Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Binary ant lion approaches for feature selection

E. Emary [a,b], Hossam M. Zawbaa [c,d,*], Aboul Ella Hassanien [a,1]

[a] *Faculty of Computers and Information, Cairo University, Egypt*
[b] *Faculty of Computer Studies, Arab Open University, Egypt*
[c] *Faculty of Computers and Information, Beni-Suef University, Egypt*
[d] *Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania*

A R T I C L E I N F O

Article history:
Received 4 November 2015
Received in revised form
2 March 2016
Accepted 5 March 2016

Keywords:
Binary ant lion optimizer
Ant lion optimizer
Feature selection
Bio-inspired optimization

A B S T R A C T

In this paper, binary variants of the ant lion optimizer (ALO) are proposed and used to select the optimal feature subset for classification purposes in wrapper-mode. ALO is one of the recently bio-inspired optimization techniques that imitates the hunting process of ant lions. Moreover, ALO balances exploration and exploitation using a single operator that can adaptively searches the domain of solutions for the optimal solution. Binary variants introduced here are performed using two different approaches. The first approach takes only the inspiration of ALO operators and makes the corresponding binary operators. In the second approach, the native ALO is applied while its continuous steps are threshold using suitable threshold function after squashing them. The proposed approaches for binary ant lion optimizer (BALO) are utilized in the feature selection domain for finding feature subset that maximizing the classification performance while minimizing the number of selected features. The proposed binary algorithms were compared to three common optimization algorithms hired in this domain namely particle swarm optimizer (PSO), genetic algorithms (GAs), binary bat algorithm (BBA), as well as the native ALO. A set of assessment indicators is used to evaluate and compare the different methods over 21 data sets from the UCI repository. Results prove the capability of the proposed binary algorithms to search the feature space for optimal feature combinations regardless of the initialization and the used stochastic operators.

1. Introduction

In many applications, data set contains relevant, irrelevant, or redundant features that degrade the classification performance and have an enormous number of features [1]. Feature selection is applied for a better understanding of the data and selecting the subset of significant features. The aim of feature selection is to improve the classifier performance and obtain comparable or even best classification accuracy than use the complete feature set [2]. In addition, feature selection is formulated as a multi-objective problem that minimizes the size of selected features and maximizes the classification accuracy [3,4]. There are two different approaches of feature selection that evaluate the quality of the selected features: wrapper-based and filter-based. More precisely, a *wrapper-based approach* uses a machine learning technique to search through the space of possible solutions [1]. On the other hand, a *filter-based approach* searches the feature space based on data-dependent criteria rather than classification-dependent criteria as in the wrapper approach [5].

The search space size is exponentially increasing according to the number of features in a given data set. In practice, the exhaustive search techniques are impossible to get the optimal solution and still suffer from stagnation in local optima [6]. Therefore, evolutionary computation (EC) algorithms are the alternative for solving these limitations and their global search capability. EC techniques are inspired by nature, social behavior, and biological behavior of animals or birds or insects like: bat, gray wolf, ant lion, moth-flame, etc. in a group [7]. EC algorithms were employed to search adaptively the feature space for the optimal subset using a number of search agents that communicate in a social behavior to reach the optimal solution [8,11]. In EC algorithms, it is essential to have a convenient balance between *exploration* and *exploitation*. In bee swarm algorithms, different behaviors of the bees give us the possibility to create the robust balancing technique between exploration and exploitation [9,10].

Genetic algorithms (GAs) were the first evolutionary based algorithm introduced in the literature and developed based on the natural process of evolution through reproduction [12,13]. In particle swarm optimizer (PSO), each solution is considered as a particle that is characterized by position, fitness, and speed vector [14,15]. Artificial fish swarm (AFS) is a robust stochastic algorithm based on the fish movement and its intelligence during the process of searching for the food [16]. The ant lion optimizer algorithm (ALO) is a comparatively recent EC algorithm that is computationally less expensive than some another EC techniques [17].

* Corresponding author at: Faculty of Mathematics and Computer Science, Babes-Bolyai University, Romania.
E-mail address: hossam.zawbaa@gmail.com (H.M. Zawbaa).
[1] Scientific Research Group in Egypt (SRGE), http://www.egyptscience.net.

http://dx.doi.org/10.1016/j.neucom.2016.03.101
0925-2312/© 2016 Elsevier B.V. All rights reserved.

A modified binary version of PSO algorithm to deal with the binary optimization problems [18] and feature selection problems [19]. The search space in BPSO is considered as a hyper-cube; a particle may be seen to move to nearer or farther corners of the hypercube by flipping various numbers of bits. Furthermore, a binary version of the gravitational search algorithm (BGSA) is used for feature selection problem [20]. A binary version of the bat algorithm (BBA) is applied for feature selection purposes, where the search space is modelled as an n-cube. It is important to assign for every bat a set of binary coordinates that indicate if this feature will belong to the final feature set [21]. Lots of research used a binary version of bio-inspired algorithms [22–25]. In addition, many bio-inspired algorithms are used to solve different problems and applications [26–30].

The aggregate aim of this paper is to propose new binary versions of ant lion optimizer (BALO) for feature selection that selects a minimal number of features and obtaining comparable or even best classification accuracy from using all features and conventional feature selection techniques. The remainder of this paper is organized as follows: Section 2 presents the background of continuous ant lion optimizer (cALO) and the proposed new binary versions of ant lion optimizer (BALO) are described in Section 3. Section 4 presents the proposed BALO algorithms' feature selection, while the experimental results with discussions are reported in Section 5. Finally, conclusions and future work are stated in Section 6.

2. Preliminaries

2.1. Continuous ant lion optimizer (cALO)

Ant lion optimizer (ALO) is a recently proposed optimization algorithm that was proposed by Mirjalili [17]. The ALO algorithm mimics the hunting mechanism of ant lions in nature. The following two subsections discuss the inspiration and operators of the artificial ant lion optimizer.

2.1.1. Inspiration

Mostly, ant lions (doodlebugs) hunt in larvae and their adulthood period is for reproduction. An ant lion larva digs a cone-shaped hole (trap) in the sand by moving along a circular path and throwing out sands with its huge jaw. After that, the larvae hide underneath the bottom of the cone and waits for insects (preferably ant) to be trapped in the trap. Once the ant lions found out that the prey is in the trap, they are trying to catch it by throwing sand toward the trap to bury the prey. When the prey is caught into the jaw, it is pulled under the soil and consumed. After consuming its prey, they throw the leftovers outside the trap and amend it for the next hunting process. Another interesting behavior observed during the lifestyle of the ant lion is that the size of the trap is influenced by two things: level of hunger and shape of the moon [17].

2.1.2. Artificial ant lion

Based on the above description of the ant lion hunting process, a set of conditions can be formulated as in the following items [17]:

- Preys and ants are moving around the search space by using different random walks.
- Ant lions can build traps proportional to their fitness.
- Ant lion with the larger hole has the higher probability of catching ants.
- The range of random walk is adaptively decreased to simulate the sliding ants toward the ant lions.
- If an ant becomes fitter than an ant lion that means it is caught and pulled under the sand by the ant lion.
- An ant lion repositions itself to the latest caught prey and re-builds the trap after each hunt.

According to the above conditions ant lion optimizer can be described as in Algorithm 1.

Algorithm 1. Continuous ant lion optimizer algorithm (CALO).

```
Input: Search space, fitness function, Number of ants, Number of ant
       lions, number of iterations ($T$)
Output: The elitist ant lion
  1. Initialize a population of $n$ ant's positions and $N$ ant lion's positions
     randomly.
  2. Calculate the fitness of all ants and ant lions.
  3. Find the elite ant lion.
  4. $t$=0.
  5. while $t \le T$ do
     foreach $ant_i$ do
         • Select an ant lion using roulette wheel selection (building trap).
         • Slide ants towards the ant lion; perform random search around
           this selected ant lion.
         • Create a random walk for this $ant_i$ around the elite ant lion and
           normalize it.
         • update the ant's position as the average of the two obtained
           random walks.
     end
       • Calculate the fitness of all ants.
       • Replace an ant lion with its corresponding ant; if its becomes
         fitter (catching prey).
       • Update the elite (if an ant lion becomes fitter than the current
         elite).
     end
```

3. The proposed binary ant lion optimizer (BALO).

The ALO algorithm is a recently proposed bio-inspired algorithm that mimics the hunting mechanism of ant lions in nature [17]. ALO has very competitive results in terms of *improved exploration*, *local optima avoidance*, *exploitation*, and *convergence* [17]. As mentioned in [17], the ALO has superior performance on the majority of unimodal and multimodal test functions. The algorithm benefits from high exploitation and convergence rates. The main reason for the high exploitation and convergence speed is due to the adaptive boundary shrinking mechanism and elitism. On the other hand, the high exploration of ALO is due to the employed random walk and roulette wheel selection mechanisms that allow for population diversity.

These attractive properties motivate using it in other applications such as wrapper-based feature selection. In the wrapper-based feature selection, the classifier is trained and evaluated at each individual optimization step and hence it requires a very intelligent optimization to minimize the number of evaluations. In addition, the search space is expected to be very nonlinear with many local minima. Continuous optimization algorithms are commonly used to find feature combinations that maximizing the classifier performance where search agents are positioned in a *d*-dimensional search space at positions in [0, 1]. Binary optimization algorithms; if appropriately used in a similar manner, use much limited search space as two values are only allowed for each dimension {0, 1} and hence is expected to perform better. Furthermore, the binary operator is expected to be much simpler than continuous counterparts.

In the continuous version of ant lion optimizer (cALO), ant lions and ants continuously change their positions to *whatever* point in the space. In feature selection problem, the solutions are restricted to the binary {0, 1} values which motivate using a binary version of the cALO. In this paper, a novel binary ant lion optimizer (BALO) is proposed for the feature selection purposes.

In cALO, we can observe that each individual ant updates its position by averaging two positions. One of them is obtained by performing random walk with suitable step size around the elite ant lion while the other position is obtained by performing random walk around a selected ant lion. Therefore, we applied the same search manner in the binary version of the ALO where the average operator is replaced by crossover operation between two binary solutions. The two solutions to be crossover are either obtained by performing mutation as a local search around ant lions with suitable mutation rate; called BALO-1, or as a threshold continuous random walk around ant lions with suitable step size; called BALO-S and BALO-V. The following two subsection explains in detail the BALO-1 and BALO-S (BALO-V) as two proposed binary versions of ALO.

3.1. Binary ant lion optimizer – approach 1 (BALO-1)

In this approach, each individual ant changes its position according to Eq. (1). The crossover operation between the two binary solutions obtained from random walk around the elite and the selected ant lion as depicted in Algorithm 2:

$$X_i^{t+1} = Crossover(RW_1, RW_2), \tag{1}$$

where $Crossover(x, y)$ is suitable cross over between solutions x and y, and RW_1 and RW_2 are binary vectors representing the effect of elite ant lion and a random selected ant lion respectively.

Algorithm 2. The algorithm of binary ant lion optimizer – approach 1 (BALO-1).

```
input  : $N$ Number of ant lions,
         $n$ Number of ants,
         $IterMax$ Number of iterations.
output: $x_{elite}$ Optimal ant lion binary position,
         $f(x_elite)$ Best fitness value.
 1. Initialize a population of $n$ ants' positions at random $\in 0, 1$.
 2. Initialize a population of $N$ ant lions' positions at random $\in 0, 1$.
 3. Calculate the fitness of all ants and ant lions.
 4. Find the fittest ant lion; $x_{elite}$.
 5. while Stopping criteria not met do
        • Calculate the mutation rate ($r$) given the random walk step size
          and current iteration number $IterMax$ as in equation (4).
          foreach $ant_j$ do
              – Select an ant lion at random using Roulette wheel selection
                (building trap) ($ant\_lion_{RW}$).
              – Apply mutation operation on $ant\_lion_{RW}$ given mutation
                rate $r$; see equation (3), called $RW_1$.
              – Apply mutation operation on $x_{elite}$ given mutation rate $r$;
                see equation (3), called $RW_2$.
              – Perform crossover between $RW_1, RW_2$ using equation (2)
                and set the new position of $ant_j$ to the output of crossover.
          end
        • Calculate the fitness of all ants.
        • Replace an ant lion with its corresponding ant it if becomes
          fitter (Catching Prey).
        • Update elite if an ant lion becomes fitter than the elite.
    end
 6. Select the elite ant lion $x_{elite}$ and its fitness.
```

The *average* operator used in the cALO is used to attract the ant toward the ant lion trap cone and hence it is the main operator for *exploration* or *global search*. Here, the *crossover* operator replaces the *average* operator used in the cALO where both operators achieve *global searching/exploration*. The *crossover* operator is a simple operator to obtain an intermediate solution between two solutions. The used crossover is simple stochastic crossover that switches between the two input vector with same probability as given in the following equation:

$$x^d = \begin{cases} x_1^d & if(rand) \geq 0.5 \\ x_2^d & otherwise \end{cases} \tag{2}$$

where x^d is the output from crossover at dimension d between vectors x_1^d and x_2^d.

RW_1 represents the attraction of an ant by the elite ant lion that is represented by a random walk around the *elite* continuous-valued ant lion with suitable step size. And it can be represented by stochastic mutation around a selected ant lion with suitable mutation rate around the binary-valued elite ant lion in the binary version as given in Eq. (3). RW_2 represents the attraction by the other ant lions and is performed by applying stochastic mutation around an ant lion in the binary mode that is selected by the roulette wheel selection method:

$$x_{out}^d = \begin{cases} x_{in}^d & if\ rand1\ \geq r \\ rand2 & otherwise \end{cases} \tag{3}$$

where $x_{out}{}^d$ is the d dimension value for the output vector from mutation, $x_{in}{}^d$ is the input vector to be mutated, $rand1$, $rand2$ are two random numbers drawn from a uniform distribution in the range [0, 1], and r is the mutation rate. It worth mentioning that r is linearly decremented with iteration number ranging from 0.9 to 0 as shown in the following equation:

$$r = 0.9 + \frac{-0.9 * (i-1)}{Iter\ Max - 1} \tag{4}$$

where r is the mutation rate at iteration i and, $IterMax$ is the total number of iterations to run the optimization.

3.2. Binary ant lion optimizer – approach 2 (BALO-S and BALO-V)

In this approach; depicted in Algorithm 3, each individual ant changes its position to the crossover between two binary solutions as in the past approach namely RW_1, RW_2 as in Eq. (2). Furthermore, the first solution RW_1 is obtained by performing random walk around the elite ant lion (E) while the second solution RW_2 is obtained by performing random walk around a selected ant lion (S).

The two solutions resulted from the random walk have *continuous* values and hence must be converted into the corresponding binary solutions. The conversion is performed by applying squashing of continuous values in each dimension using either *S-shaped* function or *V-shaped* function [32]. Transfer functions or squashing functions define the probability of changing position vector's elements from 0 to 1 and vice versa. Transfer functions force the search agents to move in a binary space. *The sigmoidal (S-shaped)* function is a common transfer function as given in Eq. (5). The hyperbolic *tan* function is a common example of *V-shaped* functions as given in Eq. (6):

$$y^d = \frac{1}{1 + e^{-10(x^d)}}, \tag{5}$$

$$y^d = |\tanh x^d| \tag{6}$$

where x^d is the continuous-valued input vector at dimension d defined as $RW_1 - E$ or $RW_2 - S$ with E current elite solution and S is a solution obtained from the roulette wheel selection and y^d is the output of sigmoidal function at dimension d.

The output from the squashing function; sigmoidal or hyperbolic tan is still in continuous mode and hence it has to be thresholded to reach the binary-valued one. The sigmoidal or hyperbolic tan functions are mapping the infinite input smoothly to a finite output. Commonly stochastic threshold is applied as in Eq. (7) to reach the binary solution in the case of *sigmoidal* function. In the case of using *V-shaped* function, the threshold into binary is performed using Eq. (8):

$$Ant_d^{t+1} = \begin{cases} 1 & \text{if } y^d \geq rand \\ 0 & otherwise \end{cases} \tag{7}$$

$$Ant_d^{t+1} = \begin{cases} sel_d^t & \text{if } y^d < rand \\ org_d^t & otherwise \end{cases} \tag{8}$$

where $rand$ is a random number drawn from a uniform distribution ∈[0, 1], Ant_d^{t+1} is the updated binary position in dimension d at iteration t, and y^d is defined in Eq. (7) in the case of S-shaped and in Eq. (6) in the case of V-shaped and Sel is either S or E solution before applying the random walk.

Random walks are all based on the following equation:

$$X(t) = [0, cumsum(2W(t_1) - 1); cumsum(2W(t_2) - 1); \ldots; \\ cumsum(2W(t_T) - 1)], \tag{9}$$

where $cumsum$ calculates the cumulative sum, T is the maximum number of iteration, t shows the step of a random walk, and $W(t)$ is a stochastic function defined in the following equation:

$$W(t) = \begin{cases} 1 \text{ if } rand > 0.5 \\ 0 \text{ if } rand \leq 0.5, \end{cases} \tag{10}$$

where t shows the step of random walk and $rand$ is a random number generated with uniform distribution in the interval of [0, 1].

In order to keep the random walks inside the search space, they are normalized using the following equation (min–max normalization):

$$X_i^t = \frac{(X_i^t - a_i) \times (d_i - c_i^t)}{(b_i^t - a_i)} + c_i, \tag{11}$$

where a_i is the minimum of the random walk of i-th variable, b_i is the maximum of random walk in i-th variable, c_i^t is the minimum of i-th variable at t-th iteration, and d_i^t indicates the maximum of i-th variable at t-th iteration.

The radius of ants's random walks hypersphere is *decreased* adaptively; see Eqs. (12)–(14):

$$c^t = \frac{c^t}{I}, \tag{12}$$

$$d^t = \frac{d^t}{I}, \tag{13}$$

where c^t is the minimum of all variables at t-th iteration, d^t is the maximum of all variables at t-th iteration, and I is a ratio that is defined in the following equation:

$$I = 10^w \frac{t}{T}, \tag{14}$$

where t is the current iteration, T is the maximum number of iterations, and w is a constant defined based on the current iteration ($w=2$ when $t>0.1T$, $w=3$ when $t>0.5T$, $w=4$ when $t>0.75T$, $w=5$ when $t>0.9T$, and $w=6$ when $t>0.95T$). Basically, the constant w can adjust the accuracy level of exploitation.

Algorithm 3. The algorithm of binary ant lion optimizer – approach 2 (BALO-S and BALO-V).

```
input  : N Number of ant lions,
         n Number of ants,
         IterMax Number of iterations.
output : x_elite Optimal ant lion binary position,
         f(x_e lite) Best fitness value.
1. Initialize a population of n ants' positions at random ∈ 0, 1.
2. Initialize a population of n ant lions' positions at random ∈ 0, 1.
3. Calculate the fitness of all ants and ant lions.
4. Find the fittest ant lion; x_elite.
5. while Stopping criteria not met do
       • Calculate the radius of the ant's random walk; see equations
         (12), (13), and (14).
         foreach ant_j do
             – Select an ant lion at random using Roulette wheel selection
               (building trap) (ant_lion_RW).
             – Apply random walk around ant_lion_RW given the current
               random walk radius; called x1.
             – Apply random walk around x_elite given the current random
               walk radius; called x2.
             – Squash the solutions in x1, x2 using squashing function as in
               equation (5) or 6; called y1, y2.
             – Apply stochastic threshold on y1, y2 using equation (7) or 8
               to output RW_1, RW_2
             – Perform crossover between RW_1, RW_2 using equation (2)
               and set the new position of ant_j to the output of crossover.
         end
       • Calculate the fitness of all ants.
       • Replace an ant lion with its corresponding ant it if becomes
         fitter (Catching Prey).
       • Update the elite (if an ant lion becomes fitter than the elite).
   end
6. Produce the elite ant lion x_elite and its fitness.
```

4. Binary ant lion optimizer applied for feature selection

In this section, the two proposed binary optimizer approaches are exploited in feature selection for classification problems. For a feature vector sized N, the different feature combinations would be 2^N which is a huge space of features to be searched *exhaustively*. Therefore, ALO is used to search adaptively the feature space for best feature subset. The best feature subset is the one with minimum *classification error rate* and the minimum *number of selected features*. The fitness function is used in BALO to evaluate individual search agents is shown in the following equation:

$$\downarrow Fitness = \alpha\gamma_R(D) + \beta\frac{|R|}{|C|}, \tag{15}$$

where $\gamma_R(D)$ is the classification error rate given classifier R relative to selection decision D of the features, R is the length of selected feature subset, C is the total number of features in the data set, α and β are two parameters corresponding to the importance of classification quality and subset length, $\alpha \in [0, 1]$ and $\beta = 1-\alpha$.

K-Nearest neighbor (KNN) [31] is a simple common method used for classification purposes and is adopted here as a simple candidate classifier to evaluate the fitness function. KNN classifier is determined based on the minimum distance from the query instance to the training samples.

5. Experimental results and discussion

5.1. Data description

Twenty-one data sets in Table 1 from the UCI machine learning repository [33] are used in the experiments and comparison of results. The data sets were selected to have various numbers of *features* and *instances* as representatives of various kinds of issues that the proposed technique will be tested on. In addition, we selected a set of respectively high dimensional data to ensure the performance of optimization algorithms in huge search spaces. The individual data set is divided in a cross-validation [34] manner for evaluation. In *K*-fold cross-validation, $K-1$ folds are used for

Table 1
List of data sets used in the experiments.

DS No.	Name	No. of features	No. of samples
1	**BreastEW**	30	569
2	**Breastcancer**	9	699
3	**CongressEW**	16	435
4	**Exactly**	13	1000
5	**Exactly2**	13	1000
6	**HeartEW**	13	270
7	**IonosphereEW**	34	351
8	**KrvskpEW**	36	3196
9	**Lymphography**	18	148
10	**M-of-n**	13	1000
11	**SonarEW**	60	208
12	**SpectEW**	22	267
13	**Tic-tac-toe**	9	958
14	**Vote**	16	300
15	**WaveformEW**	40	5000
16	**WineEW**	13	178
17	**Zoo**	16	101
18	**Clean1**	166	476
19	**Clean2**	166	6598
20	**PenglungEW**	325	73
21	**Semeion**	265	1593

Table 2
Parameter setting for experiments.

Parameter	Value(s)
K for cross validation	10
M The number of runs	5
No. of search agents	8
No. of iterations (Dimension < 100)	70
No. of iterations (Dimension ≥ 100)	200
Problem dimension	Number of features in the data
Search domain in binary algorithms	{0, 1}
Search domain in continuous algorithms	[0, 1]
Crossover fraction in GAs	0.8
Inertia factor of PSO	0.1
Individual-best acceleration factor of PSO	0.1
α parameter in the fitness function	0.99
β parameter in the fitness function	0.01
Q_{min} Frequency minimum for BBA	0
Q_{max} Frequency maximum for BBA	2
A Loudness for BBA	0.5
r Pulse rate for BBA	0.5

training and validation and the remaining fold is used for testing. This process is repeated M times. Hence, the individual optimizer is evaluated K_*M times for individual data set. The data for training, validation, and testing are equally sized. *Training* part is used to train the used classifier through optimization and at the final evaluation. *Validation* part used to assess the performance of the classifier at the optimization time. *Testing* part is used to evaluate the finally selected features given the trained classifier.

A wrapper approach for feature selection is used in this paper based on KNN classifier. KNN is utilized in the experiments based on trial and error basis where the best choice of $(K = 5)$ is selected as best performing on all the data sets [31]. Through the training process, every search agent represents one feature subset. The training set is used to evaluate the KNN on the validation set throughout the optimization to guide the feature selection process, while test data are kept hidden from the optimization and is used for final evaluation.

The proposed feature selection methods are benchmarked with particle swarm optimizer (PSO) [35], genetic algorithms (GAs) [36], continuous ALO, and binary bat algorithm (BBA) [37] for evaluation. The global and optimizer-specific parameter setting is outlined in Table 2. All the parameters are set either according to domain-specific knowledge as the α and β parameters of the used fitness function, or based on trial and error on small simulations and common in the literature such as the rest of parameters.

5.2. Evaluation criteria

In each run of individual optimizer, the following measures are calculated on test data:

- *Classification error rate*: is an indicator that describes how accurate is the classifier given the selected feature set and is formulated in the following equation:

$$Avg\_Perf = \frac{1}{M}\sum_{j=1}^{M}\frac{1}{N}\sum_{i=1}^{N} Unmatch(C_i, L_i), \tag{16}$$

  where M is the number of runs for the optimization algorithm, N is the number of points in the test set, C_i is the classifier output label for data point i, L_i is the reference class label for data point i, and *Unmatch* is a function that outputs 0 when the two input labels are the same and outputs 1 when they are different.

- *Statistical mean*: is the average of solutions acquired from running an optimization algorithm for different M running. Mean represents the average performance a given stochastic optimizer can be formulated in the following equation:

$$Mean = \frac{1}{M}\sum_{i=1}^{M} g_*^i, \tag{17}$$

  where g_*^i is the optimal solution resulted from run number i.

- *Statistical standard deviation* (*std*): is a representation for the variation of the obtained best solutions found for running a stochastic optimizer for M different runs. *Std* is used as an indicator for optimization algorithm stability and robustness as given in the following equation:

$$Std = \sqrt{\frac{1}{M-1}\sum (g_*^i - Mean)^2}, \tag{18}$$

- *Average selection size*: represents the average size of the selected features to the total number of features and can be formulated as in the following equation:

$$AVG\_Selection = \frac{1}{M}\sum_{i=1}^{M}\frac{size(g_*^i)}{D}, \tag{19}$$

  where $size(x)$ is the number of values for the vector x, and D is the number of features in the original data set.

- *Wilcoxon rank sum test*: proposed by Frank Wilcoxon [39] as a non-parametric test. The test assigns ranks to all the scores considered as one group, and then sums the ranks of each group. The null hypothesis is that the two samples come from the same population, so any difference in the two rank sums come only from sampling error. The rank sum test is often described as the non-parametric version of the t test for two independent groups. It tests the null hypothesis that data in x and y vectors are samples from continuous distributions with equal medians, against the alternative that they are not.

- *Average computational time*: is the run time for a given optimization algorithm in millisecond that calculated over the different runs as given in the following equation:

$$T_o = \frac{1}{M}\sum_{i=1}^{M} RunTime_{o,i}, \tag{20}$$

  where M is the number of runs for the optimizer O, and $RunTime_{o,i}$ is the computational time in millisecond for optimizer o at run number i.

The proposed binary versions of ant lion optimizer are compared against common optimizers namely particle swarm optimizer (PSO), genetic algorithms (GAs), continuous ALO, and binary bat algorithm (BBA) with three different initialization methods namely *Mixed initialization*, *small initialization*, and *large initialization* [40].

The different initialization methods differ from one to another as follows:

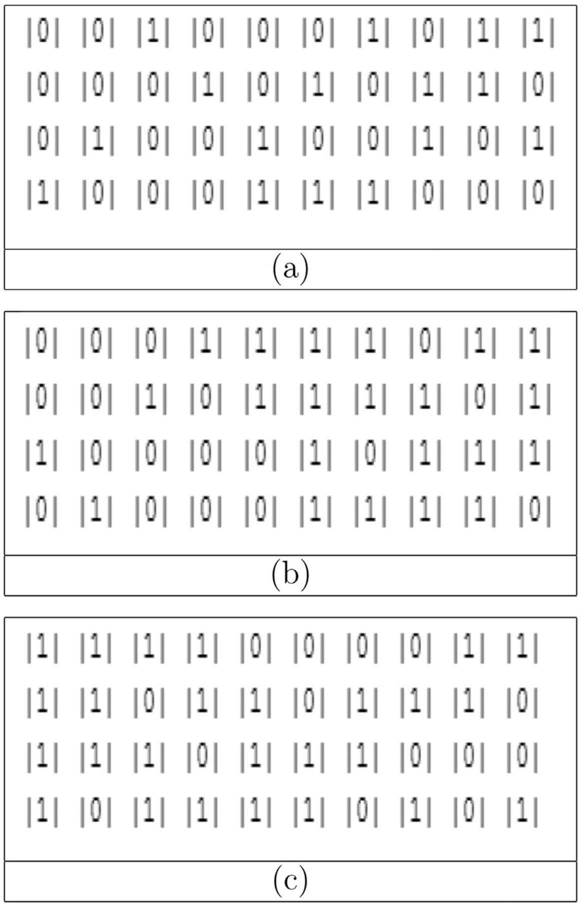
- *Population diversity*: the capability of an optimizer to produce variants of the given initial population is an important property that identifies a given algorithm.
- *Closeness to expected optimal solution*: the optimizer capability to efficiently search the space for the optimal solution is must for a successful optimizer and hence it is intended to force the initial search agents to be apart from the expected optimal solution or close to the expected optimal solution.
- *Resemblance to forward and backward selection*: forward and backward selections are two common methods for the selection of features where each has its own strengths and weaknesses and hence we would like to assess the initialization impact on the feature selection process.

Three initialization methods are used to ensure capability of the different optimizers to converge from different initial positions. *Small initialization* method is expected to test the *global searching* capability of a given optimizer as the initial search agents' positions are commonly apart from the expected optimal. Therefore, the optimizer has to use global search operators to derive better solutions.

Large initialization is expected to assess the local searching capability of a given optimizer as the search agents' positions are commonly closer to the expected optimal solution and hence just local searching operators are required to reach the optimal solution. Both the small and large initialization methods have less population diversity and hence the algorithm has to evolve the population adaptively to reach the optimal solution. Moreover, the small initialization method resembles the forward selection method while the large initialization method resembles the backward selection method which is very common in feature selection. Forward selection starts with an empty feature set and evolves with adding the extra feature by selecting the feature that achieves the highest classification performance and so on. On the other hand, backward selection starts with all the features selected; then candidate features are sequentially removed from the feature pool while it increases the classification performance. Forward selection usually selects a smaller number of features and is computationally less than backward selection. But, when the best feature subset contains a relatively large number of features, backward selection has a larger chance to obtain the best solution [41].

Mixed initialization is the case where some search agents are close to the expected optimal solution and the other search agents are apart and hence it provides much *diversity* of the population as the search agents are expected to be apart from each other. In addition, this initialization method takes both the merits of small and large initialization. This initialization method is expected to evaluate the global searching in cases of the diverse population. Some details about the three different initialization methods are provided as follows:

1. *Small initialization*: search agents are initialized with the minor number of randomly selected features. Therefore, if the number of agents is less than the number of features we will see that each search agent will have a single dimension with 1. Of course, the optimizer will search for feature(s) to be set to 1 to enhance the fitness function value as in the standard forward selection of features; as shown in Fig. 1(a).



|0| |0| |1| |0| |0| |0| |1| |0| |1| |1|
|0| |0| |0| |1| |0| |1| |0| |1| |1| |0|
|0| |1| |0| |0| |1| |0| |0| |1| |0| |1|
|1| |0| |0| |0| |1| |1| |1| |0| |0| |0|
(a)

|0| |0| |0| |1| |1| |1| |1| |0| |1| |1|
|0| |0| |1| |0| |1| |1| |1| |1| |0| |1|
|1| |0| |0| |0| |0| |1| |0| |1| |1| |1|
|0| |1| |0| |0| |0| |1| |1| |1| |1| |0|
(b)

|1| |1| |1| |1| |0| |0| |0| |0| |1| |1|
|1| |1| |0| |1| |1| |0| |1| |1| |1| |0|
|1| |1| |1| |0| |1| |1| |1| |0| |0| |0|
|1| |0| |1| |1| |1| |1| |0| |1| |0| |1|
(c)

Fig. 1. Sample initial ant lion positions using small, mixed and large initialization with 10 search agents and 4 dimensions.

2. *Large initialization*: search agents are set to it is maximum. In this case, we find a solution with all dimensions set to 1 and solutions with all dimensions set to 1 except for single random dimension set to 0. Of course, the optimizer will search for feature(s) to be removed while keeping or enhancing the fitness function value; as shown in Fig. 1(c).
3. *Mixed initialization*: where half the search agents are initialized using the small initialization and the rest is initialized using the large initialization method; as shown in Fig. 1(b).

5.3. Numerical results and discussion

Table 3 outlines the statistical mean fitness values obtained from the different optimization algorithms on the different data sets using small initialization. We can remark that although small initialization forces the initial search agents to be far away from the optimal solution, the proposed BALO-1 can reach global/near optimal solution. We can see that in almost half the data sets the BALO-1 reaches a better solution rather than the traditional continuous algorithm namely GAs and PSO and even the modern optimizers such as ALO. A similar remark can be seen by remarking the performance of BALO-V. Therefore, in general, we can see that the proposed binary algorithms with ALO inspiration have better performance in comparison to continuous ones as well as binary algorithms such as BBA. The limited search space can interpret the enhanced performance in the case of binary algorithms which eases the problem and balance between *global* and *local* searching that is inspired by the ALO principles. The balance between global and local search helps the optimization algorithm to

Table 3
Statistical mean fitness measure calculated on the different runs for all optimizers on the different data sets using small initialization.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	0.024	0.024	**0.021**	0.023	0.037	0.025	0.037
2	0.033	**0.026**	0.033	0.029	0.095	0.032	0.066
3	0.041	0.035	0.040	**0.033**	0.121	0.043	0.057
4	0.301	**0.139**	0.270	0.240	0.312	0.306	0.308
5	0.238	**0.232**	0.234	0.237	0.242	0.238	0.242
6	0.143	0.130	0.130	**0.124**	0.233	0.135	0.165
7	**0.100**	0.113	0.115	0.101	0.151	0.100	0.121
8	0.045	0.037	0.052	**0.034**	0.427	0.065	0.240
9	0.173	0.138	0.152	**0.078**	0.237	0.176	0.223
10	0.091	**0.020**	0.097	0.045	0.300	0.062	0.208
11	0.175	0.142	0.171	**0.141**	0.329	0.176	0.163
12	0.154	**0.116**	0.137	0.122	0.202	0.139	0.206
13	0.236	**0.213**	0.221	0.224	0.323	0.232	0.289
14	0.052	0.038	0.048	**0.028**	0.100	0.050	0.100
15	0.208	**0.200**	0.224	0.201	0.454	0.232	0.275
16	0.008	**0.006**	0.011	0.017	0.115	0.023	0.059
17	0.094	0.079	0.079	**0.049**	0.317	0.138	0.168
18	0.127	0.124	0.150	**0.109**	0.285	0.146	0.159
19	0.039	**0.037**	0.042	0.038	0.092	0.043	0.047
20	**0.205**	0.219	0.233	0.273	0.472	0.239	0.266
21	0.031	**0.026**	0.033	0.030	0.099	0.032	0.070

Table 5
Statistical mean fitness measure calculated on the different runs for all optimizers on the different data sets using large initialization.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	0.028	0.023	0.024	**0.021**	0.030	0.027	0.029
2	0.031	0.026	0.034	**0.022**	0.051	0.028	0.036
3	0.040	0.036	0.044	**0.032**	0.074	0.045	0.063
4	0.295	0.223	0.254	**0.143**	0.314	0.298	0.300
5	0.235	0.233	0.233	**0.229**	0.245	0.233	0.234
6	0.144	**0.124**	0.133	0.124	0.159	0.133	0.150
7	0.103	**0.097**	0.115	0.107	0.144	0.120	0.131
8	0.059	0.035	0.054	**0.033**	0.085	0.048	0.063
9	0.170	0.155	0.173	**0.135**	0.206	0.176	0.206
10	0.099	**0.011**	0.102	0.020	0.126	0.069	0.107
11	0.168	**0.147**	0.166	0.156	0.291	0.166	0.221
12	0.142	**0.114**	0.137	0.114	0.172	0.139	0.150
13	0.228	0.213	**0.213**	0.217	0.239	0.223	0.233
14	0.043	0.040	0.047	**0.037**	0.087	0.048	0.075
15	0.215	0.203	0.219	**0.200**	0.226	0.210	0.216
16	0.017	**0.008**	0.009	0.009	0.020	0.017	0.017
17	0.094	**0.079**	0.084	0.089	0.193	0.089	0.164
18	0.110	**0.108**	0.147	0.120	0.216	0.144	0.170
19	0.038	0.036	0.042	**0.034**	0.048	0.041	0.041
20	**0.205**	0.226	0.233	0.226	0.349	0.239	0.301
21	0.028	0.026	0.035	**0.025**	0.046	0.036	0.035

Table 4
Statistical mean fitness measure calculated on the different runs for all optimizers on the different data sets using mixed initialization.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	0.028	0.021	**0.021**	0.022	0.024	0.028	0.026
2	0.029	0.026	0.031	0.030	**0.025**	0.032	0.033
3	0.039	**0.030**	0.039	0.037	0.033	0.039	0.056
4	0.299	**0.144**	0.278	0.144	0.252	0.298	0.302
5	0.236	0.234	**0.234**	0.234	0.235	0.235	0.239
6	0.132	0.128	0.133	**0.122**	0.128	0.141	0.152
7	0.115	0.111	0.123	0.108	**0.101**	0.113	0.117
8	0.052	**0.033**	0.054	0.034	0.043	0.049	0.052
9	0.176	**0.125**	0.156	0.139	0.149	0.172	0.203
10	0.070	**0.006**	0.083	0.010	0.050	0.079	0.086
11	0.173	**0.132**	0.175	0.154	0.135	0.154	0.219
12	0.150	0.111	0.137	**0.109**	0.126	0.137	0.154
13	0.228	**0.213**	0.221	**0.213**	0.219	0.235	0.234
14	0.050	0.045	0.047	0.040	**0.038**	0.055	0.058
15	0.214	0.200	0.222	**0.195**	0.210	0.210	0.211
16	0.011	0.011	0.011	**0.006**	0.009	0.014	0.017
17	0.109	**0.069**	0.094	0.079	0.084	0.088	0.143
18	0.126	**0.105**	0.146	0.126	0.144	0.137	0.160
19	0.039	0.037	0.041	**0.036**	0.040	0.041	0.041
20	**0.212**	0.226	0.246	0.219	0.226	0.246	0.280
21	0.029	**0.027**	0.034	0.028	0.031	0.035	0.035

Table 6
Statistical standard deviation measure for the different optimizers on the different data sets averaged over the three initializers.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	0.007	0.008	**0.007**	0.009	0.008	0.008	0.008
2	0.006	0.006	**0.004**	0.005	0.008	0.008	0.017
3	0.024	0.016	0.017	**0.015**	0.019	0.021	0.028
4	0.023	0.109	0.034	0.101	0.041	**0.020**	0.023
5	0.014	**0.013**	0.015	0.016	0.014	0.013	0.015
6	0.023	**0.019**	0.021	0.022	0.033	0.021	0.037
7	0.028	0.028	0.028	**0.027**	0.033	0.028	0.032
8	0.012	**0.008**	0.008	0.009	0.018	0.023	0.033
9	0.038	0.038	0.030	0.030	0.058	**0.028**	0.054
10	0.044	**0.023**	0.034	0.032	0.029	0.049	0.029
11	0.045	**0.037**	0.044	0.053	0.061	0.046	0.061
12	0.041	0.032	0.032	0.030	**0.029**	0.040	0.037
13	0.023	**0.019**	0.020	0.020	0.032	0.024	0.034
14	0.022	**0.013**	0.023	0.014	0.033	0.017	0.031
15	0.009	**0.006**	0.010	0.009	0.037	0.014	0.021
16	0.011	0.012	0.014	**0.008**	0.026	0.020	0.027
17	0.076	0.062	0.071	**0.060**	0.074	0.086	0.089
18	0.027	**0.016**	0.018	0.026	0.019	0.026	0.029
19	0.003	0.004	**0.002**	0.004	0.004	0.003	0.004
20	**0.081**	0.084	0.082	0.110	0.083	0.085	0.096
21	**0.009**	0.009	0.010	0.009	0.010	0.010	0.020

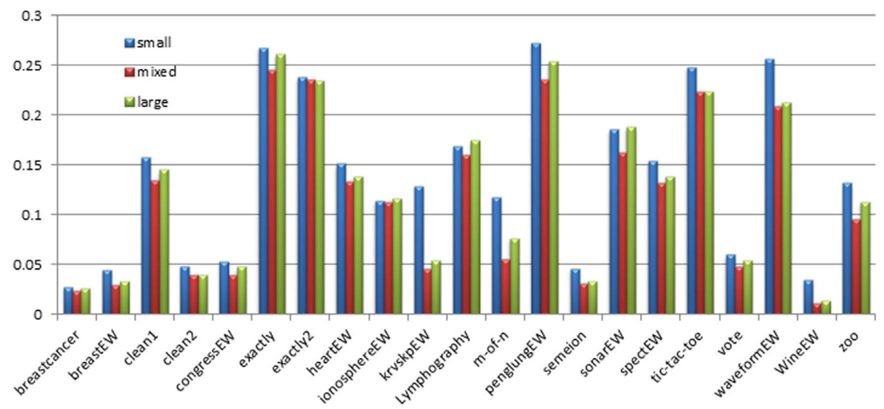


Fig. 2. Statistical mean fitness averaged for the different optimizers on the different data sets using the different initializers.

Table 7
Average selection size of different optimizers on all the data sets averaged on all initialization methods.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	0.519	**0.494**	0.537	0.537	0.562	0.525	0.574
2	0.519	**0.419**	0.513	0.443	0.530	0.494	0.541
3	**0.271**	0.313	0.399	0.309	0.483	0.413	0.604
4	0.406	0.436	0.517	0.462	0.487	**0.385**	0.427
5	0.457	0.338	0.547	**0.333**	0.410	0.457	0.449
6	0.650	**0.474**	0.573	0.504	0.564	0.551	0.543
7	**0.335**	0.431	0.350	0.399	0.500	0.469	0.556
8	0.688	**0.457**	0.559	0.468	0.552	0.551	0.591
9	0.417	**0.361**	0.475	0.432	0.509	0.512	0.531
10	0.684	**0.483**	0.675	0.513	0.556	0.624	0.564
11	**0.286**	0.443	0.444	0.462	0.493	0.456	0.506
12	0.508	0.391	0.543	**0.389**	0.462	0.492	0.422
13	0.691	0.654	0.617	0.630	**0.580**	0.722	0.623
14	**0.285**	0.326	0.292	0.330	0.431	0.427	0.535
15	0.847	0.593	0.654	**0.567**	0.611	0.707	0.657
16	0.526	0.444	0.470	**0.389**	0.470	0.504	0.547
17	0.441	**0.361**	0.458	0.396	0.441	0.389	0.472
18	**0.424**	0.434	0.426	0.469	0.513	0.475	0.568
19	0.640	**0.483**	0.510	0.496	0.531	0.574	0.632
20	**0.111**	0.393	0.329	0.374	0.474	0.427	0.460
21	0.498	**0.459**	0.468	0.465	0.553	0.549	0.515

Table 9
Classification error rate of the different optimizers on test data on the different data sets using small mixed method.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO	Full
1	0.028	**0.021**	**0.021**	0.021	0.024	0.028	0.026	0.037
2	0.029	0.026	0.031	0.030	**0.025**	0.032	0.033	0.056
3	0.039	**0.030**	0.039	0.037	0.033	0.039	0.056	0.083
4	0.299	**0.144**	0.277	0.144	0.252	0.298	0.302	0.327
5	0.236	0.234	**0.234**	0.234	0.235	0.234	0.239	0.257
6	0.131	0.128	0.133	**0.122**	0.128	0.141	0.152	0.185
7	0.115	0.111	0.123	0.108	**0.101**	0.113	0.117	0.134
8	0.052	**0.033**	0.054	0.034	0.043	0.049	0.052	0.085
9	0.176	**0.125**	0.156	0.139	0.149	0.172	0.203	0.317
10	0.070	**0.006**	0.083	0.010	0.050	0.079	0.086	0.151
11	0.173	**0.132**	0.175	0.154	0.135	0.154	0.219	0.380
12	0.150	0.110	0.137	**0.109**	0.125	0.137	0.154	0.169
13	0.228	**0.213**	0.221	**0.213**	0.219	0.235	0.234	0.285
14	0.050	0.045	0.047	0.040	**0.038**	0.055	0.058	0.123
15	0.214	0.200	0.222	**0.195**	0.210	0.210	0.211	0.232
16	0.011	0.011	0.011	**0.006**	0.008	0.014	0.017	0.068
17	0.109	**0.069**	0.094	0.079	0.084	0.088	0.143	0.208
18	0.126	**0.105**	0.146	0.126	0.144	0.137	0.160	0.206
19	0.038	0.037	0.041	**0.036**	0.040	0.041	0.041	0.049
20	**0.212**	0.226	0.246	0.219	0.226	0.246	0.280	0.342
21	0.029	**0.027**	0.034	0.028	0.031	0.035	0.035	0.043

Table 8
Classification error rate of the different optimizers on test data on the different data sets using small initialization method.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO	Full
1	0.024	0.024	**0.021**	0.038	0.037	0.025	0.037	0.037
2	0.033	**0.026**	0.033	0.062	0.095	0.032	0.066	0.056
3	0.041	**0.034**	0.040	0.059	0.121	0.043	0.057	0.083
4	0.301	**0.139**	0.270	0.282	0.312	0.306	0.308	0.327
5	0.238	**0.232**	0.233	0.252	0.242	0.238	0.242	0.257
6	0.143	**0.130**	**0.130**	0.198	0.233	0.135	0.165	0.185
7	**0.100**	0.113	0.115	0.168	0.151	**0.100**	0.121	0.134
8	0.045	**0.037**	0.052	0.038	0.427	0.065	0.240	0.085
9	0.173	**0.138**	0.152	0.224	0.237	0.176	0.223	0.317
10	0.091	**0.020**	0.097	0.055	0.300	0.062	0.208	0.151
11	0.175	**0.142**	0.171	0.298	0.329	0.175	0.163	0.380
12	0.154	**0.116**	0.137	0.193	0.202	0.139	0.206	0.169
13	0.236	**0.213**	0.221	0.244	0.323	0.232	0.289	0.285
14	0.052	**0.038**	0.048	0.082	0.100	0.050	0.100	0.123
15	0.208	**0.200**	0.224	0.219	0.454	0.232	0.275	0.232
16	0.008	**0.006**	0.011	0.079	0.115	0.023	0.059	0.068
17	0.094	**0.079**	**0.079**	0.169	0.317	0.138	0.168	0.208
18	0.127	**0.124**	0.150	0.190	0.285	0.146	0.159	0.206
19	0.039	**0.037**	0.042	0.045	0.092	0.043	0.047	0.049
20	**0.205**	0.219	0.233	0.356	0.472	0.239	0.266	0.342
21	0.031	**0.026**	0.033	0.043	0.099	0.032	0.070	0.043

Table 10
Classification error rate of the different optimizers on test data on the different data sets using large initialization method.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO	Full
1	0.028	0.023	0.024	**0.021**	0.030	0.026	0.029	0.037
2	0.031	0.026	0.034	**0.022**	0.051	0.028	0.036	0.056
3	0.040	0.036	0.044	**0.032**	0.074	0.045	0.063	0.083
4	0.295	0.223	0.254	**0.143**	0.314	0.298	0.300	0.327
5	0.234	0.233	0.232	**0.229**	0.244	0.233	0.233	0.257
6	0.144	**0.124**	0.133	**0.124**	0.159	0.133	0.150	0.185
7	0.103	**0.097**	0.115	0.107	0.144	0.120	0.131	0.134
8	0.059	0.035	0.054	**0.033**	0.084	0.048	0.063	0.085
9	0.170	0.155	0.173	**0.135**	0.206	0.176	0.206	0.317
10	0.099	**0.011**	0.102	0.020	0.126	0.069	0.107	0.151
11	0.168	**0.147**	0.166	0.156	0.291	0.166	0.221	0.380
12	0.142	**0.114**	0.137	**0.114**	0.172	0.139	0.150	0.169
13	0.228	**0.213**	**0.213**	0.217	0.239	0.223	0.233	0.285
14	0.043	0.040	0.047	**0.037**	0.087	0.048	0.075	0.123
15	0.215	0.203	0.219	**0.200**	0.226	0.210	0.216	0.232
16	0.017	**0.008**	0.008	0.008	0.020	0.017	0.017	0.068
17	0.094	**0.079**	0.084	0.089	0.193	0.089	0.164	0.208
18	0.110	**0.108**	0.147	0.120	0.216	0.144	0.170	0.206
19	0.038	0.036	0.042	**0.034**	0.047	0.041	0.041	0.049
20	**0.205**	0.226	0.233	0.226	0.349	0.239	0.301	0.342
21	0.028	0.026	0.035	**0.024**	0.046	0.036	0.035	0.043

avoid *premature convergence* and *local optima*. Furthermore, we can remark that the performance enhances using the *V-shaped* transfer function; employed in BALO-V, rather than *S-shaped* one; employed in BALO-S and BBA. This enhances in performance may be interpreted by the abrupt switching between 0 and 1 in the case of using the *V-shape* function that improves the exploration power of the algorithm.

Table 4 outlines the statistical mean fitness values obtained from the different optimization algorithms using mixed initialization. In mixed initialization, all the algorithms are initialized with some search agents closer to the optimal and some apart from the optimal and hence it provides a greater *diversity* of the population. We can highlight that this enhanced initialization helps BBA to enhance its obtained solutions while the proposed binary algorithms still keeping its enhanced performance regardless of the change in the initial population. The remark can interpret that the algorithms inspired by the ALO principles can generate much *diverse population* than the ones based on bat

Table 11
Wilcoxon rank sum test of different optimizer pairs calculated for all the data sets using different optimizers.

Optimizer_1	Optimizer_2	Small	Mixed	Large
BALO-1	**PSO**	**0.002**	**0.002**	**0.002**
BALO-1	**GAs**	**0.002**	**0.002**	**0.002**
BALO-1	**ALO**	**0.002**	**0.002**	**0.004**
BALO-1	**BBA**	**0.002**	**0.009**	**0.002**
BALO-1	**BALO-S**	**0.002**	**0.002**	**0.002**
BALO-1	**BALO-V**	**0.394**	**0.394**	**0.394**
BALO-S	**PSO**	**0.002**	**0.002**	**0.002**
BALO-S	**GAs**	**0.041**	0.699	0.589
BALO-S	**ALO**	0.699	0.818	0.589
BALO-S	**BBA**	**0.002**	**0.009**	**0.002**
BALO-S	**BALO-V**	**0.009**	**0.002**	**0.002**
BALO-V	**PSO**	**0.002**	**0.002**	**0.002**
BALO-V	**GAs**	**0.002**	**0.002**	**0.002**
BALO-V	**ALO**	**0.004**	**0.002**	**0.002**
BALO-V	**BBA**	**0.002**	**0.026**	**0.002**

Table 12
Average computational time of different optimizers averaged over the different initializers.

DS No.	ALO	BALO-1	BALO-S	BALO-V	BBA	GAs	PSO
1	54.539	58.863	61.860	62.393	53.731	**40.132**	53.547
2	62.006	57.561	67.067	66.571	51.921	**39.726**	52.254
3	56.461	59.967	71.188	71.825	56.353	**42.521**	54.642
4	67.170	78.934	84.121	83.861	66.996	**51.682**	64.257
5	66.609	74.356	83.471	78.834	65.394	**50.780**	64.933
6	43.311	47.528	51.228	51.639	39.357	**31.046**	39.771
7	51.923	48.638	58.363	58.286	43.964	**32.730**	43.534
8	560.125	456.039	464.636	466.768	481.368	**378.590**	501.268
9	42.352	44.308	50.204	50.405	38.774	**29.917**	37.528
10	79.271	80.473	82.328	85.512	70.422	**59.710**	72.661
11	55.097	45.025	62.185	62.402	40.999	**30.215**	39.253
12	44.499	46.777	52.721	53.075	38.982	**30.231**	36.557
13	75.378	72.649	71.961	75.947	65.900	**54.114**	69.213
14	49.078	49.682	54.245	54.341	44.312	**37.993**	45.567
15	1685.168	1256.171	1230.965	1276.610	1400.309	**1131.997**	1480.887
16	42.959	46.632	51.161	50.704	40.713	**30.554**	39.558
17	49.572	46.751	51.571	52.328	43.859	**34.762**	45.213
18	169.465	87.880	177.509	177.586	92.396	**22.015**	86.043
19	8598.746	6435.019	6542.570	6668.146	7189.532	**2022.297**	8356.652
20	226.738	66.486	237.080	236.964	72.810	**14.736**	54.868
21	867.887	659.962	811.685	802.000	759.117	**197.827**	710.674
Total	12,948	9819	10,418	10,586	10,757	**4363**	11,948

algorithm thanks to its selection method which adopts roulette wheel selection and the smooth switching between global and local searching while the BBA is a *global best guided* algorithm.

Table 5 outlines the mean fitness using the large initialization method. In the case of using large initialization, the search agents have low diversity, but the search agents are closer to the optimal solution. We can see that using this method did not enhance the performance of BBA as it still cannot generate diverse solution as it is *global best* guided solutions. In addition, the proposed binary algorithm is still performing better than the continuous ones and the BBA. Fig. 2 shows the effect of the initialization method on the different optimizers on the different data sets. We can remark that, the best performing initializer is the *mixed initialization* regardless of the used optimizer. This can be interpreted by the diversity in population provided by this method and the closeness of some search agents to the global optima. Furthermore, the worst performing initialization method is the small that neither provide a *diversity* of the population nor provide *closeness* to the optima.

For assessing the *repeatability* of results, Table 6 outlines the standard deviation of the obtained fitness values of different optimizers on the different data sets averaged over the different initialization methods. From the table, the proposed BALO-1 achieves the best repeatability that is evidence that the BALO-1 algorithm can reach *similar/same optima* regardless of the used initialization and the stochastic behavior of the algorithm. Moreover, although BALO-V has enhanced performance in reaching global optima, it has a lower repeatability. This lake of repeatability can be interpreted by the abrupt switching between 0 and 1 in this squashing function that enhances the global searching, but as we saw can reduce repeatability. Table 7 outlines the secondary objective in the fitness function namely average selection size. BALO-1 has much enhanced performance over the other optimizers adopted in the paper and can be interpreted by the enhanced convergence capability of BALO-1 which implicitly minimizes the selected feature number; as indicated in Table 7.

The performance of the different optimizers on the test data using small, mixed and large initialization methods is summarized in Tables 8–10. From the tables, the performance using *full feature set* selected is worse than that using wrapper-based approaches regardless of the used optimizer. Moreover, BALO-1 is still performing better than the other optimizers regardless of the used initialization method. The flavor of using BALO-1 is more certain in the case of using the small initialization as the algorithm can enhance the diversity of the population and reach the global optima. Using mixed initialization improves the performance of all optimization algorithms. Also, the performance of the proposed binary algorithms is enhanced even at respectively high dimensional data set. For assessing the significance of performance *Wilcoxon rank sum* measure is used and the results are outlined in Table 11. Wilcoxon tests the null hypothesis that data in x and y vectors are samples from continuous distributions with equal medians, against the alternative that they are not. The proposed BALO-1 has a significant enhance over all other optimizers except for the BALO-V at a significance level of 0.05 and similar results for the BALO-V with all other optimizers, as indicated in Table 11. Furthermore, there is no significant difference between BALO-S and ALO regardless of the used initialization methods while BALO-S has a significant difference in comparison to PSO and GAs in the case of using small initialization.

Table 12 outlines the average computational time of different optimization algorithms. All optimizers are using the same number of evaluation functions and hence we used the computational time to compare the performance of the algorithms. In Table 12, the GAs have the best computational time in comparison to all other algorithms. Furthermore, the proposed binary algorithms have computational speed compared with other algorithms except GAs. We can remark that the BALO-1 has minimum run time in the binary algorithms thanks to its simple operators that depend on simple mutation and crossover. In addition, BALO-S and BALO-V have comparable computational time while there is significant difference between them in convergence capability.

6. Conclusion and future work

In this paper, binary variants of the ant lion optimizer are proposed and applied for feature selection in wrapper mode. The continuous version of ant lion optimizer (CALO) is converted into the binary form using either V-shaped or S-shaped functions or simply inspiring the basic operators of ALO and applying binary corresponding ones. The proposed approaches are applied and used for feature selection in machine learning domain using different initialization methods to assess different searching capabilities of the algorithms. The proposed binary algorithms are applied in the feature selection domain for evaluation and results are compared against well-known feature selection methods particle swarm optimizer (PSO), genetic algorithm (GA), continuous ALO, and binary bat algorithm (BBA). The evaluation is performed using a set of evaluation criteria to assess different aspects of performance. Results outlined show that the proposed BALO algorithm can adaptively search the space of features optimally and be converging to optimal/near optimal solution better than the other continuous and binary algorithms. We can see that the ALO-based binary algorithm can achieve the required diversity in population and can smoothly switch between exploration and exploitation and hence can avoid premature convergence. Moreover, results prove that the binary algorithm proposed based on *V-shaped* functions performs better than the *S-shaped* ones. In addition, we can see that binary algorithms with search agents guided by the global best only have worse performance in comparison to the binary algorithms that depend on roulette wheel selection such as ALO-based ones. The proposed binary algorithms provide much repeatability of results and convergence speed better than other algorithms adopted in the paper. We can conclude that the initialization of algorithms can affect the performance of the wrapper-based algorithms and hence we recommend using the mixed initialization method.

Acknowledgment

This work was partially supported by the IPROCOM Marie Curie initial training network, funded through the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement no. 316555. This fund only applies for one co-authors (Hossam M. Zawbaa).

References

[1] I. Guyon, A. Elisseeff, An introduction to variable and attribute selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[2] D.Y. Harvey, M.D. Todd, Automated feature design for numeric sequence classification by genetic programming, in: IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 474–489.
[3] K. Michalak, Selecting best investment opportunities from stock portfolios optimized by a multiobjective evolutionary algorithm, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO), 2015, pp. 1239–1246.
[4] F. Daolio, A. Liefooghe, S. Verel, H. Aguirre, K. Tanaka, Global vs local search on multi-objective NK-landscapes: contrasting the impact of problem features, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO), 2015, pp. 559–566.
[5] B. Xue, M. Zhang, W.N. Browne, Particle swarm optimisation for feature selection in classification: novel initialisation and updating mechanisms, Appl. Soft Comput. 18 (2014) 261–276.
[6] B. Xue, M. Zhang, W.N. Browne, Particle swarm optimization for feature selection in classification: a multi-objective approach, IEEE Trans. Cybern. 43 (6) (2013) 1656–1671.
[7] F. Valdez, Bio-inspired optimization methods, in: Springer Handbook of Computational Intelligence, 2015, pp. 1533–1538.
[8] S. Shoghian, M. Kouzehgar, A comparison among wolf pack search and four other optimization algorithms, Int. J. Comput. Electr. Autom. Control Inf. Eng. 6 (12) (2012) 1619–1624, World Academy of Science, Engineering and Technology.
[9] L.N. Vitorino, S.F. Ribeiro, C.J.A. Bastos-Filho, A mechanism based on artificial bee colony to generate diversity in particle swarm optimization, Neurocomputing 148 (2015) 39–45.
[10] M. Maeda, S. Tsuda, Reduction of artificial bee colony algorithm for global optimization, Neurocomputing 148 (2015) 70–74.
[11] C. Segura, S.B. Rionda, A.H. Aguirre, S.I.V. Pena, A novel diversity-based evolutionary algorithm for the traveling salesman problem, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO), 2015, pp. 489–496.
[12] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[13] E.C. Goncalves, A. Plastino, A.A. Freitas, Simpler is better: a novel genetic algorithm to induce compact multi-label chain classifiers, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO), 2015, pp. 559–566.
[14] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Japan, 1995, pp. 39–43.
[15] G. Zhang, Y. Li, Cooperative particle swarm optimizer with improved elimination mechanism for global optimization, in: IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 117–124.
[16] R. Azizi, Empirical study of artificial fish swarm algorithm, Int. J. Comput. Commun. Netw. 3 (1) (2014) 1–7.
[17] S. Mirjalili, The ant lion optimizer, Adv. Eng. Softw. 83 (2015) 83–98.
[18] J. Kennedy, R.C. Eberhart, A discrete binary version of the particle swarm algorithm, IEEE Int. Conf. Syst. Man Cybern. 5 (1997) 4104–4108.
[19] H.A. Firpi, E. Goodman, Swarmed feature selection, in: Proceedings of the 33rd Applied Imagery Pattern Recognition Workshop, IEEE Computer Society, Washington, DC, USA, 2004, pp. 112–118.
[20] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi, BGSA: binary gravitational search algorithm, Nat. Comput. 9 (2010) 727–745.
[21] R.Y.M. Nakamura, L.A.M. Pereira, K.A. Costa, D. Rodrigues, J.P. Papa, X.S. Yang, BBA: a binary bat algorithm for feature selection, in: IEEE XXV Conference on Graphics, Patterns and Images, 2012, pp. 291–297.
[22] K. Suresh, N. Kumarappan, Hybrid improved binary particle swarm optimization approach for generation maintenance scheduling problem, Swarm Evol. Comput. 9 (2013) 69–89.
[23] Y. Zhang, S. Wang, P. Phillips, G. Ji, Binary PSO with mutation operator for feature selection using decision tree applied to spam detection, Knowl.-Based Syst. 64 (2014) 22–31.
[24] M.A.K. Azad, A.M.A.C. Rocha, E.M.G.P. Fernandes, Improved binary artificial fish swarm algorithm for the 0-1 multidimensional knapsack problems, Swarm Evol. Comput. 14 (2014) 66–75.
[25] E. Emary, H.M. Zawbaa, A.E. Hassanien, Binary grey wolf optimization approaches for feature selection, Neurocomputing 172 (2016) 371–381.
[26] X.B. Mengab, X.Z. Gaoc, L. Lude, Y. Liub, H. Zhanga, A new bio-inspired optimisation algorithm: bird swarm algorithm, J. Exp. Theor. Artif. Intell. (2015) 1–15.
[27] H. Chiroma, N.L.M. Shuib, S.A. Muaz, A.I. Abubakar, L.B. Ila, J.Z. Maitama, A review of the applications of bio-inspired flower pollination algorithm, in: International Conference on Soft Computing and Software Engineering (SCSE), Procedia Computer Science, vol. 62, 2015, pp. 435–441.
[28] B. Niua, J. Wang, H. Wang, Bacterial-inspired algorithms for solving constrained optimization problems, Neurocomputing 148 (2015) 54–62.
[29] C. Cosariu, A. Iovanovici, L. Prodan, M. Udrescu, M. Vladutiu, Bio-inspired redistribution of urban traffic flow using a social network approach, in: IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 77–84.
[30] K. Moodley, J. Rarey, D. Ramjugernath, Application of the bio-inspired Krill Herd optimization technique to phase equilibrium calculations, Comput. Chem. Eng. 74 (2015) 75–88.
[31] L.Y. Chuang, H.W. Chang, C.J. Tu, C.H. Yang, Improved binary PSO for feature selection using gene expression data, Comput. Biol. Chem. 32 (2008) 29–38.
[32] S. Mirjalili, S.Z.M. Hashim, BMOA: binary magnetic optimization algorithm, Int. J. Mach. Learn. Comput. 2 (3) (2012).
[33] A. Frank, A. Asuncion, UCI Machine Learning Repository, 2010.
[34] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, New York, 2001.
[35] J. Kennedy, R. Eberhart, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 2009, pp. 1942–1948.
[36] A.E. Eiben, P.E. Raue, Zs. Ruttkay, Genetic algorithms with multi-parent recombination, in: Proceedings of the International Conference on Evolutionary Computation, Third Conference on Parallel Problem Solving from Nature, 1994, pp. 78–87.
[37] S. Mirjalili, S.M. Mirjalili, X. Yang, Binary bat algorithm, neural computing and applications, J. Neural Comput. Appl. 25 (3) (2014) 663–681.
[39] F. Wilcoxon, Individual comparisons by ranking methods, Biometr. Bull. 1 (6) (1945) 80–83.
[40] H. Dong, X. Teng, Y. Zhou, J. He, Feature subset selection using dynamic mixed strategy, in: IEEE Congress on Evolutionary Computation (CEC), 2015, pp. 672–679.
[41] B. Xue, M. Zhang, W.N. Browne, Particle swarm optimisation for feature selection in classification: novel initialisation and updating mechanisms, Appl. Soft Comput. 18 (2014) 261–276.

E. Emary was born in Sharkia, Egypt in 1979. He received his B.Sc. degree in 2001 and M.Sc. degree in 2003, from Faculty of Computers and Information, Information Technology Dept., Cairo University, Egypt. Currently, he is a Lecturer at Information Technology Dept., Faculty of Computers and Information, Cairo University, Egypt. He has authored/co-authored over 20 research publications in peer-reviewed reputed journals, book chapters and conference proceedings. He has served as the technical program committee member of various international conferences and reviewer for various international journals. His research interests are in the areas of computer vision, pattern recognition, video and image processing, machine learning, data mining, and biometrics.

Hossam M. Zawbaa was born in Cairo, Egypt in 1987. He received his B.Sc. degree in 2008 and M.Sc. degree in 2012, Cairo University, Egypt. He is Lecturer, Faculty of Computers and Information, Beni-Suef University, Egypt. Currently, he is Research Assistant at Babes-Bolyai University, Romania. He has authored/coauthored over 45 research publications in peer-reviewed reputed journals and international conference proceedings. He has served as the technical program committee member of various international conferences and reviewer for various international journals. His research interests are in the areas of machine learning, intelligent optimization, data mining, and biometrics.

Aboul Ella Hassanien (Abo) received his B.Sc. with honors in 1986 and M.Sc. degree in 1993, both from Ain Shams University, Faculty of Science, Pure Mathematics and Computer Science Department, Cairo, Egypt. On September 1998, he received his doctoral degree from the Department of Computer Science, Graduate School of Science & Engineering, Tokyo Institute of Technology, Japan. He is a Full Professor at Cairo University, Faculty of Computer and Information, IT Department. Professor Abo is the founder and chair the scientific research group in Egypt, www.egyptscience.net/. He has authored/co-authored over 500 research publications in peer-reviewed reputed journals, book chapters and conference proceedings. He has served as the general chair, co-chair,program chair, program committee member of various international conferences and reviewer for various international journals. His research interests include Computational intelligence, medical image analysis, security, animal identification and multimedia data mining.