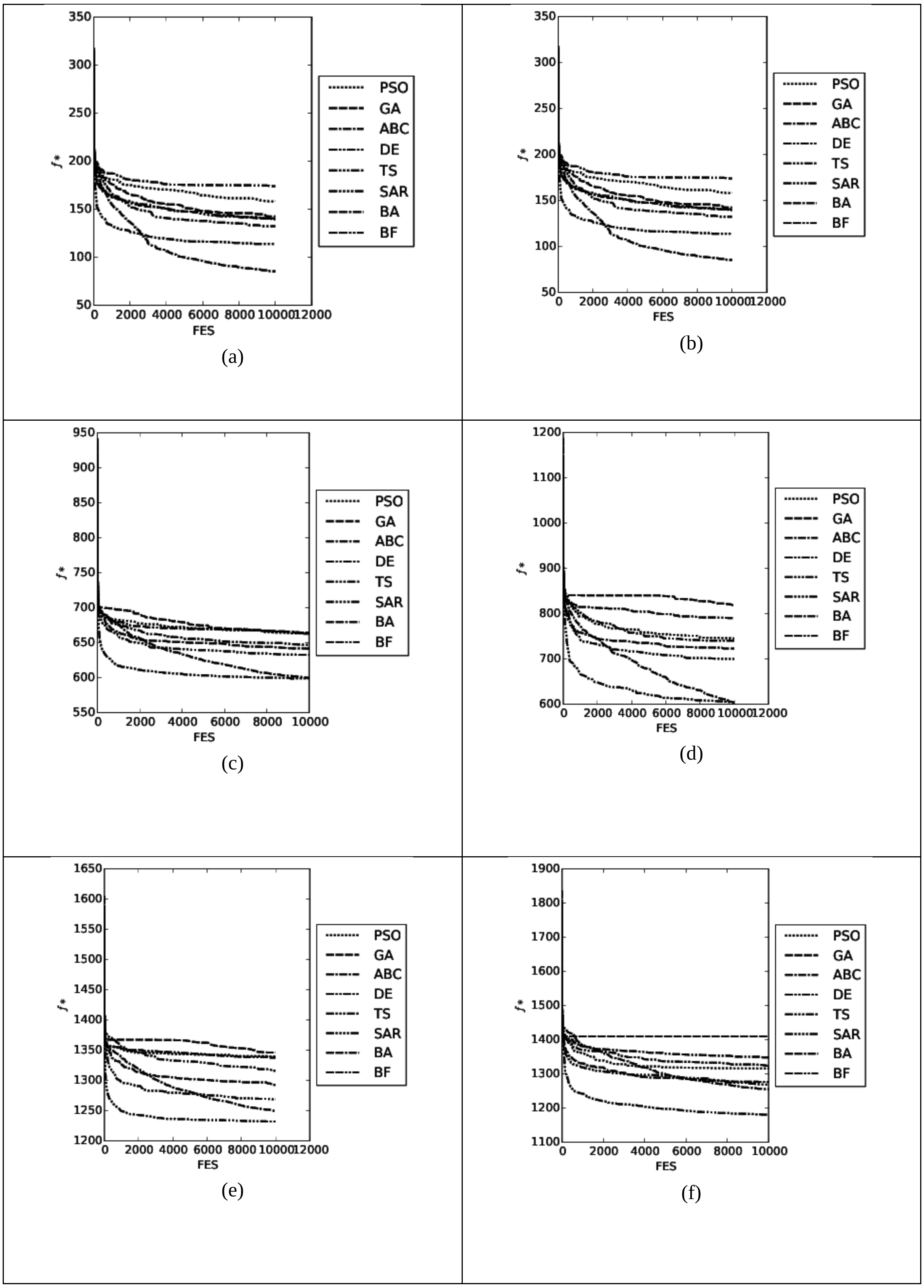

Figure 9. Convergence graphs for Problems 1-6. (a) Problem 1, (b) Problem 2, (c) Problem 3, (d) Problem 4, (e) Problem 5, (f) Problem 6. See Table 1 for definition of problems.

6. CONCLUSION AND FUTURE WORK

We have presented a comparison of Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Artificial Bee Colony (ABC), Differential Evolution (DE), Tabu Search (TS), Simulated Annealing with Random Restarts (SAR) , Bat Algorithm (BA), Bacterial Foraging Optimization Algorithm (BF) soft computing techniques for global optimization of the car-call allocation strategy of the controller in EGCS. Each algorithm was permitted 10000 function evaluations, and the convergence graphs of the global best function value were drawn. Results were provided for high-rise buildings from 10 to 24 floors, and several car configurations from 2 to 6 cars. We observe the SAR and ABC algorithms outperform the other algorithms in general. Furthermore, ABC has potential to further improve its results provided that more function evaluations are permitted.

Also, we introduced a new problem representation that sets up the problem as assignment of cars to floor requests, which allows easier programming of the simulation. In this presentation, the number of floors affects the success order of the algorithms, as it affects the dimensionality of the problem, whereas the number of elevators does not have an obvious effect on that order.

We introduced a new version of Simulated Annealing, namely Simulated Annealing with Random Restarts (SAR), that restarts the search produce when stuck in a local optimum. SAR performed as the third ranked algorithm in general in our tests. Results showed that aforementioned soft computing optimization techniques, TS and ABC in particular, help in reducing the passenger travel times greatly, hence are suitable for this problem. ABC algorithm did not seem to be converged in any of the trials, and their performances over longer runs are to be seen in future works.

REFERENCES

1. Fernandez J.R. and Cortes, P., "A survey of elevator group control systems forvvertical transportation: a look at recent literature", ***IEEE Control Systems***, 35(4): 38-55, (2015).
2. Cortes P., Munuzuri J. and Onieva L., "Design and Analysis of a Tool for Planning and Simulating Dynamic Vertical Transport", ***Simulation***, 82: 255-274, (2006).
3. Knuth D. E., "A terminological proposal", ***SIGACT News***, 6(1): 12-18, (1974).
4. Knuth D. E. (1974). Postscript about NP-hard problems. ***SIGACT News***, 6(2): 15-16, (1974).
5. Cortes P., Munuzuri J. and Onieva L., "Genetic algorithm for controllers in elevator groups: analysis and simulation during lunchpeak traffic", ***Applied Soft Computing***, 4(2): 159-174, (2004).
6. Bolat B., Cortes P., Yalçın E. and Alışverişçi M., "Optimal car dispatching for elevator groups using genetic algorithms", ***Intelligent Automation &Soft Computing***, 16(1), (2010).
7. Chen T.C., Hsu Y.J, and Huang Y.J., "optimizing the intelligent elevator group control system by using genetic algorithm", ***Advanced Science Letters***, 9(1), (2012).
8. Bolat B. and Cortes P., "Genetic and tabu search approaches for optimizing the hall call-car allocation problem in elevator group systems", ***Applied Soft Computing***, 11(2), (2011).
9. Bolat B., Altun O. and Cortes P., "A particle swarm optimization algorithm for optimal car-call allocation in elevator group control systems", 13(5), (2011).
10. Li Z, Tan H,Z, and Zhang Y., "Particle swarm optimization applied to vertical traffic scheduling in buildings in", ***11 th International Conference KES and XVII Italian Workshop on Neural Networks Conference on Knowledge-Based Intelligent Information and Enginnering Systems : Part I***, (2007).
11. Fei L., Xiaocui Z. and Yuge X., "A new hybrid elevator group control system scheduling strategy based on particle swarm simulated annealing optimization algorithm in intelligent control and automation", ***8th World Congress***, 5121-5124, (2010).
12. Li Z., Mao Za and Wu J., "Research on dynamic zoning of elevator traffic based on artifical immune algorithm", ***8th Control, Automation,Robotics and Vision Conference,*** 3: 2170-2175, (2004).
13. Liu J. and Liu Y., "Ant colony algorithm and fuzzy neural network- based intelligent dispatching algorithm of an elevator group control system, ***IEEE International Conference on Control and Automation***, (2007).
14. Cortes P., Onieva L, Munuzuri J. and Guadix J., "A viral system algorithm to optimize the car dispatching in elevatro group control sytems of tall buildings, ***Computers&Industrial Engineering***, 64(1) :403-411, (2013).
15. Cortes P., Fernandez J.R, Guadix J. and Munuzuri, J., "Fuzzy logic based controller for peak traffic detection in elevator systems", ***Journal of Computational and Theoretical Nanoscience***, 9(2): (2012).
16. Jamaludin J., Rahim N. and Hew W.P., "An elevator group control sytem with a self –tuning fuzzy logic group controller", ***IEEE Transactions on Industrial Electronics***, 57(12): 4188-4198, (2010).
17. Rashid M., Kasemi B., and Faruq A. Alam., "Design of fuzzy based controller for modern elevator group with floor priority constraints, ***4th International Conference on Mechatronics***, (2011).
18. Fernandez J.R., Cortes P., Munuzuri J. and Guadix J., "Dynamic fuzzy logic elevator group control system with relative waiting time consideration", ***IEEE Transaction on Industrial Electronics***, 61(9): (2014).
19. Fernandez J.R., Cortes P., Guadix J., and Munuzuri J., "Dynamic fuzzy logic elevator group control system for energy optimization", ***International Journal of Information Technology and Decision Making***, 12(3): (2013).

20. Kennedy J. and Eberhart R.C., "Particle swarm optimization, ***IEEE International Conference on Neural Networks***, 4: 1942-1948, (1995).
21. Clerc M. and Kennedy J., "The particle swarm-explosion, stability, and convergence in a multidimensional complex space", ***IEEE Transactions on Evolutionary Computation***, 6(11): (2002).
22. Karaboğa D., "An idea based on honey bee swarm for numerical optimization, ***Technical Report-tr06***, Erciyes University, Engineering Faculty, Computer Engineering Department, (2005).
23. Karaboğa D. and Basturk B., "A powerful and efficient algorithm for numerical function optimization : artifical bee colony (ABC) algorithm", ***Journal of Global Optimization***, 39(3): 459-471, (2007).
24. Luke S., "Essentials of Metaheuristics, Lulu, Second Edn., (2013).
25. Yang X.S., "A new metaheuristic bat-inspired algorithm, in J. Gonzalez, D. Pelta, C. Cruz, G. Terrazas, N.Krasnogar (eds.), ***Nature Inspired Cooperative Strategies for Optimization***, 284: (2010).
26. Passino K., "Biomimicry of bacterial foraging for distributed optimization and control, "***IEEE Control Systems***, 22(3): 52-67, (2002).