Applied Intelligence (2018) 48:4317–4337
https://doi.org/10.1007/s10489-018-1197-z

Explicit memory based ABC with a clustering strategy for updating and retrieval of memory in dynamic environments

Hamid Parvin[1,2] · **Samad Nejatian**[3,4] · **Majid Mohamadpour**[4,5]

Published online: 9 June 2018

Abstract
The Artificial Bee Colony (ABC) algorithm is considered as one of the swarm intelligence optimization algorithms. It has been extensively used for the applications of static type. Many practical and real-world applications, nevertheless, are of dynamic type. Thus, it is needed to employ some optimization algorithms that could solve this group of the problems that are of dynamic type. Dynamic optimization problems in which change(s) may occur through the time are tougher to face than static optimization problems. In this paper, an approach based on the ABC algorithm enriched with explicit memory and population clustering scheme, for solving dynamic optimization problems is proposed. The proposed algorithm uses the explicit memory to store the aging best solutions and employs clustering for preserving diversity in the population. Using the aging best solutions and keeping the diversity in population of the candidate solutions in the environment help speed-up the convergence of the algorithm. The proposed approach has been tested on Moving Peaks Benchmark. The Moving Peaks Benchmark is a suitable function for testing optimization algorithms and it is considered as one of the best representative of dynamic environments. The experimental study on the Moving Peaks Benchmark shows that the proposed approach is superior to several other well-known and state-of-the-art algorithms in dynamic environments.

Keywords Swarm intelligence · Optimization · Dynamic environment · Artificial bee colony · Explicit memory · Moving peaks benchmark

1 Introduction

In recent years, evolutionary algorithms have attracted much interest among researchers for solving dynamic optimization problems [52–54]. An evolutionary algorithm suitable for dynamic optimization problems should not only be able to locate the optimum, as it does in the static optimization problems, but also be capable of detecting the time when the changes in the positions of optima occur and also tracking the newly relocated optima. In the static environments for the reason that optima are not moved in the environment and each of them remains in a fixed position passing the time, it is easy for evolutionary algorithms to find the global optimum, but in an environment that the optima are subject to change it is not an easy task to find the global optimum following every change that occurs in the environment. Thus strong heuristic mechanisms are needed to solve dynamic optimization problems. One of the weaknesses in evolutionary algorithms for solving dynamic optimization problems is that they can't single-handedly solve them. Thus, they should employ a number of appropriate strategies that make them able to manage dynamic optimization problems. One proper strategy for dynamic environments is to use a combination of some auxiliary elements; for example, using of the memory element in evolutionary algorithms can be very useful. Some of these auxiliary elements are presented in [1–3]. The main weakness of using Standard Evolutionary Algorithms in a dynamic environment is that, once the algorithm starts

✉ Samad Nejatian
nejatian@iauyasooj.ac.ir

1 Department of Computer Engineering, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran

2 Young Researchers and Elite Club, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran

3 Department of Electrical Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran

4 Young Researchers and Elite Club, Yasooj Branch, Islamic Azad University, Yasooj, Iran

5 Department of Computer Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran

to converge around some optimal or near optimal solution, it will likely lose its ability to continue the search for new optima, when the environment changes. Thus, one key point in optima tracking approaches is the need for increasing or maintaining diversity among the individuals in the population, so that the algorithm keeps its ability to explore the new optima when the environments change, even after the population has incompletely converged to an optimum, i.e. the candidate solutions have become close to the optimal solution. This paper involves a comprehensive experimental study based on the Moving Peaks Benchmark (MPB) problem [25]. In order to show whether the proposed technique effects on increasing convergence speed, the examinations will be conducted on MPB. We compare the performance of the proposed approach with other well-known and state-of-the-art approaches in dynamic environments. Other algorithms selected to be compared with the proposed approach are the state-of-the-art algorithms that most of the researchers use as their competent methods.

The contributions of the paper are as follows:

1. Introducing an explicit memory based artificial bee colony algorithm for dynamic optimization,
2. Introducing a new mechanism suitable for diversity preserving in the proposed dynamic optimization algorithm based on an innovative updating-retrieving mechanism using clustering of the population,
3. Huge experimental study on tuning the proposed dynamic optimization algorithm's parameters.

The second one is a vital contribution. Indeed, through it we help memory usage be more goal-oriented. One of the most important goals of memory usage has been injecting diversity in the population after convergence of the algorithm. Through the second contribution, this is maximally satisfied. It is due to the clustering concept. Indeed, through clustering of population and letting only exchange occurrence in a cluster guarantees the diversity in the population; while through clustering of memory and letting only exchange occurrence in a cluster guarantees the diversity in the memory.

This paper is organized as follows. Section 1 is introduction. Section 2 is dedicated to memory definition and its strategies for updating and retrieving in the optimization algorithms. Section 3 includes backgrounds, i.e. the ABC algorithm and paper definitions. Section 4 presents related work. The proposed method is presented in sufficient detail in Section 5 along with its algorithmic Pseudo code. Section 6 first presents MPB problem. Then experimental results and their analysis have been presented in the rest of Section 6. Finally, the paper is concluded in Section 7 with a discussion on the possible future work of the research.

2 Memory types in dynamic environments

In dynamic environments, memory is applied to save outstanding past solutions with the assumption that the optimum may return to its previous point. When certain aspects of the problem present some variety of periodic behavior, old solutions might be used to bias the search in their vicinity and consequently the computational time is reduced. The use of memory is beneficial on those types of environments. Memory based approaches can be divided in two categories. The first one is implicit memory and the second one is explicit memory.

2.1 Implicit memory

The usage of redundant representations is the main characteristic of the implicit memory methods. Memory implicit are divided into two groups: (a) double memory and (b) diploid memory. Diploid memory representations have been first used by Goldberg et al. [10]. The diploid implicit memory functions have been described in detail at [11]. Other works using implicit memory based on diploid representations and dominance mechanisms can be found in [12–16].

2.2 Explicit memory

Using explicit memory for the storage of useful environmental information is in contrast to using implicit memory that even additional information is also reserved. Here only useful information is to be reserved (Fig. 1). Yang [17] has divided explicit memory approaches into two main categories: (a) direct memory and [18, 19] (b) associative memory [20, 21].

2.3 Updating memory

To replace the members of population in the memory, different strategies have been proposed, some of which that are discussed in [22] are presented here:

Strategy 1: In this strategy, two individuals in memory with the least distance (the closest similarity) are selected and among the two, the individual with lower efficiency becomes the candidate for replacement. For example, if $fit(i)$ is the efficiency of the ith individual and $fit(j)$ is the efficiency of the jth individual, if $fit(i) < fit(j)$, then the ith

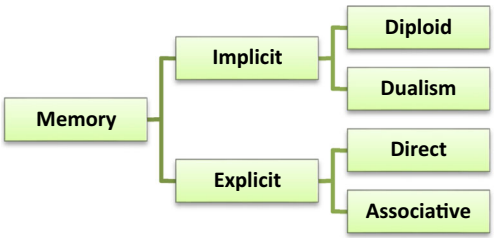

Fig. 1 Taxonomy of memory schemes for dynamic environments

individual in memory is replaced by the best individual in the population and vice versa.

Strategy 2: If $fit(j) \times \frac{d_{ij}}{d_{max}} \leq fit(new)$, then the *jth* individual in the memory will be replaced by the current individual. In this equation *fit(j)* is the individual efficiency of the *jth* person in the memory and d_{ij} is the distance between two individuals i and j and d_{max} is the maximum distance between any arbitrary pair of individuals.

Strategy 3: This strategy is known as similarity strategy in which the most similar individual is replaced by the best available individual in the population in the memory, provided that this individual has more efficiency than the individual in memory. In order to measure the similarity, Euclidean distance can be employed. Euclidean distance between *ith* and *jth* individuals, denoted by $d(i, j)$, is calculated according to equation $\sqrt{\sum_{d=1}^{D}(x_i^d - x_j^d)^2}$. In this strategy, the best individual of the population, denoted by β_{pop}, is nominated for being placed into the memory. Then the most similar individual in the memory with that β_{pop}, denoted by $\alpha_{\beta_{pop}}$, is replaced by β_{pop} if $f(\alpha_{\beta_{pop}}) \leq f(\beta_{pop})$.

2.4 Memory retrieval

The information stored in memory should be used for tracking of the optimum when optima are relocated. So the best time to retrieve data from memory is the moment when the environment is changed. Several strategies can be adopted to retrieve information from memory. One of the methods of memory information retrieval is to select the best individual in the memory then to replace it with the worst individual in the population [23].

3 Backgrounds

3.1 Artificial bee colony

In recent years a growing interest has been created in the field of swarm intelligence in dynamic optimization problems according to their importance in the real world. The collective intelligence is an almost new field of research focused on studying and modeling the behavior of social insects such as bees. Artificial bee colony algorithm [24] simulates an inquisitive and intelligent behavior of a set of bees. Artificial bee algorithm consists of six essential steps. These steps are briefly described in the following subsection.

3.1.1 ABC steps

Step (1) Initialization of the artificial bee colony algorithm parameters and the optimization problem parameters.

In general, optimization problem can be formulated based on (1)

$$\begin{aligned} &\arg\min_{\mathrm{x}} \mathrm{f(x)} \\ &\text{s. t. } \mathrm{x_i} \in \mathrm{X_i} \\ &\mathrm{g(x)} < 0 \\ &\mathrm{h(x)} = 0 \end{aligned} \tag{1}$$

In (1), $f(x)$ is the objective function that should be minimized. Each x is a set of decision variables (a bee) that $\{x_i \in X_i | i = 1, \ldots, N\}$. X_i is the possible range for ith decision variable. It means that $X = \{X_1, X_2, \ldots, X_N\}$ and $X_i \in (LB_i, UB_i)$ where LB_i and UB_i are the lower bound and upper bound of the ith decision variable $\boldsymbol{X}_i$. N is the number of decision variables (the number of features) and $g(x)$ and $h(x)$ are the equality and inequality relations respectively. Artificial bee colony algorithm includes three other parameters as well. These parameters include:

(A): Parameter *SN*, which is the number of food sources (solutions) in the population. *SN* is equal to the number of employee bees.

(B): The maximum cycle parameter for algorithm is denoted by *MCN*, which stands for the maximum number of generations that algorithm is permitted to proceed.

(C): Limit parameter is the frequency of motion of a bee to a position without improvement. This parameter is used to explore the food sources (solutions). If the frequency of motion of a bee to a position without improvement is more than the limit parameter, the bee considers that position as a deserted position.

Step (2): Creation of food source memory.

Food Source Memory (*FSM*) consists of a $SN \times N$ matrix. In each row of *FSM*, location of a food source is assumed. The *FSM* is created based on the (2). Vectors are arranged according to the adjacent cost equation in an upside trend.

$$FSM = \begin{bmatrix} \mathrm{X_1(1)} & \mathrm{X_1(2)} & \cdots & \mathrm{X_1(N)} & \mathrm{f(X_1)} \\ \vdots & & \ddots & \vdots & \vdots \\ \mathrm{X_{SN}(1)} & \mathrm{X_{SN}(2)} & \cdots & \mathrm{X_{SN}(N)} & \mathrm{f(X_{SN})} \end{bmatrix} \tag{2}$$

Generally any $x_j(i)$ (the position of *jth* bee in *ith* dimension) is generated as (3).

$$\begin{aligned} \mathbf{x}_j(i) &= LB_i + (UB_i - LB_i) \times r \\ &\forall j \in (1, 2, \ldots, SN), \forall i \in (1, 2, \ldots, N) \end{aligned} \tag{3}$$

where, r is a random number generated from uniform distribution in the closed range from 0 to 1 ($r \in [0, 1]$). $\mathrm{f(x_i)}$ indicates the fitness of x_i; i.e. ith bee.

Step (3): Allocation of employee bees to food sources.

At this stage, any employee bee moves into the food sources randomly and calculates the suitability of a position. Each bee chooses a neighbor randomly and moves toward it. The employee bee modifies its new position based on (4).

$$x_j^{new}(i) = x_j^{old}(i) + r \times (x_j^{old}(i) - x_k^{old}(i)) \quad \exists k \in (1, 2, \ldots, SN), k \neq j \text{ and } r \in [-1, 1] \tag{4}$$

In (4), $x_j(i)$ is the position of *jth* bee in the *ith* dimension and $x_k(i)$ stands for a neighbor bee position. Parameter r is a random number generated from uniform distribution in the closed range from -1 to 1 ($r \in [-1, 1]$). The pseudo code of the employee bee is presented in Algorithm 1.

Step (4): Sending the onlooker bees.

Onlooker bee phase includes three following phases.

1) Assigning a probability to each employee bee indicating probability of its selection (for exploring the environment around it) according to (5).

$$P_j = \frac{f(x_j)}{\sum_{k=1}^{SN} f(x_k)} \tag{5}$$

2) Employing the probability that an employee bee is selected by an onlooker bee is based on (5). This is done in fact by a roulette wheel; i.e. the higher probability of a position leads to the higher possibility of being selected.

Algorithm 1 Employed bee phase

```
1.  x^old = x
2.  for j = 1 ... SN do
3.      r = RandomNumberFrom([−1, 1])
4.      k = RandomSelecFrom({1, 2, ..., SN} − {j})
5.      i = RandomSelecFrom({1, 2, ..., N})
6.      x_j^new(i) = x_j^old(i) + r × (x_j^old(i) − x_k^old(i))
7.      calculatef(x_j^new)
8.      if (f(x_j^new) ≤ f(x_j^old)) then
9.          x_j = x_j^new and f(x_j) = f(x_j^new)
10.     end if
11. end for
```

Form (1): The pseudo code of the employee bee phase

Step (5): Sending the scout bees to search for new food sources.

3) Every onlooker bee randomly selects an employee bee and moves toward it. So each bee finds a new position after a move toward a randomly selected employee bee. If the efficiency of the new position is higher than the efficiency of its previous position, the bee selects to stay at the new position and resets its non-improvement counter; otherwise it returns to the previous position and adds one unit to its non-improvement counter. Non-improvement counter shows how many successive trials have been failed to find a better position by an onlooker bee. In other words, the non-improvement counter counts the number of consecutive movements of the bee with no improvement. The bee modifies its new position based on (4). If non-improvement counter of an onlooker bee exceeds a specified limit right after the onlooker bee fails to improve its position; it means that the food source has no nectar and that position should be abandoned. Pseudo code of the onlooker bee is based on Algorithm 2. In this pseudo code *sum_prob* is the cumulative probability of the employee bees being selected by onlooker bees.

Algorithm 2 Onlooker bee phase

```
1.  x^old = x
2.  for j = 1 ... SN do
3.          sum_prob = 0
4.          r = RandomNumberFrom([−1, 1])
5.          j = 0
6.          while (sum_prob ≤ r) do
7.              sum_prob = sum_prob + P_{j+1}
8.              j = j + 1
9.          end while
10.         k = RandomSelectFrom({1, 2, ..., SN}−{j})
11.         i = RandomSelectFrom({1, 2, ..., N})
12.         x_j^new(i) = x_j^old(i) + r × (x_j^old(i) − x_k^old(i))
13.         calculate f(x_j^new)
14.         if (f(x_j^new) ≤ f(x_j^old))then
15.                 x_j = x_j^new and f(x_j) = f(x_j^new)
16.         else
17.                 scout(i) = scout(i) + 1
18.         end if
19. end for
```

Form (2): The pseudo code of the onlooker bee phase

The bee that is responsible to find new solutions in the search environment is the scout bee. This bee moves in search space randomly to explore new areas. These bees must replace the found abandoned food sources with new food sources with random search according to (3). Pseudo code related to scout bee phase is presented in Algorithm 3.

Algorithm 3 Scout bee phase

```
1. for i = 1 ... SN do
2.     if(scout(i) = Limit) then
3.         generate x_j using (3)
4.         scout(i) = 0
5.     end if
6. end for
```

Form (3): The pseudo code related to the scout bee phase

Step (6): Save the best food source.

In this step the best food source position denoted by ${}^{best}x$ is stored in memory of food sources. The pseudo code related to artificial bee colony algorithm is based on Algorithm 4.

Algorithm 4 Artificial bee colony algorithm (Popfunction)

Input:
Algorithm parameters: N, SN, D, MCN, Limit, LB, UB
Output:
BEST Solution, BEST Fitness

1. *Begin*
   % *initialization*%
2. *Initialize solution population by* (3)
   % *Evaluate population*%
3. ***For*** $i = 1$ *to SN*
4. $f_i = f(x_i)$
5. ***End For***
6. $cycle = 1$

Repeat:
% *employ bee phase*%
7. ***For*** $i = 1 to SN$
8. *Producenewsolutions* x_i^{new} *foremploybeeby* (4)
9. *Apply the greedy selection process*
10. ***End For***
11. *Apply the ranking evaluation bees*

% *end of employ bee phase*%
% *Onlooker bee phase*%
12. ***For*** $i = 1$ *to SN*
13. *sharing the entire solutions between employ bees and onlooker bees*
14. *select a solution with roulette wheel strategy by* (5) *and name it k*
15. *Produce new solutions* x_k^{new} *for onlooker bees by* (4)
16. *Apply the greedy selection process*
17. ***End For***

% *end of onlooker bee phase*%
% *scout bee phase*%
18. *if Trial* > *Limit then*
19. *replace the solution with a new randomly produced solution* $\mathbf{x}_j$ *by* (3)
    % *end of scout bee phase*%
20. *Save in memory the best solution so far*
21. $cycle = cycle + 1$
22. *Until* $cycle = MCN$
23. *End*

Form (4): The pseudo code related to the ABC algorithm

3.2 Dynamic and static environment definitions

A dynamic environment is the one that changes over time. We follow this subsection by presenting the definitions of some keywords.

Dynamic environments are the ones that change continuously or discretely over time. These changes can be in large scale. Among the cases that may happen, the change in parameter values over time is considered. Consequently, after each environmental change, the problem optima have probably been subject to change. One of the most complete simulations of the dynamic environments is Moving Peaks Benchmark that possesses almost all features of a real-world dynamic environment. Indeed, a problem is considered as a dynamic one if its objective function is subject to change over time.

If the objective function is not a function of time, then it can be considered as a static objective function. For example, the problem of finding the shortest path in a graph is a static problem, while by adding the online traffic of paths to the problem, it will be a dynamic one.

Moment of environment change The moment in which the environment changes (optimization function changes) is called the moment of environment change (or the cycle of environment change) and is denoted by *MEC*.

One of the challenges in dynamic environments is to detect the moment of change. We need to identify the moment of change in the environment after it is occurred, because after a change in the environment a re-evaluation for all individuals (artificial bees) is needed. To detect an environment change, the algorithm computes the average fitness values of all artificial bees in a generation is more than the average fitness values of all artificial bees in the previous generation.

This strategy is used to identify the moment of change. Equation (6) presents a condition when it holds, then we can assume the change has been occurred. If $f(x_j^{new})$ presents the efficiency of the *jth* bee in current generation (after the change) and $f(x_j^{old})$ presents the efficiency of *jth* bee in previous generation (before the change), then (6) indicates the condition in where environment change has been occurred.

$$\frac{\sum_{j=1}^{SN} f(x_j^{new})}{SN} \geq \frac{\sum_{j=1}^{SN} f(x_j^{old})}{SN} \tag{6}$$

Colony error The error of a colony x^t (x^t shows colony at the tth function evaluation), denoted by $E(x^t, f_t(\vec{p}))$, is defined in (7) by fitness function $f_t(\vec{p})$ where $\vec{p}$ is a position

in the landscape of fitness function f_t. The fitness function f_t is a function that is defined at tth function evaluation (it is very important to note that except in the environment change moment, τ, $f_\tau \not\equiv f_{\tau-1}$. In other words, $f_\tau \equiv f_{\tau-1}$ is always a valid sentence provided that τ isn't environment change moment).

$$E(\boldsymbol{x}^{\boldsymbol{t}}, f_t(\vec{p})) = f_t(p_t^*) - \max_{j=1:SN}(f_t(\overrightarrow{x_j})) \tag{7}$$

where p_t^* is the optimal position in the landscape of fitness function $f_t(\vec{p})$ (which is not available).

OfflineError To measure the effectiveness of an evolutionary algorithm in a dynamic environment, we use a measure called offline error denoted by *OffLineError*. *OffLineError* is obtained by (8).

$$OffLineError(\pi) = \frac{1}{\pi}\sum_{t=1}^{\pi} E(x^t, f_t(\vec{p})) \tag{8}$$

where, π equals the number of fitness evaluations that have been completed.

4 Related work

In recent years, several methods have been expanded for solving dynamic optimization problems with the aim of increasing and maintaining diversity as the generations go forward during the entire run [4–6]. Ramsey and Grefenstette [7] have introduced a case-based method for initializing the genetic algorithm when a change is detected. Louis and Xu [8] have applied the same idea to the open shop scheduling problem. They have used an Evolutionary Algorithm combined with case-based reasoning. Yang et al. [9] have used a hybrid immigrant scheme. This method has combined the concepts of elitism, dualism and random immigrants. The best individual from the previous generation and its dual individual are retrieved in order to create immigrants via mutation. These elitism-based and dualism based immigrants together with some random immigrants were substituted into the current population, replacing the worst individuals in the population.

A cellular automata-based artificial immune system optimization algorithm has been proposed for dealing with dynamic environments [47]. The cellular automaton has been used as a diversity generator in population.

Xin et al. have used a self-adaptive mechanism for the determination of transfer rate in using immigrants in population to increase the diversity in the population. By the mentioned strategy, they have proposed a dynamic optimization algorithm [48].

In another work, a hybrid optimization algorithm has been proposed based on combining the artificial bee colony optimization algorithm and the particle swarm optimization algorithm. The main foundation of the algorithm is the artificial bee colony optimization algorithm where their bees have been improved by the particle swarm optimization algorithm. A cellular automaton has been employed to increase the diversity in the population [49].

A method called DMGA is an evolutionary algorithm that uses an explicit memory as a diversity generator in population [50]. The memory size is $m = 0.1 \times n$ where it is randomly initiated. In updating memory step, the memory individual that is the most similar to the best individual in the population is replaced the best individual in the population. The updating times are random. In DMGA, the fitness function values of all memory individuals should be reevaluated after each environmental change. If the change occurs, then from all population and memory individuals, a subset of $0.9 \times n$ individuals should be selected as the new population, but the memory should not be changed.

The artificial fish swarm algorithm has been modified by Yazdani et al. to become suitable for solving dynamic environment optimization problems [51]. The Dynamic Modified Artificial Fish Swarm Algorithm (DMAFSA) is modified parameters, behaviors and procedure of the standard AFSA so as to solve dynamic optimization problems.

Mohamadpour et al. [46] have proposed a memory-based approach for solving dynamic environment optimization problems. In their method, the chaotic genetic algorithm based on explicit memory has been employed. They also have proposed a suitable approach for memory updating.

5 Proposed algorithm

The most important challenge for a dynamic method in dealing with dynamic environments is how the method accomplishes the self-adaptation to the new (changed) environment. Therefore, it should track all local optima in its evolution so as to find them as soon as possible if the environment changes. To deal with the mentioned challenge, researchers have tried to improve the evolutionary and swarm intelligence algorithms by adding some special mechanisms. From another perspective, the dynamic algorithm should be (a) fast and (b) accurate. The artificial bee colony optimization algorithm is one of the swarm intelligence algorithms that has both high convergence rate and high capability of local search. The ABC optimization algorithm has been effectively used for solving the static optimization problems.

In this paper, a new method based on the ABC optimization algorithm has been proposed for solving dynamic optimization problems. It has been proved that the simple ABC optimization algorithm is unable to solve these problems [46, 49]. Therefore, the proposed ABC

optimization algorithm has been enriched by a memory mechanism and an updating strategy. One important feature of all optimization algorithms is they can converge to the optimal solution after some iterations and consequently they lose their diversity. Therefore, these methods without some improvements are unable to re-search the problem space after an environmental change in order to find the new optimal solution. Therefore, it is safe to acclaim that they are unable to appropriately solve the dynamic optimization problems. Therefore, the proposed method, as it is presented in the flowchart of Fig. 2, uses some new improvements in the ABC optimization algorithm.

5.1 Artificial bee colony algorithm based on clustering and memory

In this article we have proposed an Artificial Bee Colony algorithm based on Clustering and Memory (***ABCCM***) for dynamic environments. In this method, we employ an explicit memory to store the previously found optimum solutions with the aim of benefiting from them in the new environment at the change moment. Because in a dynamic environment there are cyclic behaviors, it is probable for a previous optimum which has been appeared in a past generation, to be a good solution in current generation.

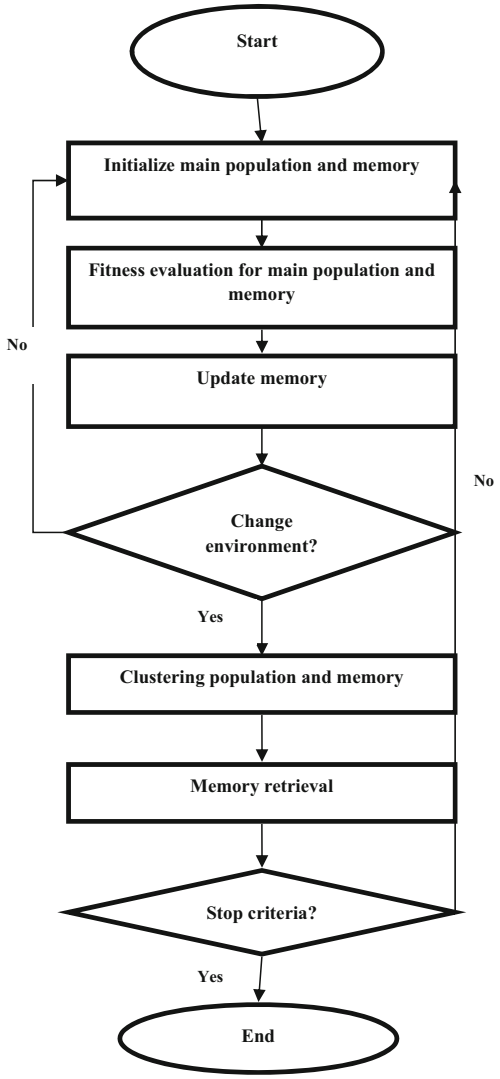

Fig. 2 Describing the proposed method by flowchart

Employment of a memory raises seven problems/questions (including four challenging problems and three approximately easy problems) to be managed. The three approximately easy problems are: (Q1) "When should we update the memory?", (Q2) "How many individuals (bees) should be swapped between population (colony) and memory when it is decided to update population (colony) and memory? (either from population to memory or vice versa)", and (Q3) "When should we update population (colony) from the memory?". The four challenging problems are: (Q4) "Which individual(s) (bee(s)) of population (colony) should be selected to be placed in memory when it is decided to update the memory?", (Q5) "Which element(s) of memory should be removed when it is decided to update the memory?", (Q6) "Which element(s) of memory should be selected to be placed in population (colony) when it is decided to update the population (colony) from the memory?", and finally (Q7) "Which individual(s) (bee(s)) of population (colony) should be removed when it is decided to update population (colony) from the memory?".

So employment of a memory can be useful. Before we go forward, we define some keywords.

Memory updating time The moment that memory is updated from population is named *Memory Updating Time* (*MUT*) throughout all this paper.

Population element to be saved in memory A population (colony) element that is needed to be saved in memory at *MUT* is named a *Population Element to be saved in Memory* (*PEM*) throughout all this paper. It is very important to note that it is possible that there is more than one individual that is of type *PEM*. If kth individual is of type *PEM*, we will show it by $x_k \in PEM$ where x_k indicates kth population individual.

Element to be deleted from memory An element that is needed to be deleted from memory at *MUT* is named an *Element to be deleted from Memory* (*EM*) throughout all this paper. It is very important to note that it is possible that there is more than one memory individual that is of type *EM*. If kth memory individual is of type *EM*, we will show it by $mem_k \in EM$ where mem_k indicates kth memory individual.

Number of individuals transferred from population to memory The number of individuals (bees) that are needed

to be transferred from population (colony) into memory at *MUT* is named *Number of individuals transferred from Population to Memory* (*NPM*) throughout all this paper.

Updated size The *NPM* is also named *Updated Size* (*UdS*) throughout all this paper.

Population updating time The moment that population (colony) is updated from memory is named *Population Updating Time* (*PUT*) throughout all this paper.

Number of individuals transferred from memory to population The number of individuals (bees) that are needed to be transferred from memory into population (colony) at *PUT* is named *Number of individuals transferred from Memory to Population* (*NMP*) throughout all this paper.

Updating size The *NMP* is also named *Updating Size* (*UgS*) throughout all this paper.

Memory element to be saved in population A memory element that is needed to be saved in population (colony) at *PUT* is named *Memory Element to be saved in Population* (*MEP*) throughout all this paper. It is very important to note that it is possible that there is more than one individual that is of type *MEP*. If kth individual is of type *MEP*, we will show it by $x_k \in MEP$.

Element to be deleted from population An element that is needed to be deleted from population (colony) at *PUT* is named an *Element to be deleted from Population* (*EP*) throughout all this paper. It is very important to note that it is possible that there is more than one population individual that is of type *EP*. If kth memory individual is of type *EP*, we will show it by $x_k \in EP$ where x_k indicates kth population individual.

Cluster in memory A number of memory individuals (bees) that are similar to each other and construct a concentrated group are considered as a *Cluster in Memory* (*CM*) throughout all this paper. Indeed a *CM* is a set containing a number of similar individuals in memory.

Cluster in population A number of population (or colony) individuals (or bees) that are similar to each other and construct a concentrated group are considered as a *Cluster in Population* (*CP*) throughout all this paper. Indeed a *CP* is a set containing a number of similar individuals in population (colony).

Cluster size in memory The number of clusters defined in the memory is named *Cluster Size in Memory* (*CSM*) throughout all this paper.

Cluster size in population The number of clusters defined in the population (colony) is named *Cluster Size in Population* (*CSP*) throughout all this paper.

Cluster center in memory Each cluster defined in the memory has a central assumptive element that is named *Cluster Center in Memory* (*CCM*) throughout all the paper. The jth feature of ith *CM* is denoted by $CCM_i(j)$ and is defined according to (9).

$$CCM_i(j) = \frac{1}{|CM_i|} \sum_{mem_k \in CM_i} mem_k(j) \tag{9}$$

where CM_i is ith *Cluster in Memory*, $|CM_i|$ is the number of memory individuals (bees) in the ith *Cluster in Memory*, and mem_k is kth memory individual.

Cluster center in population (colony) Each cluster defined in the population (colony) has a central assumptive element that is named *Cluster Center in Population* (*CCP*) throughout all the paper. The jth feature of ith *CP* is denoted by $CCP_i(j)$ and is defined according to (10).

$$CCM_i(j) = \frac{1}{|CP_i|} \sum_{x_k \in CP_i} x_k(j) \tag{10}$$

where CP_i is ith *Cluster in Population*, $|CP_i|$ is the number of population individuals in the ith *Cluster in Population*, and x_k is kth population individual.

All of the seven questions mentioned above should be responded in a right manner. If we don't control the above questions properly, it will be highly likely that the memory usage is more harmful than useful. Now we answer seven questions respectively.

(A1) Assume MUT_i be ith *MUT*. The first *MUT* is assumed to be zero at cycle zero; i.e. $MUT_0 = 0$. Also assume ϑ_i be equal to $MUT_{i+1} - MUT_i$. The ϑ_i is also a random integer number between $[L_\vartheta, U_\vartheta]$. So for obtaining MUT_{i+1}, we first compute ϑ_i and then add it to MUT_i; as presented in (11).

$$MUT_{i+1} = MUT_i + \vartheta_i \tag{11}$$

(A2) both *UdS* and *UgS* are to set by user as two parameters of algorithm throughout all the paper.

(A3) *PUT* is always done at the time of *MEC* throughout all the paper. It is assumed when environment changes, the population needs to be updated.

(A4) When it is decided to update memory, we should first of all select *UdS* individuals (it is worthy to mention that *UdS* is set by user, so in each *MUT* only *UdS* individuals are chosen) in the population that can be transformed to memory. All *PEM*s should be chosen in such a way that two constraints are satisfied. (1) Qualities of memory

individuals (bees) should be as high as possible, and (2) Diversity among memory individuals (bees) should be guaranteed.

To satisfy constraint (2), the proposed algorithm first assumes that each population individual is a data sample and each dimension is a feature; so we have a dataset. A fast clustering algorithm partitions data samples (population individuals) into CSP clusters, where $CSP = UdS$. The proposed algorithm chooses one population individual per each cluster. The proposed algorithm chooses as an PEM individual for that cluster, the population individual that have the most quality among individuals of that cluster. By selecting the best individual **in each cluster**, the proposed algorithm guarantees constraint (1).

(A5) When it is decided to update memory and after making the clustering of population individuals, in the second phase we should select UdS memory individuals that can be removed (they should be replaced with PEM individuals). All EM individuals should be chosen in such a way that the two constraints mentioned in (A4) are guaranteed to be satisfied. To satisfy constraint (1), the proposed algorithm considers each memory individual as a data sample and each dimension as a feature; so we have a dataset again; we initially place each data sample in a cluster produced in population, i.e. we cluster data samples. The mem_i is placed in the jth cluster if we have $\forall k : \{1, 2, \ldots, UdS\} \cdot |mem_i - CCP_j| \leq |mem_i - CCP_k|$, where CCP_k is the center of kth cluster produced right after clustering task accomplished on the population individuals and $|\cdot|$ indicates a norm function (here in the paper $|a| = (\sum_{i=1}^{p} a_i^2)^{0.5}$). To satisfy constraint (2), the algorithm selects the worst memory element in each cluster as EM in that cluster and it should be replaced with PEM individual in that cluster.

Indeed the algorithm first partitions the population individuals into CP.

$$CP_t^{Ud} = \{i \blacksquare \forall k : \{1, 2, \ldots, UdS\} \cdot |pop_i - CCP_t| \leq |pop_i - CCP_k|\} \tag{12}$$

where $t \in \{1, 2, \ldots, UdS\}$ and CP^{Ud} stands for clustering of populations individuals when it is wanted to update memory and CP_t^{Ud} stands for its tth cluster. Let $Best_i^{Ud} = \arg(\max_{j \in CP_i^{Ud}} f(pop_j))$. Then it places any memory element in the cluster whose center is the nearest. i.e.

$$\begin{aligned} CP_t^{Ud} = \{i \blacksquare \forall k : \{1, 2, \ldots, UdS\} \cdot |mem_i \\ - CCP_t| \leq |mem_i - CCP_k|\} \end{aligned} \tag{13}$$

Let $Worst_i^{Ud} = \arg(\min_{j \in CM_i^{Ud}} f(mem_j))$. Now we have $EM_i = mem_{Worst_i^{Ud}}$ and $PEM_i = pop_{Best_i^{Ud}}$.

(A6) When it is decided to update population, we should first of all select UgS memory elements (it is worthy to mention that UgS is set by user, so in each PUT only UgS memory elements are chosen) in the memory that can be transformed to population. All MEPs should be chosen in such a way that two constraints are satisfied. (1) Qualities of population individuals (bees) should be as high as possible, and (2) Diversity among population individuals (bees) should be guaranteed.

To satisfy constraint (2), the proposed algorithm first assumes that each memory individual is a data sample and considers each dimension as a feature; so we have a dataset. A fast clustering algorithm partitions data samples (memory elements) into CSM clusters, where $CSM = UgS$. The proposed algorithm chooses one memory individual per each cluster. The proposed algorithm chooses the memory individual that has the most quality among individuals of that cluster, as an MEP individual for that cluster. By selecting the best individual **in each cluster**, the proposed algorithm guarantees constraint (1).

(A7) When it is decided to update population and after making the clustering of memory individuals, in the second phase we should select UgS population individuals that can be removed (they should be replaced with MEP individuals). All EP individuals should be chosen in such a way that the two constraints mentioned in (A4) are guaranteed to be satisfied. To satisfy constraint (1), the proposed algorithm considers each population individual as a data sample and each dimension as a feature; so we have a dataset again; we initially place each data sample in a cluster produced over memory individuals, i.e. we cluster data samples. The pop_i is placed in the jth cluster if we have $\forall k : \{1, 2, \ldots, UgS\} \cdot |pop_i - CCM_j| \leq |pop_i - CCM_k|$, where CCM_k is the center of kth cluster produced right after clustering task accomplished on the memory individuals. To satisfy constraint (2), the algorithm selects the worst population element in each cluster as EP in that cluster and it should be replaced with MEP individual in that cluster.

Indeed the algorithm first partitions the memory individuals into CM.

$$CM_t^{Ug} = \{i \blacksquare \forall k : \{1, 2, \ldots, UgS\} \cdot |mem_i - CCM_t| \leq |mem_i - CCM_k|\} \tag{14}$$

where $t \in \{1, 2, \ldots, UgS\}$. Let $Best_i^{Ug} = \arg(\max_{j \in CM_i^{Ug}} f(mem_j))$. Then it places any population element in the cluster whose center is the nearest. i.e.

$$CP_t^{Ug} = \{i \blacksquare \forall k : \{1, 2, \ldots, UgS\} \cdot |pop_i - CCM_t| \leq |pop_i - CCM_k|\} \tag{15}$$

Let $Worst_i^{Ug} = \arg(\min_{j \in CM_i^{Ug}} f(pop_j))$. Now we have $EP_i = pop_{Worst_i^{Ug}}$ and $MEP_i = mem_{Best_i^{Ug}}$.

Algorithm 5 Proposed algorithm

Input: *N, SN, D, MCN, Limit,* x_j^{min}, x_j^{max}, *mem_size*
Output: *BEST Solution, BEST Fitness, offline error*

1. Begin
2. *initialize x and mem % x is population and mem is memory population%*
3. $f_i = fit(x_{:i})$ *% f_i is fitness for population%*
4. $m_i = fit(mem_i)$ *% m_i is fitness for memory population%*
5. $update\_mem = 1$
   $update\_time = rand(5, 10)$
6. $cycle = 1$

Repeat:

7. *update_mem:*
   $update\_mem = 0$
   $mem = update\_\boldsymbol{memfunction}(mem, x)$
   $update\_time = rand(5, 10) + cycle$
8. $cx = update\_\boldsymbol{popfunction}(x)$*% cx is current x and* $\boldsymbol{popfunction}(x)$ *is ABC algorithm%*
9. $cf_i = fit(x_{:i})$*% fitness for current x%*
10. $if cf_i \neq f_i$ *then chang_flag* $= 1$ *% change detected%*
11. $chang\_flag$: % reuse memory%
    1- $\mathbf{m_i} = \boldsymbol{ft}(\boldsymbol{mem}_{:i})$, $\boldsymbol{f}_i = \boldsymbol{fit}(\boldsymbol{x}_{:i})$
    2- $\boldsymbol{mq}_m = \boldsymbol{cluster}(\boldsymbol{memp})$ *%p is number of clusters %*
    3- Select $j_r \in \boldsymbol{mq}_r \forall \boldsymbol{r} \in \{\boldsymbol{1} \ldots \boldsymbol{p}\}$
    Subject to $\boldsymbol{m}_{j_r} \leq \boldsymbol{m}_l \forall \boldsymbol{l} \in \boldsymbol{q}_r$
    4- $\boldsymbol{xq}_x = \boldsymbol{cluster}(\boldsymbol{x}, \boldsymbol{p})$
    5- Select $\boldsymbol{d}_r \in \boldsymbol{xq}_r \forall \boldsymbol{r} \in \{\boldsymbol{1} \ldots \boldsymbol{p}\}$
    Subject to $\boldsymbol{f}_{dr} \geq \boldsymbol{f}_l \forall \boldsymbol{l} \in \{\boldsymbol{1} \ldots \boldsymbol{p}\}$
    6- $\boldsymbol{x}_{d_r} = \boldsymbol{mem}_{j_r} \forall \boldsymbol{r} \in \{\boldsymbol{1} \ldots \boldsymbol{p}\}$
12. *if update_time* $\geq$ *cycle then update_mem* $= 1$
13. $cycle = cycle + 1$
14. *Until cycle = MCN*
15. *End*

Form (5): The pseudo code of the proposed algorithm

One of the main challenges in dynamic environments is to maintain the diversity when implementing the algorithm and in this method the diversity has been maintained stable in the population using clustering of memory and the population. In this method the memory and the population in the first phase are initialized. Then memory and the population are clustered after being initialized based (12) and (13). Each bee needs to be placed in his cluster for insertion and retrieval. Clustering is an unsupervised learning field in which the samples are divided into a series of clusters as the samples in a cluster are similar and they are different than the samples in other clusters. There are many criteria to measure the similarities between any pair of samples. One of them is Euclidean distance between the two samples. Similar samples have lower Euclidean distance and they are placed in a cluster. This clustering is called *distance-based clustering*. One of the *distance-based clustering* is k-means clustering. In k-means clustering the center of a cluster is the average amount of data within each cluster. Then the data are clustered based on the Euclidean distance to cluster centers. This method in each iteration improves the inside cluster changes of the model which is done by estimating the new cluster center in iteration phase. In this way the data are allocated to different clusters based on updated average. This work will continue until the center of the cluster is fixed and the value will remain unchanged in successive iterations and the clusters are stable. Pseudo-code of the proposed method is based on Form (5).

As mentioned previously in order to update the memory an alternative strategy should be used. The pseudo-code of the update function in the memory is presented in Form (6). In this pseudo-code it is mentioned that if there are two similar persons (with the closest distance to each other) exist in the memory the person with lower efficiency is removed from the memory and the person with higher efficiency remains in the memory.

Algorithm 6 *memfunction*(mem, x)

Inputs: M: set of solution in the memory; s_{new} : new good solution found.
Outputs: $\emptyset$.
function
{

1. $dist\ (s, s')$ *and* $\forall\ s \in M | s' \in M$
2. ***if*** $dist\ (s, s')$ *is minimum*
   if $fitness(s, s_{new}) \leq fitness(s', s_{new})$
   then s is removed from memory
   else
   s′ is removed from memory
3. ***end of if***
4. *s_new replaced with worst solution*

}

Form (6): The pseudo code of the memory update

5.2 Descriptions of the proposed algorithm pseudo code

At the beginning of the algorithm the required parameters are defined. In Phase 2 the population must be created. To initialize the population (16) and (17) are used. Initializing

population for the base and memory population is done based on (16) and (17).

$$pop_j(i) = LB_i + r_n(UB_i - LB_i), \quad r_n \in [0,1] \\ \forall j \in (1,2,\ \ldots, |pop|), \forall i \in (1,2,\ \ldots, N) \tag{16}$$

$$mem_j(i) = LB_i + r_n(UB_i - LB_i), \quad r_n \in [0,1] \\ \forall j \in (1,2,\ \ldots, |mem|), \forall i \in (1,2,\ \ldots, N) \tag{17}$$

where $|mem|$ and $|pop|$ are memory and population size. In step 3 and 4 based on efficiency function, competence for each individual memory is calculated by the base or memory population. In phase 6 it becomes clear that memory update is done and a randomized range of rand (5,10). If the memory update time occurs in cycle iteration then the next update for memory is in $Rand(5,10) + cycle$. The algorithm cycle begins at stage 6. Steps 7 and 8 state the function of updating for the bees and memory population (in step 8 popfunction(x) is ABC algorithm (algorithm 4)). In step 9 the reassess is performed to calculate the efficiency of the bees and if the efficiency is changed even for a single bee, we understand that the environment has changed. Step 11 based on the changes in the environment, the data stored in the memory should be applied foe the new environment which is done in 6 various phases as follows:

1. Efficiency is calculated for the base and memory population.
2. Memory population is clustered.
3. The best person in each cluster of the population is selected and the best person in this population is the one who has the best efficiency.
4. The based population is clustered.
5. The worst person in each base population cluster is selected (the worst individual from each cluster is the least efficient one).
6. The worst person in each cluster is replaced by the best individual from each cluster in the base population.

Step 12 indicates that if update time is greater than the algorithm cycle, we activate update for active memory and otherwise one unit is added to the cycle and finally when we get to the cycle stop condition we leave the cycle and reach the end of algorithm.

Example Figure 3 shows an example that helps to explain the proposed approach. As it is presented in Fig. 3, the problem space is divided into three clusters and in each cluster, the position of the bees is defined by a small star, the center of cluster is defined by the plus, the position of the memory is determined by a circle, the center of cluster of memory population is defined by the square, the center of each peak is marked by a diamond and the new center of the peal is marked by a bug star sign. As discussed before the worst individual of the population is the one who has the greatest distance from the center of the peak and in fact it can be said that the worst person has the least efficiency. If the environment change occurs and the peak center is changed then the efficiency of the members can be changed.

The closest memory to the new peak is considered as the best memory and this memory after the change and displacement of the optimum peak can lead members of the population toward the new peak. The question that arises here is that how a memory understands is close or far from the new peak center. In response to this question, we can say that, if the efficiency of a memory after the change of environment is increased the memory understands that it is closer to the peak and if the efficiency is lowered the memory understands that is far from it. The best individual of the population is the closest one to the center of the peak center.

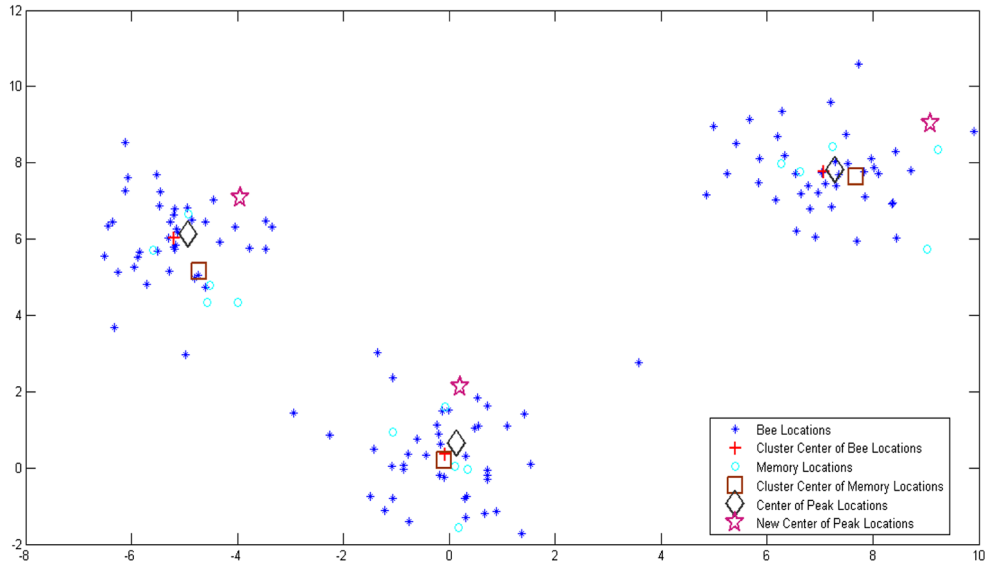

Fig. 3 Describing the proposed method by image

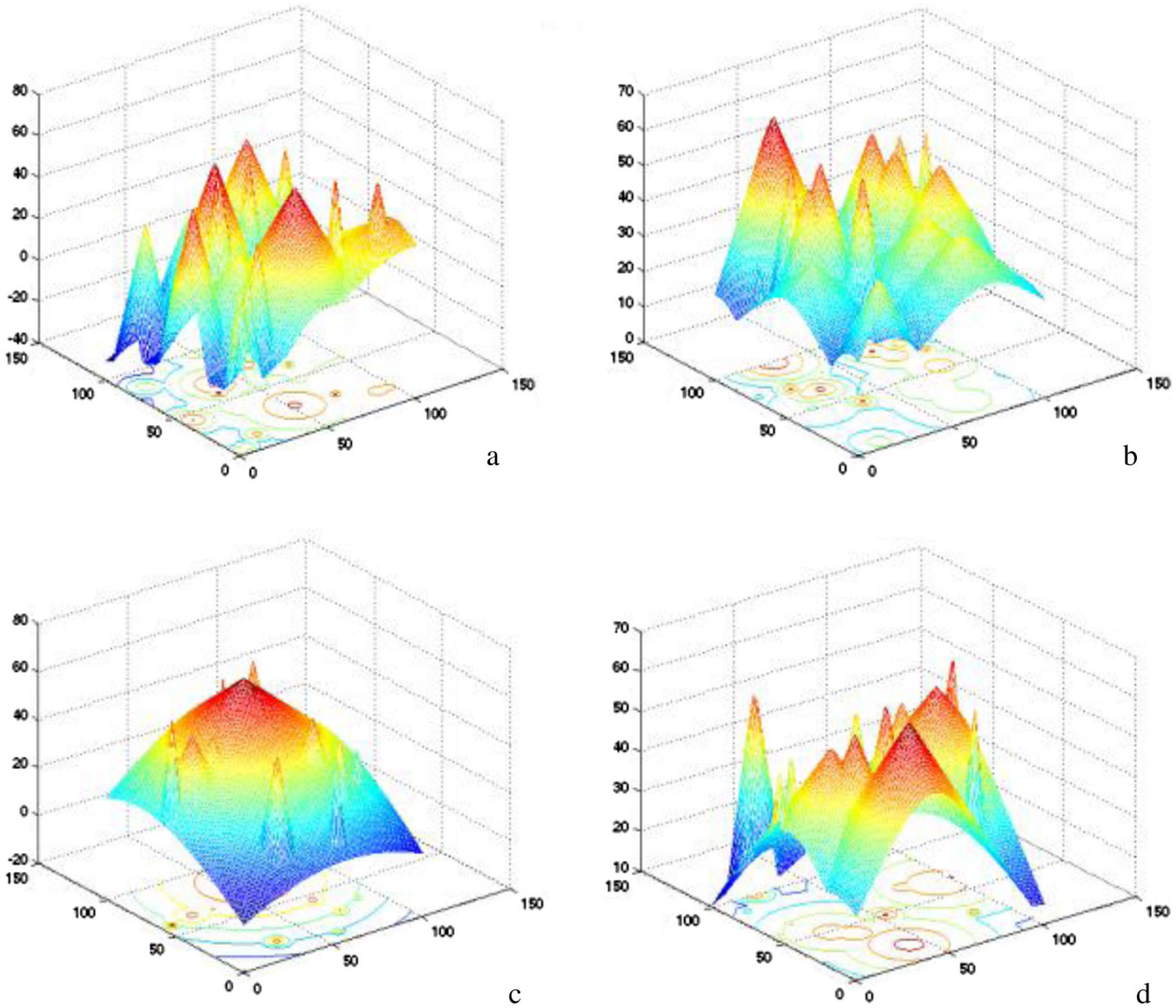

Fig. 4 The changes in the peaks in MPB function

Table 1 The standard configuration of the parameters for MPB

Parameter	Value
peaks (number of peaks)	10
Change frequency (U)	5000
Height severity	7.0
Width severity	1.0
Peak shape	Con
Basic function	No
Shift length s	1.0
Number of dimensions (D)	5
Correlation coefficient (λ)	0
S	[0, 100]
H	[30.0, 70.0]
W	[1, 12]
I	50.0

6 Tests and results

In order to perform the tests on the proposed algorithm and its comparison with other algorithms in a dynamic

Table 2 The proposed algorithm parameters

Parameter	Value
Lower bound	0
Upper bound	100
Total population	100
Memory Size	10
Number of Food	50
Limit	0.2
Number of main population clusters	2.0
The number of population clusters Momery	2.0

Table 3 Average Offline Errors for Different Algorithms on the MPB with Different Shift Severities

Algorithm	Shift Severity (s)						
	0	1	2	3	4	5	6
Proposed algorithm	**0.0853**	**0.0995**	**0.1473**	**0.8629**	**0.9933**	**1.017**	**1.1096**
CPSO [26]	0.465	0.715	0.843	0.911	0.997	1.08	1.23
mQSO [27]	1.17	1.75	2.40	3.0	3.59	4.24	4.79
rSPSO [28]	0.74	1.50	1.87	2.4	2.90	3.25	3.87
ESCA [29]	1.72	1.53	1.57	1.67	1.72	1.78	1.79
CESO [30]	0.58	1.38	1.78	2.03	2.23	2.52	2.74
mCPSO [31]	1.18	2.05	2.80	3.57	4.18	4.89	5.53
SPSO [32]	0.95	2.51	3.78	4.96	2.56	6.76	7.68
CGAR [33]	1.48	2.62	2.76	2.96	3.16	3.46	3.8
CDER [33]	2.56	2.52	7.47	8.62	9.81	10.7	11.4
PSO–CP [34]	0.87	1.31	1.98	2.21	2.61	3.20	3.93
CPSOR [33]	0.418	0.599	0.849	0.964	1.38	1.69	2.07

Bold means the best performance

environment Moving Peaks Benchmark [20] is used to test the efficiency of the proposed method.

6.1 Moving peaks benchmark problem

MPB problem [25] is a good simulator for simulating the dynamic environment. This problem includes m peaks in a dimension space of the size n with real value parameters and the heights, widths and positions of peaks may change over time. MPB function is formulated based on (18):

$$F(\vec{x}, t) = max(B(\vec{x}), \max_{i=1...m} P(\vec{x}, h_i(t), w_i(t), \vec{p}_i(t))) \tag{18}$$

Table 4 Average offline errors for different algorithms on the MPB with different numbers of peaks, where the suggested configuration for the framework and the default settings for the MPB available in Table 1

Algorithm	Peaks number									
	1	2	5	7	10	20	30	50	100	200
Proposed algorithm	**0.0595**	**0.0645**	**0.0635**	**0.0952**	**0.0995**	**0.1938**	**0.1871**	**0.2842**	**0.6922**	**0.5434**
CGAR	2.02	1.88	2.56	2.98	2.62	3.66	3.12	3.26	2.68	2.39
CDER	0.903	2.6	8.02	6.74	5.52	7.49	5.51	5.79	4.12	3.71
CPSO	2.29	0.005	0.361	0.675	0.715	1.18	1.34	1.42	1.09	0.955
mCPSO	4.93	3.36	2.07	2.11	2.08	2.64	2.63	2.65	2.49	2.44
mQSO	5.07	3.47	1.81	1.77	1.80	2.42	2.48	2.50	2.26	2.36
mCPSO*	4.93	3.36	2.07	2.11	2.05	2.95	3.38	3.68	4.07	3.97
mQSO*	5.07	3.47	1.81	1.77	1.75	2.74	3.27	3.65	3.93	3.86
CESO	1.04	–	–	–	1.38	1.72	1.24	1.45	1.28	–
rSPSO	1.42	1.10	1.04	1.21	1.50	2.20	2.62	2.72	2.93	2.79
SPSO	2.64	2.31	2.15	1.98	2.51	3.21	3.64	3.86	4.01	3.82
ESCA	0.98	–	–	–	1.54	1.89	1.52	1.67	1.61	–
PSO – CP	3.41	–	–	–	–	1.31	202	2.14	2.04	–
HmSO [35]	0.87	–	1.18	–	1.42	1.5	1.65	1.66	1.68	1.71
RVDEA [36]	1.02	–	–	–	3.54	3.87	3.92	3.87	3.37	3.54
FMSO [37]	3.44	–	2.94	–	3.11	3.36	3.28	3.22	3.06	2.84
Cellular PSO [38]	2.55	–	1.68	–	1.78	2.60	2.93	3.26	3.41	3.40
rPSO [39]	0.56	–	12.58	–	12.98	12.79	12.35	11.34	9.73	8.90
Adaptive mQSO [40]	0.51	–	1.01	–	1.51	2.00	2.19	2.43	2.68	2.62

Bold means the best performance

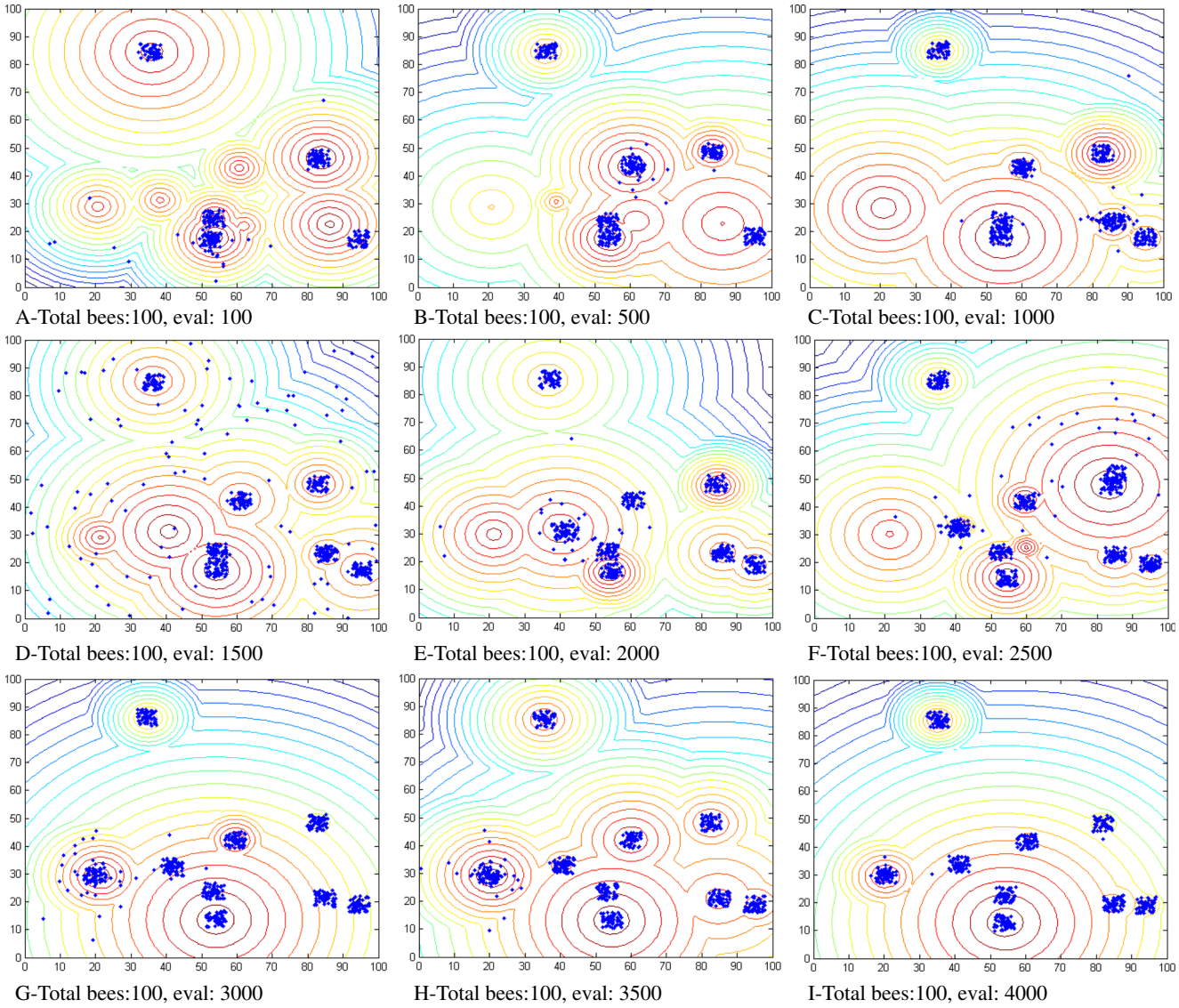

Fig. 5 Cover of the peaks via bees for 10 peaks and the default setting of the MPB after different numbers of the fitness evaluations

where, $B(\vec{x})$ is the base value of the environment that is independent of time and P is a function that defines the shape of the peak, that each of m peaks has their time variable parameters, height (h), width (w), and their own position (p). In each ΔE fitness evaluations, height, width and position of each peak are changed. The height and width of each peak changes by adding a Gaussian random variable. The change frequency parameter indicates when the environment is changed or when the algorithm must respond to changes in the environment. Moving

Table 5 Results of the proposed method vs. the state-of-the-art methods for different dimensions when number of the peaks is 10, frequency of changes is 5000 and shift severity is 1

Algorithm	Dimension						
	2	3	4	5	10	15	20
Proposde method	**0.0485**	**0.0643**	**0.0731**	**0.0995**	**0.1517**	**0.2625**	**0.3547**
Adaptive mQSO	0.71	1.16	1.33	1.51	3.37	4.91	5.83
mQSO	1.01	1.49	1.47	1.85	4.22	6.50	8.88
rPSO	2.62	6.61	10.43	12.98	16.87	18.48	18.48
mPSO	1.24	1.42	1.35	1.61	4.32	7.07	10.77

Bold means the best performance

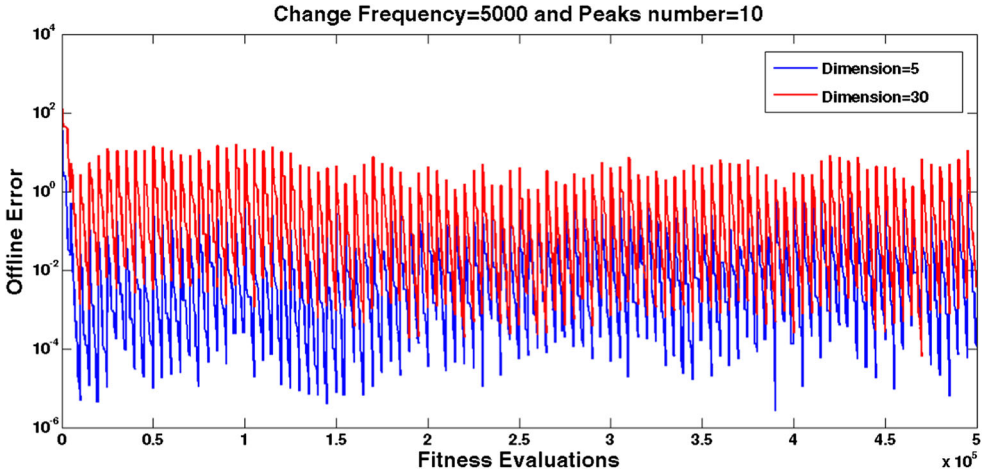

Fig. 6 Diagrams of the offline error for the proposed method with dimension sizes 5 and 30 in frequency change of 5000 with 10 peaks

Peaks Benchmark function has various parameters that by changing each one of these parameters, the problem nature can be changed. Figure 4 presents the change of the peaks in this problem with multiple peaks.

The parameter s controls the amount of variation, ΔE determines the frequency of changes, the parameter λ determines how the position of a peak is changed based on its previous motion.

If $\lambda = 0$, every motion can be random and if $\lambda = 1$, peaks can move in a determined path. Whenever a change occurs in the environment this change is mention on the location, height and width of a peak as the equations mentioned in (19).

$$\begin{aligned} h_i(t) &= h_i(t-1) + height_{severity} \cdot \sigma \\ w_i(t) &= w_i(t-1) + width_{severity} \cdot \sigma \\ \vec{p}_i(t) &= \vec{p}_i(t-1) + \vec{v}_i(t) \\ \sigma &\in N(0,1) \end{aligned} \tag{19}$$

Transmission vector $\vec{v}_i(t)$ combines a random vector $\vec{r}_i$ with the previous transmission vector $\vec{v}_i(t-1)$. The random vector $\vec{v}_i(t)$ is generated by producing the random numbers in [0,1] for each dimension and normalizing it to a length of s. Vector $\vec{v}_i(t)$ can be created based on the previous change where the position of the peaks is aligned to the previous changes or it is created randomly which changes the position of the peaks and they would have no dependence to the last change. Vector $\vec{v}_i(t)$ is calculated based on (20):

$$\vec{v}_i(t) = \frac{S}{|\vec{r} + \vec{v}_i(t-1)|}((1-\lambda)\vec{r} + \lambda\vec{v}_i(t-1)) \tag{20}$$

Peak function for height, width and position of each peak is calculated based on (21).

$$P(\vec{x}, h(t), w(t), \vec{p}(t)) = h(t) - w(t).\sqrt{\sum_{j=1\ldots n}(x_j - p_j)^2} \tag{21}$$

The part related to the radical mentions the distance between a point and a peak position.

Numerical experiments concerning the Moving Peaks Benchmark, scenario 2, as proposed by Branke (2001) were performed in order to test behavior of the proposed method. The default settings and definition of the benchmark used in the experiments of this paper can be found in Table 1. Parameter settings for the proposed algorithm are presented in Table 2.

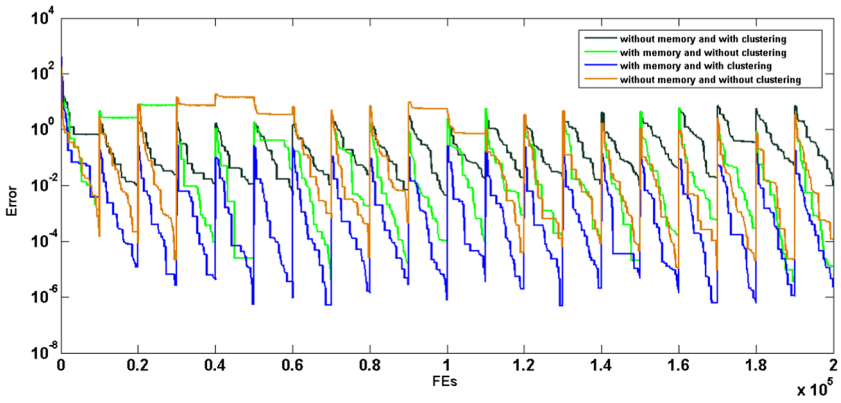

Fig. 7 Effect of the memory and clustering on the proposed algorithm

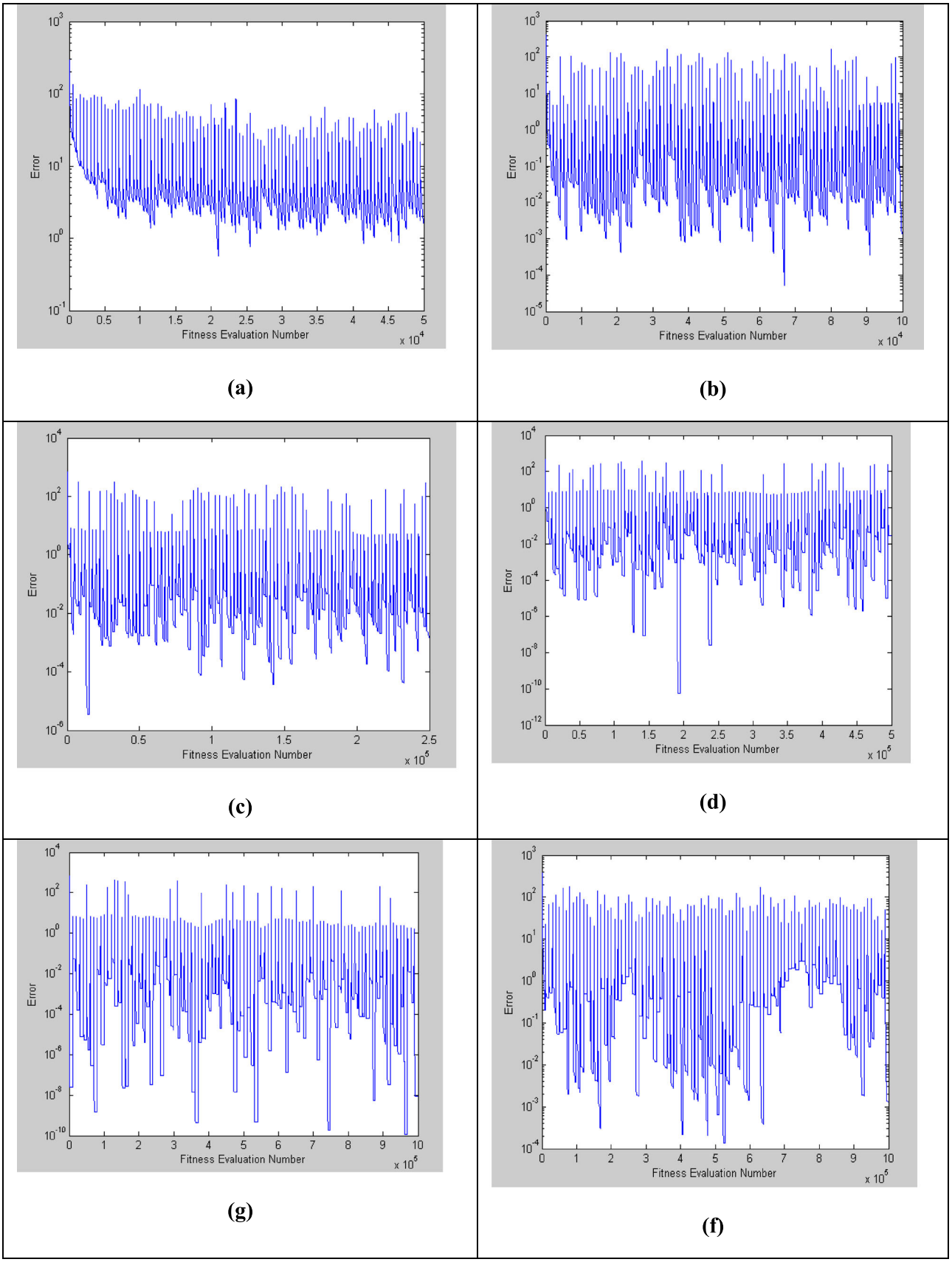

Fig. 8 Offline error of the proposed algorithm when change frequency is 500 (**a**), 1000 (**b**), 2500 (**c**), 5000 (**d**), 10000 (**e**), and 15000 (**f**)

6.2 Varying shift severity

The shift severity parameter of the MPB controls the severity of the change in height, width and position of peaks. From Table 3, it can be seen that the results achieved by the proposed algorithm are much better than the results of the other 11 state-of-the-art algorithms on the MPB problems with different shift severity. As we know, the peaks are more and more difficult to track with the increasing of the shift severity. Of course, the performance of any optimization algorithm degrades when the shift severity increases. However, the offline error of the proposed algorithm is better than the other 11 state-of-the-art algorithms. These results indicate the proposed algorithm to adapt better than others algorithm to more severe changes in the landscape.

6.3 Varying number of peaks

Table 4 presents the experimental results in terms of the offline error for 19 algorithms, where the results of the other 18 state-of-the-art algorithms are provided by the corresponding papers with their optimal configuration that enables them to achieve their best performance. In Table 4, mCPSO* and mQSO* denote mCPSO without anti-convergence and mQSO without anti-convergence, respectively. From Table 4, it can be seen that the performance of the proposed algorithm is not influenced too much when the number of peaks is increased. Usually, increasing the number of peaks makes it harder for algorithms to track the optima. However, the offline error is less when the number of peaks is larger than 50 for the proposed algorithm.

6.4 Coverage of bees around peaks

Figure 5 shows the distribution of bees around peaks that is drawn in 2 dimensions for 10 peaks and the default setting of the MPB after different numbers of the fitness evaluations.

6.5 Effect of the dimension size on the proposed algorithm

Table 5 shows the results of the proposed method with different dimensions when the number of peaks is 10, the change frequency is 5000 and shift severity is 1; it also represents the same results for mQSO, Adaptive mQSO, rPSO and mPSO. The results presented in Table 5 shows "the more dimensions of the landscape, the better the performance of the proposed algorithm comparing to the other algorithms". Figure 6 presents the diagrams of the offline error for the proposed method when number of dimensions is 5 or 30, the frequency of changes is 5000 and the number of peaks is 10.

6.6 Effect of memory and clustering

Memory and clustering are of those facilities that are able to significantly improve both of the convergence time and the performance metric of dynamic optimizers. In previous sections, we have discussed about clustering in details. In this subsection, the effect of these two concepts is evaluated on dynamic optimization paradigm.

To determine how effective these two concepts are, the offline errors for four paradigms have been produced: With-Memory-withOut-Clustering (WMOC), withOut-Memory-With-Clustering (OMWC), withOut-Memory-withOut-Clustering (OMOC), and With-Memory-With-Clustering (WMWC). These results are depicted in Fig. 7. The parameters of these results are the same mentioned as default parameters.

6.7 Effect of frequency change on offline error

In this section, the offline error is plotted for different frequency changes (i.e. 500, 1000, 2500, 5000, 10000, and 15000) when there are 10 peaks, 5 dimensions, and parameter s is set to 1.

Table 6 The parameters employed for the sate-of-the-art methods

Algorithm	Setting
PCAFSA	Population size (Parent) = 2, Population size (Best child) = 2, Population size (Non-best child) = 2, Swarm number(Parent) = 2, Swarm number(Best child) = 1, Swarm number(Non-best child)= $\frac{N}{A}$, Try number(Parent) = 4, Try number(Best child) = 10, Try number (Non-best child) = 2, Initial visual(Parent) = 25, Initial visual (Best child) = 1× Shift severity, Initial visual(Non-best child) = 25
Adaptive-SFA	*Population size* = 100, $\alpha = 0.01$, $\gamma = 1$, num_seq_iteration = 2, *min_activating_discoverer* = 2, $w_{min} = 0.6$, $w_{max} = 0.9$, $r_{cloud} = 0.2 \times severity$
Mohammadpour	*Population size* = 100, *Memory Size* = 10, *Probability of Crossover* = 0.6, *Probability of Mutation* = 0.2, *Logistic factor* $(A) = 4$
CMBGA	*Population size* = 100, *DE scheme* = *DE/rand/1/exp*, $F = 0.5$, $w = 0.729844$, $v_{clamp} = [-50,50]$, $r_s = 0.5$
mNAFSA	*Population size* = 100, *Try number* (*Parent*) = 4
Multi-pop-ABC	Maximum number of iterations (*MaxIt*) = 50,000 fitness evaluations , Population size (*Ps*) = 100, Limit parameter (*Lit*) = 30, Change strength threshold (T_v) = 0.05

Table 7 Comparison of the proposed method with some of the sate-of-the-art methods

Algorithm	Number of peaks						
	1	5	10	20	30	50	100
proposed method	0.692(0.01)	0.284(0.01)	**0.187**(0.00)	**0.193**(0.00)	**0.099**(0.02)	**0.095**(0.01)	**0.063**(0.00)
PCAFSA	0.33(0.01)	0.48(0.01)	0.65(0.02)	1.03(0.02)	1.46(0.02)	1.53(0.03)	1.60(0.02)
Adaptive-SFA	0.66(0.05)	0.79(0.06)	0.95(0.05)	1.29(0.07)	1.51(0.06)	1.71(0.04)	1.84(0.04)
Mohammadpour	0.92(0.09)	1.06(0.7)	1.15(0.10)	1.18(0.06)	1.35(0.05)	1.65(0.07)	1.80(0.06)
CDEPSO	1.02(0.14)	0.99(0.15)	1.75(0.10)	1.93(0.11)	2.28(0.10)	2.74(0.10)	2.84(0.12)
mNAFSA	0.38(0.06)	0.55(0.04)	0.90(0.03)	1.25(0.06)	1.47(0.05)	1.68(0.05)	1.83(0.05)
Multi-pop-ABC	0.14(0.00)	**0.20 (0.00)**	0.22 (0.01)	0.35 (0.00)	0.46 (0.00)	0.44(0.01)	0.52 (0.00)

Bold means the best performance

Figure 8a depicts the offline error curve of the proposed method when the frequency change is 500 and the number of the fitness evaluations is 50000 (it means that 100 changes occur). The number of peaks is 10 in all experiments of this section. Figure 8b depicts the offline error curve of the proposed method when the frequency change is 1000 and the number of the fitness evaluations is 100000 (it means that 100 changes occur again here). Figure 8c, d, e and f depict the offline error curves of the proposed method when the frequency changes are 2500, 5000, 10000, and 15000 respectively and the numbers of the fitness evaluations are 250000, 500000, 1000000, and 1500000 (it means that 100 changes occur in all of these cases).

6.8 Comparing with the state-of-the-art methods

In this section we have compared our method with 6 state-of-the-art methods including PCAFSA [41], Adaptive-SFA [42], CDEPSO [43], mNASA [44], Multi-pop-ABC [45], and CMBGA [46] in terms of the offline and standard errors. The employed parameters for these methods have been extracted based on their papers and have been presented in Table 6. The experimental comparison of the proposed method with the algorithms mentioned in Table 6 has been presented in Table 7. The results indicate that by increasing the number of the peaks, our method becomes more superior to state-of-the-art methods.

Effect of λ value on the offline error of the proposed method when the change frequency is 500 and there are 10 peaks has been investigated here. The experimental results presented in Table 8 indicate as the value of the parameter λ increases, our method performs better. It has been predictable, because by increasing λ value, the randomness in the movements of the peaks becomes more limited.

The memory size is the last parameter that should be investigated in our experimentations so as to find out what value is its best option. This parameter plays an important role in effectuality of the proposed method. Figure 9 depicts the proposed method efficacy in terms of the number of peaks when change frequency is 5000 for memory sizes 10, 20, 30, and 40. The best value for memory size is 10 according to these results.

The effect of the population size on performance of the proposed method in terms of average offline errors has been examined in Table 9.

Table 8 Effect of λ value on the offline error when the change frequency is 500 and there are 10 peaks

Offline error	λ value
0.887(0.00)	0
0.650(0.00)	0.2
0.521(0.00)	0.4
0.401(0.00)	0.6
0.280(0.00)	0.8
0.187(0.00)	1.0

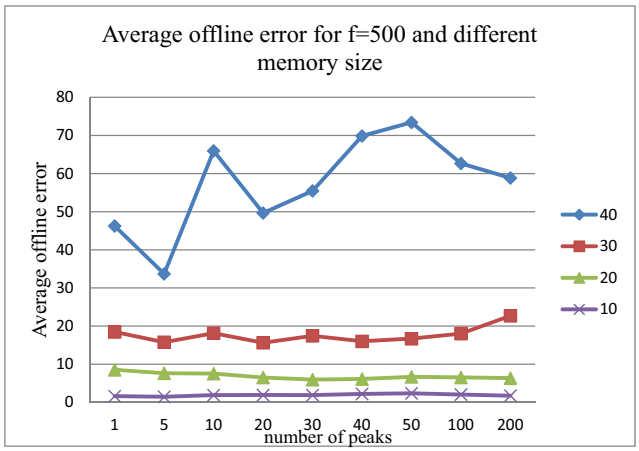

Fig. 9 The proposed method efficacy in terms of the number of peaks when change frequency is 5000 for memory sizes 10, 20, 30, and 40

Table 9 Average offline error of the proposed method of the proposed method with different population sizes and default parameters

Population size	Average offline error of the proposed method
50	0.687(0.03)
100	**0.187**(0.00)
150	0.320(0.05)
200	0.591(0.04)
250	0.683(0.06)
300	0.916(0.09)

Bold means the best performance

The results presented in Table 9 indicate the best population size is no more than 150 and no less than 50. It can be 100 according to the results of Table 9. Due to two reasons: (a) if the population size is a large value, the algorithm should accomplish many fitness function evaluations in one generation; and (b) if population is very small, then the algorithm exploration is not guaranteed

7 Conclusions and future work

Intelligent optimization algorithms in dynamic environments should be designed in such a way that they could track the optima efficiently. In this article we use a combination of the memory and artificial bee colony algorithm to maintain good solutions. We increase algorithms' efficiency by clustering of population and memory individuals. We employed an appropriate strategy to maintain diversity in population. We experimentally have shown that the proposed algorithm completely outperforms the state-of-the-art algorithms in terms of convergence speed. Usage of the proposed algorithm with chaos theory can be a good idea for future works.

Acknowledgements We want to be thankful of Yasooj Branch, Islamic Azad University, Yasooj, Iran, for supporting this research.

References

1. Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. Proc Congr Evol Comput 3:1875–1882
2. Yang S (2006) Associative memory scheme for genetic algorithms in dynamic environments. In: Proceedings of EvoWorkshops: Appl. Evol. Comput., LNCS 3907, pp 788–799
3. Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. IEEE Trans Evol Comput 12(5):542–561
4. Cobb HG, Grefenstette JJ (1993) Genetic algorithms for tracking changing environments. In: Proceedings of the 5th international conference on genetic algorithms, pp 523–530
5. Grefenstette JJ (1992) Genetic algorithms for changing environments. In: Proceedings of the 2nd international conference on parallel problem solving from nature, pp 137–144
6. Yang S (2008) Genetic algorithms with memory and elitism-based immigrants in dynamic environment. Evol Comput 16(3):385–416
7. Ramsey CL, Grefenstette JJ (1993) Case-based initialization of genetic algorithms. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann, pp 84–91
8. Louis SJ, Xu Z (1996) Genetic algorithms for open shop scheduling and re-scheduling. In: Cohen ME, Hudson DL (eds) Proceedings of the eleventh international conference on computers and their applications (ISCA), pp 99–102
9. Yang S, Tinos R (2007) A hybrid immigrants scheme for genetic algorithms in dynamic environments. Int J Autom Comput 3(4):243–254
10. Goldberg DE, Smith RE (1987) Non-stationary function optimization using genetic algorithms with dominance and diploidy. In: Grefenstette JJ (ed) Proceedings of the second international conference on genetic algorithms (ICGA 1987). Lawrence Erlbaum Associates, pp 5968
11. Ryan C (1997) Diploidy without dominance. In: Nordic workshop on genetic algorithms, pp 45–52
12. Ryan C, Alander JT (1997) Dyploidy without dominance. In: Proceedings of the nordic workshop on genetic algorithms, pp 63–70
13. Lewis EHJ, Ritchie G (1998) A comparison of dominance mechanisms and simple mutation on non-stationary problems. In: Schoenauer M, Deb K, Rudolf G, Yao X, Lutton E, Merelo JJJ, Schwefel H-P (eds) Proceedings of the parallel problem solving from nature (PPSN V), vol 1917 of Lecture notes on computer science. Springer, pp 139–148
14. Uyar AS, Harmanci AE (1999) Investigation of new operators for a diploid genetic algorithm. In: Proceedings of SPIE's annual meeting
15. Uyar AS, Harmanci AE (2005) A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. Soft Comput 9(11):803–814
16. Yang S (2006) Dominance learning in diploid genetic algorithms for dynamic optimization problems. In: Keijzer M et al (eds) Proceedings of the eighth international genetic and evolutionary computation. Conference (GECCO 2006). ACM Press, pp 1435–1436
17. Yang S (2007) Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Yang S, Ong Y-S, Jin Y (eds) Evolutionary computation in dynamic and uncertain environments, vol 51 of Studies in computational intelligence, pp 3–28
18. Ramsey CL, Grefenstette JJ (1993) Case-based initialization of genetic algorithms. In: Forrest S (ed) Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann, pp 84–91
19. Louis SJ, Xu Z (1996) Genetic algorithms for open shop scheduling and rescheduling. In: Cohen ME, Hudson DL (eds) Proceedings of the eleventh international conference on computers and their applications (ISCA), pp 99–102
20. Trojanowski K, Michalewicz Z (1999) Searching for optima in non-stationary environments. In: Proceedings of the IEEE congress on evolutionary computation (CEC 1999). IEEE Press, pp 1843–1850
21. Barlow GJ, Smith SF (2008) A memory enhanced evolutionary algorithm for dynamic scheduling problems. In: Springer (ed) Applications of evolutionary computing, vol 4974 of Lecture notes in computer science, pp 606–615
22. Ryan C (1997) Dyploidy without dominance. In: Alander JT (ed) Proceedings of the nordic workshop on genetic algorithms, pp 63–70

23. Bird S, Li X (2007) Using regression to improve local convergence. In: Proceedings of congress on evolutionary computation, pp 592–599
24. Karaboga D, Basturk B (2009) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39:459–471
25. Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: Proceedings of the IEEE congress on evolutionary computation (CEC 1999). IEEE Press, pp 1875–1882
26. Yang S, Li C (2009) A clustering particle swarm optimizer for dynamic optimization. In: Proceedings of congress on evolutionary computation, pp 439–446
27. Blackwell T, Branke J, Li X (2008) Particle swarms for dynamic optimization problems. In: Swarm intelligence. Springer, Berlin, pp 193–217
28. Blackwell TM, Branke J (2006) Multiswarms, exclusion, and anticon vergence in dynamic environments. IEEE Trans Evol Comput 10(4):459–472
29. Lung RI, Dumitrescu D (2010) Evolutionary swarm cooperative optimization in dynamic environments. Nat Comput 9(1):83–94
30. Lung RI, Dumitrescu D (2007) A collaborative model for tracking optima in dynamic environments. In: Proceedings of congress on evolutionary computation, pp 564–567
31. Blackwell TM, Branke J (2006) Multiswarms, exclusion, and anticon vergence in dynamic environments. IEEE Trans Evol Comput 10(4):459–472
32. Li X (2004) Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization. In: Proceedings of genetic and evolutionary computation conference, pp 105–116
33. Yang S, Li C (2012) A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans 4:16
34. Liu L, Yang S, Wang D (2010) Particle swarm optimization with composite particles in dynamic environments. IEEE Trans Syst Man Cybern B Cybern 40(6):1634–1648
35. Kamosi M, Hashemi AB, Meybodi MR (2010) A hibernating multiswarm optimization algorithm for dynamic environments. In: Proceedings of world congress on NaBIC, pp 363–369
36. Woldesenbet YG, Yen GG (2009) Dynamic evolutionary algorithm with variable relocation. IEEE Trans Evol Comput 13(3):500–513
37. Yang S, Li C (2008) Fast multi-swarm optimization for dynamic optimization problems. In: Proceedings of international conference on natural computation, vol 7, no 3, pp 624–628
38. Hashemi B, Meybodi M (2009) Cellular PSO: a PSO for dynamic environments. In: Advances in computation and intelligence. Lecture notes in computer science, vol 5821, pp 422–433
39. Wang H, Yang S, Ip WH, Wang D (2012) A memetic particle swarm optimization algorithm for dyanamic multi modal optimization problems. Int J Syst Sci 43(7):1268–1283
40. Blackwell T, Branke J, Li X (2008) Particle swarms for dynamic optimization problems. In: Swarm S, Yang C, Li A (eds) Clustering particle swarm optimizer for locating and intelligence. Springer, Berlin, pp 193–217
41. Yazdani D, Sepas-Moghaddam A, Dehban A, Horta N (2016) A novel approach for optimization in dynamic environments based on modified artificial fish swarm algorithm. Int J Comput Intell Appl 15(2):1650010 (23 pages)
42. Nasiri B, Meybodi MR (2016) Improved speciation-based firefly algorithm in dynamic and uncertain environment. Int J Bio-Inspir Comput (in press)
43. Kordestani JK, Rezvanian A, Meybodi MR (2014) CDEPSO: a bi-population hybrid approach for dynamic optimization problems. Appl Intell 40:682–694
44. Yazdani D, Nasiri B, Sepas-Moghaddam A, Meybodi M, Akbarzadeh-Totonchi M (2014) MNAFSA: a novel approach for optimization in dynamic environments with global changes. Swarm Evolut Comput 18:38–53
45. Shams KN, Abdullah S, Turky A, Kendall G (2016) An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems. In: Knowledge-based systems · 104 April 2016 with 138 reads. https://doi.org/10.1016/j.knosys.2016.04.005
46. Mohammadpour M, Parvin H, Sina M (2018) Chaotic genetic algorithm based on explicit memory with a new strategy for updating and retrieval of memory in dynamic environments. J AI Data Min 6:191–205 (in press)
47. Rezvanian A, Meybodi MR (2010) Tracking extrema in dynamic environments using a learning Automata-Based immune algorithm, grid and distributed computing. Control Autom 121:216–225
48. Xin Y, Ke T, Xin Y (2011) Immigrant schemes for evolutionary algorithms in dynamic environments: adapting the replacement rate. Science in China Series F - Information Sciences II:543–552
49. Baktash N, Mahmoudi F, Meybodi MR (2012) Cellular PSO-ABC: a new hybrid model for dynamic environment. Int J Comput Theory Eng 4(3):365–368
50. Yang S (2007) Explicit memory schemes for evolutionary algorithms in dynamic environments. In: Evolutionary computation in dynamic and uncertain environments, vol 51. Springer, Heidelberg, pp 3–28
51. Yazdani D, Akbarzadeh-Totonchi MR, Nasiri B, Meybodi MR (2012) A new artificial fish swarm algorithm for dynamic optimization problems. In: IEEE congress on evolutionary computation (CEC), pp 1–8
52. Saxena N, Mishra KK (2017) Improved multi-objective particle swarm optimization algorithm for optimizing watermark strength in color image watermarking. Appl Intell 47(2):362–381
53. Sharma B, Prakash R, Tiwari S, Mishra KK (2017) A variant of environmental adaptation method with real parameter encoding and its application in economic load dispatch problem. Appl Intell 47(2):409–429
54. Tripathi A, Saxena N, Mishra KK, Misra AK (2017) A nature inspired hybrid optimisation algorithm for dynamic environment with real parameter encoding. IJBIC 10(1):24–32

Hamid Parvin received a B.D. degree from Shahid Chamran Uni., Ahvaz, Iran, in 2006 and an M.S. degree from Iran University of Science and Technology, Tehran, Iran, in 2008.

He then received his Ph.D. degree from Iran University of Science and Technology, Tehran, Iran. His research interests include data mining, machine learning, and ensemble learning.

Samad Nejatian received the M.Eng. in Telecom. Tech. and Ph.D. degree in Data Com. from the UTM in 2008 and 2014, respectively. He holds University Assistant Professor position at the Faculty of Elec. Eng., Islamic Azad Uni., Yasooj Branch, Yasooj, Iran.

His research interests are in cognitive radio networks, software-defined radio and wireless sensor networks. He is a registered member of professional organizations such as IEEE and IET.

Majid Mohamadpour received a B.D. degree from Kerman Uni., and an M.S. degree from Islamic Azad University, Yasooj, Iran.

His research interests include data mining, dynamic optimization.