Electronic Letters on Science & Engineering 11(1) (2015)
Available online at www.e-lse.org

Classification using neural networks trained by swarm intelligence

Hasan Makas [1,*] Nejat Yumusak[2]
[1]Ereğli Iron & Steel Works Co., Department of Electronic Automation,
67330, Karadeniz Ereğli, Zonguldak, Turkey
[2]Sakarya University, Faculty of Computer and Information Sciences,
Department of Computer Engineering, 54187, Serdivan, Sakarya, Turkey

Abstract: The metaheuristics are the algorithms that are designed to solve many optimization problems without needing knowledge about the corresponding problems in detail. Similar to other metaheuristics, the Migrating Birds Optimization (MBO) algorithm which is introduced recently is a nature inspired neighbourhood search method. It simulates migrating birds' V flight formation which is an effective flight shape for them to save the energy. In this paper, 20 different data sets were used for classification. Firstly, the MBO algorithm was employed to train neural networks which were designed for classification. Then, the same networks were trained by using other well-known powerful metaheuristic algorithms. These are the Artificial Bee Colony (ABC) algorithm, the Particle Swarm Optimization (PSO) algorithm, the Differential Evolution (DE) algorithm and the Genetic Algorithm (GA). Finally, the Levenberq-Marquardt (LM) algorithm, a classical gradient based training method, was added to implementations so that clear comparisons could be done among algorithm performances. Results show that the MBO algorithm has better performance than the others' performances. It gets the highest accuracies in tests and reaches to the lowest mean squared errors in trainings for most of the experiments.

***Keywords**:* Migrating birds optimization; swarm intelligence; neural network training; neighbourhood search; classification

Özet: Meta-sezgisel algoritmalar ilgili problem hakkında detaylı bilgiye ihtiyaç duymaksızın pek çok optimizasyon problemini çözebilecek şekilde tasarlanmış algoritmalardır. Yeni bir algoritma olan Göçmen Kuşlar Optimizasyon (GKO) algoritması diğer meta-sezgisellere benzer şekilde doğadan esinlenilerek oluşturulmuş sistematik bir komşuluk araştırma yöntemidir. Algoritma, harcanan enerjiyi minimize etmek için göçmen kuşların kullandıkları efektif bir uçuş şekli olan V uçuş biçimini simüle eder. Bu makalede 20 farklı veri seti kullanılarak ilgili sınıflandırma problemlerine çözümler aranmıştır. İlk olarak, ilgili sınıflandırma problemleri için tasarlanan Yapay Sinir Ağlarının (YSA) eğitimlerinde GKO algoritması kullanılmıştır. Daha sonra, aynı YSA'lar diğer güçlü ve yaygın meta-sezgisel algoritmalar kullanılarak eğitilmişlerdir. Bu algoritmalar yapay arı koloni algoritması, parçacık sürü optimizasyon algoritması, fark gelişim algoritması ve genetik algoritmadır. Son olarak, algoritmalar arasında net mukayeseler yapılabilmesi için eğim tabanlı klasik bir eğitim yöntemi olan Levenberq-Marquardt algoritması çalışmaya ilave edilmiştir. Sonuçlar, GKO algoritmasının YSA eğitimindeki performansının diğerlerininkine nazaran daha iyi olduğunu göstermiştir. GKO algoritması deneylerin çoğunda en yüksek doğruluk ve en düşük ortalama karesel hata değerlerine ulaşmıştır.

Anahtar Kelimeler: Göçmen kuşlar optimizasyon algoritması; sürü zekâsı; yapay sinir ağı eğitimi; komşuluk araştırması; sınıflandırma

1. Introduction

The metaheuristic methods which are based on swarm intelligence are getting more and more popular in the areas of optimization, neural network training, scheduling, clustering, classification etc. These algorithms can be thought as higher level heuristics in comparison with problem-specific heuristic methods [1-2]. They are generally applied to the problems for which there is no satisfactory problem-specific algorithm to solve.

Most of the metaheuristic algorithms are neighbourhood search methods. They constitute an important and large class among improvement algorithms [3]. They search neighbourhoods of

[*] *Corresponding author;* e-mail : hasanmakas@gmail.com

the existing solutions to get much better solutions. So, the choice of neighbourhood structure is thought to be a critical issue for design of the corresponding neighbourhood search method [4].

A great many metaheuristic algorithms have been introduced by researchers so far. Some of the most popular metaheuristics are the GA [5], the Simulated Annealing (SA) algorithm [6], the Tabu Search (TS) algorithm [7], the Ant Colony Optimization (ACO) algorithm [8] and the PSO algorithm [9]. Some of the recent well-known metaheuristic algorithms are the DE algorithm [10], the Harmony Search (HS) algorithm [11], the Monkey Search (MS) algorithm [12], the ABC algorithm [13], the Firefly Algorithm (FA) [14], the Intelligent Water Drops (IWD) algorithm [15], the Cuckoo Search (CS) algorithm [16-17], the Bat Algorithm (BA) [18-19] and the MBO algorithm [3].

Metaheuristics are generally inspired by the nature. Mobile agents interact locally, and they somehow generate self-organized attitude under the right conditions to perform global convergence. Agents are aided by randomization to increase diversity of solutions on global scale when they explore the search space locally. Thus, there is a good balance between global exploration and local powerful exploitation [20]. In addition, swarming agents are suitable to work in parallel and this leads to a reduction in computation time.

A variety of Artificial Neural Network (ANN) training techniques are available in literature such as the Error Back-Propagation (EBP) algorithm [21-22], the LM algorithm [23-24], the Scaled Conjugate Gradient (SCG) algorithm [25] etc. In these methods, ANNs are trained in order to minimize the mean squared error (MSE) at network output by using gradient of the error function. They need differentiable search spaces (error functions). So, they may not give reasonable results for non-differentiable search spaces by falling into local minimum traps of the error functions. On the other hand, since population based metaheuristic search algorithms are independent from the search space characteristics, they can give good performances on trainings of ANNs. They approach the training process as an optimization problem, and they perform training without needing much knowledge about the real system.

In this paper, we especially concentrated on ANN training performance of the MBO algorithm by using 20 different data sets. We performed a comparison among the GA, DE, PSO, ABC and the MBO algorithms. We also added the LM algorithm training results for the same data sets to emphasize superiority of swarm intelligence in ANN training. For comparison among the algorithms, we focussed on the MSE values in training stage as optimization performance criterion and on the accuracy values in test stage as training performance criterion. The paper is organized as follows. Section 2 introduces theoretical backgrounds of the GA, DE, PSO, ABC and the MBO algorithms briefly and gives the tricks about ANN training by using metaheuristics. Section 3 describes the performance evaluation method in test stage, gives brief information about data sets used in this paper and introduces the experimental setups. Section 4 shows experimental results and gives their discussions. Section 5 concludes the whole study.

2. Methods and background

2.1. The PSO algorithm

The PSO algorithm is a swarm-intelligence based metaheuristic algorithm inspired by the behaviour of birds' flocking. Algorithm deals with the swarm of particles as population. Each member of the swarm represents a possible solution for the corresponding optimization problem, and its position is calculated by adding its velocity in next time step to its current position as [26]

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

where x_i and v_i are the position and the velocity of particle i respectively. Velocity vector conducts the optimization process, and velocity of particle i in dimension j for next time step is described by

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

where c_1 and c_2 are acceleration coefficients, w is inertia weight, rand is uniform random number ranging from 0 to 1, x_{ij} is current position of particle i in dimension j, p_{ij} is the best position achieved by particle i in dimension j and p_{gj} is the best position achieved by population so far in dimension j. Second and third terms in Eq. (2) are called cognition and social terms respectively.

Particle velocities should be in a predefined range of [-V_{max}, V_{max}] to keep the global search capability under control. So, calculated velocities are shifted to these predefined limits when they exceed the limits. Inertia weight w in Eq. (2) is an important variable to control the convergence characteristic of the PSO algorithm. If inertia weight is greater than 1, velocities converge to the limit velocities and swarm diverges from global minimum. Higher values of w in the range of [0, 1] increase global search capability and decrease local search capability of the PSO algorithm or vice versa. In the study of Shi and Eberhart, it was reported that performance of the PSO algorithm increases significantly if w decreases from 0.9 to 0.4 linearly through the iterations [27]. So, we used this linear decreasing method in our benchmark tests.

Diversity of initial population directly affects performance of the PSO algorithm. So, initialization of the population is an important process to get better optimization results. Initial positions can be assigned to the particles by using

$$x_{ij} = x_j^{min} + \text{rand} \cdot \left(x_j^{max} - x_j^{min}\right) \tag{3}$$

where x_{ij} is the position of solution i in dimension j, x_j^{min} and x_j^{max} are minimum and maximum limit values for dimension j, and rand is the uniform random number ranging from 0 to 1. Initial positions are assigned to initial personal best positions, and zero or small random values can be assigned to initial velocities.

2.2. The ABC algorithm

The ABC algorithm is a swarm-intelligence based metaheuristic algorithm inspired by bees' foraging behaviours. The position of each food source represents an available solution for the corresponding optimization problem, and its nectar amount matches to the fitness of this solution [28].

There are three phases in calculations of each iteration. These are employed bee phase, onlooker bee phase and scout bee phase. firstly, the algorithm generates randomly distributed initial solutions to the food sources via Eq. (3) at the beginning. Then, algorithm phases start running. Finally, the algorithm stops when termination criteria are met. The first phase of calculations is the employed bee phase in which employed bees try to enhance their own solutions via

$$\hat{x}_{ij} = x_{ij} + \phi \cdot \left(x_{ij} - x_{kj}\right) \tag{4}$$

where x_{ij} is the position of solution i in dimension j, k is randomly selected index value different from i, x_{kj} is the position of solution k in dimension j, ϕ is a random number ranging from -1 to 1, $\hat{x}_{ij}$ is generated new neighbourhood position in dimension j for solution i. If $\hat{x}_{ij}$ exceeds predefined problem space limits x_j^{min} and x_j^{max}, then it is shifted to those limit values.
Selections between existing food sources and generated new neighbours are performed by using a greedy selection mechanism based on fitness values. Fitness value of a solution is calculated by

$$Fitness_i = \begin{cases} 1/(1+f_i) & , \ f_i \geq 0 \\ 1+\text{abs}(f_i) & , \ f_i < 0 \end{cases} \tag{5}$$

where f_i and $Fitness_i$ are cost and fitness values of source i respectively. After performing greedy selections, selection probabilities by onlooker bees for updated new sources are calculated via

$$p_i = Fitness_i \Big/ \sum_{j=1}^{SN} Fitness_j \tag{6}$$

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

In the second phase, onlooker bee phase, onlooker bees make their selections according to the corresponding p_i probabilities by using roulette wheel method. In other words, the higher the selection probability is, the more chance the corresponding food source can be selected and enhanced by onlooker bees. Each onlooker generates a neighbourhood to its selection via Eq. (4). Then, it makes a greedy selection between the selected source and new generated source in order to enhance its own selection.

In the third phase, scout bee phase, new food sources are generated by scouts for the sources whose nectar are abandoned. In implementations of the algorithm, a fail counter is assigned to each source. After completing a neighbourhood creation by corresponding bee, this counter is increased by one for an unsuccessful trial or set to zero for a successful trial [29]. Whenever the fail counter of a food source exceeds a predefined limit value, employed bee of this source is transformed into a scout bee to find a new food source by making global search via Eq. (3). The scout bee is re-transformed into an employed bee after it generates a new source by this method.

2.3. The GA

The GA simulates genetic evolution process. Firstly, initial population of the GA is created randomly by using Eq. (3) at the beginning, and fitness values of the solutions are calculated. Then, genetic operators are applied to population until the best solution meets the termination criteria.

Parents are determined in the first step. In our implementations, the number of parent pairs is equal to half of the population size, and roulette wheel method is used to determine parent pairs before crossover. Since we used real coded GA in implementations, we used a combination of extrapolation and crossover methods for crossover operation as follows [30]. Let chromosomes are

$$parent = [\ p_1\ p_2 \cdots p_{N_{\text{var}}}\ ] \tag{7}$$

where p represents variables and N_{var} is the number of variables. The parents are

$$\begin{aligned} parent_1 &= [\, p_{m1}\ p_{m2} \cdots p_{m\alpha} \cdots p_{mN_{\text{var}}} \,] \\ parent_2 &= [\, p_{d1}\ p_{d2} \cdots\ p_{d\alpha} \cdots\ p_{dN_{\text{var}}} \,] \end{aligned} \tag{8}$$

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

$$\begin{aligned} p_{new1} &= p_{m\alpha} - \beta\,[p_{m\alpha} - p_{d\alpha}] \\ p_{new2} &= p_{d\alpha} - \beta\,[p_{m\alpha} + p_{d\alpha}] \end{aligned} \tag{9}$$

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

$$\begin{aligned} offspring_1 &= [\, p_{m1}\ p_{m2} \cdots p_{new1}\ \cdots\ p_{dN_{\text{var}}} \,] \\ offspring_2 &= [\, p_{d1}\ p_{d2} \cdots\ p_{new2} \cdots\ p_{mN_{\text{var}}} \,] \end{aligned} \tag{10}$$

Third genetic operator is the mutation. For the mutation operation, we selected a predefined percentage of variables from offsprings randomly; then, we employed Eq. (3) to change the selected variables. High mutation rates increase the randomness in search operation and cause divergence from optimum. On the other hand, too low mutation rates decrease the diversity in population and give rise to insufficient problem space search. We used 5% mutation rate in our implementations.

2.4. The DE algorithm

The DE algorithm has very similar operators with GA. These are mutation, recombination and selection. The main difference of the DE algorithm is that the GA focuses on crossover; however, the DE algorithm focuses on mutation. DE's basic calculation is performed by using differences of randomly sampled solution pairs selected from a population [31]. Firstly, the algorithm initializes the population by using Eq. (3). It performs search task by using mutation operation. Mutant solutions are generated for the solutions by using

$$v_{i,G+1} = x_{r_1,G} + F \cdot \left(x_{r_2,G} - x_{r_3,G}\right) \tag{11}$$

where $v_{i,G+1}$ is generated mutant solution in next generation for solution i; $x_{r_n,G}$ is a solution in current generation; r_1, r_2 and r_3 are randomly selected different integer indexes which point different solutions and none of them is equal to running index i, and F is real amplification constant.
Then, mutant and parent solutions are mixed in recombination operation to generate trial solutions. Trial solutions are defined as

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} \ , & \text{if } randb(j) \le CR \vee j = rnbr(i) \\ x_{ji,G} \ , & \text{else} \end{cases} \tag{12}$$

where $u_{ji,G+1}$ is generated trial solution *i* in dimension *j* for next generation; $v_{ji,G+1}$ is generated mutant solution *i* in dimension *j* for next generation; $x_{ji,G}$ is current solution *i* in dimension *j*; *randb*(*j*) is a random number generated for dimension *j* ranging from 0 to 1; *CR* is the crossover constant and *rnbr*(*i*) is randomly chosen dimension index. Finally, among the trial and current solutions, a greedy selection is performed in the selection operation. Algorithm runs until termination criteria are met.

2.5. The MBO algorithm

The MBO algorithm is a recently introduced another nature inspired metaheuristic neighbourhood search method. It simulates V flight formation of migrating birds. With its induced drag reduction, V shaped flight is an effective formation for birds to save the energy [3]. For instance, each one of the birds can reduce the induced drag up to 65% in a V shaped flight formation having 25 members. Thus, an increase of 70% for maximum flight distance can be achieved [32]. The benefit mechanism of this formation can be explained briefly as follows. A pair of vortices, which is seen in Fig. 1, is created owing to the wing movements. If it is examined in flight direction, the vortices from the left and right wing tips rotate in clockwise and counter clockwise directions respectively [3,32]. Vortices create downwash and upwash forces for the birds just flying behind. Downwash force is undesirable because it increases the induced drag on bird's wings. On the other hand, upwash force is beneficial because it decreases the induced drag on a wing in flight. So, all the birds in V formation except for the leader bird locate mostly in upwash regions of vortices. Thus, they get benefit of this upwash forces and reduce their energy consumption. To sum up, the leader bird in that formation spends the most energy and the other birds get benefit coming from the birds in front [3].

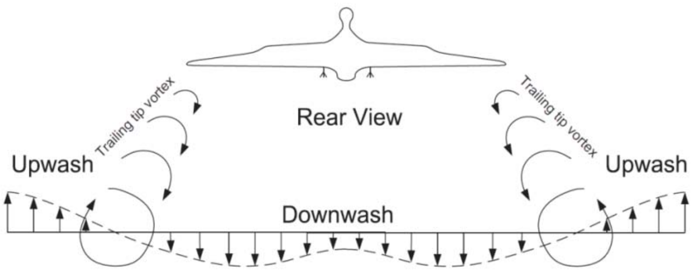


Fig. 1. Regions of upwash and downwash created by trailing vortices [3]

The flow diagram of the MBO algorithm, which simulates this benefit mechanism by sharing the solutions, is given in Fig. 2. The parameters of the MBO algorithm are the number of solutions or flock size, which is represented by *n*; the number of neighbour solutions to be tried for improvement, which is represented by *k*; the number of neighbour solutions for sharing with the next solution, which is represented by *x*; the number of tours to change the leader, which is represented by *m*; and the maximum iteration or tour number, which is represented by *K*.

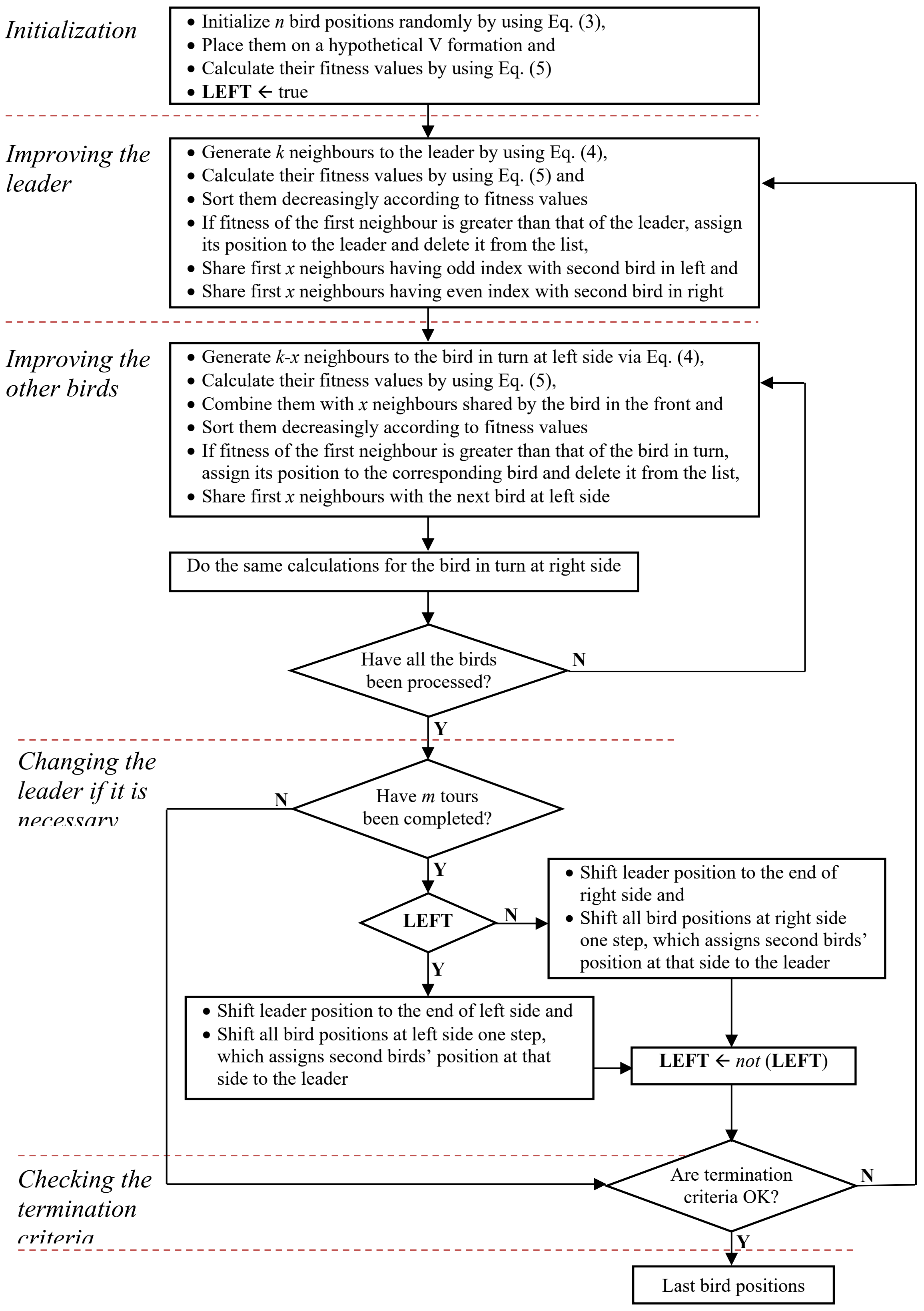

Initialization
• Initialize n bird positions randomly by using Eq. (3),
• Place them on a hypothetical V formation and
• Calculate their fitness values by using Eq. (5)
• LEFT ← true
Improving the leader
• Generate k neighbours to the leader by using Eq. (4),
• Calculate their fitness values by using Eq. (5) and
• Sort them decreasingly according to fitness values
• If fitness of the first neighbour is greater than that of the leader, assign its position to the leader and delete it from the list,
• Share first x neighbours having odd index with second bird in left and
• Share first x neighbours having even index with second bird in right
Improving the other birds
• Generate k-x neighbours to the bird in turn at left side via Eq. (4),
• Calculate their fitness values by using Eq. (5),
• Combine them with x neighbours shared by the bird in the front and
• Sort them decreasingly according to fitness values
• If fitness of the first neighbour is greater than that of the bird in turn, assign its position to the corresponding bird and delete it from the list,
• Share first x neighbours with the next bird at left side
Do the same calculations for the bird in turn at right side
Have all the birds been processed?
N
Y
Changing the leader if it is necessary
Have m tours been completed?
N
Y
LEFT
N
Y
• Shift leader position to the end of right side and
• Shift all bird positions at right side one step, which assigns second birds' position at that side to the leader
• Shift leader position to the end of left side and
• Shift all bird positions at left side one step, which assigns second birds' position at that side to the leader
LEFT ← not (LEFT)
Checking the termination criteria
Are termination criteria OK?
N
Y
Last bird positions

Fig. 2. Flow diagram of the MBO algorithm

The algorithm runs as follows. In the first step, the population is initialized. We used the method defined in Eq. (3) to initialize positions of the birds which represent possible solutions. After initialization, one of the solutions is chosen as the leader and all of the solutions are placed on a hypothetical V formation arbitrarily.

In the second step, the leader bird is tried to be improved. Hence, *k* neighbours are generated and their fitness values are calculated. We used the neighbour creation method given in Eq. (4). If the best neighbour solution shows an improvement on leader, its position is assigned to the leader bird and 2*x* unused best solutions are shared with 2 next birds in second row.

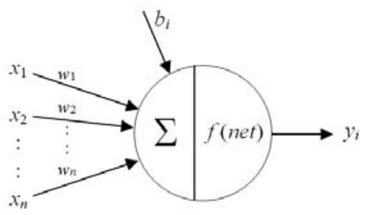
In the third step, other birds are tried to be improved. For each bird in lines, (*k*-*x*) neighbours are generated by using Eq. (4) and their fitness values are calculated. These neighbours are combined with *x* unused neighbours coming from the bird in the front. So, total number of neighbour solutions to be considered for each bird is *k* like in the leader bird. If the best neighbour solution shows an improvement on the corresponding bird, the position of that neighbour solution is assigned to the corresponding bird. Then, *x* unused best solutions are shared with the next bird. This neighbourhood sharing mechanism simulates the benefit of upwash force caused by trailing tip vortex in V flight formation. One iteration ends after completing improvement trials for all birds. Briefly, algorithm starts from the first solution which corresponds to the leader bird, and progresses on two lines towards the tails in each iteration in order to improve each solution by using its neighbour solutions [3].

In the fourth step, the leader bird is thought to be tired after performing a predefined number of iterations (*m*). If predefined iteration is reached, the leader solution is shifted to the end of one side on hypothetical V formation and the second solution at that side is assigned to the leader position. Steps from step 2 to step 4 continue until predefined termination criteria is satisfied by generated solutions. Then, the algorithm stops and gives solution with the best fitness value as overall solution.

Parameter *k* and *x* directly affect the algorithm performance and should be chosen appropriately. Parameter *k* is inversely proportional with the flight speed of real birds. If it is chosen at small values, it is assumed that the birds are flying at higher speeds. For the MBO algorithm, higher speeds enable algorithm to reduce total execution time. This is an advantageous choice for the problems having small number of parameters. But for high dimensional problems, *k* may need to be increased to get a satisfactory solution. Parameter *x* represents the upwash benefit of trailing tip vortex in bird flock. Since benefit mechanism of the MBO algorithm is defined as the number of good neighbour solutions obtained from the predecessor solution, high values of *x* cause solutions to be similar to each other. Thus, a premature convergence may happen. Thus, we choose $x = 1$ as recommended [3].

2.6. Training feed-forward ANNs by using metaheuristics

An ANN consists of a set of interconnected processing elements which are known as neurons [33]. Schematic representation of a neuron is given in Fig. 3a. The neurons in the ANN form a topology like depicted in Fig. 3b which shows the ANN having *n* inputs, *m* hidden layer neurons and *k* outputs. Each neuron in this topology gets the signals coming from the neurons located in previous layer. These signals are multiplied by weight values before they reach to the corresponding neuron. The neuron sums these weighted signals and its bias value. Then, it applies the summation result to the transfer function and transmits the transfer function output to the neurons in next layer.



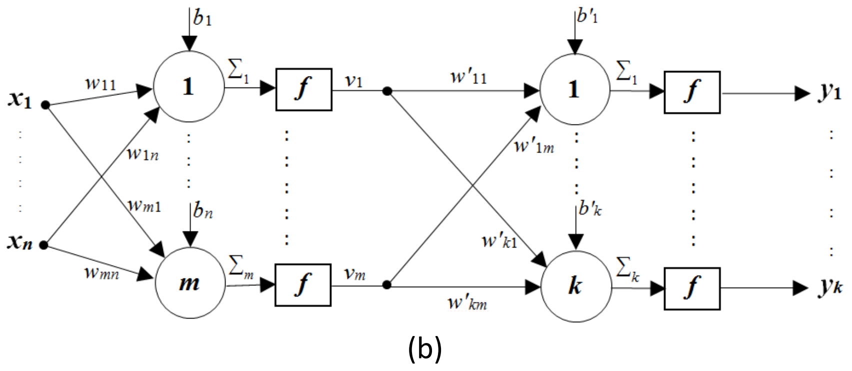


Fig. 3 (a) Neuron of an ANN (b) An ANN with *n-m-k* topology

The relation between input signals and i^{th} output signal of the ANN is given by

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij} x_j + b_i\right) \tag{13}$$

where y_i is i^{th} neuron's output value, x_j is j^{th} input value, w_{ij} is the connection weight between i^{th} neuron and j^{th} input, b_i is neuron's bias value and f_i is the transfer function. Since an ANN is used to find a possible solution to a nonlinear problem, nonlinear transfer functions such as logarithmic sigmoid given in Eq. (14) are used. The input signals are normalized in accordance with the selected transfer function before they are applied to the ANN.

$$f(x) = 1/(1+e^{-x}) \tag{14}$$

An ANN training algorithm aims to minimize the MSE at output during training process. The lower the MSE is, the higher the output accuracy for training data set is. The MSE of an ANN at t^{th} iteration is calculated by

$$MSE(t) = \frac{1}{P}\sum_{j=1}^{P}\sum_{i=1}^{K}(d_i - y_i)^2 \tag{15}$$

where d_i and y_i are desired and actual outputs respectively, P is the total number of input patterns existing in training data set and K is the number of nodes at output.
An ANN like in Fig. 3b can be thought as an optimization problem having many parameters. These parameters consist of connection weights and bias values. In this context, considering an ANN having only one hidden layer, total number of parameters to be optimized is calculated by

$$N_{par} = n \cdot m + m \cdot k + m + k \tag{16}$$

where N_{par} is the total number of parameters, n is the number of inputs, m is the total number of hidden layer neurons and k is the total number of output layer nodes. Similarly, considering an ANN having two hidden layers, total number of parameters to be optimized is calculated by

$$N_{par} = n \cdot m_1 + m_1 \cdot m_2 + m_2 \cdot k + m_1 + m_2 + k \tag{17}$$

where m_1 and m_2 are number of neurons in first and second hidden layers respectively.

In ANN training, metaheuristic algorithms start optimization process for N_{par} parameters. Aim of this optimization is to minimize the MSE at the output of the ANN. Metaheuristics calculate new connection weights and bias values in each iteration, and the ANN is run with these new parameters. Then, MSE is calculated for the last run and a decision is made about whether the parameters are updated or not. If the MSE decreases by using new parameters, the parameters are updated with last calculated parameters. Otherwise, existing parameters are maintained. A general flow diagram of the ANN training via metaheuristics is shown in Fig. 4.

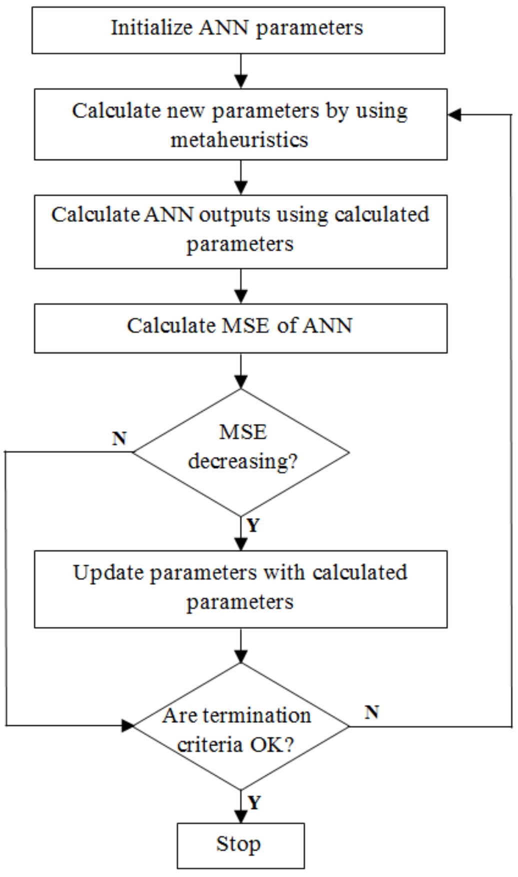

Initialize ANN parameters
Calculate new parameters by using metaheuristics
Calculate ANN outputs using calculated parameters
Calculate MSE of ANN
MSE decreasing?
N
Y
Update parameters with calculated parameters
Are termination criteria OK?
N
Y
Stop

Fig. 4. Flow diagram of ANN training strategy by using metaheuristics

3. Experimental study

3.1. Performance evaluation and comparison criteria

Training process searches error space and tries to find the global minimum point of that space. So, main performance criterion is the MSE reached at the end of training, and it is calculated via Eq. (15). Since we think the training process as an optimization problem, it is also a measure of optimization performance.
In this study, classification accuracies for test data sets are calculated via Eqs. (18)-(19) [34] and these calculated accuracies are used as another performance measure.

$$Accuracy = \frac{1}{|N|}\sum_{i=1}^{|N|} \text{assess}(n_i) , \quad n_i \in N \tag{18}$$

$$\text{assess}(n) = \begin{cases} 1 & \text{If classify}(n) = nc \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

where N is the set of test items, n is a member of N, nc is actual class value of the item n and function classify(n) returns the class value calculated by neural network for item n.

3.2. Data sets and experimental setup

20 data sets, which were downloaded from UCI Machine Learning Repository [35] and KEEL Data Set Repository [36] web sites, were used in this study. Some brief information about these data sets is available in Table 1. We used attributes of the data sets as inputs and outputs of the ANNs in implementations.

Table 1. Dataset properties, ANN topologies and corresponding problem dimensions

Data Set	Attribute (Input + Output)	Instance	ANN Topology	Problem Dimension
Acute Inflammations [35]	6 + 2	120	6 - 10 - 2	92
Blood Transfusion [35]	4 + 1	748	4 - 10 - 1	61
Breast Cancer [35]	9 + 1	683	9 - 25 - 1	276
Fertility [35]	9 + 1	100	9 - 21 - 1	232
Indian Liver Patient [35]	10 + 1	579	10 - 25 - 1	301
Lenses [35]	4 + 1	24	4 - 25 - 3	203
Liver Disorders [35]	6 + 1	345	6 - 25 - 1	201
Pima Indians Diabetes [35]	8 + 1	768	8 - 25 - 1	251
Planning Relax [35]	12 + 1	182	12 - 36 - 1	505
SPECT Heart [35]	22 + 1	267	22 - 30 - 1	721
Thyroid Disease [35]	5 + 1	215	5 - 15 - 20 - 3	473
Vertebral Column [35]	6 + 1	310	6 - 21 - 1	169
Appendicitis [36]	7 + 1	106	7 - 25 - 1	226
Titanic [36]	3 + 1	2201	3 - 15 - 1	76
Phoneme [36]	5 + 1	5404	5 – 20 – 1	141
Iris [36]	4 + 1	150	4 - 25 – 3	203
Mammographic Mass [35]	5 + 1	830	5 – 25 – 1	176
Banknote Authentication [35]	4 + 1	1372	4 – 10 – 1	61
Balance Scale [35]	4 + 1	625	4 – 25 – 3	203
Haberman's Survival [35]	3 + 1	360	3 – 20 – 1	101

Firstly, we partitioned the data sets into two equal parts as training and test sets randomly. Then, we applied the GA, DE, PSO, ABC and the MBO algorithms for ANN trainings. As mentioned in previous sections, MSE values of the networks were calculated in each iteration during trainings, and the metaheuristics tried to minimize those MSE values by changing parameters of the networks systematically. The population sizes were set to 150 members and the numbers of iterations were set to 500 cycles in metaheuristic trainings.

Finally, as a conventional method, trainings were also performed by using LM algorithm to make proper comparisons. In LM training; firstly, we used 500 iterations in LM algorithm like in metaheuristics. Thinking that the total number of calculations in metaheuristics is equal to population size times number of iterations, that is, 150 × 500 = 75000 for our case; secondly, we performed trainings by using LM algorithm one more time with 75000 iterations to make fair comparisons. Each training case was repeated 20 times by using the corresponding training data set and by using the network topology given in Table 1. Network accuracy of each training case was calculated. Average accuracies and average MSE values of the ANNs are given in Table 2 in next section.

4. Results

Numerical results of the experiments are given in Table 2. The highest accuracies reached and the final MSEs for the corresponding network trainings are shown in bold fonts for each data set. From the numerical results, it is clearly seen that the accuracy values obtained by metaheuristic trainings are competitive with gradient calculation based conventional training method LM. Their accuracies are a little bit higher than those of the LM algorithm. In addition, MSE values of them are extremely good in comparison with the LM algorithm.

Table 2. MSE and accuracy values of the trainings (150 members for metaheuristics, 500 iterations for metaheuristics and LM1, 75000 iterations for LM2)

Data Sets		GA	DE	PSO	ABC	MBO	LM1	LM2
Acute Inflammations	*MSE*	1.06E-13	1.06E-21	1.05E-22	6.98E-15	**1.38E-26**	2.56E-05	3.47E-07
	Accuracy	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Blood Transfusion	*MSE*	0.1386	0.1409	**0.1361**	0.1414	0.1367	0.1584	0.2297
	Accuracy	79.41	79.14	79.41	**79.68**	79.41	79.41	74.87
Breast Cancer	*MSE*	0.0061	0.0066	0.0058	0.0078	**6.09E-06**	0.0327	0.0571
	Accuracy	96.19	96.48	**97.07**	96.48	**97.07**	96.19	93.84
Fertility	*MSE*	0.0101	**0.0100**	0.0400	0.0101	**0.0100**	0.0847	0.1035
	Accuracy	90.00	90.00	90.00	90.00	**92.00**	**92.00**	90.00
Indian Liver Patient	*MSE*	0.1596	0.2005	0.1201	0.1524	**0.0984**	0.2772	0.3143
	Accuracy	70.93	71.63	72.66	70.93	71.97	71.97	68.17
Lenses	*MSE*	7.47E-12	**5.82E-47**	7.03E-32	4.71E-16	5.28E-46	0.3568	0.3687
	Accuracy	**81.82**	**81.82**	**81.82**	**81.82**	**81.82**	**81.82**	**81.82**
Liver Disorders	*MSE*	0.1563	0.2075	0.1049	0.1628	**0.0982**	0.2900	0.3590
	Accuracy	**71.51**	**71.51**	70.93	68.60	**71.51**	69.77	62.21
Pima Indians Diabetes	*MSE*	0.1710	0.1442	0.1160	0.1434	**0.1062**	0.2325	0.2818
	Accuracy	76.30	**77.08**	76.56	74.22	76.56	73.96	69.79
Planning Relax	*MSE*	0.1758	0.2044	0.1322	0.1403	**0.0790**	0.3066	0.3405
	Accuracy	72.53	72.53	72.53	**74.73**	72.53	69.23	62.64

SPECT Heart	***MSE***	0.0590	0.0434	0.1068	0.0659	**0.0396**	0.1227	0.1646
	Accuracy	84.21	84.21	84.21	84.96	**85.71**	84.21	79.70
Thyroid Disease	***MSE***	0.0242	3.70E-07	**2.49E-09**	0.0284	0.0034	0.0381	0.0403
	Accuracy	96.26	**97.20**	**97.20**	96.26	**97.20**	**97.20**	**97.20**
Vertebral Column	***MSE***	0.0626	0.0849	0.0463	0.0569	**0.0341**	0.1780	0.2501
	Accuracy	**83.23**	**83.23**	82.58	**83.23**	**83.23**	81.29	73.55
Appendicitis	***MSE***	0.0103	0.0368	0.0185	0.0191	**1.90E-04**	0.1307	0.1344
	Accuracy	84.62	86.54	**90.38**	**90.38**	**90.38**	86.54	86.54
Titanic	***MSE***	0.1570	0.1584	0.1567	0.1569	0.1567	**0.1530**	0.1531
	Accuracy	**79.64**	**79.64**	**79.64**	**79.64**	**79.64**	**79.64**	**79.64**
Phoneme	***MSE***	0.1295	0.1509	0.1131	0.1333	**0.1117**	0.1193	0.1598
	Accuracy	81.64	80.76	83.64	81.35	**84.04**	83.64	83.68
Iris	***MSE***	0.0044	7.71E-05	2.58E-05	0.0060	**1.05E-06**	0.0466	0.0497
	Accuracy	**97.33**	**97.33**	**97.33**	96.00	**97.33**	**97.33**	**97.33**
Mammographic Mass	***MSE***	0.1383	0.1485	0.1159	0.1285	**0.1100**	0.4987	0.3231
	Accuracy	**80.72**	80.00	80.00	78.31	**80.72**	49.64	66.99
Banknote Authentication	***MSE***	3.16E-04	2.50E-04	3.17E-06	0.0013	**1.18E-06**	2.41E-04	4.64E-05
	Accuracy	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**
Balance Scale	***MSE***	0.1380	0.0782	0.0812	0.1180	**0.0690**	0.1511	0.1874
	Accuracy	86.86	87.50	88.46	87.82	**89.74**	87.82	88.14
Haberman's Survival	***MSE***	0.1434	0.1501	0.1290	0.1378	**0.1228**	0.2764	0.2928
	Accuracy	74.03	**74.68**	74.03	**74.68**	**74.68**	70.78	70.78

Especially, the MBO algorithm drawn the best performance figure among the algorithms in this study by giving the best MSE results. Owing to its unique benefit mechanism, MBO algorithm can achieve a good convergence to the global minimum. Normally, metaheuristics given in this paper except for the MBO algorithm generate new neighbourhoods to their existing population members in each iteration. Then, they make greedy selections between generated and existing solutions in order to improve their fitness values. The improvement chances of the solutions completely depend on the neighbourhoods which are generated by using systematic search techniques of the corresponding algorithms. If the neighbourhood having the best fitness can show an improvement on the corresponding solution, the algorithm can be said to be successful in improvement. On the other hand, the MBO algorithm generates new neighbourhoods to its existing population members like in the others. But, it gets the solutions shared by the predecessors for the corresponding members. Therefore, improvement chances of the solutions partially depend on the neighbourhoods which are generated by using systematic search technique of the MBO algorithm. In short, even if it could not achieve to improve the fitness values of the solutions via the generated neighbourhoods, it has additional chances to improve them via the shared solutions. That is, it has an additional chance for each improvement trial. Having both neighbourhood creation and neighbourhood sharing in one application is the novelty of MBO algorithm and this unique benefit mechanism makes it competitive.

Fig. 5 shows MSE progresses of the algorithms during trainings. It is seen that the MBO algorithm converges to the global minimum faster than the others in most of the experiments, and it avoids from going into saturation by using its benefit mechanism. Its saturation values are extremely lower than that of the others in general. It reached to the lowest MSE values except for four data sets in implementations. However, the network accuracies for these data sets are also good enough.

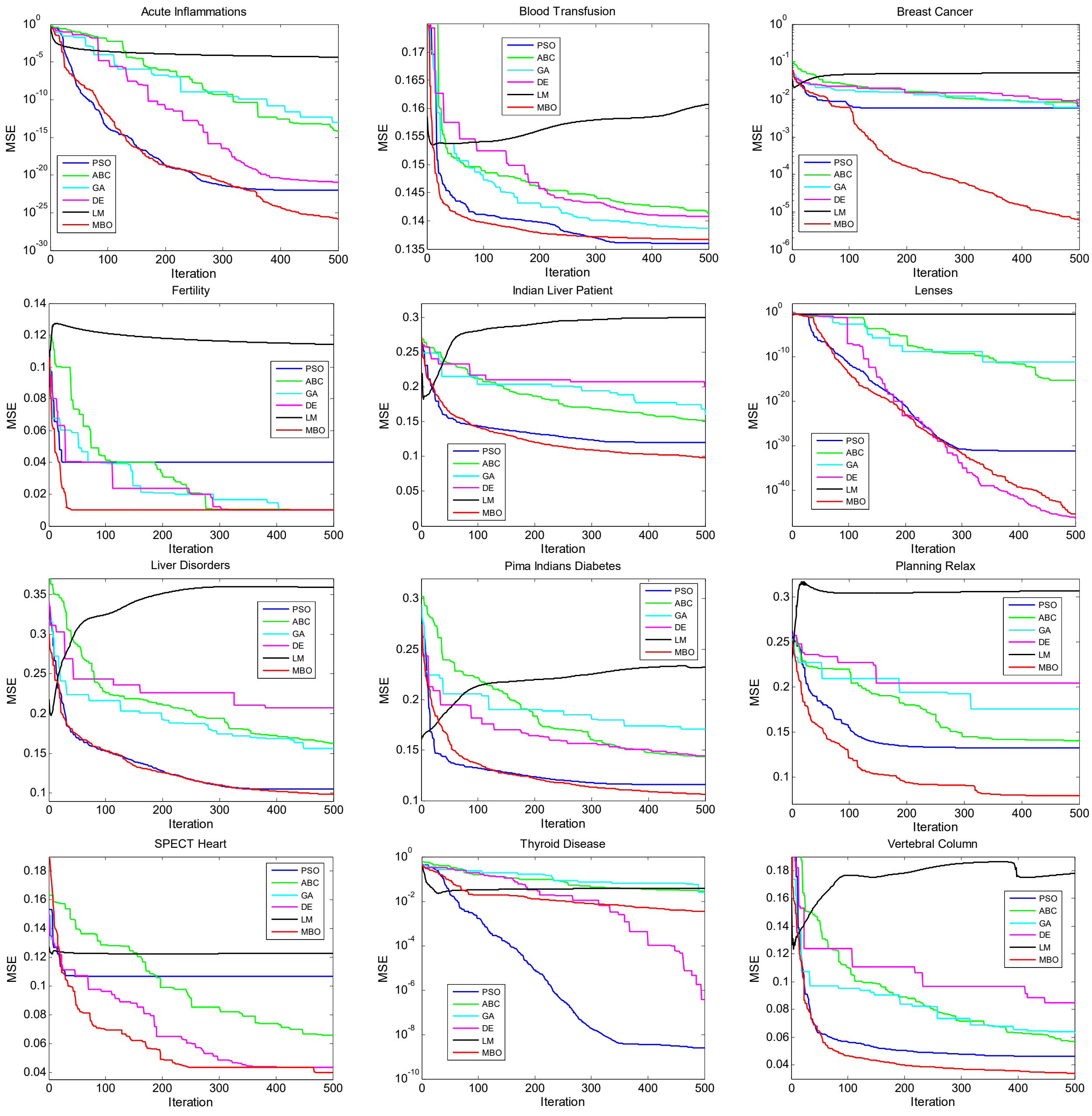


Fig. 5 MSE Progresses of the algorithms during trainings

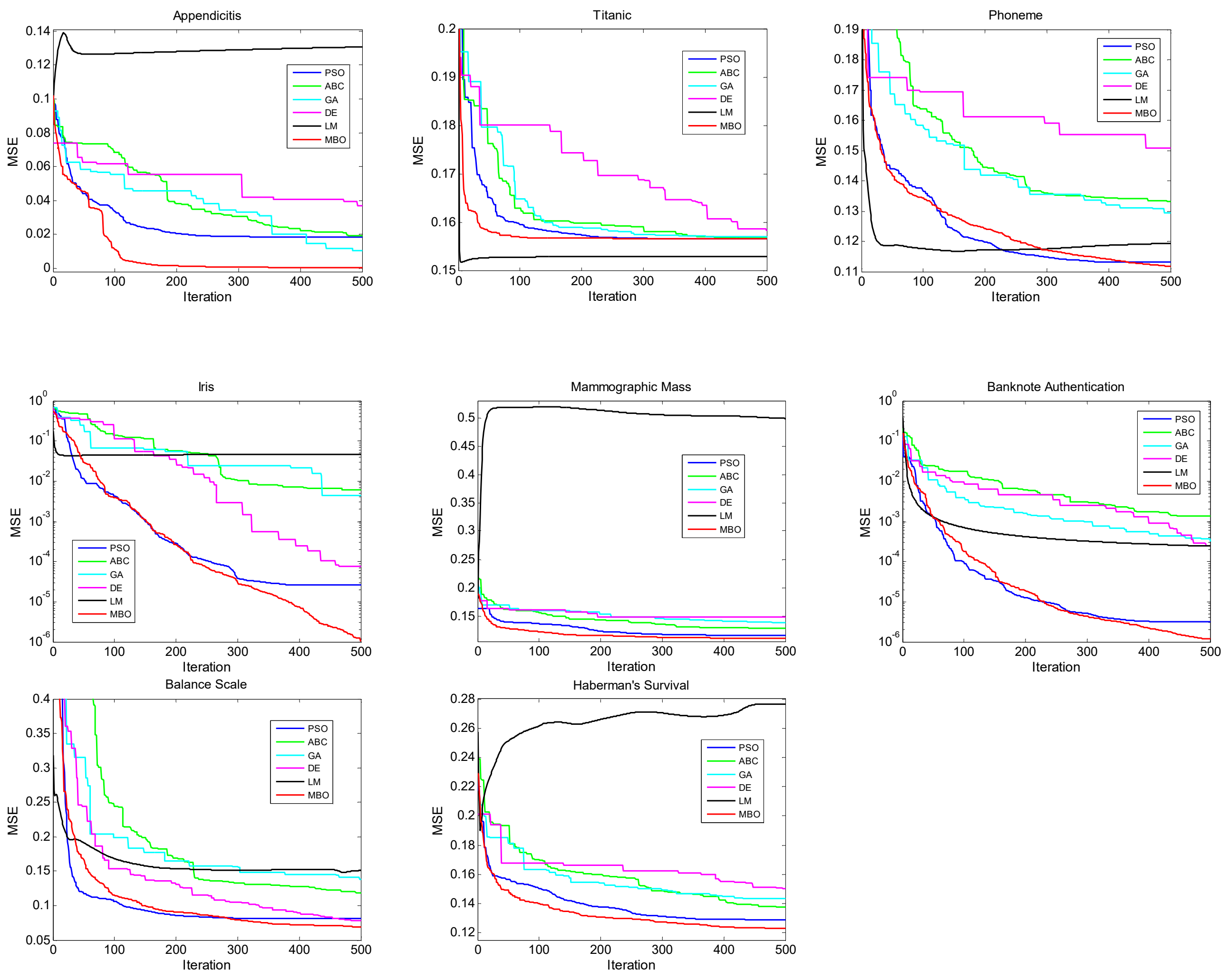


Fig. 5 (Continue) MSE Progresses of the algorithms during trainings

As to the other algorithms, their performances are changing experiment to experiment. Since performance of the PSO algorithm highly depends on initial population quality, it may fall into local minimums of the error space and may not give better solutions for high dimensional problems. Its exploitation (local search) capability is good, but its exploration (global search) capability needs to be improved. The GA is seen to be poor to converge to the global minimum. Its exploration is good, but its exploitation needs to be improved. The ABC algorithm generally succeeds to escape from local minimums of the search space. However, it may not achieve to converge to the global minimum as desired. Its exploration capability is good, but its exploitation capability needs to be improved. The DE algorithm generally performs good optimization and training, but its performance is not better than that of the MBO algorithm. The experiments were performed twice for the LM algorithm. The same number of iterations with the metaheuristics was used in the first experiment. The same number of total calculations with the metaheuristics was performed in the second experiment as mentioned in Section 3.2. It is seen that the increased number of iterations in the second experiment does not make the results better.

Looking at the bold written values in Table 2, the MBO algorithm performed the highest accuracies for 80% of the data sets; it performed the lowest MSE values for 80% of the data sets, and it performed both lowest MSE and highest accuracy values for 65% of the data sets. Fig. 6 shows these success rate distributions of the algorithms on data sets.

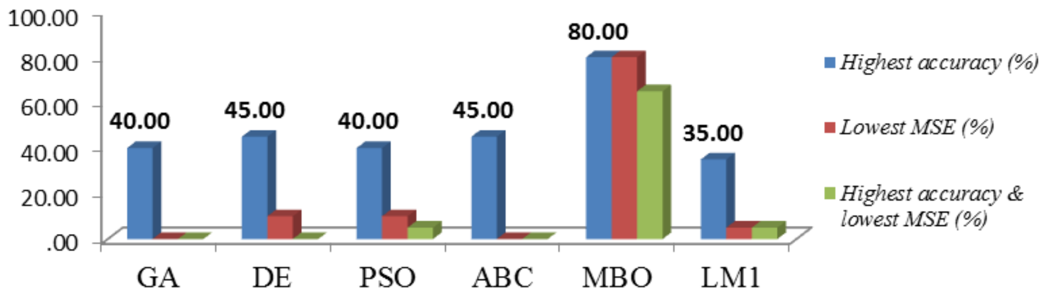


Fig. 6 Success rates of the algorithms on 20 data sets

To sum up, the results can be concluded that the metaheuristic algorithms mentioned in this paper are superior to the conventional LM training algorithm in point of ANN training. It is also seen that the MBO algorithm with its competitive optimization performance outperforms the others in points of the accuracy and MSE.

5. Conclusions

Because the MBO algorithm is recently introduced metaheuristic algorithm, its performance on engineering and computational applications may not been tested satisfactorily. In this study, the GA, DE, MBO, PSO and the ABC algorithms were used to train ANN implementations. The main objective is to test the ANN training performances of these metaheuristics and to compare success of the MBO algorithm with that of the others.

In experimental results, ANN training performances of the algorithms were compared by using 20 different data sets. According to the good results obtained in experiments, it can be said that swarm intelligence can be used in ANN training process successfully. Considering the MSE values in trainings and the accuracy values in tests, the MBO algorithm has the best success rates in ANN trainings among the algorithms examined in this paper. It gets this good and promising performance by using its unique benefit mechanism. Simulating the upwash benefits in real migrating birds, the MBO algorithm has more chance to improve its members and performs a good optimization.

References

[1] Boussaïd I, Lepagnot J, Siarry P (2013) A survey on optimization metaheuristics. Information Sciences. 237: 82–117.

[2] Pappa G L, Ochoa G, Hyde M R, Freitas A A, Woodward J, Swan J (2014) Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. Genetic Programming and Evolvable Machines. 15: 3-35.

[3] Duman E, Uysal M, Alkaya A F (2012) Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem. Information Sciences. 217: 65–77.

[4] Ahuja R K, Ergun O, Orlin J B, Punnen A P (2002) A survey of very large scale neighborhood search techniques. Discrete Applied Mathematics. 123: 75–102.

[5] Holland JH (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

[6] Kirkpatrick S¸ Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598): 671–680.

[7] Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13 (5): 533–549.

[8] Dorigo M (1992) Optimization, learning and natural algorithms. Ph.D. Thesis, Politecnico di Milano.

[9] Eberhart RC, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the sixth international symposium on micromachine and human science, Nagoya, Japan, pp. 39–43.

[10] Storn R, Price K (1997) Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11: 341–359.

[11] Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. Simulation 76: 60–68.

[12] Mucherino A, Seref O (2007) A novel meta-heuristic approach for global optimization. In: Proceedings of the conference on data mining, system analysis and optimization in biomedicine, Gainesville, Florida, pp. 162–173.

[13] Karaboğa D (2005) An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.

[14] Yang XS (2008) Firefly algorithm. In: Nature-Inspired Metaheuristic Algorithms, pp. 79–90. Luniver Press, Frome, UK.

[15] Shah-Hosseini H (2009) The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. International Journal of Bio-inspired Computation (IJBIC) 1: 71–79.

[16] Yang XS and Deb S (2009) Cuckoo search via Lévy flights. In: Proceedings of World Congress on Nature & Biologically Inspired Computing (NaBIC), Coimbatore, India, pp. 210-214.

[17] Yang XS and Deb S (2010) Engineering optimisation by cuckoo search. Int. J. Math. Modelling & Num. Optimisation 1: 330-343.

[18] Yang XS (2010) A New Metaheuristic Bat-Inspired Algorithm. In: JR Gonzalez et al. (Eds.). Inspired Cooperative Strategies for Optimization (NICSO 2010), Vol 284, pp. 65-74.

[19] Nature Yang XS (2011) Bat algorithm for multi-objective optimisation. Int. J. Bio-Inspired Computation 3(5): 267-274.

[20] Blum C, Roli A (2003) Metaheuristics in combinatorial optimisation: Overview and conceptural comparision. ACM Comput. Surv. 35: 268-308.

[21] Rumelhart D E, Hinton G E, Williams R J (1986) Learning representations by back-propagating errors. Nature. 323: 533–536.

[22] Werbos P J (1988) Back-propagation: Past and future. In: Proceedings of International Conference on Neural Networks, San Diego, CA, pp. 343–354.

[23] Levenberg K (1944) A method for the solution of certain problems in least squares. Quarterly of Applied Mathematics. 5: 164–168.

[24] Marquardt D (1963) An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics. 11(2): 431–441.

[25] Moller M F (1993) A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks. 6: 525-533.

[26] Engelbrecht A P (2006) Fundamentals of computational swarm intelligence. Wiley.

[27] Shi Y, Eberhart RC (1999) Empirical Study of Particle Swarm Optimization. In: Proceedings of CEC'99 congress of evolutionary computation, pp. 1945-1950.

[28] Karaboga D, Akay B (2009) A comparative study of Artificial Bee Colony algorithm. Applied Mathematics and Computation 214: 108–132.

[29] Akay B (2009) Performance analysis of artificial bee colony algorithm on numerical optimization problems. Ph.D. Thesis, Erciyes University, Turkey.

[30] Haupt R L, Haupt S E (2004) Practical Genetic Algorithms. Second ed., Wiley, New York, USA.

[31] Karaboğa D, Ökdem S (2004) A Simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm. Turkish Journal of Electrical Engineering. 12: 53 - 60.

[32] Lissaman P B S, Schollenberger C A (1970) Formation Flight of Bird. Science. 168: 1003-1005.

[33] Yao X (1999) Evolving artificial neural networks. In Proceeedings of the IEEE, 87 (9): 1423–1447.

[34] Temurtas F (2009) A comparative study on thyroid disease diagnosis using neural networks. Expert Systems with Applications 36: 944–949.

[35] UCI, Machine Learning Repository web site, Available: http://archive.ics.uci.edu/ml/index.html. Accessed 10 Jun 2013.

[36] KEEL, Data Set Repository web site, Available: http://sci2s.ugr.es/keel/category.php?cat=clas#sub2. Accessed 04 Mar 201