Contents lists available at ScienceDirect

Applied Soft Computing

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

# Multi swarm bare bones particle swarm optimization with distribution adaption

Reza Vafashoar*, Mohammad Reza Meybodi

*Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran*

A R T I C L E  I N F O

*Article history:*
Received 9 November 2015
Received in revised form 30 April 2016
Accepted 19 June 2016
Available online 23 June 2016

*Keywords:*
Particle swarm optimization
Multi-swarm PSO
Global numerical optimization
Learning automata
Cellular learning automata

A B S T R A C T

Bare bones PSO is a simple swarm optimization approach that uses a probability distribution like Gaussian distribution in the position update rules. However, due to its nature, Bare bones PSO is highly prone to premature convergence and stagnation. The characteristics of the probability distribution functions used in the update rule have a tense impact on the performance of the bare bones PSO. As a result, this paper investigates the use of different methods for estimating the probability distributions used in the update rule. Four methods or strategies are developed that are using Gaussian or multivariate Gaussian distributions. The choice of an appropriate updating strategy for each particle greatly depends on the characteristics of the fitness landscape that surrounds the swarm. To deal with issue, the cellular learning automata model is incorporated with the proposed bare bones PSO, which is able to adaptively learn suitable updating strategies for the particles. Through the interactions among its elements and the learning capabilities of its learning automata, cellular learning automata gradually learns to select the best updating rules for the particles based on their surrounding fitness landscape. This paper also, investigates a new and simple method for adaptively refining the covariance matrices of multivariate Gaussian distributions used in the proposed updating strategies. The proposed method is compared with some other well-known particle swarm approaches. The results indicate the superiority of the proposed approach in terms of the accuracy of the achieved results and the speed in finding appropriate solutions.

© 2016 Elsevier B.V. All rights reserved.

## 1. Introduction

Particle swarm optimization (PSO) is a stochastic search approach inspired from the intelligent social behavior that exists among some animals like bird and fish [1]. It consists of a swarm of particles flying in a search space for better solutions. The flight which is guided by the velocity of particles resembles an iterative search process. Particles adjust their velocities in this iterative search based on their own and social experiences. They can memorize their historical best visited locations and the historical best visited location of the whole swarm.

Traditional PSO suffers from premature convergence and low exploration ability, especially, when dealing with complex multi modal problems [2,3]. As a result, many researches proposed various approaches to remedy this problem. Parameter adaption [4–7], hybridization [8–10], incorporation of different updating, selection, mutation strategies [11–13], topology variations [14–17], and integration of learning schemes [18–21] are some of the well-studied methods.

Studying the population dynamics and its convergence behavior [22], Kennedy suggested that the new positions of particles can be obtained by sampling from an explicit probability distribution rather than using velocity information. In their work, the Gaussian distribution is used for the acquisition of new positions, and its parameters for a particular particle are exclusively defined by the informers (particles utilized in its position updating) of the particle. The mean of the Gaussian distribution used in the position update rule of a particular particle is defined as the average of the particle's own historical best position and the swarm historical best position. Also, the distance of these two positions defines the standard deviation of the distribution [23].

Bare bones PSO (BBPSO) still suffers from slow and premature convergence and stagnation. In the latter case, the swarm converges to non-optimal points [24–27]. Considering the definition of updating distribution, the best particle of each iteration remains unchanged during that iteration, which may result in stagnation. BBPSO with mutation and crossover operations (BBPSO-MC) handles this issue by employing the mutation strategy of differential evolution algorithms in the update rule of the best particle [24].

* Corresponding author.
*E-mail addresses:* vafashoar@aut.ac.ir (R. Vafashoar), mmeybodi@aut.ac.ir (M.R. Meybodi).

http://dx.doi.org/10.1016/j.asoc.2016.06.028
1568-4946/© 2016 Elsevier B.V. All rights reserved.

Other mutation strategies like Gaussian or Cauchy are also examined for improving bare bones PSO [25,28,29]. Krohling and Mendel proposed a jumping mechanism for avoiding local optima. Their jumping mechanism is based on the Gaussian or Cauchy noises which are added to the position of particles when they don't improve for a period of time [28].

Hybridizing bare bones PSO with other methods is also studied by some researchers. Gao et al. introduced a bee colony algorithm which produces candidate individuals in the onlooker phase based on the update strategies of bare bones PSO. Because the particles in bare bones PSO are attracted to the best individuals, the authors considered this step as an improved exploitation phase [30].

Chen introduced a variant of bare bones PSO which uses a mixture of two centers in the Gaussian distribution updating the particle positions. One of the centers is using local best positions and is considered to have exploration characteristics, while the other is based on the global best position and is used for better exploitation around the best particle. At early iterations, exploration has a bigger weight which gradually decreases in the favor of exploitation [31,32].

Some other variants of bare bones PSO utilize heavy tail probability distributions instead of Gaussian distribution for more explorative behavior [25,29]. Li and Yao introduced a Cooperative Particle Swarms optimization method (CCPSO) for Large Scale Optimization. The population of CCPSO is divided into several swarms, and the swarms are updated cooperatively. The position of a particle at each step is updated similar to bare bones PSO; however, it uses different probability distributions for position sampling: a Gaussian distribution with its center at the best historical position of the particle itself, and a Cauchy distribution whose center is at the local best position. Either of the two distributions will be selected with a fixed probability for updating the particle position in each dimension. CCPSO, also, changes the number of swarms when the global best fitness value is not improving; this way, it can adaptively adjust the number of swarms. BBPSO with scale matrix adaptation (SMA-BBPSO) uses multivariate t-distribution with adaptive scale matrices in its update rule. It employs a simple, yet effective, method for adapting the scale matrices associated with the particles: the best position found by any particle in the neighborhood of a particular particle is sampled in its associated scale matrix for the next iteration.

This paper investigates the use of various probability models for updating the positions of particles in BBPSO. Each model can benefit the search process depending on the shape of the landscape where the particles reside. Some of the studied models involve multivariate Gaussian distributions for updating particle positions. Accordingly, a simple method for adapting the covariance matrices of these Gaussian distributions will be introduced. To achieve a good performance, a strategy selection mechanism is needed. This mechanism should select an appropriate updating strategy for each particle based on the properties of the particle's surrounding landscape. To achieve this objective, the swarms are embedded in a cellular learning automata (CLA) [33–36], which adaptively learns the best updating models for each particle. CLA is a hybrid mathematical model for many decentralized problems (decentralized problems can be considered as multiple systems where each system has its own processing unit, and can make decisions according to on its own knowledge). It can be considered as a network of agents with learning abilities. These agents have the ability to interact and learn from each other through a set of CLA rules. CLA has been utilized in many scientific domains such as deployment and clustering in wireless sensor networks [37,38], dynamic channel assignment in cellular networks [39], evolutionary computing [40], and graph coloring [41].

The rest of the paper is organized as follows: Section 2 gives a brief review on the involved concepts such as learning automata, cellular learning automata and particle swarm algorithm. Section 3 gives the detailed description of the proposed approach. Experimental results and comparison to other algorithms comes in Section 4, and finally Section 5 contains conclusions.

## 2. The involved concepts

This section provides a brief description of the concepts that are used in the paper. It starts with an introduction on particle swarm optimization and Bare Bones PSO; then, reviews the basic concepts of cellular learning automata.

### 2.1. PSO and related topics

Traditional PSO utilizes a population of particles called swarm. Each particle $X_i$ in the swarm keeps information about its current position, its personal best visited position known as pbest, and its flying velocity, respectively, in three $D$ dimensional real valued vectors: $x_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]$, $p_i = [p_{i1}, p_{i2}, \ldots, p_{iD}]$, and $v_i = [v_{i1}, v_{i2}, \ldots, v_{iD}]$. All particles know the best position experienced by the whole swarm known as gbest, which is also represented by a vector like $p_g = [p_{g1}, p_{g2}, \ldots, p_{gD}]$. The search in a $D$ dimensional space is an iterative process, during which the velocity and position of a particle like $X_i$ are updated according to the following equations:

$$\begin{aligned} v_{id}(k+1) &= \omega v_{id}(k) + c_1 r_1 (p_{id} - x_{id}(k)) + c_2 r_2 (p_{gd} - x_{id}(k)) \\ x_{id}(k+1) &= x_{id}(k) + v_{id}(k+1) \end{aligned} \tag{1}$$

where $v_{id}(k)$ is the *dth* dimension of the velocity vector of the particle in step $k$; $x_{id}(k)$ and $p_{id}(k)$ are respectively the *dth* dimension of its position and historical best position vectors; $p_{gd}(k)$ represents the *dth* dimension of the historical best position of the whole swarm in step $k$; $\omega$ is the inertia weight, which was introduced to bring a balance between the exploration and exploitation characteristics [3]; $c_1$ and $c_2$ are acceleration constants that represent cognitive and social learning weights; and, finally, $r_1$ and $r_2$ are two random numbers from the uniform distribution $u(0, 1)$. After acquiring the new positions of the particles, the historical best position of each particle is updated; which may, also, affect the historical global best position of the swarm.

### 2.2. Bare bones particle swarm optimization

The observation of the motion of individual particles in the original PSO inspired the development of bare bones PSO. In traditional PSO, the distribution of the positions visited by a single particle resembles the bell shaped curve of a Gaussian distribution. Accordingly, BBPSO uses Gaussian probability distribution to generate the new position of a particle like $X_i$ [23]:

$$x_{id}(k+1) = \frac{p_{id} + pl_{id}}{2} + N(0, 1) \times |p_{id} - pl_{id}| \tag{2}$$

where $N(0,1)$ denotes a random number taken from the normal distribution with mean 0 and standard deviation 1. *pl* is the best location, visited so far, in the neighborhood of the particle $X_i$, and $p_i$ is the personal best location of the particle itself.

### 2.3. Learning automaton

A variable structure learning automaton (LA) can be represented by a sextuple like $\{\Phi, \alpha, \beta, A, G, P\}$ [42]. where $\Phi$ is a set of internal states; $\alpha$ is a set of outputs or actions of the learning automaton; $\beta$ is a set of inputs or environmental responses; $A$ is a learning algorithm; $G(.)$: $\Phi \to \alpha$ is a function that maps the current state into the current output; and $P$ is a probability vector that determines the selection probability of an state at each stage. There is usually a one to one correspondence between $\Phi$ and α, as a results, the two terms can be used interchangeably.

The objective of a learning automaton is to identify the best action which maximizes the expected received payoff from the environment [43]. To attain this goal, the learning automaton selects an action according to its action probability distribution, and then applies this action to the environment. The environment measures the favorability of the received action, and responds the LA with a noisy reinforcement signal β. The learning automaton uses this response to adjust its internal action probabilities via its learning algorithm. Consider the environmental response to a selected action like $a_i$ at step $k$ be $\beta \in \{0,1\}$, where 0 and 1 are used as pleasant and unpleasant responses respectively. The internal action probability can be updated according to equation 3 when the response is pleasant and according to equation 4 when it is not.

$$p_j(k+1) = \begin{cases} p_j(k) + a(1-p_j(k)) & if\ i=j \\ p_j(k)(1-a) & if\ i \neq j \end{cases} \tag{3}$$

$$p_j(k+1) = \begin{cases} p_j(k)(1-b) & if\ i=j \\ \dfrac{b}{r-1} + p_j(k)(1-b) & if\ i \neq j \end{cases} \tag{4}$$

In these two equations, $a$ and $b$ are called reward and penalty parameters, respectively. For $a = b$ the learning algorithm is called $L_{R-P}$; when $b \ll a$, it is called $L_{R\varepsilon P}$; and when $b$ is zero it is called $L_{R-I}$

### 2.4. Cellular automaton

A cellular automaton (CA) [44] is a collection of cells that are usually arranged in a regular form such as grid. Each cell has a finite number of states, and in each time step, it is in one of these states. The cells evolve in time using a set of finite rules and local interactions. The local interactions happen in the neighborhood of cells which is defined by the topology of the CA. At each time step, the set of rules defines the state of a cell based on the state of its neighboring cells at the previous time step.

A cellular automaton starts from some initial state, and evolves for some time based on its rules, which results in complex patterns of behavior. The ability to produce different patterns using only local interactions and a set of simple rules has made cellular automaton a popular tool for modeling complicated systems and natural phenomena.

### 2.5. Cellular learning automata

Cellular learning automata (CLA) is a hybrid model which is based on cellular automata and learning automata theories. It is a collection of learning automata residing in the cells of a cellular automaton. Each cell of the cellular automaton contains one or more learning automata, and the state of the cell is determined by the states of its beholding learning automata. The neighboring cells of a particular cell like $c$ constitute the environment of the learning automata residing in $c$. The interactions among the cells are controlled using a set of rules, termed as CLA rules.

A CLA starts from some initial state, which is the state of all of its cells. During each step, the learning automata select actions based on their internal probability distributions. The actions are applied to the environment of the corresponding learning automata, and based on the CLA rules, reinforcement signals are returned to the learning automata. Then, each learning automaton adjusts its internal probability distribution according to its received signals. This procedure continues for period of some steps until a termination condition is satisfied. Mathematically, a CLA can be defined as follows. A $d$-dimensional CLA with a single learning automaton in each cell is a structure $A = \left(Z^d, N, \phi, A, F\right)$ where:

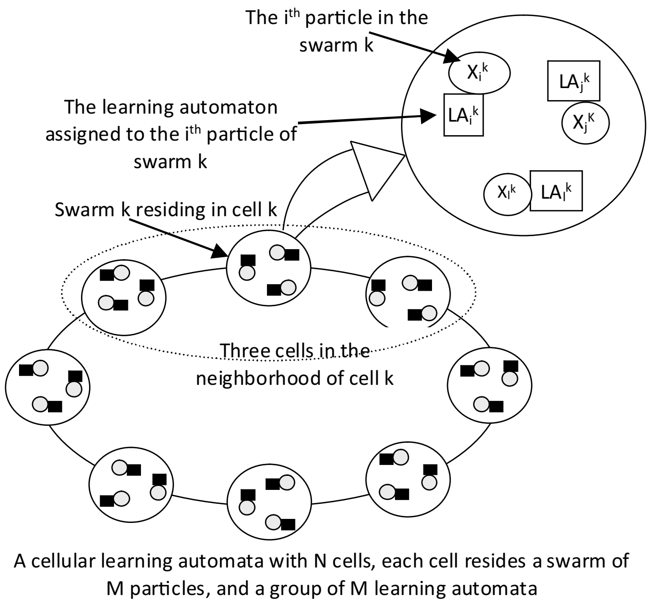

The i^th particle in the swarm k
The learning automaton assigned to the i^th particle of swarm k
Swarm k residing in cell k
$X_i^k$
$LA_j^k$
$LA_i^k$
$X_j^K$
$X_l^k$
$LA_l^k$
Three cells in the neighborhood of cell k
A cellular learning automata with N cells, each cell resides a swarm of M particles, and a group of M learning automata

**Fig. 1.** Topology of CLA-BBPSO.

$Z^d$ is a lattice of $d$-tuples of integer numbers.

$N = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_m\}$ is a finite subset of $Z^d$ called neighborhood vector, where $\bar{x}_i \in Z^d$.

$\phi$ is a finite set of states. The state of a cell like $c_i$ is denoted by $\phi_i$.

$A$ is a set of learning automata each of which is assigned to one cell of the CLA.

$F^i : \underline{\phi}_i \rightarrow \underline{\beta}$ is the local rule of the CLA in each cell $c_i$, where $\underline{\beta}$ is the set of values that the reinforcement signal can take.

## 3. A CLA based multi swarm bare bones optimization: CLA-BBPSO

After its initial development, several variants of BBPSO have been introduced in the literature; some authors have studied different probability distributions or strategies for updating the particle positions in the search space. Some strategies are considered to have better exploration or exploitation characteristics. In order to obtain better results, like other swarm approaches, BBPSO requires an appropriate balance between these two characteristics. However, achieving this issue is not an easy task, as it is highly dependent on the problem landscape and the evolutionary state of the search. Also, some strategies may show different exploration (exploitation) behaviors depending on the region of the search landscape occupied by the population. Accordingly, an appropriate updating strategy depends on the characteristics of the fitness landscape and the positions of particles. In the current work we developed four updating strategies, with different properties, for each particle. The choice of an appropriate strategy for each particle requires an understanding of the fitness landscape, which is not known priori. To tackle these issues, this paper introduces a CLA based multi-swarm BBPSO which learns the appropriate strategy for each particle based on the characteristics of the region of the fitness landscape that surrounds the particles.

In the proposed model, the whole population is organized into several swarms, each of which is embedded in one cell of a CLA which has a ring topology (Fig. 1). Every cell of the CLA has a neighborhood radius of 1, and the immediate left and right cells (beside itself) in the ring constitute its neighbors. Each cell contains a group of learning automata; each LA has four actions, and is associated with a particular particle residing in the cell. There is a one to

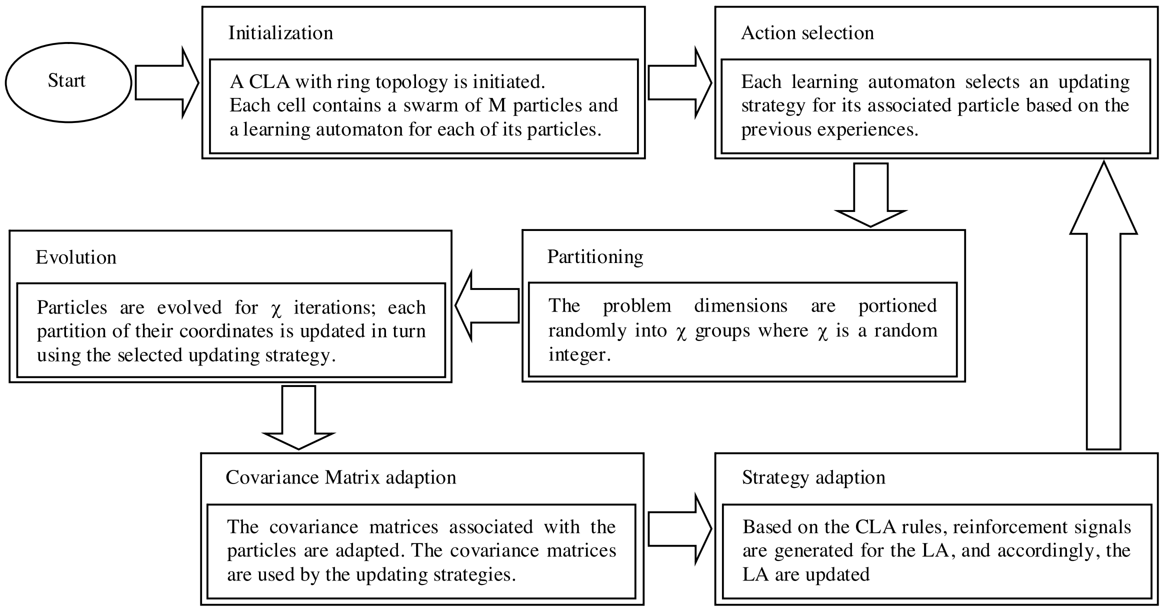

Start
Initialization
A CLA with ring topology is initiated. Each cell contains a swarm of M particles and a learning automaton for each of its particles.
Action selection
Each learning automaton selects an updating strategy for its associated particle based on the previous experiences.
Partitioning
The problem dimensions are portioned randomly into χ groups where χ is a random integer.
Evolution
Particles are evolved for χ iterations; each partition of their coordinates is updated in turn using the selected updating strategy.
Covariance Matrix adaption
The covariance matrices associated with the particles are adapted. The covariance matrices are used by the updating strategies.
Strategy adaption
Based on the CLA rules, reinforcement signals are generated for the LA, and accordingly, the LA are updated

**Fig. 2.** A block diagram for CLA-BBPSO.

**Table 1**
A brief definition of notations used in the paper.

| Notation | Description | Notation | Description |
|---|---|---|---|
| $X_i^k$ | the $i^{th}$ particle in the $k^{th}$ swarm | $x_i^k(t)$ | the current position of $X_i^k$ at time $t$ |
| $p_i^k(t)$ | personal best position of $X_i^k$ at time $t$ | $p_g^k(t)$ | the global best position of the $k^{th}$ swarm |
| $p_{id}^k(t)$ | the value of $p_i^k(t)$ in the $d^{th}$ dimension | $\theta(k,j)$ | the $j^{th}$ neighbor of the $k^{th}$ cell (or swarm). $j=0, 1, 2$ respectively refer to the $k^{th}$ cell (swarm), its left cell (swarm), and right cell (swarm). |
| $p_{iI}^k(t)$ | The value of $p_i^k(t)$ in dimensions presented by $I$, where $I \subseteq \{1, 2, \ldots, D\}$ | $I_i^k(t)$ | the partition of dimensions which is used by $X_i^k$ at time $t$. |
| $g^k(t)$ | the index of the best particle in the $k^{th}$ swarm | $D$ | problem dimension |
| $M$ | The size of each swarm | $LA_i^k$ | The learning automaton associated with $X_i^k$ |

one correspondence between the actions of a learning automaton and the updating strategies of its associated particle. The learning automata govern the strategy application of their related particles. The proposed model uses multiple swarms rather than a single one to maintain a natural diversity in the population. Also, the interactions that exist among the neighboring cells and swarms enable CLA to learn suitable strategies (actions) for its beholding particles. A general block diagram for CLA-BBPSO is given in Fig. 2, and its details are described in the following sections. Before continuing, Table 1 summarizes the notations that will be used in the rest of the paper.

## 3.1. Action selection and partitioning phase

Section 3.2 defines four updating rules and their corresponding operators for the proposed BBPSO. At each time step, a particle can only use one of these operators to steer its movement. Therefore, at each step, a particle needs to select an operator out of the available ones. However, as discussed earlier, an appropriate strategy for a particle depends on the region of a fitness landscape where the population resides. Accordingly, the suitable strategies for a particle may change in the course of the search. Strategy selection may be considered as an optimization problem: for each particle select a sequence of operators in a way that maximizes the probability of reaching a global optimum and minimizes the time required to do so. Finding such a sequence may be much harder than solving the original optimization problem, and in fact may be impossible.

Some of the recent intelligent strategy selection mechanisms developed for evolutionary or swarm optimization methods are based on reinforcement learning. Gong et al. [48,49] used probability matching and adaptive pursuit for adaptive strategy selection in deferential evolution algorithm. They used reinforcement learning to adaptively learn the favorability of considered strategies during the evolutionary process. Self-Learning Particle Swarm Optimizer [19] uses the same concept for adaptive strategy selection in PSO. CLA-BBPSO utilizes a CLA which gradually learns the most appropriate operators for each particle as the search proceeds. CLA is based on ideas from reinforcement learning and cellular automata. It can be considered as a complex system of learning automata where each automaton, besides its own actions, can learn from the actions of other learning automata in its neighborhood. This way, a learning automaton can utilize the experience of other nearby entities.

The evolutionary cycle of CLA-BBPSO starts with the action selection phase. During this phase, each learning automaton selects an action based on its internal probability vector. Each selected action corresponds to one updating strategy that will be applied to the particle associated with the corresponding learning automaton. At the beginning, all actions of a learning automaton have equal probabilities. As the search proceeds, the learning automata adjust their action probabilities in the strategy adaption phase of the algorithm, and accordingly, more appropriate strategies can be selected for the particles.

The next phase of the algorithm involves partitioning the coordinates of each particle. The coordinates associated with each partition will be evolved separately using the selected strategy for the particle. Accordingly, if the coordinates of a particle are divided into $\chi$ partitions, the particle will be updated for $\chi$ times during the next evolutionary phase. As we intend to evolve all of the particles for the same number of times during each evolutionary phase (to attain a fair comparison of their performances), the coordinates of each particle are partitioned in a same manner as other particles. To achieve this objective, first, $D$ is partitioned into $\chi$ random summands ($D$, problem dimensionality, is an integer number):

$$D=\sum_{i=1}^{\chi}a_i$$
$$where:\ \ a_i=2^k,\ \ k=\{1,2,\ldots,MAXP\};\ i<\chi\ and\ a_\chi\in\mathbb{Z}^+-\{1\} \tag{5}$$

Here *MAXP* is a parameter controlling the maximum size of each partition. As the summands are generated randomly, their number ($\chi$) may vary in different partitioning phases. Using these ordered summands, the coordinates of each particle are partitioned randomly into $\chi$ groups where the size of the *ith* group equals to the *ith* summand ($a_i$):

$$\bigcup_{j=1}^{\chi}I_i^k(t+j)=\{1,2,\ldots,D\}$$
$$where: \tag{6}$$
$$|I_i^k(t+j)|=a_i$$

Here $I_i^k(t+j)$ is the *jth* partition of coordinates for the *ith* particle of swarm $k$, which will be used during the $(t+j)th$ iteration of the evolution phase.

### 3.2. Updating rules of CLA-BBPSO

Considering different aspects of problems, we define four updating strategies for CLA-BBPSO. The strategies have different characteristics that enhance various aspects of the search. The first strategy is the classical update rule of the BBPSO algorithm. It uses a Gaussian sampling which is based on the information present in the neighborhood of a particle.

**Operator A, Gaussian operator:**

If $d\in I_i^k(t)$:

$$x_{id}^k(t)=\begin{cases}p_{id}^k(t-1)+p_{gd}^k(t-1)\ /2+N\ 0,1\ \times|p_{gd}^k(t-1)-p_{id}^k(t-1)|if(i,k)\neq(g^k(t-1),k)\\ p_{id}^k(t-1)+p_{gd}^{\theta(k,r)}(t-1)\ /2+N\ 0,1\ \times|p_{gd}^{\theta(k,r)}(t-1)-p_{id}^k(t-1)|otherwise\end{cases} \tag{7}$$

If $d\notin I_i^k(t)$:

$$x_{id}^k(t)=p_{id}^k(t-1)$$

This operator alters the *dth* dimension of a particle using a random number taken from Gaussian probability distribution. For particles other than a swarm's best particle, $\left(p_{id}^k(t-1)+p_{gd}^k(t-1)\right)/2$ and $|p_{gd}^k(t-1)-p_{id}^k(t-1)|$ will be used as the mean and standard deviation of the Gaussian distribution. Therefore, non-global best particles in a swarm use inter swarm information to guide their movements. These particles aim for better exploitation of the found promising locations in the swarm. In order to explore the best locations out of the swarm, the global best particle of the swarm uses $\left(p_{id}^k(t-1)+p_{gd}^{\theta(k,r)}(t-1)\right)/2$ and $|p_{gd}^{\theta(k,r)}(t-1)-p_{id}^k(t-1)|$ as the parameters of the Gaussian operator where $r\in\{1,2\}$ is a random number identifying a random neighboring swarm. In order to prevent drastic random changes and to utilize previously learned information, a particle retains some of its historical best position's information in its new position and updates the others as stated previously. $I(t)$ identifies the dimensions of a particle to be updated during the iteration.

When the new position in dimension $d$ falls out of the search range, it is re-generated (in dimension $d$) using the operator A until it sticks inside the search range.

**Operator B, multivariate Gaussian operator with covariance matrix Σ:**

$$x_i^k(t)=\begin{cases}N\ p_i^k(t-1)+p_g^k\ /2,\sum_i^k\ if(i,k)\neq(g^k(t-1),k)\\ N\ p_i^k(t-1)+p_g^{\theta(k,r)}(t-1)\ /2,\sum_i^k\ otherwise\end{cases} \tag{8}$$

Operator A changes the value of each coordinate in the position vector independent from the values of other coordinates. However, it is useful to consider the existent interdependencies among the coordinates of the problem landscape. Covariance is a measure of how much two random variables change together, and a multivariate Gaussian distribution utilizing a covariance matrix can represent the dependencies among the dimensions of a landscape. Motivated from this idea, some evolutionary algorithms manage the correlations between coordinates through covariance matrices [45–47]. Operator B changes the position of a particle based on the multivariate Gaussian distribution with mean $\mu$ and covariance matrix $\Sigma$. When a particle is not the best particle of its swarm, $\left(p_i^k+p_g^k\right)/2$ will be used as the mean of its associated distribution; otherwise, $\left(p_i^k+p_g^{\theta(k,r)}\right)/2$ will be used (similar to operator A). The covariance matrix associated with a particular particle is adapted during the search (as will be discussed later).

Re-generating a vector for handling out of range search may be computationally inefficient; therefore, the method presented in [19] will be used when operator B results in out of range positions:

$$x_{id}^k(t)=\begin{cases}x_{id}^k(t)\ if\ x_{id}^k(t)\in[x_{d,\min},x_{d,\max}]\\ u\ p_{id}^k(t-1),x_{d,\max}\ \ if\ x_{id}^k(t)>x_{d,\max}\\ u\ x_{d,\min},p_{id}^k(t-1)\ \ if\ x_{id}^k(t)<x_{d,\min}\end{cases} \tag{9}$$

where $u(a,b)$ generates a random number between $a$ and $b$.

**Operator C, multivariate Gaussian operator with partial covariance matrix:**

Operator B resamples all of the coordinates of a personal best position to generate a new position; therefore, some promising information may be lost. In order to relax this issue, Operator C is defined to affect only some coordinates. Let $I=I_i{}^k(t)\subseteq\{1,2,3,\ldots,D\}$ represent these coordinates, and let $\sum_{il}^k$ be the projection of $\sum_i^k$ into these coordinates; then, operator C defines a new position as:

If $d\in I_i^k(t)$:

$$x_{il}^k(t)=\begin{cases}N\ p_{il}^k(t-1)+p_{gl}^k\ /2,\sum_{il}^k\ \ if\ (i,k)\neq(g^k(t-1),k)\\ N\ p_{il}^k(t-1)+p_{gl}^{\theta(k,r)}(t-1)\ /2,\sum_{il}^k\ otherwise\end{cases} \tag{10}$$

If $d\notin I_i^k(t)$:

$$x_{id}^k(t)=p_{id}^k(t-1)$$

Operator C handles the out of range search in the same way as Operator B.

**Operator D, multivariate Gaussian operator with partial sample covariance matrix:**

Estimation of distribution algorithms use explicit distribution models like multivariate Gaussian distribution for sampling new points. Then, employing the promising newly generated points, they refine their probabilistic models. Inspiring from this idea, operator D uses the sample covariance matrix and the sample mean in the Multivariate Gaussian distribution (Let $I=I_i{}^k(t)$):

If $d \in I_i^k(t)$:

$$x_{il}^k(t) = N\left(\mu_{il}^k, \hat{\Sigma}_{il}^k\right) \tag{11}$$

If $d \notin I_i^k(t)$:

$$x_{id}^k(t) = p_{id}^k(t-1)$$

where the sample mean and the sample covariance matrices are defined as follows:

$$\mu_{il}^k(t) = \begin{cases} \dfrac{1}{M}\sum_{j=1}^{M} p_{jl}^k(t-1)\ if\ (i,k) \neq (g^k(t-1),k) \\ \dfrac{1}{3}\sum_{j=0}^{2} p_{gl}^{\theta(k,j)}(t-1) otherwise \end{cases} \tag{12}$$

$$\hat{\Sigma}_{il}^k = \begin{cases} \dfrac{1}{M}\sum_{j=1}^{M} \left(p_{jl}^k(t-1) - \mu_{il}^k(t)\right)\left(p_{jl}^k(t-1) - \mu_{il}^k(t)\right)^T if\ (i,k) \neq (g^k(t-1),k) \\ \dfrac{1}{3}\sum_{j=0}^{2} \left(p_{gl}^{\theta(k,j)}(t-1) - \mu_{il}^k(t)\right)\left(p_{gl}^{\theta(k,j)}(t-1) - \mu_{il}^k(t)\right)^T otherwise \end{cases} \tag{13}$$

### 3.3. Covariance matrix adaption

Each particle has a covariance matrix incorporated with it, which is evolved alongside the particle itself. Operators B and C use this covariance matrix in order to direct the particle in the landscape. Hansen and his colleagues proposed some covariance matrix adaptation mechanisms for evolution strategies [45–47]. They introduced rank-1 and rank-μ update rules to ensure a reliable estimator. At each generation, the information from the previous generations is used to refine the covariance matrix. Rank 1 uses the concept of evolutionary path which is the whole path taken by the population over a number of generations, and is defined as follows:

$$p_c^{(g+1)} = (1-c_c)p_c^{(g)} + \sqrt{c_c(2-c_c)\mu_{eff}}\frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}} \tag{14}$$

where $c_c$ is the backward time horizon, and $p_c^{(0)} = 0$; $m^{(g)}$ is the weighted mean of the best selected samples at generation $g$. rank-μ update rule uses an estimation for the distribution of the selected steps along with the previous information about the covariance matrix:

$$C^{(g+1)} = (1-c_{cov})C^{(g)} + c_{cov}\sum w_i {y_i}^{(g+1)} {y_i}^{(g+1)^T} \tag{15}$$

where ${y_i}^{(g+1)} = \left(x_i^{(g+1)} - m^{(g)}\right)/\sigma^{(g)}$, and $w_i$ is the weight of the selected sample; $\sigma$ represents the step size, which is also updated at each generation. Using these rules together, the overall update rule can be like the following:

$$C^{(g+1)} = (1-c_{cov})C^{(g)} + c_{cov}\left(1 - \frac{1}{\mu_{\text{cov}}}\right)\sum w_i {y_i}^{(g+1)} {y_i}^{(g+1)^T} + \frac{c_{cov}}{\mu_{\text{cov}}} {p_c}^{(g+1)} {p_c}^{(g+1)^T} \tag{16}$$

BBPSO with scale matrix adaptation (SMA-BBPSO) uses multivariate $t$-distribution to update the position of particles. It employs a simple method for adapting the scale matrix of the $t$-distribution. At each iteration, the scale matrix related to the $kth$ particle is updated using the following equation:

$$\sum\nolimits_k = (1-\beta)\sum\nolimits_k + \beta n_k n_k^T \tag{17}$$

$\beta$ is the learning rate controlling the portion of the scale matrix to be inherited from the past experience, and $n_k$ is the best position in the neighborhood of the $kth$ particle. The idea is simple, and is based on sampling the best found positions into the scale matrix.

Employing the previous concepts, CLA-BBPSO uses two mechanisms to adapt the covariance matrix of each particle. First, it utilizes the evolution direction of the particle. This concept resembles the concept of evolution path. The evolution direction ($\kappa$) of particle $X_i^k$ will be defined as follows:

$$\kappa_i^k(t) = \left(p_i^k(t) - p_i^k(t')\right)\left(p_i^k(t) - p_i^k(t')\right)^T \tag{18}$$

where $t$ and $t'$, respectively, represent the current iteration number and the iteration number during the previous covariance matrix adaption phase. Accordingly, $p_i^k(t)$ is the current personal best location of the particle $X_i^k$, and $p_i^k(t')$ is its personal best position at the time of previous covariance matrix adaption phase. Evolution direction samples the last improvement of a particle. Like Eq. (17), we also sample the best nearby position into the covariance matrices. However, considering the sample covariance matrix definition, which is given in Eq. (13), one can interpret the sample covariance as the average dependencies of the data samples with respect to their mean. In this interpretation if we only consider the dependency of the best point, we can obtain:

$$\begin{gathered} n_i^k(t) - m\ \ n_i^k(t) - m^{\ T} \\ where: n_i^k(t) = \arg\min f(p)\ , p \in \left\{p_j^l(t) | l \in \theta(k,s), s = 0,1,2\right\} - p_i^k(t) \end{gathered} \tag{19}$$

Here $n_i^k(t)$ is the best position in the neighborhood of the particle $X_i^k$. $m$ is the mean of the distribution, and as it is unknown, it should somehow be estimated. We, simply, will use $p_i^k(t)$ as the mean of the distribution. In this way, the best nearby dependency with respect to $p_i^k(t)$ will be sampled into the covariance matrix. Using this interpretation, if $X_i^k$ happens to be the best particle in its own neighborhood, then the above equation evaluates to zero. Therefore, we defined $n_i^k(t)$ as the best nearby position other than $p_i^k(t)$ itself. Equation 19 is very similar to Equation (17); though, in 17, $m$ is missing which is similar to assuming a zero value mean. As we shall see in the experimental section, assuming $m = 0$ may be troublesome on shifted test functions that have their optimum in a location other than the origin. Based on the aforementioned ideas, the overall equation for covariance matrix adaption in CLA-BBPSO is:

$$\sum\nolimits_i^k(t) = (1-\beta)\sum\nolimits_i^k(t') + \beta S_i^k \tag{20}$$

where:

$$S_i^k = \frac{1}{2}\left(p_i^k(t) - p_i^k(t')\right)\left(p_i^k(t) - p_i^k(t')\right)^T + \frac{1}{2}\left(n_i^k(t) - p_i^k(t)\right)\left(n_i^k(t) - p_i^k(t)\right)^T \tag{21}$$

### 3.4. Strategy adaption

After action selection phase, each particle is evolved for a period of several iterations. Afterwards, the strategy adaption phase is applied to adjust the action probabilities in the favor of the most promising ones. The favorability of a selected action for a learning automaton is measured based on the comparison of its outcome with the outcomes of the selected actions in the adjacent cells. The outcome of an action will be measured according to the amount of improvement in the fitness value of the associated particle, which is defined as follows:

$$\psi_i^k = \frac{1}{f(p_i^k(t))}\left(f(p_i^k(t')) - f(p_i^k(t))\right) \tag{22}$$

where $t$ is the current iteration number, and $t'$ is the iteration number in the previous strategy adaption phase. Accordingly, $f(p_i^k(t))$ and $f(p_i^k(t'))$ represent the fitness of the particle $X_i^k$, respectively, in the current and previous adaption phases. Based on the improvement values of the particles in its adjacent cells, the reinforce signal for a learning automaton like $LA_i^k$ is obtained as follows:

$$\beta_i^k = \begin{cases} 0 \; if \; \psi_i^k > median \left\{ \psi_j^l | l = \theta(k,h), h = 1, 2 \right\} \\ 1 otherwise \end{cases} \tag{23}$$

Here *median* returns the statistical median of its argument set. Based on the received reinforcement signal, each learning automaton updates its internal action probabilities using equations 3 or 4. This definition for reinforcement signals has some advantages. On average, the promising actions of about half of the learning automata are rewarded, hence, these actions can be selected with higher probabilities next times. The selected actions of the other half are panelized; therefore, these learning automata will explore other actions (other operators) with higher probabilities than before. This way, we can ensure that each operator always would have a chance of selection.

After strategy adaption phase, the algorithm continues a new cycle starting with the action selection phase. A general pseudocode for the proposed model is given in Fig. 3.

## 4. Experimental study

This section demonstrates the effectiveness of the proposed method through experimental study and comparison. The behavior of the method is studied on a set of benchmark functions that are defined in Table 2. These functions possess various characteristics; as a result, they can exhibit different properties of the stochastic search methods. The set contains 2 unimodal functions $f_8$ and $f_9$, while the others are multimodal. Although, the two dimensional Rosenbrock's function ($f_{11}$) is unimodal, its higher dimensional cases are considered to be multimodal. In higher dimensional cases, it possesses two optima with the global one residing inside a long, narrow, and parabolic shaped valley with a flat bottom. The functions can also be categorized according to the separability of their coordinates. Separable functions, including functions $f_1$–$f_9$, can be solved using independent searches in each of their coordinates. As a result, some mutation or crossover based simple search methods, which are somehow simulating coordinate independent search, can mislead evaluations. In order to alleviate this problem, Table 2, also, contains a set of inseparable functions, problems $f_{10}$–$f_{16}$. Some of these inseparable problems are the rotated versions of their original separable cases. By rotating a function one can make its coordinates inseparable. To rotate a function like $f(x)$, the original vector $x$ is left multiplied with an orthogonal matrix $M$: $y = M \times x$, and the new generated vector $y$ is used to calculate the fitness of its associated individual. We use the Salomon's method to generate the orthogonal matrices [51].

Many stochastic search approaches start with random initial populations, and using some stochastic operators, explore the search space. The result of some operators tends to have a bias toward the mean point of the population. In the case of random initialization and symmetric problem landscapes, this mean point approaches the center of the search space. Looking at Table 2, it can be seen that except $f_1$ all of the test problems have their global optimum at the center of the search space. These issues may result in misleading performance evaluations as in the case of the two recent published works [25,52]. Hence, we employed the shifted version of the given functions, where $f(x - \alpha)$ is used instead of $f(x)$ to determine the fitness. We set $\alpha$ to $[0.2U_1, 0.2U_2, \ldots, 0.2U_D]$ where $U_i$ is the upper boundary of the *ith* coordinate, and applied this shift to all of the functions except $f_1$, which already has its optimum point near the boundaries.

### 4.1. Compared PSO algorithms

CLA-BBPSO is compared with a group of PSO algorithms whose configurations are summarized in Table 3. The results of all algorithms are taken from 30 independent runs. Each run of an algorithm on a benchmark function is terminated after $10^5$ fitness function evaluations on 30-*D* functions, and $3 \times 10^5$ fitness function evaluations on 50-*D* functions.

SMA-BBPSO is a variant of Bare Bones PSO which uses a multivariate t-distribution for sampling the new positions of particles. It, also, adapts the scale matrix of each particle by sampling the best position in its neighborhood with the maximum likelihood. $PS^2OS_R$ resembles CLA-BBPSO in that both are utilizing multiple swarms of particles. In $PS^2OS_R$, a particle is attracted to three informers: historical best positions of itself and its own swarm along with the best historical position in its neighboring swarms. Comprehensive learning particle swarm optimizer (CLPSO) is mainly developed for dealing with complex multimodal functions. Each coordinate of a particle learns comprehensively from the corresponding coordinate of another particle or the corresponding coordinate of the historical best position of itself until it fails to improve for a period of time. CLPSO updating procedure results in a diversified population, which ensures its high exploration capability; however, the diversified population results in a low convergence rate. In fully informed PSO (FIPS), all the neighbors contribute to the velocity adjustment. The sum of acceleration coefficients is considered to be a constant value, which is equally divided between the acceleration coefficients. Frankenstein PSO (FPSO) starts with a fully connected topology, and gradually decreases the connectivity of the particles. Adaptive PSO (APSO) uses the population distribution and fitness information to adjust the inertia weight and acceleration coefficients. Starling PSO is based on the collective response of starlings. When the global best position of the swarm doesn't improve for a period, the velocity and position of particles are adjusted based on their neighboring particles. The way to handle the out of rang search is not discussed clearly in the original paper for Starling PSO. The method of handling search boundaries may have a critical impact on the performance of the PSO approaches. For a fair comparison, we implemented the boundary handling methods of CLA-BBPSO for Starling PSO. In summary, Starling PSO is implemented with max velocity of 0.2 of the search ranges, and Eq. (9) is used for handling the search boundaries. The same equation, also, will be used for SMA-BBPSO.

### 4.2. A note on the max partition size

One advantage of multi swarm approaches is their locality, which means that each swarm operates utilizing only local information. However, if we have to partition the coordinates of every particle in a same way, as discussed previously, this locality will be violated. This partitioning can be done locally if we set *MAXP* to 1. Furthermore when *MAXP* = 1, the search process can get better information on how much two coordinates of a particle belonging to a same group are correlating with each other. Therefore, from the next sections the value of 1 will be used for *MAXP*. Though, this section investigates the effect of different values of *MAXP* on the performance of the proposed method. Fig. 4a to 4e show the effect of different values of *MAXP* on a group of benchmarks selected from Table 2. As the results suggest, the algorithm performed better with *MAXP* = 1 on most of the tested problems. The definition of rotated hyper-ellipsoid function ($f_{10}$) suggests that to refine a candidate solution all of its coordinates somehow should change together as they are highly correlated. For this function, changing a value in one coordinate of a fine candidate solution generally worsens the candidate's fitness. Also, as we will see in Section 4.6, the learning mechanism tends to select mostly operator B for $f_{10}$.

```
I.   Initialization:
     1. Initialize a cellular learning with $N$ cells; each cell is composed of a swarm of $M$ randomly generated
        particles and a group of $M$ learning automata with four actions of equal probability.
II.  Set time to zero: $t = 0$.
III. Set: $t' = 0$.
IV.  While termination condition is not satisfied, repeat:
     1. Perform action selection: each LA selects an action based on its internal probability vector. Let $Ac_i^k$
        denote the selected action of $LA_i^k$.
     2. Partition $D$ into random summands according to Eq. 5, and let $\chi$ denote the number of these
        summands.
     3. Using the generated summands, partition the coordinates of each particle according to Eq. 6. Let
        $I_i^k(t+j), j=1..\chi$ represent the generated partitions for particle $X_i^k$.
     4. For $t = t+1$ to $t+\chi$ do:
            a. Compute the new position of each particle $X_i^k$ using $I_i^k(t)$ and the updating rule related to
               $Ac_i^k$.
            b. Evaluate the new positions, and update the personal best position of each particle.
     5. Update the associated covariance matrix of each particle using Eq. 20.
     6. Generate reinforcement signals for each learning automata using Eq. 23.
     7. Based on the generated reinforcement signals, updated each learning automata using Eq. 3 or Eq. 4.
     8. Set $t' = t$.
```

**Fig. 3.** A general pseudocode for CLA-BBPSO.

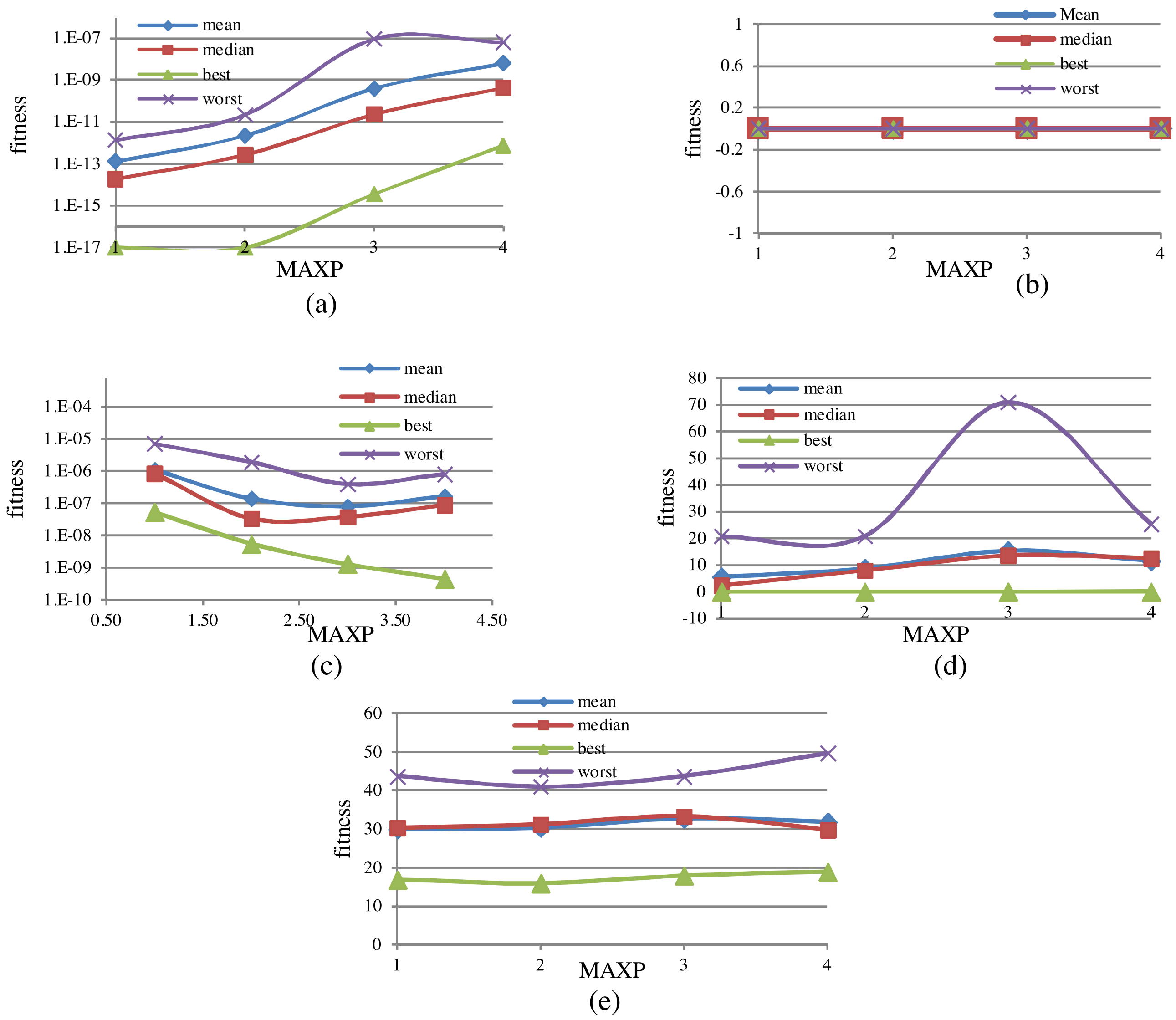


**Fig. 4.** The statistical results on $f_3$(a), $f_5$(b), $f_{10}$(c), $f_{11}$(d), and $f_{12}$(e) for different values of MAXP. The results are obtained from 30 independent runs.

**Table 2**
Benchmark functions.

| function | Search range | Opt value | Required precision |
|---|---|---|---|
| $f_1 = 418.9829 \times D + \sum_{i=1}^{N} \left(-x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right)$ | $[-500,500]^D$ | 0 | 0.001 |
| $f_2 = \sum_{i=1}^{N} \left(-x_i^2 - 10\cos(2\pi x_i) + 10\right)$ | $[-5.12,5.12]^D$ | 0 | 0.001 |
| $f_3(X) = f_2(Z),\ z_i = \begin{cases} x_i\ if\ \lvert x_i \rvert < \frac{1}{2} \\ \frac{round(2x_i)}{2}\ if\ \lvert x_i \rvert \ge \frac{1}{2} \end{cases}$ | $[-5.12,5.12]^D$ | 0 | 0.001 |
| $f_4 = -20\exp\left[-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right] - \exp\left[\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right] + 20 + \exp(1)$ | $[-32,32]^D$ | 0 | 0.001 |
| $f_5 = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ | $[-600,600]^D$ | 0 | 0.00001 |
| $f_6 = \frac{\pi}{N}\left\{ 10\sin^2(\pi y_1) + (y_N - 1)^2 + \sum_{i=1}^{N-1}(y_i - 1)^2\left[1 + 10\sin^2(\pi y_{i+1})\right]\right\} + \sum_{i=1}^{N} u(x_i, 10, 100, 4),\ y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - 1)^m x_i > a \\ 0 - a \le x_i \le a \\ k(-x_i - a)^m x_i < -a \end{cases}$ | $[-50,50]^D$ | 0 | 0.001 |
| $f_7 = 0.1\left\{\sin^2(3\pi x_1) + (x_N - 1)^2\left[1 + \sin^2(2\pi x_N)\right] + \sum_{i=1}^{N-1}(x_i - 1)^2\left[1 + \sin^2(3\pi x_{i+1})\right]\right\} + \sum_{i=1}^{N} u(x_i, 10, 100, 4)$ | $[-50,50]^D$ | 0 | 0.001 |
| $f_8 = \sum_{i=1}^{N} x_i^2$ | $[-100,100]^D$ | 0 | 0.001 |
| $f_9 = \sum_{i=1}^{N} \lvert x_i \rvert + \prod_{i=1}^{N} \lvert x_i \rvert$ | $[-10,10]^D$ | 0 | 0.001 |
| $f_{10} = \sum_{i=1}^{N}\left[\sum_{j=1}^{i} x_j\right]^2$ | $[-100,100]^D$ | 0 | 0.001 |
| $f_{11} = \sum_{j=1}^{N-1}\left[100(x_{j+1} - x_j^2)^2 + (x_j - 1)^2\right]$ | $[-100,100]^D$ | 0 | 30 |
| $f_{12}: f_2(y), y = M^* x$ | $[-5.12,5.12]^D$ | 0 | 50 |
| $f_{13}: f_5(y), y = M^* x$ | $[-600,600]^D$ | 0 | 0.00001 |
| $f_{14}: f_4(y), y = M^* x$ | $[-32,32]^D$ | 0 | 0.001 |
| $f_{15}: f_3(y), y = M^* x$ | $[-5.12,5.12]^D$ | 0 | 50 |
| $f_{16}: f_1(y), y = M^* x$ | $[-500,500]^D$ | 0 | 3000 |

**Table 3**
Parameter settings for the compared PSO algorithms.

| Method | parameter settings |
|---|---|
| $PS^2OS_R$ [17] | $c = 1.3667$, $\chi = 0.729$ |
| CLPSO [38] | $c = 1.49445$, $w \in [0.4, 0.9]$ |
| FIPS [11] | $\varphi = 4.1$, $\chi = 0.7298$ |
| FPSO [14] | $\varphi = 4.0$, $w \in [0.4, 0.9]$ |
| APSO [6] | $1.5 < c_1, c_2 < 2.5$ and $c_1 + c_2 < 4$, $w \in [0.4, 0.9]$ |
| SMA-BBPSO[25] | *Max M* = 5, *neighborhood size* = 3, *Swarm size* = 30 |
| Starling PSO[52] | *Swarm Size* = 50; *Stagnant limit* = 2; *Max_Num* = 14; $c_1 = c_2 = 2$; $w_{max} = 0.4$; $w_{min} = 0.2$; |

Operator B changes the coordinates of a particle all together which approves the latter idea. As operator B somehow plays the rule of a large *MAXP*, the tendency of functions like $f_{10}$ for larger partition sizes becomes clearer if we disable this operator. Finally, as operator B plays the rule of a large *MAXP*, *MAXP* = 1 shows to be a less restriction.

### 4.3. Parameter settings for CLAB-BPSO

The proposed CLA-based BBPSO involves some parameters that should be adjusted. The population size is set to 40 as suggested in the PSO literature. This population can be divided into 20, 13, 10, 8, ... swarms according to the number of particles in each swarm. However, using few swarms with large swarm sizes may spoil the multi swarm features of the algorithm. Therefore, we keep the number of swarms large enough. We have tested CLA-BBPSO with the number of swarms equal to 20, 13 (the population size is 39 in this case) and 10, and used 10 in the experiments of this paper. The other topologies yield very similar results.

The other parameters are the reward and punishment factors of the learning automata. For these two parameters a vast range of values can be used, and their effect is deeply investigated in the related literature. As a result, based on our previous experience, we used $a = 0.2$ and $b = 0.02$. The only remaining parameter is $\beta$ which controls the portion of the covariance matrix to be inherited from the past experiences (equation 20). The effect of this parameter on a set of benchmarks is studied in this section. Fig. 5a to 5e show the results of the algorithm on a set of benchmark problems when $\beta$ changes in the range [0.1 0.9].

For two problems $f_1$ and $f_2$, different values for $\beta$ have a little impact on the achieved final results. On $f_{10}$, the algorithm performed better when $\beta$ is near 0.4. On $f_{11}$, $\beta < 0.5$ yields, somehow, better final results. The algorithm has, somehow, better behavior on $f_{12}$ when $\beta \in [0.3, 0.8]$. From these observations $\beta = 0.4$ seems to be an appropriate choice.

### 4.4. Experimental results on 30-D test problems

The first test of this section examines the speed of the compared methods in finding a global optimum or nearing an acceptable solution. The study is performed by obtaining the average number of fitness function evolutions required to reach the specified precisions given in Table 2. A run of an algorithm on a benchmark is allowed to continue until the maximum number of fitness evaluations. If a run of an algorithm can't reach the required precision, then the run will be considered as unsuccessful. The average number of successful runs on a benchmark (which will be called the success rate of the method on the benchmark) can represent the robustness of an algorithm. This section compares the methods based on their success rates and the average number of fitness evolutions on their successful runs.

The precisions that are given in Table 2 are defined in a way to ensure the detection of global optima; however, none of the compared methods can reach the global optimum of some benchmarks with a high precision in the allowable number of fitness evolutions. Therefore, some larger precisions are considered for these functions to make the comparisons sensible.

The comparison results are given in Table 4. CLA-BBPSO is superior to the other peer algorithms according to the success rate criteria; it achieved the specified threshold on almost all of the runs (except for one run on $f_{16}$) on every tested benchmark function. All of the other peer PSO methods have failed to reach the specified precisions on many runs. All of the runs of APSO on $f_1$, $f_2$, and $f_{10}$ were unsuccessful. The same is true for CLPSO on $f_3$, $f_{10}$, $f_{13}$, and $f_{14}$. FIPS has zero success rates on four benchmark functions. FPSO has zero success rates on 10 benchmarks; while, the success rate of $PS^2OS_R$ is zero on 4 benchmarks. From another point of view, APSO has a success rate lower than one on 13 benchmarks; CLPSO has lower success rate on 10 benchmarks; FIPS has lower success rate on 9 benchmarks; the success rate of FPSO is lower than 1 on 13 benchmarks; and finally, $PS^2OS_R$ is unsuccessful on some runs of 10 tested benchmark functions. From these observations, the robustness and superiority of the purposed method in comparison to the other peer methods can easily be concluded.

Although CLAB-BPSO isn't the fastest compared method in reaching the specified precisions, its speed in nearing the specified thresholds is very competitive to the fastest ones. Some of the compared methods possess higher convergence speeds, which resulted in their insufficient explorative ability and premature convergence.

The next experiment is conducted to compare the accuracy of the compared algorithms. Each method is tested on each of the benchmark functions for 30 runs, and the statistical information of the achieved final results on the benchmark functions is used for comparison. Table 5 gives these statistical information in terms of average, median, best, and worst on each benchmark, the distribution of the final results are illustrated in Fig. 6. Also, the Wilcoxon rank sum test for a confidence level of 95% [50] has been utilized to compare the methods from statistical point (Table 6). Wilcoxon test is a nonparametric alternative to the paired *t*-test [50], and is based on the null hypothesis that the two compared algorithms are statistically equivalent. For each test '+' is used if the proposed approach is better than a compared algorithm, '−' for the opposite case, and '=' if the two algorithms are statistically indistinguishable.

The first note on the results of Table 5 is about Starling PSO. The obtained results of Starling PSO are much weaker than the ones which could be obtained on the un-shifted versions of the test functions. We believe that using equations 3 and 4 (in [52]) leads to this difference, and the orientation change introduced by these two equations results in a bias toward the center of the search space. Also, the results of SMA-BBPSO in Table 5 are much weaker than the ones reported in [25]. We believe that this difference is caused because of two reasons:

1. Eq. (14) in [25], which is used to update the covariance matrix, resembles equation 19 when the mean is considered to be zero. It samples the dependency of the best point into the covariance matrix with respect to $m = 0$. However, symmetric test functions have their optimum point at the center of the search space; therefore, Eq. (14) (in [25]) learns the dependency of the best sample with respect to the problem solution (which should be considered unknown).
2. The second reason may be due to the fact that we used equation 9 for boundary handling in SMA-BBPSO. However, as it isn't mentioned how boundaries are handled in SMA-BBPSO, we utilized the boundary handling mechanisms of CLA-BBPSO into SMA-BBPSO.

Observing Table 5, it is clear that the proposed method has the highest accuracy among all of the compared methods on most of the tested problems. CLA-BBPSO has the best average on 12 benchmarks; while, its average results on the other four benchmarks are very near to the best ones. The obtained results of the purposed method are very distinctive from all of the other peer methods on the benchmarks: $f_1$, $f_2$, $f_{10}$, and $f_{11}$. Also, only 3 algorithms were capable of finding the global optimum of $f_3$, with CLA-BBPSO having the highest average. Although CLA-BBPSO doesn't possess the best average results on $f_4$ and $f_{14}$, its obtained results are very competitive to the best obtained ones. Also, CLA-BBPSO has the second best results on $f_{12}$ and $f_{16}$, and its achieved results are very near to those of FPSO (the best method on these benchmarks).

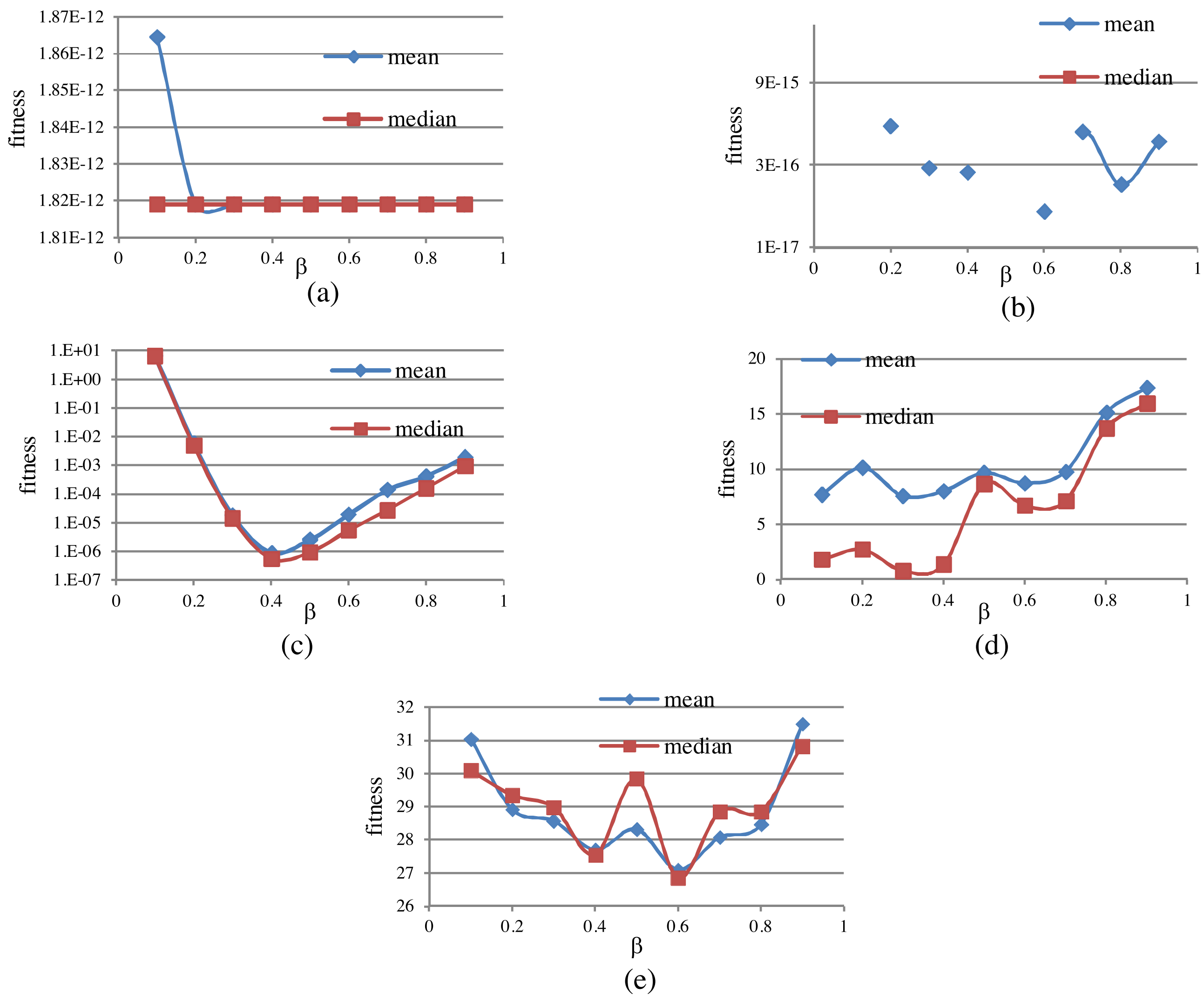
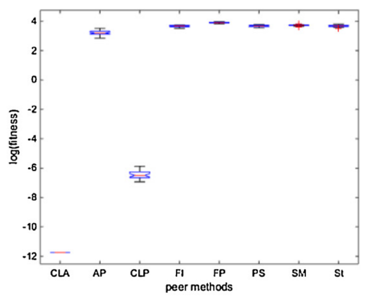


**Fig. 5.** The statistical results on $f_1$(a), $f_2$(b), $f_{10}$(c), $f_{11}$(d), $f_{12}$(e) when β changes in the range [0.1, 0.9].

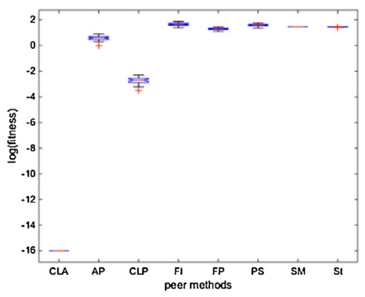
**Table 4**
Average number of evaluations needed to reach the specified accuracy for successful runs along with the average success rates inside parenthesis.

| Test Funs | Methods | | | | | |
|---|---|---|---|---|---|---|
| | CLABBPSO | APSO | CLPSO | FIPS | FPSO | $PS^2OS_R$ |
| $f_1$ | 48926(1) | –(0) | 74389(1) | –(0) | –(0) | –(0) |
| $f_2$ | 57770(1) | –(0) | 98214(0.16) | –(0) | –(0) | –(0) |
| $f_3$ | 68069(1) | 63241(1) | –(0) | –(0) | –(0) | –(0) |
| $f_4$ | 26108(1) | 23828(0.96) | 84462(1) | 18028(1) | –(0) | 14227(1) |
| $f_5$ | 30265(1) | 16627(0.43) | 95942(.96) | 32261(0.83) | –(0) | 16783(0.8) |
| $f_6$ | 16332(1) | 34744(0.93) | 56444(1) | 22260(1) | 41626 (1) | 10127(0.86) |
| $f_7$ | 19273(1) | 13723(0.8) | 66033(1) | 17555 (1) | 85924(0.26) | 10977(1) |
| $f_8$ | 20873(1) | 12242(1) | 67365(1) | 13670(1) | 95321(0.87) | 11155(1) |
| $f_9$ | 24648(1) | 13801(1) | 73310(1) | 17690(1) | 96529(0.93) | 13932(1) |
| $f_{10}$ | 70908(1) | –(0) | –(0) | –(0) | –(0) | 87847(1) |
| $f_{11}$ | 44742(1) | 26579(0.8) | 96962(0.16) | 19136(0.93) | –(0) | 16544(1) |
| $f_{12}$ | 35430(1) | 26843(0.6) | 83431(0.43) | 79984(0.03) | 26496(1) | 7528(0.8) |
| $f_{13}$ | 41329(1) | 18031(0.36) | –(0) | 18918(1) | –(0) | 18021(0.8) |
| $f_{14}$ | 31261(1) | 16640(0.13) | –(0) | 19479(1) | –(0) | 14509(0.93) |
| $f_{15}$ | 26548(1) | 34603(0.63) | 76330(0.8) | 99843(0.03) | 31957(1) | 85978(0.63) |
| $f_{16}$ | 43553(0.96) | 42688(0.23) | 61658(0.86) | –(0) | –(0) | –(0) |

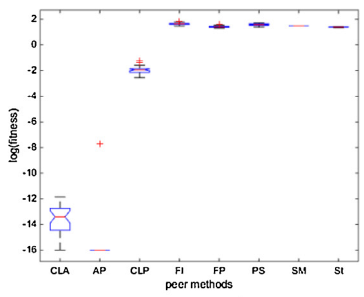
Considering the results of Wilcoxon test we can conclude the statistical superiority of the proposed algorithm to both APSO and CLPSO. CLA-BBPSO is statistically superior to CLPSO on all of the tested problems. Comparing CLA-BBPSO to APSO, CLA-BBPSO is statistically inferior to APSO on $f_3$, is statistically indistinguishable on $f_4$, and is statistically superior on the rest of tested problems. In Comparison to $PS^2OS_R$, CLA-BBPSO is better on 9 benchmark problems, is statistically indistinguishable on 4 benchmarks, and is statistically worst only in 2 cases.



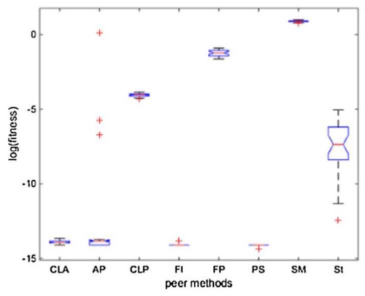
(a) $f_1$



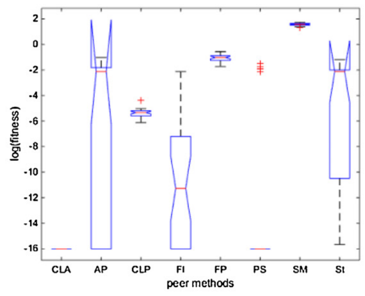
(b) $f_2$



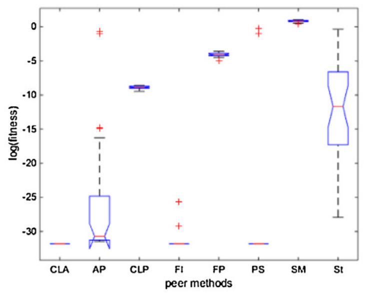
(c) $f_3$



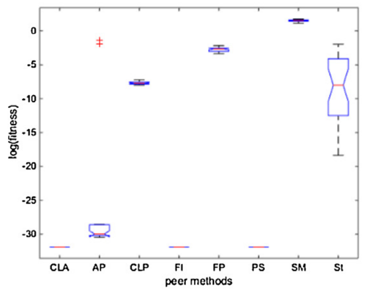
(d) $f_4$



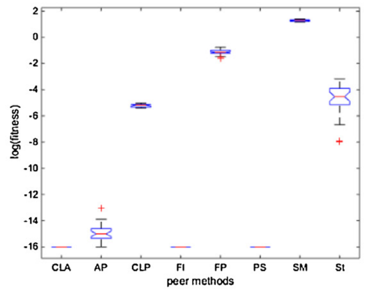
(e) $f_5$



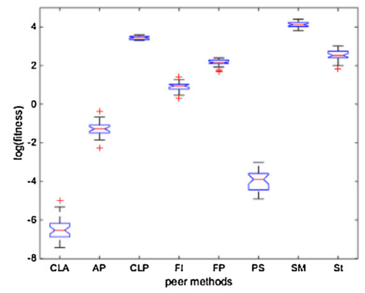
(f) $f_6$



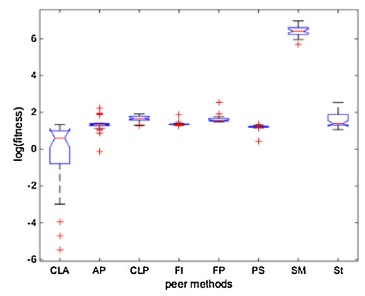
(g) $f_7$



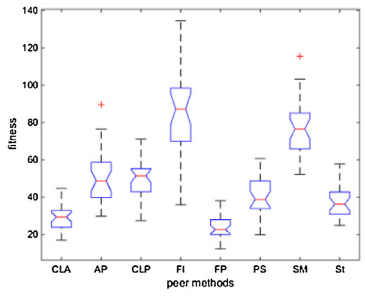
(h) $f_9$



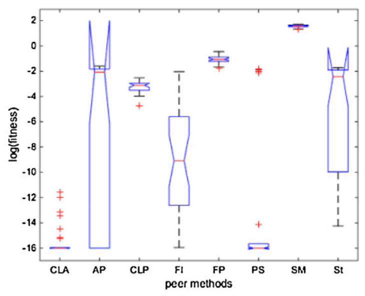
(i) $f_{10}$



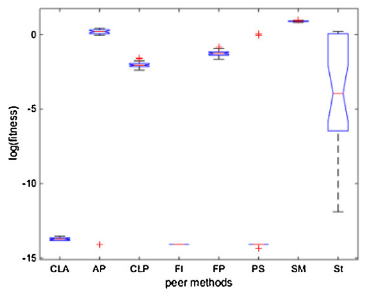
(j) $f_{11}$



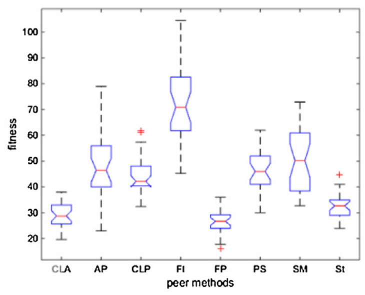
(k) $f_{12}$



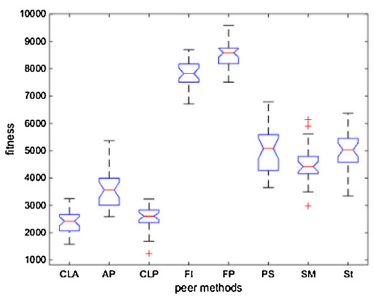
(l) $f_{13}$



(m) $f_{14}$



(n) $f_{15}$



(o) $f_{16}$

**Fig. 6.** Box plot of the compared methods on the benchmark set. CLA, AP, CLP, FI, FP, PS, SM, and St are used to refer to CLA-BBPSO, APSO, CLPSO, FIPS, FPSO, PS$^2$OS$_R$, SMA-BBPSO, and Starling PSO respectively.

**Table 5**
Statistical information (in this order: average, median, best, and worst) of obtained results for the compared algorithms.

| | CLABBPSO | APSO | CLPSO | FIPS | FPSO | $PS^2OS_R$ | SMA-BBPSO | Starling-PSO | |
|---|---|---|---|---|---|---|---|---|---|
| $f_1$ | average | **1.81e-12** | 1722.92 | 4.27e-7 | 4718.4 | 7998.8 | 4809.3 | 5212.88 | 4836.68 |
| | median | **1.81e-12** | 1598.96 | 3.24e-7 | 4730.5 | 7915.1 | 4709.7 | 5238.36 | 4826.47 |
| | best | **1.81e-12** | 710.63 | 1.18e-7 | 3339.8 | 6619.8 | 3594.4 | 4088.79 | 3138.66 |
| | worst | **1.81e-12** | 3396.83 | 1.34e-6 | 5531.8 | 9226.1 | 6021.3 | 6378.59 | 6317.15 |
| $f_2$ | average | **0** | 4.04 | 2.09e-3 | 46.63 | 20.87 | 39.63 | 29.94 | 28.92 |
| | median | **0** | 3.97 | 1.97e-3 | 43.48 | 20.68 | 38.30 | 29.93 | 29.84 |
| | best | **0** | 0.99 | 2.92e-4 | 24.27 | 12.63 | 22.88 | 29.89 | 24.87 |
| | worst | **0** | 7.95 | 4.97e-3 | 74.16 | 28.09 | 58.70 | 30.00 | 29.84 |
| $f_3$ | average | **1.18e-13** | 6.30e-10 | 1.32e-2 | 42.903 | 25.58 | 37.4 | 30 | 23.81 |
| | median | 3.64e-14 | **0** | 1.10e-2 | 42.95 | 25.00 | 37.5 | 30 | 24.00 |
| | best | **0** | **0** | 2.70e-3 | 29.1 | 20.15 | 24 | 30 | 21.00 |
| | worst | **1.34e-12** | 1.89e-8 | 5.46e-2 | 61.108 | 41.00 | 52 | 30 | 26.11 |
| $f_4$ | average | 1.32e-14 | 4.43e-2 | 9.20e-5 | **8.23e-15** | 6.38e-2 | **7.40e-15** | 7.87 | 1.28e-6 |
| | median | 1.15e-14 | 1.50e-14 | 9.48e-5 | **7.99e-15** | 6.06e-2 | **7.99e-15** | 7.76 | 4.41e-8 |
| | best | **7.99e-15** | **7.99e-15** | 4.81e-5 | **7.99e-15** | 2.41e-2 | **4.44e-15** | 5.67 | 3.49e-13 |
| | worst | 2.22e-14 | 1.34 | 1.39e-4 | 1.50e-14 | 1.24e-1 | **7.99e-15** | 10.11 | 9.06e-6 |
| $f_5$ | average | **0** | 1.33e-2 | 5.64e-6 | 5.02e-4 | 9.64e-1 | 2.70e-3 | 37.24 | 9.55e-3 |
| | median | **0** | 7.31e-3 | 4.45e-6 | 5.53e-12 | 9.36e-2 | **0** | 37.25 | 7.39e-3 |
| | best | **0** | **0** | 7.27e-7 | **0** | 1.87e-2 | **0** | 20.48 | **0** |
| | worst | **0** | 9.52e-2 | 3.95e-5 | 7.4128e-3 | 2.69e-1 | 2.94e-2 | 52.99 | 6.34e-2 |
| $f_6$ | average | **1.57e-32** | 1.03e-2 | 1.44e-9 | 7.49e-28 | 9.81e-5 | 3.10e-2 | 6.72 | 6.56e-2 |
| | median | **1.57e-32** | 2.01e-31 | 1.34e-9 | **1.57e-32** | 8.04e-5 | **1.57e-32** | 6.15 | 2.30e-12 |
| | best | **1.57e-32** | **3.63e-32** | 3.61e-10 | **1.57e-32** | 9.70e-6 | **1.57e-32** | 2.35 | 1.22e-28 |
| | worst | **1.57e-32** | 2.07e-1 | 2.55−9 | 2.24e-26 | 2.56e-4 | 6.21e-1 | 10.94 | 4.14e-1 |
| $f_7$ | average | **1.34e-32** | 3.29e-3 | 2.34e-8 | **1.34e-32** | 2.36e-3 | **1.34e-32** | 34.18 | 8.25e-4 |
| | median | **1.34e-32** | 1.11e-30 | 1.91e-8 | **1.34e-32** | 2.23e-3 | **1.34e-32** | 34.21 | 1.85e-8 |
| | best | **1.34e-32** | 3.29e-31 | 1.00e-8 | **1.34e-32** | 4.31e-4 | **1.34e-32** | 13.72 | 4.44e-19 |
| | worst | **1.34e-32** | 4.39e-2 | 5.42e-8 | **1.34e-32** | 6.81e-3 | **1.34e-32** | 54.96 | 1.09e-2 |
| $f_8$ | average | **0** | 6.22e-29 | 3.80e-8 | **0** | 3.88e-3 | **0** | 4678.86 | 1.59e-8 |
| | median | **0** | 5.04e-29 | 3.79e-8 | **0** | 3.81e-3 | **0** | 4544.64 | 2.78e-15 |
| | best | **0** | **0** | 1.49e-8 | **0** | 1.03e-2 | **0** | 3035.65 | 1.18e-27 |
| | worst | **0** | 1.64e-28 | 8.46e-8 | **0** | 1.20e-3 | **0** | 7013.92 | 3.81e-7 |
| $f_9$ | average | **0** | 4.57e-15 | 6.39e-6 | **0** | 7.99e-3 | **0** | 19.08 | 1.04e-4 |
| | median | **0** | 9.99e-16 | 6.65e-6 | **0** | 7.33e-3 | **0** | 19.64 | 2.95e-5 |
| | best | **0** | **0** | 3.96e-6 | **0** | 2.50e-3 | **0** | 14.31 | 1.06e-8 |
| | worst | **0** | 8.85e-14 | 9.45e-6 | **0** | 1.77e-3 | **0** | 24.38 | 6.42e-4 |
| $f_{10}$ | average | **9.15e-7** | 9.501e-2 | 2842.7 | 9.99 | 154.35 | 1.67e-3 | 14430.98 | 410.97 |
| | median | **2.89e-7** | 7.1836e-2 | 2868.8 | 9.03 | 149.71 | 1.08e-3 | 13488.13 | 338.98 |
| | best | **3.71e-8** | 1.56e-2 | 2001.5 | 2.10 | 48.27 | 1.09e-4 | 6522.47 | 66.35 |
| | worst | **9.89e-6** | 4.48e-1 | 3917.3 | 26.285 | 249.4 | 5.02e-3 | 25609.62 | 1039.72 |
| $f_{11}$ | average | **6.20** | 34.02 | 47.14 | 25.85 | 51.66 | 17.54 | 3.34e+6 | 52.93 |
| | median | **3.98** | 21.88 | 46.96 | 22.09 | 36.33 | 17.91 | 2.60e+6 | 24.72 |
| | best | **3.37e-6** | 7.21e-1 | 18.04 | 18.93 | 30.32 | 6.71 | 5.00e+5 | 11.45 |
| | worst | **21.89** | 175.01 | 81.61 | 76.58 | 350.59 | 22.73 | 9.34e+6 | 345.11 |
| $f_{12}$ | average | 29.16 | 50.42 | 49.18 | 85.20 | **23.71** | 40.89 | 77.15 | 36.54 |
| | median | 29.35 | 48.75 | 51.42 | 87.23 | **22.65** | 38.83 | 76.58 | 36.31 |
| | best | 16.92 | 29.84 | 27.42 | 35.88 | **12.35** | 19.89 | 52.30 | 24.87 |
| | worst | 44.77 | 89.54 | 71.06 | 134.69 | **38.09** | 60.69 | 115.61 | 57.70 |
| $f_{13}$ | average | **1.28e-13** | 8.94e-3 | 8.03e-4 | 5.82e-4 | 1.08e-1 | 2.13e-3 | 37.51 | 6.06e-3 |
| | median | **0** | 8.62e-3 | 7.54e-4 | 8.63e-10 | 8.85e-2 | **0** | 37.03 | 4.63e-3 |
| | best | **0** | **0** | 1.82e-5 | **1.11e-16** | 1.62e-2 | **0** | 19.82 | 5.44e-15 |
| | worst | **2.64e-12** | 2.46e-2 | 2.89e-3 | 9.12e-3 | 3.59e-1 | 1.47e-2 | 53.44 | 1.95e-2 |
| $f_{14}$ | average | 1.88e-14 | 1.53 | 1.01e-2 | **7.99e-15** | 6.05e-2 | 6.95e-2 | 7.93 | 5.43e-1 |
| | median | 1.86e-14 | 1.57 | 8.96e-3 | **7.99e-15** | 5.55e-2 | **7.99e-15** | 8.01 | 1.15e-4 |
| | best | 1.50e-14 | **7.99e-15** | 4.21e-3 | **7.99e-15** | 2.22e-2 | **4.44e-15** | 6.72 | 1.27e-12 |
| | worst | 2.93e-14 | 2.66 | 2.79e-2 | **7.99e-15** | 1.47e-1 | 1.15 | 9.54 | 1.64 |
| $f_{15}$ | average | 28.64 | 47.68 | 44.38 | 72.77 | **26.21** | 46.51 | 50.72 | 32.53 |
| | median | 28.76 | 46.52 | 42.16 | 70.94 | **26.65** | 46.59 | 50.21 | 32.67 |
| | best | 19.65 | 23.39 | 32.43 | 45.23 | **16.12** | 30.27 | 32.72 | 24.00 |
| | worst | 38.00 | 79.07 | 61.91 | 104.64 | **36.01** | 62.91 | 72.89 | 44.74 |
| $f_{16}$ | average | **2370.52** | 3614.3 | 2583.4 | 7797.6 | 8543.5 | 5035.2 | 4533.16 | 4986.91 |
| | median | **2422.65** | 3561.9 | 2606.4 | 7828.5 | 8585.5 | 5087.5 | 4418.56 | 5034.05 |
| | best | 1578.12 | 2591.2 | **1238.1** | 6715.2 | 7505.2 | 3651.3 | 2973.62 | 3345.77 |
| | worst | 3249.96 | 5367.1 | **3235.8** | 8698.7 | 9592.3 | 6789.6 | 6132.55 | 6376.29 |

We finish this section with a discussion on the behavior of CLA-BBPSO according to the obtained results. The adaptability of the algorithm is very beneficial to its performance on problems like $f_{11}$ (see Fig. 8d). The strategy adaption mechanism plays a significant role in the achieved results, as it is able to select suitable strategies for the particles based on the characteristics of the problems and the fitness information of the swarm. This idea will be obvious if we consider the average selection probability of different operators during the search, as in Section 4.6. For problems like $f_2$, $f_3$ and $f_5$ CLA-BBPSO decreases the selection probability of operators B and C. There exists no correlation among the coordinates of these problems which can be learned into the covariance matrices of the particles. However, CLA-BBPSO increases the probability of operator B on problems like $f_{10}$ where its coordinates are fully correlated.

**Table 6**
The results of Wilcoxon test. + indicates the superiority of the approach, on a benchmark, − indicates the converse, and = indicates failure in rejecting the null hypothesis.

| Test function | APSO | | $PS^2OS_R$ | | CLPSO | |
|---|---|---|---|---|---|---|
| | p-value | result | p-value | result | p-value | result |
| $f_1$ | 1.20e-12 | + | 1.21e-12 | + | 1.21e-12 | + |
| $f_2$ | 1.21e-12 | + | 1.21e-12 | + | 1.21e-12 | + |
| $f_3$ | 3.99e-8 | – | 2.90e-11 | + | 2.94e-11 | + |
| $f_4$ | 0.46 | = | 2.57e-11 | – | 1.78e-11 | + |
| $f_5$ | 2.20e-6 | + | 1.10e-2 | + | 1.21e-12 | + |
| $f_6$ | 1.17e-12 | + | 4.19e-2 | + | 1.21e-12 | + |
| $f_7$ | 1.12e-12 | + | – | = | 1.21e-12 | + |
| $f_8$ | 4.30e-12 | + | – | = | 1.21e-12 | + |
| $f_9$ | 1.61e-11 | + | – | = | 1.21e-12 | + |
| $f_{10}$ | 3.01e-11 | + | 3.01e-11 | + | 3.01e-11 | + |
| $f_{11}$ | 1.20e-8 | + | 2.15e-6 | + | 4.97e-11 | + |
| $f_{12}$ | 4.99e-9 | + | 2.31e-6 | + | 3.19e-9 | + |
| $f_{13}$ | 8.66e-5 | + | 0.33 | = | 1.10e-11 | + |
| $f_{14}$ | 9.10e-7 | + | 1.65e-9 | – | 2.12e-11 | + |
| $f_{15}$ | 9.81e-8 | + | 2.84e-10 | + | 4.61e-10 | + |
| $f_{16}$ | 8.10e-10 | + | 3.01e-11 | + | 3.51e-2 | + |

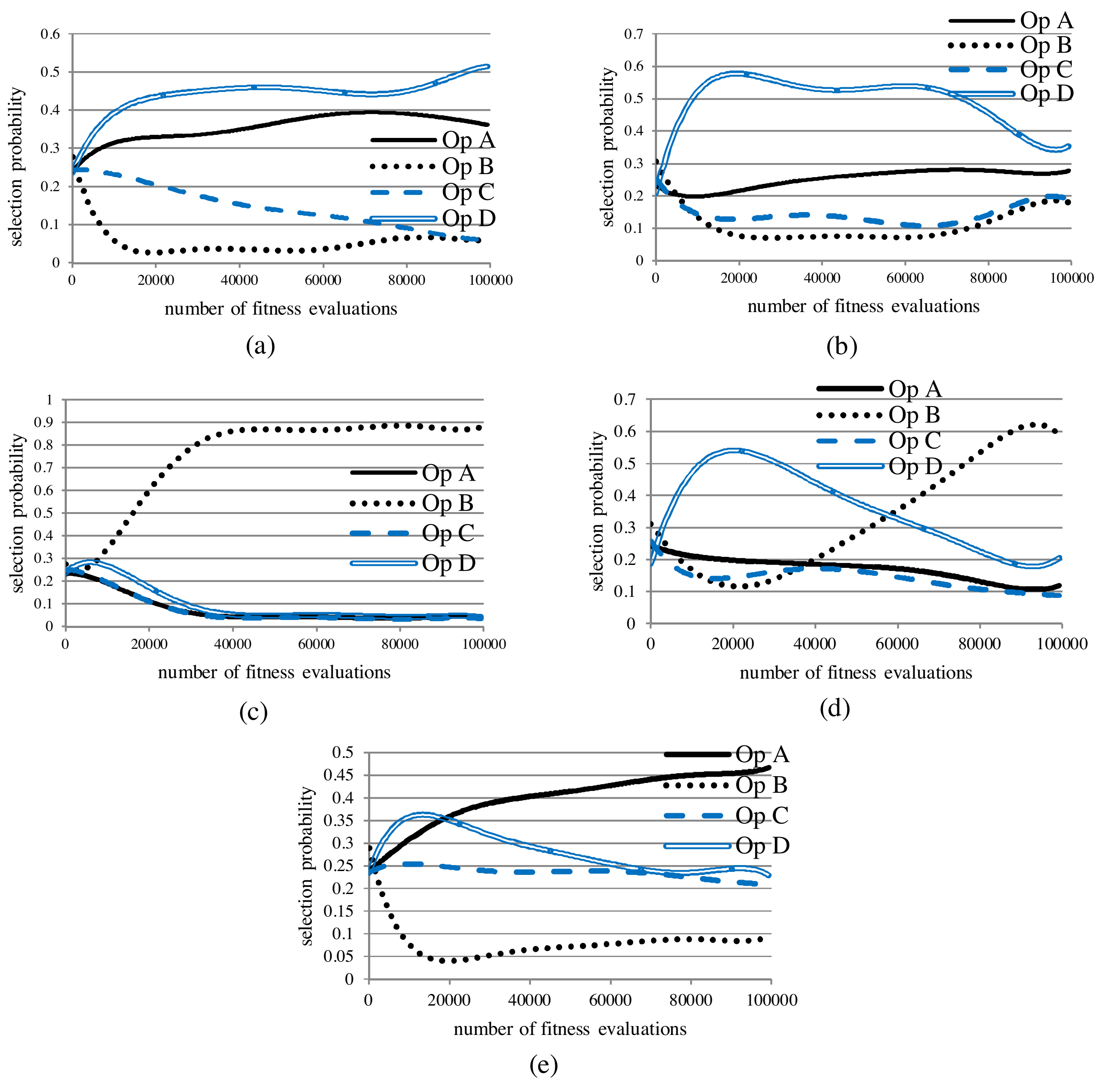


**Fig. 8.** The average probability of each operator in the population throughout the search process on: $f_3$ (a), $f_5$ (b), $f_{10}$ (c), $f_{11}$ (d), $f_{12}$ (e).

Also, the algorithm balances the exploration and exploitation characteristics of the search through operator selection; as, depending on the fitness dynamics, some of the proposed operators tend to have more exploration or exploitation characteristics. Considering a problem like $f_1$, which has its optimum point near the boundaries of the search space, CLA-BBPSO is capable of diversifying the population to discover the boundaries of the search space, which CLPSO is also capable of. It also can achieve a proper balance between exploration and exploitation, which enables the proposed method to achieve more accurate results than CLPSO on this function.

As CLA-BBPSO seems to be more explorative than the other approaches, its convergence speed may be low on some problems like $f_8$ and $f_9$ (Table 5). However, the developed operators may have some impact in this matter. Other operators would be developed that can acquire more knowledge of the fitness landscape instead of operators A, and D. These operators are based on simple Gaussian or multivariate Gaussian distributions, and utilize little information about the landscape properties or the past experiences of the particles. Finally, FPSO obtained a little better results than CLA-BBPSO on $f_{12}$ and $f_{15}$ (also consider FIPS uses a higher connectivity, and its results are weaker than the other peer methods on these functions). FPSO uses a dynamic topology where it starts with a fully connected population and gradually decreases its connectivity. In general, the topology of the population can greatly affect the performance of the swarm approaches. Like different strategies, the appropriate topology for the population greatly depends on the fitness landscape. Accordingly, CLA-BBPSO can be extended to utilize dynamic topologies to achieve better performances.

## 4.5. Experimental results on 50-D test problems

This section conducts the experiments of the second part of Section 4.4 on the 50-D test problems where the accuracy of the methods on the given benchmarks is compared after a maximum number of fitness evaluations. Table 7 provides the statistical results of the compared algorithms on each of the benchmark problems. Also, Fig. 7 provides the distribution of the solutions of each method on the benchmark set.

The experimental results on 50-D functions are much similar to their 30-D cases. CLABBPSO obtained the best average on almost all of the tested functions except $f_{12}$, where it achieved the second best average fitness just behind FPSO. The achieved average results of none of the peer methods are better than CLA-BBPSO (except in one case), and CLA-BBPSO is superior to each of them in some instances. In terms of average obtained results, CLA-BBPSO is better than APSO on 13 benchmark problems and is somehow equivalent on the other 3; it is better than CLPSO on all of the benchmarks except $f_1$; in comparison with FPSO, CLA-BBPSO is better on 15 benchmarks and is a little worst on only $f_{12}$; comparing CLABBPSO with FIPS and $PS^2S_R$, its better on respectively 10 and 14 functions and is equivalent on the reminder. From these observations, the superiority of CLA-BBPSO over other peer methods can be easily deducted.

## 4.6. A study on the learning capabilities of CLA-BBPSO

This section studies the effect of the proposed action selection mechanism on the performance of the algorithm, and illustrates its functioning. Fig. 8 shows how the operator probabilities vary throughout the algorithm's iterations on a group of benchmarks from Table 2. Each figure represents the average operator probability in the population over the spent number of fitness evaluations. As it can be seen from the figures, each test problem requires operators with different characteristics. For separable problems the probabilities of operators B and C reduce over time, and get lower than the probabilities of operators A and D. This may be due to the fact that no dependency exists among their coordinates which can be learned over time. However, the type of dependencies among the coordinates of hyper-ellipsoid function completely fits with operator B. Therefore, the probability of operator B is increased over time. For the other two test functions, the probabilities change according to the state of the population at different stages of the search process.

To demonstrate that the algorithm actually has learning capabilities, it is statistically compared with its pure chance version (actions are selected with equal probability all the time). For this purpose, Wilcoxon test is used to demonstrate the statistical difference of the two versions. The results of this test are provided in Table 8. From Table 8, CLA-BBPSO is statistically superior to the pure chance version on 11 benchmarks, and is statistically indistinguishable on the reminder. In addition, on the latter instances, CLA-BBPSO was better than its pure chance version in terms of the convergence speed (which would have been clear if we compared them using convergence curves).

## 4.7. Time complexity of the proposed approach

This section investigates the time complexity of the proposed approach. As the evolutionary phase of the proposed method is analogous to other simple evolutionary or swarm optimization methods, this section analyzes the overhead of other phases of the proposed approach on the evolutionary phase. There are 4 phases to consider: action selection, partitioning, covariance matrix adaption, and strategy adaption. Before continuing, let us assume that the population size is $n$, i.e. $N \times M = n$, also, the problem dimensionality is $D$. Action selection simply selects an action for each learning automaton based on its probability vector. As the number of actions is a small constant (4), this phase can be performed in a linear time with respect to $n$, i.e. $o(n)$. Action selection is repeated once for every $D/2$ iterations of the evolution phase, so its overhead to each iteration of the evolution phase is $o(2n/D) = o(n/D)$. As we are using partitions of size two, the partitioning phase can be performed independently for each particle by randomly permutating its coordinates, which can be performed in $o(D)$; hence, its overall contribution to each iteration of the evolution phase is $o(n)$. Covariance matrix adaptation phase includes finding the best particle in the neighborhood of each particle, and simple matrix operations. The first part can be performed efficiently in $o(n)$ for all particles, and the second one in $o(D^2)$ for each particle. So, the overhead of this phase on each iteration is $o(n/D + nD) = o(nD)$. The overhead of strategy adaption phase can be similarly computed, which is $o(n/D)$. The used updating operators, except operator B are simple and have a low time complexity. While generating samples from multivariate Gaussian distribution using operator B is time consuming, this operator may be selected fewer times than the others.

To conclude this section, we provided the average run times of each peer method on the benchmark functions given Table 2. The run time of an algorithm greatly depends on the efficiency of the implementation, used data structures, and the running platform; however, it may give an insight on the complexity of the proposed method. Looking at Table 9, it is obvious that the proposed method is slower than the other peer methods on many instances, which was predictable. Algorithms like FPSO do not have much overhead as CLA-BBPSO, and each of their generations involves only a few simple operations. However, the proposed method is not much slower. Besides, on time consuming functions like $f_6$ and $f_7$ the runtime ratio of different methods approaches to 1.

There remains two more points to discuss. First, as the proposed updating strategies have different time complexities, and the method may select the most complex one more than the others (or vice versa), its runtime on some problems may be lower than some simpler ones. Second, APSO has a very long run time on

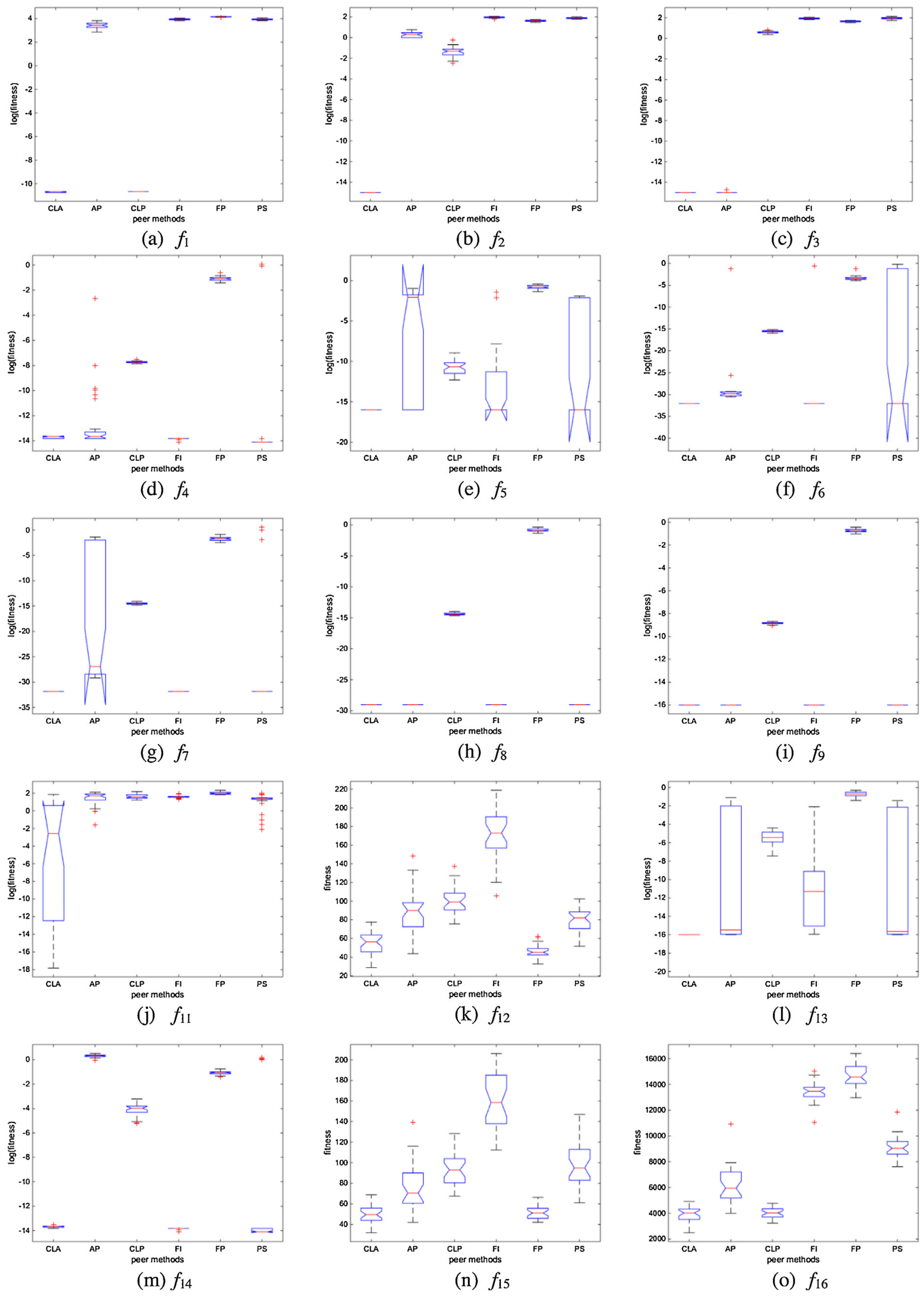


**Fig. 7.** Box plot of compared methods on the benchmark set. CLA, AP, CLP, FI, FP, PS, SM, and St are used to denote CLA-BBPSO, APSO, CLPSO, FIPS, FPSO, $PS^2OS_R$, SMA-BBPSO, and Starling PSO respectively.

**Table 7**
Statistical information (in this order: average, median, best, and worst) of obtained results for the compared algorithms.

| | CLAMS | APSO | CLPSO | FIPS | FPSO | $PS^2OS_R$ | |
|---|---|---|---|---|---|---|---|
| $f_1$ | average | **2.08e-11** | 2990.6 | **2.18e-11** | 8614.3 | 14003 | 8723.3 |
| | median | **2.18e-11** | 2732.8 | **2.18e-11** | 8621.4 | 14122 | 8737.8 |
| | best | **1.81e-11** | 710.63 | **2.18e-11** | 6704.7 | 12196 | 6830.3 |
| | worst | **2.18e-11** | 6761.8 | **2.18e-11** | 10624 | 15167 | 11311 |
| $f_2$ | average | **0** | 2.42 | 7.49e-2 | 93.72 | 42.79 | 80.46 |
| | median | **0** | 1.98 | 4.47e-2 | 93.19 | 41.12 | 77.60 |
| | best | **0** | 0.99 | 3.23e-3 | 58.23 | 30.56 | 61.68 |
| | worst | **0** | 5.96 | 5.42e-1 | 112.11 | 55.64 | 116.41 |
| $f_3$ | average | **0** | **0** | 3.97 | 88.45 | 44.67 | 92.41 |
| | median | **0** | **0** | 4.06 | 89.31 | 44.24 | 92.85 |
| | best | **0** | **0** | 2.34 | 67.07 | 33.17 | 55.32 |
| | worst | **0** | **0** | 6.61 | 114.09 | 55.04 | 135.50 |
| $f_4$ | average | **1.93e-14** | 7.33e-5 | 1.85e-8 | **1.42e-14** | 9.03e-2 | 8.52e-2 |
| | median | 2.22e-14 | 2.22e-14 | 1.80e-8 | 1.50e-14 | 8.25e-2 | **7.99e-15** |
| | best | 1.51e-14 | 1.50e-14 | 1.32e-8 | **7.99e-15** | 3.76e-2 | **7.99e-15** |
| | worst | **2.22e-14** | 2.19e-3 | 2.91e-8 | **1.50e-14** | 2.47e-1 | 1.37 |
| $f_5$ | average | **0** | 1.49e-2 | 8.45e-11 | 1.47e-3 | 1.76e-1 | 2.21e-3 |
| | median | **0** | 8.62e-3 | 2.16e-11 | **0** | 1.54e-1 | **0** |
| | best | **0** | **0** | 5.20e-13 | **0** | 4.21e-2 | **0** |
| | worst | **0** | 1.06e-1 | 1.11e-9 | 3.67e-2 | 3.77e-1 | 1.23e-2 |
| $f_6$ | average | **9.42e-33** | 8.29e-3 | 3.33e-16 | 1.65e-2 | 2.51e-3 | 7.47e-2 |
| | median | **9.42e-33** | 1.61e-30 | 3.10e-16 | **9.42e-33** | 3.42e-4 | **9.42e-33** |
| | best | **9.42e-33** | 2.94e-31 | 1.05e-16 | **9.42e-33** | 1.19e-4 | **9.42e-33** |
| | worst | **9.42e-33** | 6.22e-2 | 7.04e-16 | 2.48e-1 | 6.30e-2 | 6.86e-1 |
| $f_7$ | average | **1.34e-32** | 6.22e-3 | 3.66e-15 | **1.34e-32** | 2.62e-2 | 1.22e-1 |
| | median | **1.34e-32** | 1.49e-27 | 3.26e-15 | **1.34e-32** | 2.06e-2 | **1.34e-32** |
| | best | **1.34e-32** | 6.63e-30 | 1.55e-15 | **1.34e-32** | 3.50e-3 | **1.34e-32** |
| | worst | **1.34e-32** | 4.39e-2 | 8.15e-15 | **1.34e-32** | 1.36e-1 | 3.60 |
| $f_8$ | average | **0** | **0** | 4.29e-15 | **0** | 1.63e-1 | **0** |
| | median | **0** | **0** | 3.81e-15 | **0** | 1.21e-1 | **0** |
| | best | **0** | **0** | 1.79e-15 | **0** | 4.37e-2 | **0** |
| | worst | **0** | **0** | 1.03e-14 | **0** | 4.17e-1 | **0** |
| $f_9$ | average | **0** | **0** | 1.44e-9 | **0** | 1.85e-1 | **0** |
| | median | **0** | **0** | 1.42e-9 | **0** | 1.77e-1 | **0** |
| | best | **0** | **0** | 8.33e-10 | **0** | 9.33e-2 | **0** |
| | worst | **0** | **0** | 2.07e-9 | **0** | 3.58e-1 | **0** |
| $f_{10}$ | average | **1.92e-6** | 0.742 | 4563.93 | 4.89 | 139.91 | 6.65e-3 |
| | median | **1.34e-6** | 0.624 | 4184.92 | 6.98 | 153.76 | 7.81e-3 |
| | best | **2.09e-7** | 0.563 | 3950.48 | 0.98 | 34.90 | 6.03e-4 |
| | worst | **1.17e-5** | 1.644 | 5284.03 | 12.98 | 410.01 | 3.78e-2 |
| $f_{11}$ | average | **3.69** | 51.65 | 50.17 | 46.66 | 106.45 | 35.74 |
| | median | **2.83e-3** | 52.56 | 38.57 | 36.00 | 93.73 | 27.82 |
| | best | **1.49e-18** | 2.39e-2 | 17.01 | 19.78 | 66.86 | 7.589e-3 |
| | worst | **72.73** | 130.15 | 154.42 | 91.43 | 216.32 | 108.26 |
| $f_{12}$ | average | 54.67 | 89.97 | 100.87 | 169.98 | **46.29** | 78.85 |
| | median | 56.33 | 90.04 | 99.05 | 172.98 | **45.19** | 80.59 |
| | best | 28.85 | 43.77 | 75.64 | 105.53 | **32.75** | 51.73 |
| | worst | 77.60 | 148.25 | 137.14 | 218.91 | **62.25** | 102.48 |
| $f_{13}$ | average | **0** | 1.18e-2 | 7.18e-6 | 2.73e-4 | 2.26e-1 | 6.86e-3 |
| | median | **0** | **3.33e-16** | 3.64e-6 | 5.98e-12 | 1.93e-1 | **2.22e-16** |
| | best | **0** | **0** | 3.57e-8 | **1.11e-16** | 3.97e-2 | **0** |
| | worst | **0** | 8.01e-2 | 3.98e-5 | 8.13e-3 | 4.99e-1 | 6.10e-2 |
| $f_{14}$ | average | **2.05e-14** | 2.14 | 1.38e-4 | **1.41e-14** | 9.20e-2 | 2.56e-1 |
| | median | 2.22e-14 | 2.17 | 1.06e-4 | 1.50e-14 | 8.82e-2 | **7.99e-15** |
| | best | 1.50e-14 | 8.79e-1 | 5.96e-6 | **7.99e-15** | 3.87e-2 | **7.99e-15** |
| | worst | **2.93e-14** | 3.25 | 6.03e-4 | **1.50e-14** | 1.75e-1 | 1.80 |
| $f_{15}$ | average | **50.22** | 75.35 | 94.09 | 160.03 | 51.30 | 95.76 |
| | median | **49.53** | 70.50 | 93.00 | 158.50 | 51.07 | 94.00 |
| | best | **32.00** | 42.47 | 67.63 | 112.42 | 42.08 | 61.19 |
| | worst | 69.00 | 139.29 | 128.37 | 206.18 | **66.32** | 147.81 |
| $f_{16}$ | average | **3912.49** | 6242.7 | 4029.4 | 13477 | 14733 | 8872.4 |
| | median | **4016.95** | 5945.5 | 4020.9 | 13473 | 14575 | 8823.2 |
| | best | **2488.64** | 3991.5 | 3233.3 | 11039 | 12962 | 7586.6 |
| | worst | 4923.70 | 10940 | **4772.7** | 15033 | 16408 | 11845 |

some problems, which is due to the method it uses for out of search handling.

## 5. Conclusion

This paper presented a new multi swarm version of Bare Bones PSO, which uses different probability distributions for directing the movements of a particle. These distributions possess different characteristics that can be helpful on various types of landscapes and different stages of the optimization process. The parameters of these distributions are learned using different methods; meanwhile, the paper also suggested a new approach for adaptive determination of covariance matrices, which are used in the updating distributions. A CLA is responsible for the selection of

**Table 8**
The results of Wilcoxon test: CLA-BBPSO is compared with its pure chance version.

| function | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ |
|---|---|---|---|---|---|---|---|---|
| Result | + | + | + | + | = | + | + | + |
| P-Value | 1.5e-8 | 6.4e-12 | 2.9e-11 | 1.9e-11 | NaN | 1.5e-11 | 3.1e-12 | 1.2e-12 |
| function | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ | $f_{16}$ |
| Result | + | + | + | = | = | + | = | = |
| P-Value | 1.2e-12 | 3.0e-11 | 1.3e-6 | 0.66 | 0.52 | 2.2e-11 | 0.54 | 0.09 |

**Table 9**
The average run time of different methods on each benchmark problem (in seconds).

| | | CLAMS | APSO | CLPSO | FIPS | FPSO | $PS^2OS_R$ |
|---|---|---|---|---|---|---|---|
| $f_1$ | 30-D | 13.16 | 15.24 | 11.91 | 13.35 | 9.63 | 10.29 |
| | 50-D | 52.54 | 66.33 | 37.17 | 40.92 | 31.95 | 33.78 |
| $f_2$ | 30-D | 12.89 | 12.71 | 10.88 | 12.34 | 8.57 | 9.67 |
| | 50-D | 46.88 | 40.97 | 34.27 | 37.78 | 27.07 | 31.12 |
| $f_3$ | 30-D | 14.10 | 13.20 | 11.72 | 13.39 | 9.51 | 10.24 |
| | 50-D | 50.93 | 43.02 | 36.34 | 40.22 | 29.89 | 33.29 |
| $f_4$ | 30-D | 15.57 | 13.51 | 11.57 | 13.15 | 9.42 | 10.45 |
| | 50-D | 55.68 | 43.37 | 37.52 | 40.17 | 29.60 | 33.65 |
| $f_5$ | 30-D | 13.64 | 12.92 | 11.20 | 12.66 | 8.97 | 10.01 |
| | 50-D | 53.38 | 42.30 | 33.89 | 39.05 | 28.38 | 32.45 |
| $f_6$ | 30-D | 31.79 | 29.45 | 28.75 | 29.77 | 26.04 | 26.94 |
| | 50-D | 129.10 | 127.33 | 114.72 | 121.15 | 109.39 | 120.66 |
| $f_7$ | 30-D | 31.09 | 29.79 | 28.30 | 29.41 | 25.72 | 26.61 |
| | 50-D | 127.73 | 122.81 | 111.54 | 118.74 | 107.87 | 116.95 |
| $f_8$ | 30-D | 15.06 | 12.51 | 10.47 | 12.07 | 8.37 | 9.42 |
| | 50-D | 45.52 | 40.18 | 31.56 | 38.07 | 26.19 | 31.50 |
| $f_9$ | 30-D | 15.33 | 13.02 | 11.05 | 12.47 | 8.78 | 10.64 |
| | 50-D | 46.72 | 42.21 | 33.02 | 39.19 | 27.47 | 32.84 |
| $f_{10}$ | 30-D | 20.23 | 19.24 | 17.97 | 18.68 | 15.07 | 16.96 |
| | 50-D | 80.46 | 75.05 | 67.16 | 70.94 | 59.36 | 65.69 |
| $f_{11}$ | 30-D | 14.12 | 13.14 | 11.36 | 12.93 | 9.04 | 10.71 |
| | 50-D | 48.76 | 44.65 | 34.01 | 39.79 | 28.12 | 32.81 |
| $f_{12}$ | 30-D | 13.10 | 13.41 | 11.91 | 13.47 | 9.24 | 10.99 |
| | 50-D | 43.34 | 44.42 | 36.16 | 40.03 | 29.34 | 33.59 |
| $f_{13}$ | 30-D | 15.56 | 14.60 | 12.82 | 14.85 | 10.52 | 12.33 |
| | 50-D | 57.74 | 50.08 | 40.06 | 45.38 | 35.26 | 39.20 |
| $f_{14}$ | 30-D | 15.74 | 14.10 | 12.34 | 13.63 | 9.89 | 11.62 |
| | 50-D | 57.02 | 46.48 | 37.08 | 41.86 | 31.27 | 33.39 |
| $f_{15}$ | 30-D | 15.03 | 14.34 | 12.81 | 13.81 | 10.01 | 11.67 |
| | 50-D | 47.47 | 56.38 | 38.85 | 42.78 | 31.55 | 33.97 |
| $f_{16}$ | 30-D | 13.56 | 28.16 | 13.37 | 14.05 | 10.26 | 11.72 |
| | 50-D | 44.84 | 280.51 | 41.46 | 44.02 | 32.69 | 33.88 |

an appropriate distribution based operator for each particle. CLA learns the best operator for each particle through a reinforcement learning process. The experimental study in Section 4.6 illustrated the learning behavior of the method, where CLA chooses a suitable operator based on the problem landscape and the search stage of the execution. Also, the learning ability of the algorithm is demonstrated through comparison with its pure chance version. The effectiveness of the proposed method has examined experimentally through comparison with some other PSO techniques. The comparison results proved the high exploration capabilities of the approach along with its appropriate convergence speed.

In summary, the major contributions of the paper are as follows:

- Development and investigation of new updating operators for bare bones PSO.
- The integration of CLA model into bare bones PSO for adaptive control of updating distributions.
- Development of a new technique for refining the covariance matrices of Multivariate Gaussian operators.

For future work, other operators can be developed and investigated to improve the convergence speed of the approach. Also, considering the effect of an appropriate population topology on the performance of swarm approaches, the strategy adaption mechanism can be extended for dynamic topology control. Finally, the proposed covariance matrix adaption scheme will be further investigated in the future works to improve knowledge acquisition into the covariance matrices.

## References

[1] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, Piscataway, 1995, pp. 1942–1948.

[2] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (2006) 281–295.

[3] Y. Shi, R. Eberhart, A modified particle swarm optimizer, Proceedings of IEEE World Congress on Computational Intelligence (1998) 69–73.

[4] S.-K.S. Fan, Y.-Y. Chiu, A decreasing inertia weight particle swarm optimizer, Eng. Optim. 39 (2007) 203–228.

[5] Y.-T. Juang, S.-L. Tung, H.-C. Chiu, Adaptive fuzzy particle swarm optimization for global optimization of multimodal functions, Inf. Sci. 181 (2011) 4539–4549.

[6] X. Cai, Y. Cui, Y. Tan, Predicted modified PSO with time-varying accelerator coefficients, Int. J. Bio-Inspired Comput. 1 (2009) 50–60.

[7] Z.-H. Zhan, J. Zhang, Y. Li, H.S.-H. Chung, Adaptive particle swarm optimization, Eng. Optim. 39 (2009) 1362–1381.

[8] A. Kaveh, T. Bakhshpoori, E. Afshari, An efficient hybrid particle swarm and swallow swarm optimization algorithm, Comput. Struct. 143 (2014) 40–59.

[9] Y.-P. Chen, W.-C. Peng, M.-C. Jian, Particle swarm optimization with recombination and dynamic linkage discovery, IEEE Trans. Syst. Man Cybern. Part B Cybern. 37 (2007) 1460–1470.

[10] G. Wu, D. Qiu, Y. Yu, W. Pedrycz, M. Ma, H. Li, Superior solution guided particle swarm optimization combined with local search techniques, Expert Syst. Appl. 41 (2014) 7536–7548.
[11] P.J. Angeline, Using selection to improve particle swarm optimization, Proceedings of IEEE International Conference on Evolutionary Computation (1998).
[12] Y.V. Pehlivanoglu, A new particle swarm optimization method enhanced with a periodic mutation strategy and neural networks, IEEE Trans. Evol. Comput. 17 (2013) 436–452.
[13] M. Pant, R. Thangaraj, A. Abraham, A new PSO algorithm with crossover operator for global optimization problems, in: Innovations in Hybrid Intelligent Systems, Springer, 2007, pp. 215–222.
[14] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler maybe better, IEEE Trans. Evol. Comput. 8 (2004) 204–210.
[15] W.H. Lim, N.A.M. Isa, Particle swarm optimization with adaptive time-varying topology connectivity, Appl. Soft Comput. 24 (2014) 623–642.
[16] X. Zhao, Z. Liu, X. Yang, A multi-swarm cooperative multistage perturbation guiding particle swarm optimizer, Appl. Soft Comput. 22 (2014) 77–93.
[17] H. Chen, Y. Zhu, K. Hu, Discrete and continuous optimization based on multi-swarm coevolution, Nat. Comput. 9 (2010) 659–682.
[18] A.B. Hashemi, M.R. Meybodi, A note on the learning automata based algorithms for adaptive parameter selection in PSO, Appl. Soft Comput. 11 (2011) 689–705.
[19] C. Li, S. Yang, T.T. Nguyen, A self-learning particle swarm optimizer for global optimization problems, IEEE Trans. Syst. Man Cybern. 42 (2012) 627–646.
[20] C. Li, S. Yang, An adaptive learning particle swarm optimizer for function optimization, Proceedings of IEEE Congress on Evolutionary Computation, CEC'09 (2009) 381–388.
[21] G.S. Piperagkas, G. Georgoulas, K.E. Parsopoulos, C.D. Stylios, A.C. Likas, Integrating particle swarm optimization with reinforcement learning in noisy problems, in: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, ACM, 2012, pp. 65–72.
[22] M. Clerc, J. Kennedy, The particle swarm-explosion stability, and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 6 (2002) 58–73.
[23] J. Kennedy, Bare bones particle swarms, Proceedings of IEEE Swarm Intelligence Symposium, SIS'03 (2003) 80–87.
[24] H. Zhang, D.D. Kennedy, G.P. Rangaiah, A. Bonilla-Petriciolet, Novel bare-bones particle swarm optimization and its performance for modeling vapor–liquid equilibrium data, Fluid Phase Equilib. 301 (2011) 33–45.
[25] H.-I. Hsieh, T.-S. Lee, A modified algorithm of bare bones particle swarm optimization, IJCSI (2010) 11.
[26] M. Campos, R. Krohling, I. Enriquez, Bare bones particle swarm optimization with scale matrix adaptation, IEEE Trans. Cybern. 44 (2014) 1567–1578.
[27] J. Yao, D. Han, Improved barebones particle swarm optimization with neighborhood search and its application on ship design, Math. Prob. Eng. 2013 (2013).
[28] R. Krohling, E. Mendel, Bare bones particle swarm optimization with Gaussian or Cauchy jumps, Proceedings of Evolutionary Computation CEC'09 (2009) 3285–3291.
[29] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Trans. Evol. Comput. 16 (2012) 210–224.
[30] W. Gao, F.T. Chan, L. Huang, S. Liu, Bare bones artificial bee colony algorithm with parameter adaptation and fitness-based neighborhood, Inf. Sci. 316 (2015) 180–200.
[31] C.-H. Chen, A variant of unified bare bone particle swarm optimizer, Proceedings of IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT) (2013) 18–22.
[32] C.-H. Chen, Bare bone particle swarm optimization with integration of global and local learning strategies, Proceedings of IEEE International Conference OnMachine Learning and Cybernetics (ICMLC) (2011) 692–698.
[33] H. Beigy, M.R. Meybodi, A mathematical framework for cellular learning automata, Adv. Complex Syst. 7 (2004) 295–319.
[34] H. Beigy, M.R. Meybodi, Asynchronous cellular learning automata, Automatica 44 (2008) 1350–1357.
[35] H. Beigy, M.R. Meybodi, Open synchronous cellular learning automata, Adv. Complex Syst. 10 (2007) 527–556.
[36] M. Esnaashari, M.R. Meybodi, Irregular cellular learning automata, IEEE Trans. Cybern. 45 (2014) 1622–1632.
[37] M. Esnaashari, M.R. Meybodi, Deployment of a mobile wireless sensor network with k-coverage constraint: a cellular learning automata approach, Wirel. Netw. 19 (2013) 945–968.
[38] M. Esnaashari, M. Meybodi, A cellular learning automata based clustering algorithm for wireless sensor networks, Sens. Lett. 6 (2008) 723–735.
[39] H. Beigy, M.R. Meybodi, Cellular learning automata based dynamic channel assignment algorithms, Int. J. Comput. Intell. Appl. 8 (2009) 287–314.
[40] R. Vafashoar, M. Meybodi, A.M. Azandaryani, CLA-DE: a hybrid model based on cellular learning automata for numerical optimization, Appl. Intell. 36 (2012) 735–748.
[41] J.A. Torkestani, M.R. Meybodi, A cellular learning automata-based algorithm for solving the vertex coloring problem, Expert Syst. Appl. 38 (2011) 9237–9247.
[42] K.S. Narendra, M.A. Thathachar, Learning Automata: An Introduction, Courier Corporation, 2012.
[43] M.A. Thathachar, P.S. Sastry, Networks of Learning Automata: Techniques for Online Stochastic Optimization, Springer Science & Business Media, 2011.
[44] S. Wolfram, A New Kind of Science, Wolfram media Champaign, 2002.
[45] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, Proceedings of IEEE International Conference on Evolutionary Computation (1996) 312–317.
[46] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2001) 159–195.
[47] T. Suttorp, N. Hansen, C. Igel, Efficient covariance matrix update for variable metric evolution strategies, Mach. Learn. 75 (2009) 167–197.
[48] W. Gong, Á. Fialho, Z. Cai, Adaptive strategy selection in differential evolution, in: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, ACM, 2010, pp. 409–416.
[49] W. Gong, Á. Fialho, Z. Cai, H. Li, Adaptive strategy selection in differential evolution for numerical optimization: an empirical study, Inf. Sci. 181 (2011) 5364–5386.
[50] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[51] R. Salomon, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms, BioSystems 39 (1996) 263–278.
[52] N. Netjinda, T. Achalakul, B. Sirinaovakul, Particle swarm optimization inspired by starling flock behavior, Appl. Soft Comput. 35 (2015) 411–422.