Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 1879 – 1900

doi:10.3906/elk-1311-45

System identification by using migrating birds optimization algorithm: a comparative performance analysis

Hasan MAKAS*, Nejat YUMUŞAK
Department of Computer Engineering, Faculty of Computer and Information Sciences, Sakarya University, Sakarya, Turkey

Received: 06.11.2013 • **Accepted/Published Online:** 01.07.2014 • **Final Version:** 23.03.2016

Abstract: System identification is an important process to investigate and understand the behavior of an unknown system. It aims to establish an interface between the real system and its mathematical representation. Conventional system identification methods generally need differentiable search spaces and they cannot be used for nondifferentiable multimodal search spaces. On the other hand, metaheuristic search algorithms are independent from the search space characteristics and they do not need much knowledge about the real system. The migrating birds optimization algorithm is a recently introduced nature-inspired metaheuristic neighborhood search approach. It simulates the V flight formation of migrating birds, which enables birds to save energy during migration. In this paper, first, a set of comparative performance tests by using benchmark functions are performed on the migrating birds optimization algorithm and some other well-known metaheuristics. The same metaheuristic algorithms are then employed to solve several system identification problems. The results show that the migrating birds optimization algorithm achieves promising optimizations both for benchmark tests and for system identification problems.

Key words: Migrating birds optimization, system identification, neighborhood search, swarm intelligence, metaheuristics

1. Introduction

Metaheuristics are algorithms that are designed to solve hard optimization problems in a wide range of applications and they do not have to deeply adapt to each problem. The prefix "meta" indicates that they are higher-level heuristic algorithms in contrast with problem-specific heuristics. They are generally used to solve problems for which there is no satisfactory problem-specific algorithm to get a reasonable solution. These problems are complex problems in industry and services ranging from finance to production management and engineering [1]. Most of the metaheuristic algorithms are called neighborhood search methods, and they constitute an important and large class among the improvement algorithms. They search the neighborhoods of the existing solutions and try to get better solutions [2]. Therefore, the choice of the neighborhood structure is a critical issue for the design of a neighborhood search algorithm [3].

There are many metaheuristic algorithms that have been introduced by researchers so far. Some of the most popular and commonly used metaheuristics are the genetic algorithm (GA) introduced by Holland [4], the simulated annealing algorithm introduced by Kirkpatrick et al. [5], the tabu search algorithm introduced by Glover [6], the ant colony optimization algorithm introduced by Drigo [7], and the particle swarm optimization

*Correspondence: hasanmakas@gmail.com

(PSO) algorithm introduced by Eberhart and Kennedy [8]. Some of the recent metaheuristic algorithms are the differential evolution (DE) algorithm [9], the harmony search algorithm [10], the monkey search algorithm [11], the artificial bee colony (ABC) algorithm [12], the firefly algorithm [13], the intelligent water drops algorithm [14], the cuckoo search algorithm [15,16], the bat algorithm [17,18], and the migrating birds optimization (MBO) algorithm [2]. In parallel to these studies, many researchers have worked to enhance the metaheuristics. Some of them have worked on the best tunings of these metaheuristics [19]; on the other hand, some of them have developed new modified or hybrid forms of the metaheuristics [20–22], and some others have established parallel running methodologies [23] to get much better optimization performances.

Most of the metaheuristics are inspired by nature. This shows that the perfectness of nature is still an inspiration source for mankind to develop new methodologies. In these algorithms, while the mobile agents interact locally, they somehow form emergent and self-organized behaviors under the right conditions, leading to global convergence. The agents generally explore the search space locally and they are aided by randomization to increase the diversity of solutions. Thus, there is a fine balance between local intensive exploitation and global exploration [24]. In addition, swarming agents are suitable to work in parallel, leading to reduction in computation time.

System identification is an approach that establishes an interface between real unknown systems and their mathematical representations. System identification methods aim to find appropriate models for the systems. They generally use input and output signals of the unknown systems for modeling. There are many system identification techniques, including the autoregressive with exogenous inputs model [25], the output error model [26], and the Box–Jenkins model [27]. Nowadays, some metaheuristic methods are used for system identification, and they give promising and competitive results [28–30].

In this paper, first we make a performance test among the PSO, ABC, DE, GA, and MBO algorithms by using multidimensional benchmark test functions. We compare the performance of the MBO algorithm with that of the others. We then employ these metaheuristics to solve some system identification problems considering that they are commonly encountered in optimal parameter tunings of the transfer functions in areas ranging from digital filter design to PID controller tuning [31,32]. The paper is organized as follows. Section 2 gives the theoretical background of the metaheuristics used in this paper and clarifies the philosophy of the system identification approach. Section 3 lists the benchmark functions used in experimental works, gives the system identification problem examples, and explains the experimental setups. Section 4 gives the experimental results and discusses them. Section 5 concludes the study.

2. Methods and background

2.1. The PSO algorithm

The PSO algorithm is a swarm intelligence-based metaheuristic algorithm. It is inspired by the behavior of flocking birds. The algorithm uses a swarm of particles as its population. It is assumed that each particle is in motion inside a multidimensional search space and represents a possible solution for the problem. The next position of a particle is calculated by Eq. (1), which adds its velocity in the next time step to its current position [33]:

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

where x_i and v_i are the position and velocity of the ith particle. The velocity vector controls the optimization

process and it is calculated by Eq. (2) [33]:

$$v_{ij}\,(t{+}1) = w \cdot v_{ij} + c_1 \cdot rand \cdot (p_{ij}\,(t) - x_{ij}\,(t)) + c_2 \cdot rand \cdot (p_{gj}\,(t) - x_{ij}\,(t))\,, \tag{2}$$

where v_{ij} is the velocity of the ith particle in the jth dimension, w is inertia weight, c_1 and c_2 are acceleration coefficients, rand is a uniform random number ranging from 0 to 1, x_{ij} is the current position of the ith particle in the jth dimension, p_{ij} is the best position in the jth dimension achieved by the ith particle so far, and p_{gj} is the best position in the jth dimension achieved by the population so far. Particle velocities should be in a predefined range of $[-V_{\max}, V_{\max}]$ to keep the global search capability under control. Therefore, they are shifted to predefined limits as given in Eq. (3) when they exceed the limits. The $V_{\max}$ value was considered the same for each dimension in benchmark tests, but this limit value can be different for each dimension j depending on the optimization problem. We chose $V_{\max}$ as 1.

$$v_{ij}(t+1) = \begin{cases} -V_{\max} & , \quad v_{ij}(t+1) \le -V_{\max} \\ v_{ij}(t+1) & , \quad -V_{\max} < v_{ij}(t+1) < V_{\max} \\ V_{\max} & , \quad v_{ij}(t+1) \ge V_{\max} \end{cases} \tag{3}$$

Inertia weight w in Eq. (2) is an important parameter to control the convergence characteristic of the PSO algorithm. If an inertia weight $w > 1$ is chosen, the velocities converge to limit velocities and the swarm diverges from the global minimum. Higher values of w in the range of [0, 1] increase global search capability and decrease local search capability of the algorithm, or vice versa. It was reported experimentally that if w decreases from 0.9 to 0.4 linearly through the iterations, the PSO performance increases significantly [34]. Although the acceleration coefficients c_1 and c_2 are chosen as equal in most of the applications, the ratio of these values depends on the problem type. While small values of acceleration coefficients cause smooth particle trajectories with slow velocity changes, high values of them cause sharply changing particle trajectories with rapid velocity changes. The static value of 1.494 can be used for each one of the acceleration coefficients to get a good optimization performance [35]. We used these recommended c_1, c_2, and w values in our experiments.

Because the diversity of the initial population directly affects the PSO performance, initialization of the population is a very important issue to get better optimization. Initial positions are assigned to particles by using Eq. (4):

$$x_{ij} = x_j^{min} + rand \cdot \left(x_j^{max} - x_j^{min}\right), \tag{4}$$

where x_{ij} is the position of the ith solution in the jth dimension, x_j^{min} and x_j^{max} are the limit values for the jth dimension, and rand is a uniform random number ranging from 0 to 1. Initial positions are assigned to initial personal best positions and initial velocity values can be set to zero. Another important point to be considered is the limit check for the next positions of the particles, which are calculated by Eq. (2). All of the particles are forced to be inside the problem space by making proper shifts via Eq. (5) when it is needed:

$$x_{ij}(t+1) = \begin{cases} x_j^{min} & , \quad x_{ij}(t+1) \le x_j^{min} \\ x_{ij}(t+1) & , \quad x_j^{min} < x_{ij}(t+1) < x_j^{max} \\ x_j^{max} & , \quad x_{ij}(t+1) \ge x_j^{max} \end{cases}, \tag{5}$$

where $x_{ij}(t+1)$ is the next position of the ith particle in the jth dimension, and x_j^{min} and x_j^{max} are the limit values of the jth dimension. This shifting operation is also valid for all of the following algorithms.

2.2. The ABC algorithm

The ABC algorithm is another swarm intelligence-based metaheuristic algorithm. It is inspired by bees' foraging behaviors. The position of a food source represents a possible solution for the optimization problem and the nectar amount of a food source corresponds to the quality or fitness of that solution. There are employed bees and onlooker bees. The numbers of employed bees and onlooker bees are equal to that of food sources [36]. Therefore, the total number of food sources is equal to half of the colony size. There are three calculation phases in each iteration: the employed bee phase, the onlooker bee phase, and the scout bee phase. At the beginning, the algorithm generates randomly distributed initial positions for the food sources via Eq. (4). After completing the food source initialization, algorithm phases start to run and continue until the termination criteria are met.

The first phase is the employed bee phase. Employed bees try to enhance their solutions by generating new neighbors via Eq. (6) in this phase:

$$\hat{x}_{ij} = x_{ij} + \phi \cdot (x_{ij} - x_{kj}) \tag{6}$$

where x_{ij} is the position of the ith solution in the jth dimension, k is a randomly selected index value different from i, x_{kj} is the position of the kth solution in the jth dimension, ϕ is a random number ranging from –1 to 1, and $\hat{x}_{ij}$ is the generated new neighbor position in the jth dimension for the ith solution. This calculation controls the neighborhood creation at the same time by making a comparison between two bees' positions in parentheses. That is, the smaller the difference between x_{ij} and x_{kj} is, the closer the solutions are. In other words, closer solutions make the change in x_{ij} decrease. The fitness value of a source is calculated by Eq. (7):

$$Fitness_i = \begin{cases} \dfrac{1}{(1+f_i)}, & f_i \geq 0 \\ 1 + abs\,(f_i)\,, & f_i < 0 \end{cases}, \tag{7}$$

where f_i and $Fitness_i$ are the cost and fitness values of the ith source, respectively. A greedy selection mechanism, which is only based on the fitness values of the food sources, is used to make a selection between existing food sources and new generated neighbors. After performing the greedy selections, selection probabilities of updated sources by onlooker bees are calculated via Eq. (8):

$$p_i = \frac{\text{Fitness}_i}{\sum_{j=1}^{SN} \text{Fitness}_j}, \tag{8}$$

where p_i is the selection probability of the ith source by onlooker bees and SN is the total number of sources.

In the second phase, the onlooker bee phase, the onlooker bees make their selections depending on the corresponding p_i probabilities. The higher the selection probability is, the more chance there is that the corresponding food source may be selected and enhanced by onlooker bees. Roulette wheel or some other probabilistic selection mechanism can be used by onlooker bees in selection operation. Each onlooker generates a neighborhood for its selection via Eq. (6). It then makes a greedy selection between the selected source and new generated source in order to enhance the selected food source.

In the third phase, the scout bee phase, new food sources instead of the food sources whose nectar was abandoned are generated by scouts via a global search operation. A failure counter is assigned to each source in implementations. After completing a neighborhood creation for the corresponding food source, the failure

counter of this source is increased by one if no improvement is achieved. Otherwise, that counter is set to zero. Whenever the failure counter of a food source exceeds a predefined limit, the employed bee of that source is transformed into a scout bee to find a new food source by making a global search via Eq. (4). The scout bees are retransformed into employed bees after they generate new sources. The limit for the failure counter can be determined by using Eq. (9) depending on the problem dimension and colony size [37]:

$$FC_{\text{Limit}} = SN * D, \tag{9}$$

where FC_{Limit} is the maximum limit for failure counters of the food sources; SN is the total number of food sources, which is equal to half of the colony size; and D is the problem dimension (number of variables in optimization problem). We used Eq. (9) in our implementations.

2.3. The DE algorithm

The DE algorithm is a powerful population-based algorithm. It uses operators very similar to those of the GA. These are mutation, recombination, and selection operators. These operators are applied repeatedly until the termination criteria are met. The main difference between the DE algorithm and the GA in constructing better solutions is that the GA focuses on crossover operation while the DE algorithm focuses on mutation operation. The main operation of the DE algorithm is based on the differences of randomly sampled solution pairs in a population [38]. Details of the DE algorithm operations are as follows. First, the DE algorithm initializes the population randomly. We used Eq. (4) for initialization. Iterative operations then start. The DE algorithm performs a search task by using a mutation operation in which a mutant solution is generated for each solution. Mutant solutions are generated via Eq. (10):

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}), \tag{10}$$

where $v_{i,G+1}$ is the generated mutant solution in the next generation for the ith solution; $x_{r_n,G}$ is a solution in the current generation; r_1, r_2, and r_3 are randomly selected different integer indexes that point to different solutions and they are also different from the running index, i; and F is a real amplification constant ranging from 0 to 2. In summary, three different indexes, which are also different from the running index i, are generated. Solutions pointed out by the generated indexes are then used in Eq. (10) to generate a new neighborhood, which is called a mutant solution. After generating mutant solutions from parent solutions, a recombination operation is performed. In this operation, mutant and parent solutions are mixed via Eq. (11) to generate a new solution, which is called trial solution:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if} \quad randb(j) \le CR \vee j = rnbr(i) \\ x_{ji,G}, & \text{else} \end{cases}, \tag{11}$$

where $u_{ji,G+1}$ is the generated ith trial solution in the jth dimension for the next generation, $v_{ji,G+1}$ is the generated ith mutant solution in the jth dimension for the next generation, $x_{ji,G}$ is the ith current solution in the jth dimension, $randb(j)$ is a random number generated for the jth dimension ranging from 0 to 1, CR is the crossover constant, and $rnbr(i)$ is a randomly chosen dimension index. This index guarantees that the trial solution gets at least one parameter from the mutant solution. A greedy selection among the trial and current solutions is performed in the selection step. Considering the minimization problems, greedy selection

can be defined as:

$$x_{i,G+1} = \begin{cases} u_{i,G+1} , & \text{if} \quad f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} , & \text{else} \end{cases} , \tag{12}$$

where $x_{i,G}$, $x_{i,G+1}$, and $u_{i,G+1}$ are the ith current solution, ith solution in the next generation, and ith trial solution in the next generation, respectively, and f is the cost function. Suitable F and CR values should be chosen to get a good optimization performance. We used F and CR values ranging from 0.2 to 0.8. We used small parameter values for high-dimensional problems and large parameter values for low-dimensional problems.

2.4. The GA

In the GA, the actual solutions, which are called phenotypes, are represented by chromosomes, which are called genotypes. It is essential in the GA to design a proper chromosome representation and to determine a proper fitness calculation method. The phenotypes are converted to genotypes before the application of genetic operators and the genotypes are converted to phenotypes before the fitness calculations.

In the GA process, first, the initial population is created randomly. We used Eq. (4) to generate initial population. The fitness of each solution is then calculated. If the solution with the best fitness value does not meet the termination criteria, then the genetic operators are applied to the chromosomes.

In the application of genetic operators, the first step is determination of the parents. There are several GA implementations and each of them uses different application methods. In our implementations, the number of parent pairs is equal to half of the population size and we used the roulette wheel method for determination of the parent pairs.

The second genetic operator is the crossover. The most common crossover method for binary coded chromosomes is the one-point crossover, which is shown in Figure 1. The same crossover point is selected for each parent in this method. The parts delimited by crossover points are interchanged by the parents. Two new offsprings are generated from two parents by using this method.

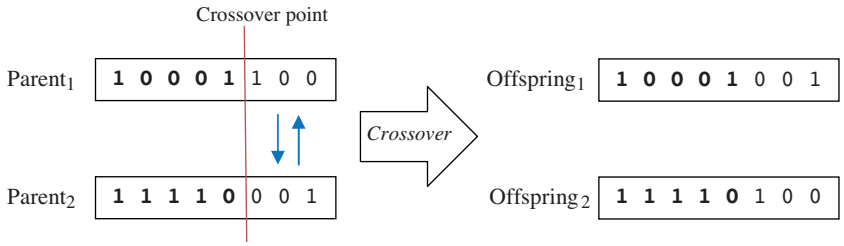

Figure 1. One-point crossover for binary coded chromosomes.

There are several crossover methods for the real coded GAs. We used a method that closely mimics the binary coded GA. It is the combination of an extrapolation method and a crossover method [39]. In this method, a chromosome is thought to be in the form of

$$\text{parent} = [p_1 p_2 \dots p_{N_{var}}], \tag{13}$$

where p represents variables and N_{var} is the number of variables. Let the parents be

$$\begin{aligned} \text{parent}_1 &= [p_{m1} p_{m2} \dots p_{m\alpha} \dots p_{mN_{var}}] \\ \text{parent}_2 &= [p_{d1} p_{d2} \dots p_{d\alpha} \dots p_{dN_{var}}] \end{aligned} , \tag{14}$$

where α is a randomly produced integer ranging from 1 to N_{var}, and subscripts m and d represent mom and dad parents, respectively. The new variables that will appear in offsprings are calculated via

$$\begin{aligned} p_{new1} &= p_{m\alpha} - \beta[p_{m\alpha} - p_{d\alpha}] \\ p_{new2} &= p_{d\alpha} - \beta[p_{m\alpha} + p_{d\alpha}] \end{aligned}, \tag{15}$$

where β is a randomly produced number ranging from 0 to 1. The offsprings with new variables are then generated by replacing $p_{m\alpha}$ and $p_{d\alpha}$ with p_{new1} and p_{new2} respectively and swapping the right side of the selected variables in parents.

$$\begin{aligned} \text{offspring}_1 &= [p_{m1}p_{m2}\dots p_{new1}\dots p_{dN_{var}}] \\ \text{offspring}_2 &= [p_{d1}p_{d2}\dots p_{new2}\dots p_{mN_{var}}] \end{aligned} \tag{16}$$

The third genetic operator is the mutation. For the binary coded GA, a predefined percentage of bits in chromosomes are toggled. On the other hand, because we used a real coded GA, we selected a predefined percentage of variables among the offsprings randomly and then employed Eq. (4) to change their values. The mutation operation enables new regions of the problem space to be included inside the search region. High mutation rate values increase the randomness in the search operation and give rise to divergence from the global optimum. However, mutation rate values that are too low decrease the diversity in the population and cause an insufficient problem space search. We used a 5% mutation rate in our implementations.

2.5. The MBO algorithm

The MBO algorithm is a recently introduced and nature-inspired metaheuristic neighborhood search method. It simulates the V flight formation of migrating birds. V-shaped flight with its induced drag reduction is an effective formation for birds to save energy [2]. For instance, in a V-shaped flight formation consisting of 25 members, each bird can achieve a reduction in induced drag as large as 65%. This could result in a range increase of about 70% [40].

The benefit mechanism of the V flight formation can be explained briefly as follows. A pair of vortices is created owing to the wing movements as seen in Figure 2. Looking in the direction of the flight, the vortices from the left and the right wing tips rotate in clockwise and counter-clockwise directions, respectively [2,40]. Vortices create downwash and upwash for the birds flying behind. Downwash is undesirable because it increases the induced drag on a wing in flight. However, upwash is beneficial because it decreases the induced drag on a wing in flight. Thus, all the birds in the V formation except for the leader bird are located mostly in the upwash regions of vortices. Thus, they get the benefit of these upwashes and reduce their energy consumptions. In summary, the leader bird in that formation is the one expending the most energy and the birds in other positions get benefit from the birds in front [2].

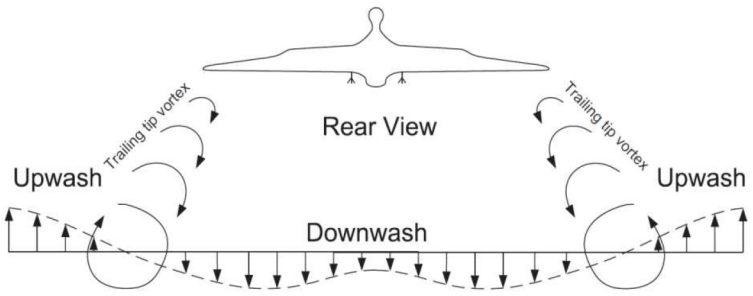

Figure 2. Regions of upwash and downwash created by trailing tip vortices.

A flow diagram of the MBO algorithm is given in Figure 3. It simulates the benefit mechanism of the real birds by sharing the solutions. The parameters of the MBO algorithm are the number of solutions representing the flock size (n), the total number of neighbor solutions to be considered (k), the number of neighbor solutions to be shared with the next solution (x), the number of tours to change the leader (m), and the maximum iteration number (K).

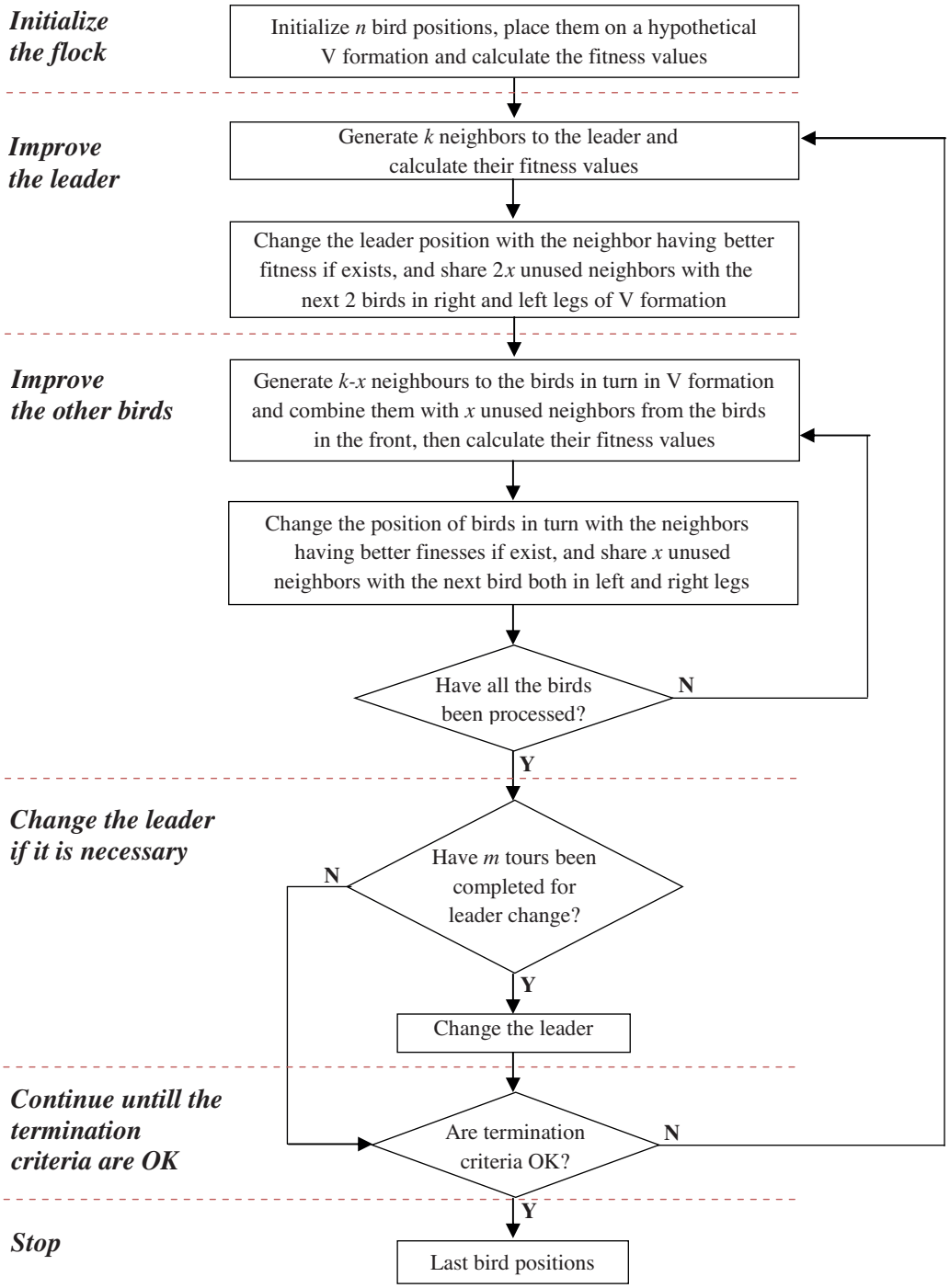

Figure 3. Flow diagram of the MBO algorithm.

The algorithm runs as follows. The population is initialized in the first step. We used the method defined in Eq. (4) to initialize the positions of the birds, which represent possible solutions. After initialization, one of the solutions is chosen as the leader and all of the generated solutions are placed in a hypothetical V formation arbitrarily. Starting with the first solution, which corresponds to the leader bird, and progressing along the lines towards the tails, the MBO algorithm aims to improve each solution by using its neighbor solutions [2].

The leader bird is tried to be improved in the second step. Hence, k neighbors are generated and their fitness values are calculated. We used the method given in Eq. (6) for neighbor creation. After creating these neighbors, if the neighbor solution having the best fitness shows an improvement for the leader, the position of that neighbor solution is assigned to the leader solution and then $2\,x$ unused best solutions are shared with the two birds in the second row.

The other birds are tried to be improved in the third step. For each bird in turn, $(k-x)$ neighbors are generated and their fitness values are calculated. These neighbors are combined with x unused neighbors coming from the birds in the front. Thus, the total number of neighbor solutions to be considered for each bird is k, like in the improvement of the leader bird. If the neighbor solution having the best fitness shows an improvement for the corresponding bird, the position of that neighbor solution is assigned to the corresponding bird, and then x unused best solutions are shared with the next bird. This neighborhood sharing mechanism simulates the benefit of upwash caused by trailing tip vortices in V flight formation. One iteration ends after completing improvement trials for all birds. In short, the leader bird spends the most energy by creating k neighbors in iterations. However, the birds in other positions benefit from the birds in front and spend less energy by creating $(k-x)$ neighbors in iterations.

In the fourth step, the leader bird is thought to be tired after performing a predefined number of iterations (m). The leader solution is then shifted to the end of one side in the hypothetical V formation, and the second solution at that side is shifted to the leader position. We chose $m = 10$ as recommended in [2]. Steps from step 2 to step 4 run until predefined termination criteria are satisfied by the generated solutions. The algorithm then stops and gives the solution having the best fitness as the overall solution.

Parameters k and x directly affect the algorithm performance and should be chosen appropriately. Parameter k is inversely proportional with the flight speed of the real birds. If it is chosen at small values, the birds are assumed to be flying at higher speeds. Higher speeds enable the MBO algorithm to reduce the total execution time. This is an advantageous choice for problems with small number of parameters. However, the search deepness of the algorithm increases if parameter k increases. Therefore, although the execution time increases, k may need to be increased to get a satisfactory solution for high-dimensional problems. Parameter x represents the upwash benefit of trailing tip vortices in a real bird flock. Since the benefit mechanism of the MBO algorithm is defined as the number of good neighbor solutions obtained from the predecessor solution, high values of x cause solutions to be similar to each other. Thus, premature convergence may happen [2]. In our implementations, we chose $x = 1$ to prevent the algorithm from premature convergence, and we chose $k = 3$ to reduce the total execution time. The MBO algorithm makes $k + (population\_size - 1) \times (k - x)$ fitness calculations in one iteration. The total number of fitness calculations in one iteration for the other algorithms is equal to the population size. That is, the total number of fitness calculations in the MBO algorithm for our k and x choices is approximately equal to two times that in the other algorithms. Therefore, we set the total number of iterations in the MBO algorithm implementations to half of those in the other algorithms in order to make fair comparisons between the algorithms.

2.6. System identification by metaheuristics

System identification is a methodology that aims to construct the model of an unknown system and tunes parameters of the model constructed by using experimental data. Methods known as black box modeling use input signal data and the resultant outputs. That is, they are not interested in what the system looks like or what the system is used for [30]. They make parameter adjustment in order to minimize the error between unknown system output and the model output. Gradient-based conventional methods are commonly used for system identification. Because these methods need smooth and differentiable search spaces, they cannot be applied to systems having nondifferentiable multimodal error functions [29]. Because of this fact, researchers have been focused on new system identification methods.

Metaheuristics are systematic search techniques and applicable to system identification problems. If we consider a system with the input $x(n)$ and the output $u(n)$, then the input-output relation of the system can be defined as:

$$u(n) + \sum_{i=1}^{P} a_i u(n-i) = \sum_{i=0}^{R} b_i x(n-i), \tag{17}$$

where $x(n-i)$ and $u(n-i)$ are the values of previous observations and a_i and b_i are the coefficients of these observations. The autoregressive moving average (ARMA) model representation of this system is given in Figure 4 and its transfer function is defined as:

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{i=0}^{R} b_i z^{-i}}{1 + \sum_{i=1}^{P} a_i z^{-i}}. \tag{18}$$

Coefficients of the unknown system transfer function in Eq. (18) can be written in the vector form of:

$$[b_0, b_1, \ldots, b_R, a_1, a_2, \ldots, a_P]. \tag{19}$$

If the system identification problem is thought to be an optimization problem of a population-based metaheuristic algorithm, each member of the population contains the transfer function coefficients described in Eq. (19). Metaheuristic optimization algorithms try to minimize the error $e(n)$ shown in Figure 5 by using systematic techniques. They produce new population members (coefficient vectors) in each iteration and calculate the mean squared errors (MSEs) of these members.

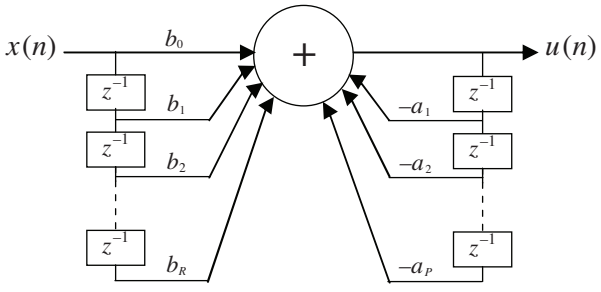

Figure 4. ARMA model representation of an unknown system.

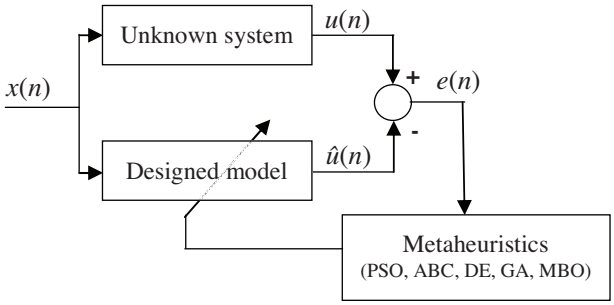

Figure 5. System identification process by using metaheuristics.

The MSE is defined as:

$$E(t) = \frac{1}{N}\sum_{i=1}^{N}(u(i) - \hat{u}(i))^2, \tag{20}$$

where $u(i)$ and $\hat{u}(i)$ are the desired output and model output of the ith input, respectively; N is the total number of inputs used for the test; and $E(t)$ is the MSE at the tth iteration. The system identification process using a metaheuristic runs until the predefined termination criteria are met, as shown in Figure 6. In summary, a metaheuristic algorithm creates new coefficients from the existing coefficients in each iteration. It then makes greedy selections between new and existing coefficients depending on the MSE calculations. In this way, it tries to find much better solutions during the progress of iterations. Finally, it reaches a reasonable solution having minimum error.

3. Problems and experimental setups

3.1. Benchmark function test

A function that has only one maximum or minimum in the region to be searched is called unimodal. Conversely, it is called a multimodal function if it has many local minima or maxima in its search region. Performances of the optimization algorithms on benchmark functions are highly dependent on this characteristic of the benchmark functions. As a generalization, an optimization algorithm that is poor at exploration may not show good performances for high-dimensional multimodal benchmark functions. They require good exploration performance to escape from falling into local minima of the search space. Otherwise, the algorithm may fall into a local minimum trap and may not escape from there. On the other hand, the convergence performance of an algorithm depends on its exploitation characteristics. If an optimization algorithm is good at exploitation, it shows a good convergence performance. Otherwise, it cannot perform satisfactory convergence even if it finds the region including the global minimum. As a result, there should be a fine balance between exploration and exploitation characteristics of an algorithm to perform good optimization.

We used 2-, 5-, 10-, 30-, and 50-dimensional versions of the benchmark functions to test optimization performances of the algorithms. These benchmark functions are given in Table 1. First, 100 different initial populations were created for each selected dimension by using Eq. (4), and they were stored in files. It was aimed to make fair comparisons among the metaheuristics by using the same initial populations from the stored files. Each algorithm was executed 100 times for each dimension of each benchmark function. A new initial population from the stored files was selected for each execution. Average values of the final results are given in Table 2–6. These average values are calculated by using the best 50 results having the smallest cost values among 100 executions. The aim of this experiment is to obtain the average minimum of the benchmark functions for each case and to compare the results.

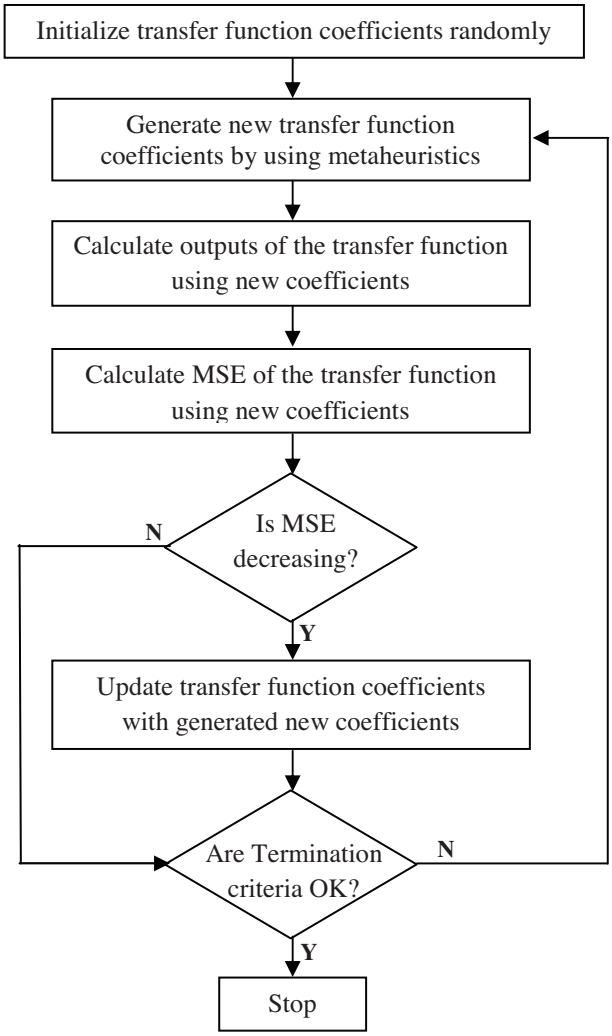

Figure 6. Flow diagram of system identification by using a metaheuristic.

3.2. System identification problems

In this experiment, we used white noise sequences as the input signals $x(n)$. We applied $x(n)$ signals to 5 different systems to get resultant output signals $u(n)$. The second-, third-, fourth-, fifth-, and eighth-order transfer functions of these systems are given in Eqs. (21)–(25) [41].

$$H_1(z) = \frac{2 + 3z^{-1} + 4z^{-2}}{1 - 0.8z^{-1} + 0.15z^{-2}} \tag{21}$$

$$H_2(z) = \frac{1 - 1.4z^{-1} - 1.71z^{-2} + 2.34z^{-3}}{1 - 0.5z^{-1} - 0.29z^{-2} + 0.105z^{-3}} \tag{22}$$

$$H_3(z) = \frac{2 - 1.5z^{-1} + 4.1z^{-2} + 6.8z^{-3} - 2z^{-4}}{1 - 0.2z^{-1} - 0.55z^{-2} + 0.116z^{-3} + 0.042z^{-4}} \tag{23}$$

$$H_4(z) = \frac{1.41 - 2.0404z^{-1} - 0.1739z^{-2} + 1.0973z^{-3} - 0.2595z^{-4} - 0.0339z^{-5}}{1 - 0.443z^{-1} - 1.067z^{-2} + 0.43z^{-3} + 0.187z^{-4} - 0.045z^{-5}} \tag{24}$$

Table 1. Benchmark functions (n: problem dimension, C: characteristic, U: unimodal, M: multimodal).

Function	Definition	Ranges & parameters	Global minimum	C
Sphere	$f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-5.12,5.12]^n$	0	U
Rastrigin	$f_2(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i)\right]$	$[-5.12,5.12]^n$	0	M
Axis parallel hyperellipsoid	$f_3(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$	$[-5.12,5.12]^n$	0	U
Griewangk	$f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	0	M
Michalewicz	$f_5(x) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right]^{2m}$	$[0,\pi]^n$ $(m = 10)$	–1.801 (n= 2) –4.687 (n= 5) :	M
Alpine	$f_6(x) = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i \rvert$	$[-10,10]^n$	0	M
Step	$f_7(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$	$[-100,100]^n$	0	U
Schwefel	$f_8(x) = \sum_{i=1}^{n} \left[-x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right]$	$[-500,500]^n$	$-418.9829n$	M
Ackley	$f_9(x) = -a\exp\left(-b \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(cx_i)\right) + a + \exp(1)$	$[-32.8, 32.8]^n$ $a = 20$, $b = 0.2$, $c = \pi$	0	M
Schawefel	$f_{10}(x) = \max\{\lvert x_i \rvert, 1 \le i \le n\}$	$[-100,100]^n$	0	U

Table 2. Mean cost values reached by the algorithms for 2-dimensional benchmark functions (K: iteration, n: population size).

F	Global min.	MBO $K = 750$ $n = 15$	ABC $K = 1500$ $n = 16$	PSO $K = 1500$ $n = 15$	DE $K = 1500$ $n = 15$	GA $K = 1500$ $n = 16$
f_1	0	0	3.7719E-17	0	0	1.7836E-11
f_2	0	0	3.7067E-04	0	0	3.7503E-08
f_3	0	0	2.8856E-17	0	0	4.9702E-13
f_4	0	3.1104E-03	1.8944E-02	3.2543E-03	0	7.6190E-03
f_5	–1.8013	–1.8013	–1.8013	–1.8013	–1.8013	–1.8013
f_6	0	0	9.7922E-06	0	0	6.8008E-08
f_7	0	0	2.5502E-16	0	0	1.7879E-09
f_8	–8.3797E+02	–8.3797E+02	–8.3796E+02	–7.5032E+02	–8.3797E+02	–1.0440E+04
f_9	0	8.8818E-16	1.9223E-09	8.8818E-16	8.8818E-16	9.7299E-05
f_{10}	0	0	1.8925E-02	0	0	5.6174E-03

Table 3. Mean cost values reached by the algorithms for 5-dimensional benchmark functions (K: iteration, n: population size).

F	Global min.	MBO $K = 1500$ $n = 31$	ABC $K = 3000$ $n = 32$	PSO $K = 3000$ $n = 31$	DE $K = 3000$ $n = 31$	GA $K = 3000$ $n = 32$
f_1	0	0	6.5052E-17	0	0	1.0206E-07
f_2	0	0	0	1.3133	0	1.2427E-05
f_3	0	0	6.9817E-17	0	0	1.2859E-07
f_4	0	1.2911E-02	1.7489E-07	7.7363E-02	0	1.8813E-02
f_5	–4.6877	–4.6877	–4.6877	–4.5642	–4.6877	–4.6877
f_6	0	0	1.6728E-16	1.1463E-16	0	6.5403E-05
f_7	0	0	6.2974E-17	0	0	3.3636E-05
f_8	–2.0949E+03	–2.0949E+03	–2.0949E+03	–1.5765E+03	–2.0949E+03	–1.3374E+04
f_9	0	3.4461E-15	4.2277E-15	4.2277E-15	8.8818E-16	4.3284E-03
f_{10}	0	0	1.3134E-08	0	4.5152E-84	5.0126E-02

Table 4. Mean cost values reached by the algorithms for 10-dimensional benchmark functions (K: iteration, n: population size).

F	Global min.	MBO $K = 2500$ $n = 41$	ABC $K = 5000$ $n = 42$	PSO $K = 5000$ $n = 41$	DE $K = 5000$ $n = 41$	GA $K = 5000$ $n = 42$
f_1	0	0	8.2649E-17	0	0	7.2550E-07
f_2	0	0	0	5.6315	0	1.2293E-04
f_3	0	0	8.8269E-17	0	0	3.7363E-06
f_4	0	2.6046E-02	8.1595E-14	3.3098E-01	0	2.9947E-02
f_5	–9.6600	–9.6359	–9.6602	–8.5504	–9.6602	–9.6601
f_6	0	3.5527E-17	2.4721E-16	4.4720E-16	0	2.1913E-04
f_7	0	0	8.6227E-17	0	0	2.7967E-04
f_8	–4.1898E+03	–4.1849E+03	–4.1898E+03	–2.6354E+03	–4.1898E+03	–1.9546E+04
f_9	0	5.9330E-15	7.9226E-15	6.9988E-15	3.3040E-15	7.3117E-03
f_{10}	0	1.8978E-38	5.3854E-14	7.8663E-86	5.6296E-10	1.0803E-01

Table 5. Mean cost values reached by the algorithms for 30-dimensional benchmark functions (K: iteration, n: population size).

F	Global min.	MBO $K = 3500$ $n = 101$	ABC $K = 7000$ $n = 102$	PSO $K = 7000$ $n = 101$	DE $K = 7000$ $n = 101$	GA $K = 7000$ $n = 102$
f_1	0	0	4.9986E-16	0	0	6.8423E-06
f_2	0	0	0	3.0247E+01	0	1.2403E-03
f_3	0	0	4.3213E-16	0	0	9.0918E-05
f_4	0	0	7.5495E-17	2.4649E-03	0	8.6388E-03
f_5	–2.9583E+01	–2.9566E+01	–2.9631E+01	–2.3646E+01	–2.5930E+01	–2.9628E+01
f_6	0	7.9495E-16	5.7107E-16	1.1644E-12	5.1108E-60	1.0217E-03
f_7	0	0	4.5768E-16	7.3956E-34	0	2.4184E-03
f_8	–1.2570E+04	–1.2555E+04	–1.2569E+04	–6.5279E+03	–1.2569E+04	–6.0910E+04
f_9	0	2.1849E-14	3.0589E-14	3.2365E-14	4.4409E-15	1.1875E-02
f_{10}	0	2.9023E-10	5.5636E-02	1.1495E-06	1.7504E-02	4.0999E-01

Table 6. Mean cost values reached by the algorithms for 50-dimensional benchmark functions (K: iteration, n: population size).

F	Global min.	MBO $K = 4000$ $n = 151$	ABC $K = 8000$ $n = 152$	PSO $K = 8000$ $n = 151$	DE $K = 8000$ $n = 151$	GA $K = 8000$ $n = 152$
f_1	0	0	9.0469E-16	3.5142E-38	0	9.0755E-05
f_2	0	0	0	5.9837E+01	0	1.5338E-02
f_3	0	0	8.2402E-16	1.3864E-36	0	1.9948E-03
f_4	0	0	9.1038E-17	1.6431E-14	0	4.2609E-02
f_5	–4.9513E+01	–4.9544E+01	–4.9624E+01	–3.8896E+01	–3.2365E+01	–4.9590E+01
f_6	0	1.9429E-15	2.1666E-15	1.0866E-07	6.9538E-03	4.0222E-03
f_7	0	0	8.2391E-16	4.1723E-32	0	3.3912E-02
f_8	–2.0949E+04	–2.0933E+04	–2.0949E+04	–9.9439E+03	–2.0949E+04	–1.0752E+05
f_9	0	3.7623E-14	5.5316E-14	9.9224E-13	7.7094E-15	3.4396E-02
f_{10}	0	2.8918E-04	4.2017E-01	4.2010E-02	3.3912E-05	1.5517

$$H_5(z) = \frac{0.01 - 0.041z^{-2} + 0.061z^{-4} - 0.041z^{-6} + 0.01z^{-8}}{1 - 2.472z^{-1} + 4.309z^{-2} - 4.886z^{-3} + 4.477z^{-4} - 2.914z^{-5} + 1.519z^{-6} - 0.5z^{-7} + 0.12z^{-8}} \tag{25}$$

Considering $x(n)$ and $u(n)$ as the input and output signals of an unknown system and considering that transfer function coefficients of this system form a population like in Eq. (19), metaheuristics can be used to optimize these transfer function coefficients. The main philosophy of a metaheuristic system identification process in each iteration is as follows. First, it generates new transfer function coefficients (new population members), then it calculates the system output $\hat{u}(n)$ by using generated new coefficients, and finally it makes greedy selections between existing and new population members depending on the calculated MSE values.

For each system, each algorithm was executed 100 times and the average values of the results were calculated by using the best 50 results having the smallest MSE values among 100 executions. Average results are given in Tables 7 and 8. The same initial populations were also used in calculations to make fair comparisons as mentioned in Section 3.1.

If the system transfer function is higher than the fourth order, it is difficult to find the original transfer function parameters of an unknown system. Since a small change in any coefficient of the transfer function gives rise to a large fluctuation in the system response, algorithms most probably cannot find the original transfer function coefficients at that time. Although the algorithms generally cannot find the original transfer function coefficients for these cases, predicted systems give very close responses to unknown systems' responses. In fact, the predicted systems are another near optimal equivalent of the unknown systems. The MSE reached by the optimization algorithm is the main measurement for the similarity check between the unknown original system and the predicted system. That is, the lower the MSE value is, the higher the similarity between the original and predicted systems is.

4. Results

4.1. Benchmark function test results

The benchmark function test results are given in Tables 2–6. Mean cost values reached by the algorithms for 2-, 5-, 10-, 30-, and 50-dimensional benchmark functions in these tables were calculated by using the method mentioned in Section 3.1. The results can be concluded as follows.

- The ABC algorithm generally performs reasonable optimizations owing to its good exploration, which occurs in the scout bee phase. It generally finds reasonably optimal solutions to all functions in all dimensions. However, it cannot perform good convergence to the global minima. While the MBO and DE algorithms converge to global minima as closely as possible, the ABC algorithm cannot achieve a closer convergence than they accomplish. Its exploitation capability thus needs to be enhanced to get a better convergence performance.

- The PSO algorithm gives reasonable results for unimodal benchmark functions in all dimensions. It performs good convergence owing to its good exploitation capability. However, the results for multimodal functions are not good, especially in high dimensions. This is caused by its weakness in exploration and its dependence on initial population quality.

- Performance of the GA decreases while the problem dimension increases. Since its exploitation capability is at a medium level, it needs many more iterations in order to get good results, and this increases the execution time. The overall optimization performance given in Figure 7 proves that the exploration performance that emerges in the mutation phase is at a medium level. It thus falls into local minima for some high-dimensional functions.

- The DE algorithm achieves reasonable optimizations for all benchmark functions in all dimensions. It has a fine balance between exploration and exploitation.

- The MBO algorithm gives good convergence performance owing to its good exploitation property. This is caused by the fact that the MBO algorithm makes many more calculations in one iteration depending on the value of parameter k. That is, increasing the parameter $k$enables the algorithm to get better optimization results, but this gives rise to an increase in the total number of fitness calculations and total execution time. We took this fact into account to make a fair competition between the algorithms and we determined the maximum iteration numbers to be lower for the MBO algorithm so that an equal number of fitness calculations could be performed for each algorithm in the same experiment.

- Figure 7 shows the overall optimization performances of the algorithms by giving the total number of successful benchmark function tests for each algorithm in each dimension. The MBO algorithm showed the best performance. Among the ten benchmark functions, it found optimal results for nine of them for the 5- and 10-dimensional cases. It found optimal results for all of the benchmark functions for the 2-, 30-, and 50-dimensional cases. It was followed by the DE and ABC algorithms.

- How fast and how stable an algorithm approaches to the global minimum is another important issue to be focused on. Figure 8 shows cost values reached by the algorithms vs. iteration graphs for the 10-dimensional unimodal sphere function and 10-dimensional multimodal Rastrigin function. It is clearly seen that the MBO algorithm reaches reasonable results at lower iterations for the unimodal sphere function, and it reaches the exact global minimum at lower iterations for the multimodal Rastrigin function.

4.2. System identification results

After performing the experimental study on system identification mentioned in Section 3.2, we got the results given in Table 7 for the H_1, H_2, and H_3 systems, and we got the results given in Table 8 for the H_4 and H_5 systems. The results can be concluded as follows.

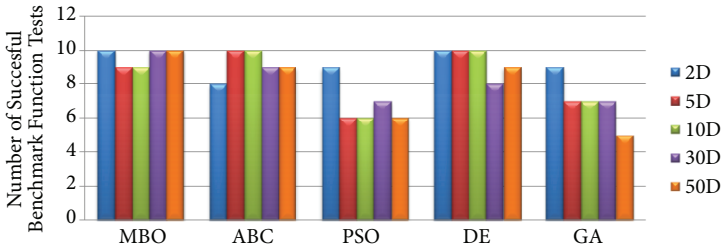

Figure 7. Successful benchmark function test distributions over problem dimensions.

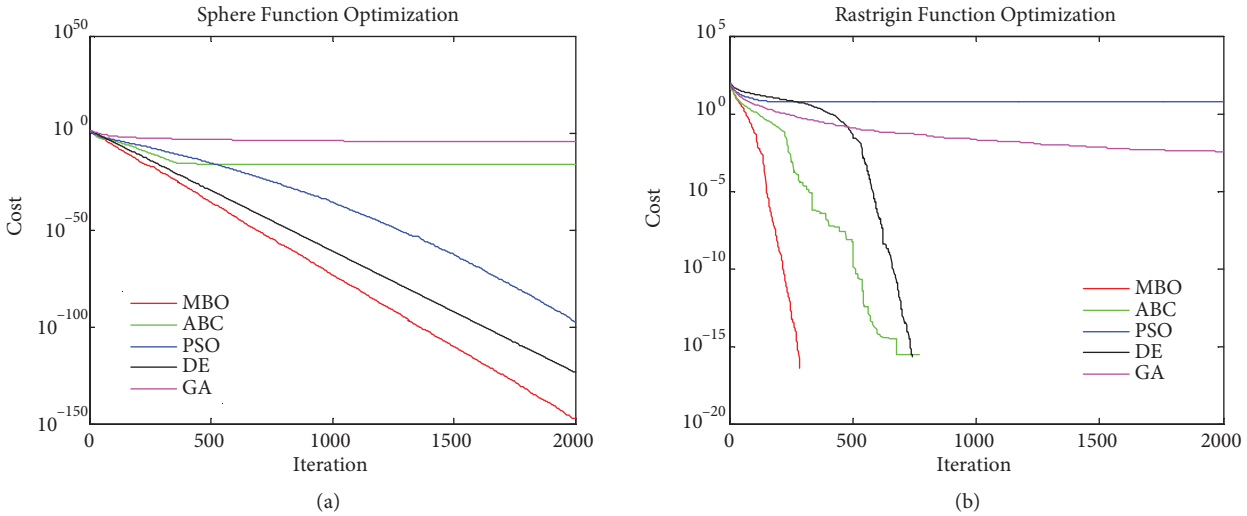

Figure 8. Trends of the cost values reached by the algorithms (a) for 10-dimensional unimodal sphere function and (b) for 10-dimensional multimodal Rastrigin function.

- The GA system identification results mostly have the highest MSE values. The GA needs many more iterations to reach a reasonable MSE value because of its typical characteristics, but this increases the total execution time needed.
- The ABC algorithm results are generally reasonable, but not the best. Its coefficient predictions are very close to the unknown system coefficients for low-order systems, and the MSEs of its predictions are low enough for high-order systems.
- The DE algorithm results are good enough for all systems. In particular, the MSEs of its predictions for low-order systems are very low. For instance, it was able to find the global minimum easily and gave the exact coefficients of H_1 in every trial. Its predictions for the other systems are good enough.
- The PSO algorithm results are reasonable for all systems. In particular, it has a good performance for H_1, like the DE algorithm. Although it was not able to find the exact global minimum in every trial for H_1, it gave nearly exact coefficients.
- The MBO algorithm has good system identification performances for all systems. It minimized the MSEs as desired. For the H_1, H_2, and H_3 systems, it gave prediction values very close to the coefficients of the unknown systems. For the H_4 and H_5 systems, it reached MSE values as low as desired. These low MSE values prove that although it could not get the original transfer function parameters for high-order systems, it gets optimal systems that are very close to the real unknown systems.

Table 7. System identification results for low-order systems H_1, H_2, and H_3 (K: iteration, n: population size).

	Parameters	Unknown system	MBO	ABC	PSO	DE	GA
			$K = 1000$	$K = 2000$			
			$n = 31$	$n = 30$			
H_1	b_0	2.00000	2.00000	2.00170	2.00000	2.00000	2.00440
	b_1	3.00000	3.00050	2.99410	3.00000	3.00000	3.05200
	b_2	4.00000	4.00070	3.98600	4.00000	4.00000	4.07440
	a_1	–0.80000	–0.79982	–0.80235	–0.80000	–0.80000	–0.78049
	a_2	0.15000	0.14985	0.15169	0.15000	0.15000	0.13414
	MSE		4.682E-05	1.344E-03	3.752E-31	0	2.229E-02
			$K = 2000$	$K = 4000$			
			$n = 39$	$n = 38$			
H_2	b_0	1.00000	1.00000	1.00000	1.04660	0.99987	0.99691
	b_1	–1.40000	–1.40000	–1.40000	–1.35940	–1.39960	–1.39370
	b_2	–1.71000	–1.71000	–1.71000	–1.88400	–1.71000	–1.71060
	b_3	2.34000	2.34000	2.34000	2.35870	2.33930	2.32860
	a_1	–0.50000	–0.50000	–0.50000	–0.47877	–0.49975	–0.49537
	a_2	–0.29000	–0.29000	–0.29000	–0.30218	–0.28988	–0.28749
	a_3	0.10500	0.10500	0.10500	0.10968	0.10499	0.10506
	MSE		3.209E-32	8.779E-12	1.238E-02	3.821E-07	4.510E-05
			$K = 3000$	$K = 6000$			
			$n = 51$	$n = 50$			
H_3	b_0	2.00000	2.00120	2.00070	2.00160	2.00320	2.02550
	b_1	–1.50000	–1.43210	–1.49450	–1.41220	–1.41490	–1.26280
	b_2	4.10000	4.06680	4.09970	4.05700	4.06280	4.02140
	b_3	6.80000	6.93280	6.79720	6.97180	6.94220	7.10720
	b_4	–2.00000	–1.72960	–1.98340	–1.65040	–1.66270	–0.99449
	a_1	–0.20000	–0.16594	–0.19833	–0.15596	–0.15881	–0.08655
	a_2	–0.55000	–0.54791	–0.54948	–0.54731	–0.54633	–0.52747
	a_3	0.11600	0.09825	0.11446	0.09305	0.09264	0.03892
	a_4	0.04200	0.04217	0.04217	0.04223	0.04303	0.04672
	MSE		7.113E-05	2.825E-04	7.978E-05	2.862E-04	1.954E-02

- Figure 9 shows the parameter calculation trends of the algorithms for H_1 system modeling. These trends were obtained by averaging the trends of the best 50 models as mentioned in Section 3.2. It is clearly seen that the MBO algorithm reaches near-optimum values at about the 25th iteration. Considering the total fitness calculation, this is equivalent to the 50th iteration of the other algorithms. The others, except for the ABC algorithm, could not reach the desired coefficients at the 50th iteration. This proves that the MBO algorithm finishes the system identification problem faster in comparison with the others.

5. Conclusions

The MBO algorithm is a recently introduced metaheuristic optimization algorithm and has a unique benefit mechanism. This paper is the first study to employ the MBO algorithm in system identification problems. In this paper, good optimization capabilities and the performance of the MBO algorithm were first investigated via a set of benchmark function tests. It was then proved that the identification of an unknown system can be performed easily by using metaheuristics including the MBO algorithm.

Table 8. System identification results for high-order systems H_4 and H_5 (K: iteration, n: population size).

	Parameters	Unknown system	MBO	ABC	PSO	DE	GA
			$K = 2500$	$K = 5000$			
			$n = 59$	$n = 58$			
H_4	b_0	1.41400	1.41240	1.41300	1.46880	1.41040	0.01260
	b_1	–2.04040	–1.24930	–0.98289	–0.84852	–1.19470	0.00831
	b_2	–0.17390	0.22646	0.04390	–0.22938	0.11894	–0.00263
	b_3	1.09730	0.09728	0.02168	0.10685	–0.00642	–0.07510
	b_4	–0.25590	–0.49338	–0.43686	–0.34856	–0.37424	0.00845
	b_5	–0.03390	0.15051	0.12206	0.07243	0.18511	0.18332
	a_1	–0.44300	0.11707	0.30221	0.38575	0.15586	–0.63028
	a_2	–1.06700	–0.22301	–0.16957	–0.26717	–0.25801	0.85377
	a_3	0.43000	0.28252	0.19105	0.11044	0.15311	0.32330
	a_4	0.18700	–0.26819	–0.25548	–0.17571	–0.27800	–0.16278
	a_5	–0.04500	–0.08654	–0.08784	–0.07993	–0.04360	0.47587
	MSE		2.988E-04	5.509E-04	1.784E-03	6.859E-04	3.943E-03
			$K = 4000$	$K = 8000$			
			$n = 91$	$n = 90$			
H_5	b_0	0.01000	0.00958	0.01273	0.00982	0.00771	0.00728
	b_1	0	0.02042	0.04974	0.01985	0.01862	0.01765
	b_2	–0.04100	–0.02847	0.06309	–0.02836	–0.02197	–0.02492
	b_3	0	–0.07883	–0.03790	–0.07672	–0.09756	–0.09398
	b_4	0.06100	–0.01118	0.01724	0.00038	–0.03538	–0.02399
	b_5	0	0.10633	0.07659	0.10181	0.15224	0.14779
	b_6	–0.04100	0.06153	0.05958	0.04461	0.13248	0.10576
	b_7	0	–0.06171	0.01078	–0.05995	–0.08742	–0.09479
	b_8	0.01000	–0.06805	–0.01882	–0.04875	–0.17069	–0.13733
	a_1	–2.47200	–0.42854	–0.16695	–0.52595	–0.15186	–0.27205
	a_2	4.30900	0.82151	0.88426	0.88877	0.31134	0.39061
	a_3	–4.88600	0.36758	–0.17745	0.38345	0.33339	0.31193
	a_4	4.47700	0.20769	0.43342	0.06808	0.13762	0.09285
	a_5	–2.91400	0.28702	–0.07887	0.46050	–0.05005	–0.01935
	a_6	1.51900	0.42684	0.22607	0.31384	0.09981	0.11957
	a_7	–0.50000	–0.12332	–0.00746	–0.08003	–0.00285	–0.02962
	a_8	0.12000	0.19351	0.14485	0.20715	–0.18008	–0.15381
	MSE		1.514E-04	4.453E-02	3.017E-05	2.306E-03	1.825E-03

The system identification process can be thought of as an optimization problem in metaheuristic approaches. Thus, metaheuristics can be employed to optimize the coefficients of transfer functions aiming to minimize the MSE between the model output and real unknown system output. Experimental results show that the MBO algorithm performs good optimizations for the benchmark functions, and it has a good performance to solve the system identification problems. It makes very close predictions to the unknown systems having low-order transfer functions. For the systems with higher-order transfer functions, it can reach the desired low MSE values. These low MSE values prove that even if it cannot get the original transfer function parameters for higher-order systems, it can construct their near-optimal equivalent transfer functions.

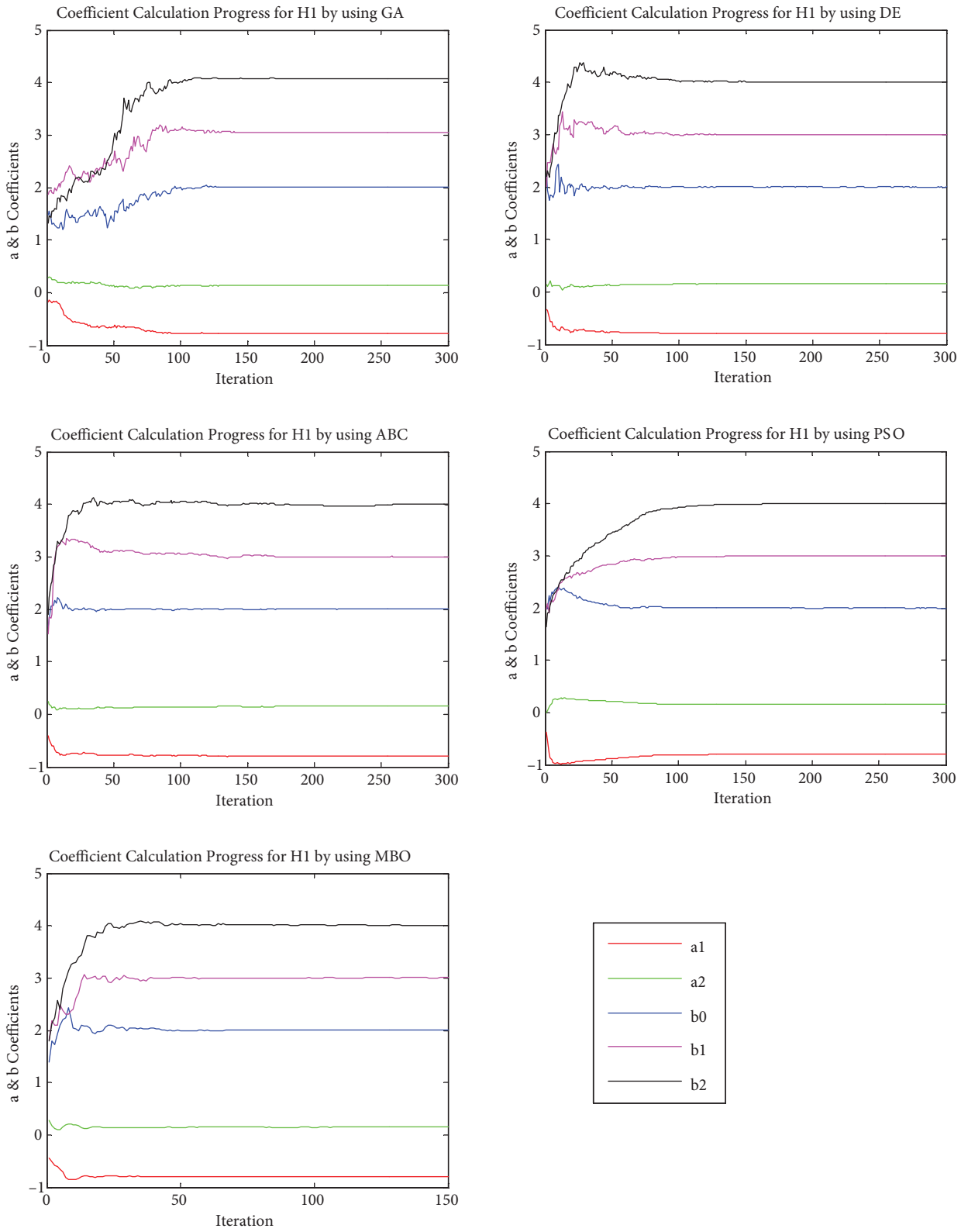

Figure 9. Parameter calculation trends of the algorithms for H_1 system.

References

[1] Boussaïd I, Lepagnot J, Siarry P. A survey on optimization metaheuristics. Inform Sciences 2013; 237: 82-117.

[2] Duman E, Uysal M, Alkaya AF. Migrating birds optimization: a new metaheuristic approach and its performance on quadratic assignment problem. Inform Sciences 2012; 217: 65-77.

[3] Ahuja RK, Ergun O, Orlin JB, Punnen AP. A survey of very large scale neighborhood search techniques. Discrete Appl Math 2002; 123: 75-102.

[4] Holland JH. Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[5] Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983; 220: 671-680.

[6] Glover F. Future paths for integer programming and links to artificial intelligence. Comp Oper Res 1986; 13: 533-549.

[7] Dorigo M. Optimization, learning and natural algorithms. PhD, Polytechnic University of Milan, Milan, Italy, 1992.

[8] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science; 4–6 October 1995, Nagoya, Japan. New York, NY, USA: IEEE. pp. 39-43.

[9] Storn R, Price K. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 1997; 11: 341-359.

[10] Geem ZW, Kim JH, Loganathan GV. A new heuristic optimization algorithm: harmony search. Simulation 2001; 76: 60-68.

[11] Mucherino A, Seref O. A novel meta-heuristic search for global optimization. In: Proceedings of the Conference on Data Mining, System Analysis and Optimization in Biomedicine; 28–30 March 2007; Gainesville, FL, USA. Melville, NY, USA: AIP Publishing. pp. 162-173.

[12] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Global Optim 2007; 39: 459-471.

[13] Yang XS. Nature-Inspired Metaheuristic Algorithms. 2nd ed. Frome, UK: Luniver Press, 2010.

[14] Shah-Hosseini H. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. International Journal of Bio-Inspired Computation 2009; 1: 71-79.

[15] Yang XS, Deb S. Cuckoo search via Lévy flights. In: Proceedings of World Congress on Nature & Biologically Inspired Computing (NaBIC); 9–11 December 2009; Coimbatore, India. New York, NY, USA: IEEE. pp. 210-214.

[16] Yang XS, Deb S. Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling & Numerical Optimisation 2010; 1: 330-343.

[17] Yang XS. A new metaheuristic bat-inspired algorithm. In: Cruz C, González JR, Krasnogor N, Pelta DA, Terrazas G, editors. Nature Inspired Cooperative Strategies for Optimization (Studies in Computational Intelligence 284). Granada, Spain: Springer, 2010. pp. 65-74.

[18] Yang XS. Bat algorithm for multi-objective optimisation. International Journal of Bio-Inspired Computation 2011; 3: 267-274.

[19] Eiben AE, Smit SK. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 2011; 1: 19-31.

[20] Pandian SMV, Thanushkodi K. Considering transmission loss for an economic dispatch problem without valve-point loading using an EP-EPSO algorithm. Turk J Electr Eng Co 2012; 20: 1259-1267.

[21] Mutluer M, Bilgin O. Application of a hybrid evolutionary technique for efficiency determination of a submersible induction motor. Turk J Electr Eng Co 2011; 19: 877-890.

[22] Jaddi NS, Abdullah S. Hybrid of genetic algorithm and great deluge algorithm for rough set attribute reduction. Turk J Electr Eng Co 2013; 21: 1737-1750.

[23] Fornarelli G, Giaquinto A. An unsupervised multi-swarm clustering technique for image segmentation. Swarm and Evolutionary Computation 2013; 11: 31-45.

[24] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 2003; 35: 268-308.

[25] Lennart L. System Identification: Theory for the User. Upper Saddle River, NJ, USA: Prentice Hall, 1997.

[26] Rosenqvist F, Karlström A. Realisation and estimation of piecewise-linear output-error models. Automatica 2005; 41: 545-551.

[27] Pintelon R, Schoukens J. Box-Jenkins identification revisited - Part I: theory. Automatica 2006; 42: 63-75.

[28] Karaboğa N, Çetinkaya MB. A novel and efficient algorithm for adaptive filtering: artificial bee colony algorithm. Turk J Electr Eng Co 2011; 19: 175-190.

[29] Kalınlı A. Simulated annealing algorithm-based Elman network for dynamic system identification. Turk J Electr Eng Co 2012; 20: 569-582.

[30] Erçin Ö, Çoban R. Identification of linear dynamic systems using the artificial bee colony algorithm. Turk J Electr Eng Co 2012; 20 (Suppl. 1): 1175-1188.

[31] Ercin O, Coban R. Comparison of the artificial bee colony and the bees algorithm for PID controller tuning. In: Proceedings of the International Symposium on Innovations in Intelligent Systems and Applications (INISTA); 15–18 June 2011; İstanbul, Turkey. New York, NY, USA: IEEE. pp. 595-598.

[32] Karaboga N. A new design method based on artificial bee colony algorithm for digital IIR filters. J Frankl Inst 2009; 346: 328-348.

[33] Engelbrecht AP. Fundamentals of Computational Swarm Intelligence. New York, NY, USA: Wiley, 2006.

[34] Shi Y, Eberhart RC. Empirical study of particle swarm optimization. In: Proceedings of the CEC'99 Congress of Evolutionary Computation; 6–9 July 1999; Washington, DC, USA. New York, NY, USA: IEEE. pp. 1945-1950.

[35] Eberhart RC, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the IEEE International Congress on Evolutionary Computation; 16–19 July 2000; La Jolla, CA, USA. New York, NY, USA: IEEE. pp. 84-88.

[36] Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. Appl Math Comput 2009; 214: 108-132.

[37] Karaboga D, Basturk B. On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 2008; 8: 687-697.

[38] Karaboga D, Okdem S. A simple and global optimization algorithm for engineering problems: differential evolution algorithm. Turk J Electr Eng Co 2004; 12: 53-60.

[39] Haupt RL, Haupt SE. Practical Genetic Algorithms. 2nd ed. New York, NY, USA: Wiley, 2004.

[40] Lissaman PBS, Schollenberger CA. Formation flight of birds. Science 1970; 168: 1003-1005.

[41] Makas H. PORLA metodunun sistem modellemedeki performans analizi. MSc, Erciyes University, Kayseri, Turkey, 1998 (in Turkish).