Swarm and Evolutionary Computation ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Regular Paper

mNAFSA: A novel approach for optimization in dynamic environments with global changes

Danial Yazdani [a,*], Babak Nasiri [b], Alireza Sepas-Moghaddam [c], Mohammadreza Meybodi [d], Mohammadreza Akbarzadeh-Totonchi [e]

[a] *Young Researchers and Elite Club, Mashhad Branch, Islamic Azad University, Mashhad, Iran*
[b] *Department of Computer Engineering, Islamic Azad University of Qzvin, Qazvin, Iran*
[c] *Young Researchers and Elite Club, Science and Research Branch, Islamic Azad University, Tehran, Iran*
[d] *Department of IT and Computer Engineering, Amirkabir University of Tehran, Tehran, Iran*
[e] *Department of Computer and Electronic Engineering, Ferdowsi University, Mashhad, Iran*

ARTICLE INFO

Article history:
Received 10 January 2013
Received in revised form
30 March 2014
Accepted 21 May 2014

Keywords:
Artificial fish swarm algorithm
Swarm intelligence
Optimization in dynamic environments
Moving peak benchmark
Evolutionary algorithms
Meta-heuristic algorithms

ABSTRACT

Artificial fish swarm algorithm (AFSA) is one of the state-of-the-art swarm intelligence algorithms that is widely used for optimization purposes in static environments. However, numerous real-world problems are dynamic and uncertain, which could not be solved using static approaches. The contribution of this paper is twofold. First, a novel AFSA algorithm, so called NAFSA, has been proposed in order to eliminate weak points of standard AFSA and increase convergence speed of the algorithm. Second, a multi-swarm algorithm based on NAFSA (mNAFSA) was presented to conquer particular challenges of dynamic environment by proposing several novel mechanisms including particularly modified multi-swarm mechanism for finding and covering potential optimum peaks and diversity increase mechanism which is applied after detecting an environment change. The proposed approaches have been evaluated on moving peak benchmark, which is the most prominent benchmark in this domain. This benchmark involves several parameters in order to simulate different configurations of dynamic environments. Extensive experiments show that the proposed algorithm significantly outperforms previous algorithms in most of the tested dynamic environments modeled by moving peaks benchmark.

1. Introduction

Optimization plays a major role in business and engineering domains. It is also the heart of many processes that take place in nature. Optimization problems include many simple routine problems such as choosing the best path to destination, as well as very complex industrial controls. In fact, in many optimization problems, the best result is sought while spending the least amount of cost. In order to solve optimization problems, a cost function is normally designed first. Then, the goal is to find the input parameters that minimize the cost function by the use of mathematical or intelligent methods. When optimization problems are too complex, using mathematical methods becomes extremely difficult or even impossible. For such problems, intelligent methods adopted from nature can be used.

Optimization problems can be categorized according to different criteria. One criterion is whether the optimization problem is static or dynamic. In problems with static environments, the problem remains unchanged in the course of time, whereas in dynamic problems, the problem changes over time. Since many real-world problems have parameters that are time-variant, it can be concluded that optimization in dynamic environments is of paramount importance. However, dynamic factors have been ignored in most of the research conducted so far on optimization. This causes the solutions to get distant from those usable in the real world. For instance, the job scheduling problem may be pointed out. In this problem, the goal is to find the best order of carrying out n independent tasks on m sources, in a manner that the completion time of all of the tasks is minimized. Different types of dynamicity may be observed in these problems that have been overlooked by considering the problem as a static one. Introduction of a new job to the system while jobs are being performed, failure of a resource during operation, addition of a resource after repair, etc. are instances of changes that may occur to the system.

In view of the importance of optimization in dynamic environments, numerous researchers and scientists attempt nowadays to

* Corresponding author. Tel.: +98 935 1185556.
E-mail addresses: danial.yazdani@yahoo.com, d_yazdani@mshdiau.ac.ir (D. Yazdani).

http://dx.doi.org/10.1016/j.swevo.2014.05.002
2210-6502/© 2014 Elsevier B.V. All rights reserved.

design algorithms for solving these problems [8]. Optimization problems in dynamic environments are defined as follows in more scientific terms:

"Given a dynamic problem f_t, an optimization algorithm G to solve f_t, and a given optimization period $[t^{\text{begin}}; t^{end}]$, f_t is called a dynamic optimization problem (DOP) in the period $[t^{\text{begin}}; t^{\text{end}}]$ if during $[t^{\text{begin}}; t^{\text{end}}]$ the underlying fitness landscape that G uses to represent f_t changes and G has to react to this change by providing new optimal solutions."[32]

In the recent decade, using evolutionary algorithms and swarm intelligence for optimization in dynamic environments has become the focus of attention of many researchers in the field of machine learning and artificial intelligence. The reason behind this attention may be found in the nature of these methods, their path of evolution, and their adaptability when facing changes in the environment. However, designing optimization algorithms in dynamic environments faces many challenges that require new algorithms. As a matter of fact, except for the fact that the algorithm should be able to find optimum position(s) in dynamic environments, it should also be able to track them. Resolving existing challenges in DOPs, improvement of results, and reduction of computational costs are the most important goals in designing optimization problems in dynamic environments.

Artificial fish swarm algorithm (AFSA) is one of the algorithms inspired both from the nature and swarm intelligence algorithms. AFSA was presented by Lei [1]. This algorithm is an approach based on swarm behaviors that was inspired from social behaviors of fish swarm in the nature. This algorithm has some characteristics such as high convergence rate, insensibility to initial values, flexibility and high fault tolerance. AFSA has been utilized in optimization applications e.g. PID controller parameters setting [2], multi-objective optimization [3], global optimization [4], neural network learning [5], data clustering [6], color quantization [7] and etc.

In this paper, a modified AFSA is proposed for optimization in dynamic environments. All requirements of dynamic environments were satisfied in the proposed algorithm. In this algorithm, first, a novel model of AFSA (NAFSA) has been presented to increase convergence speed of the algorithm. The basic behaviors of standard AFSA which are corresponded to artificial fish movements along with their procedures involve several weak points that will be discussed in Section 3.3. These behaviors are considerably modified in NAFSA in order to solve the weak points and improve algorithm efficiency. Furthermore, some parameters are eliminated and several supplementary parameters are added and the overall procedure of AFSA is modified.

Subsequently after designing NAFSA, it is extended to conquer particular challenges of dynamic environment including outdated memory, diversity losing and covering several potential optimum peaks. The proposed dynamic optimization algorithm which is called mNAFSA, is a multi-swarm algorithm based on NAFSA which involves several mechanisms to conquer the challenges. Modified multi-swarm mechanism for finding and covering potential optimum peaks and new diversity increase mechanism which is applied after detecting an environment change are amongst the proposed mechanisms.

Another novel mechanism has been employed in the proposed algorithm which is sleeping–awakening. In this mechanism, the swarms placed on non-best peaks are slept after approaching to the new position of the peak (after an environment change). This issue leads to more time for the swarm which is placed on the best peak in the current environment in order to perform a better exploitation.

The proposed algorithm has been applied on different configurations of moving peaks benchmark (MPB) [11,12] which is one of the most well-known problems for dynamic environments. Several researchers [9,10,17–19,23,24,27–31] used MPB to test their algorithms, so it is a good test suited for a fair comparison of different dynamic optimization algorithms. The performance of the proposed algorithm is compared with 10 other algorithms which were presented for optimizing MPB. The comparison metrics is *offline_error* which is the main metric in this domain. Experimental results show that the proposed algorithm has an appropriate efficiency. This paper is organized as follows: Section 2 is dedicated to review the related work. In Section 3, the standard artificial fish swarm algorithm is discussed and subsequently, the modification of the algorithm for optimization in dynamic environments is proposed in the next section. Experimental study is investigated in Section 5 and Section 6 concludes the paper.

2. Related work

Multi-swarm is considered as a well-known solution for designing DOPs [43]. Several algorithms for DOPs based on multi-swarm approach have been proposed in the literature. In [33], a method called Shifting Balance Genetic Algorithm (SBGA) has been proposed in which a number of small subpopulations were responsible for global search in the problem space, and a large subpopulation was responsible for tracking the peaks. Another approach was presented in [34] called Self Organizing Scouts (SOS), which utilized a big subpopulation for global search and a number of small subpopulations for tracking changes. This strategy has also been proposed with other meta-heuristic methods such as Genetic Algorithm in [35] and Differential Evolution in [36]. In [23,19], two methods similar to SOS were proposed, respectively called fast multi-swarm optimization (FMSO) and multi-swarm optimization algorithm (MPSO), in which a parent type explored the search space to discover existing promising areas in the environment, and a series of child types performed local search. Another approach was to use a population for both local search and global search simultaneously. In [37], a population was used for performing global search, and after discovering an optimum, the population was divided into two subpopulations. The first and second subpopulations were responsible for tracking optimum changes and conducting global search, respectively. In [38–40] a speciation-based PSO approach was proposed for optimization in dynamic environments. Li et al. [38] describe an extension to a speciation-based particle swarm optimizer to improve performance in dynamic environments. Parrott and Li [39] propose an improved particle swarm optimizer using the notion of species to determine its neighborhood best values for solving multimodal optimization problems and for tracking multiple optima in a dynamic environment. In [40], a form of speciation allowing development of parallel subpopulations is used. The model employs a mechanism to encourage simultaneous tracking of multiple peaks by preventing overcrowding at peaks. Also in [22], a regression-based approach (RSPSO) was presented in order to enhance the convergence rate using speciation-based methods. In that approach, every subpopulation was considered as a hypersphere and was developed through a certain radius of the best solution. In [21], a method called multi-strategy ensemble particle swarm optimization (MEPSO) was proposed in which every cluster was divided into two. The first cluster was responsible for exploitation and the second one was in charge of exploration. In that research, Gaussian local search and differential mutation have been used in order to improve diversity in the environment. The proposed algorithm in [41] employs hierarchical clustering method to track multiple peaks based on a nearest neighbor search strategy. A fast local search method is also proposed to find the near optimal solutions in a local promising region in the search space. In [30], the proposed method in [41] has been improved, in which some simplifications, e.g. eliminating the learning procedure, and reducing the number of phases for clustering from two phases to only one phase were made. In [9], two multi-population methods, called MQSO and MCPSO, were proposed. In the former one, quantum particles and in the latter one, charged particles were used to generate diversity. The number of solutions in this technique was equal for every subpopulation, and the number of subpopulations was also initialized. In [10], an approach for enhancing this method was proposed by adapting the number of subpopulations, which was called adaptive MQSO (AMQSO). It has significantly improved the performance of the algorithm. Finally, a method called PSO with composite particles (PSO-CP) was presented in [24], to address dynamic optimization problems. PSO-CP partitions the swarm into a set of composite particles based on their similarity using a "worst first" principle. Inspired by the composite particle phenomenon in physics, the elementary members in each composite particle interact via a velocity-anisotropic reflection scheme to integrate valuable information for effectively and rapidly finding the promising optima in the search space. Each composite particle maintains the diversity by a scattering operator.

3. Artificial fish swarm algorithm

3.1. Basic AFSA algorithm

In biology, an aggregation of fish is the general term for any collection of fish that have gathered together in some locality. Fish aggregations can be structured or unstructured. An unstructured aggregation might be a group of mixed species and sizes that have gathered randomly near some local resource, such as food or nesting sites.

If the aggregation comes together in an interactive, social way, they are said to be shoaling. Although shoaling fish can be related to each other in a loose way, with each fish swimming and foraging somewhat independently, they are nonetheless aware of the other members of the group as shown by the way they adjust behavior such as swimming, so as to remain close to the other fish in the group. Shoaling groups can include fish of disparate sizes and can include mixed-species subgroups.

If, as a further addition, the shoal becomes more tightly organized, with the fish synchronizing their swimming so they all move at the same speed and in the same direction, then the fish are said to be schooling. Schooling fish are usually of the same species and the same age/size. Fish schools move with the individual members precisely spaced from each other. The schools undertake complicated maneuvers, as though the schools have minds of their own.

Many hypotheses to explain the function of schooling have been suggested, such as better orientation, synchronized hunting, predator confusion, finding more food, and reduced risk of being found.

AFSA is one of the swarm intelligence methods and evolutionary optimization techniques, which was proposed by Lei et al. [1]. The framework of this algorithm is based on functions that are inspired from social behaviors of fish schools in the nature.

In this algorithm, there are four functions inspired from fish behaviors in fish swarms. The first function is free-move behavior: in nature, when fish do not prey, they move freely in the swarm. The second function is prey behavior: naturally, every fish looks for its prey independently by means of its senses. These senses are mostly sense of vision, smell and available sensors on their bodies. In AFSA, the area where an artificial fish can sense a prey is modeled as a neighborhood with a *Visual*-sized radius. The next function is follow behavior: in fish swarm, when a fish finds food, other swarm members also go after it to reach the food. The last function is swarm behavior: in nature, to be secure from hunters, fish always try to be in the swarm and not to leave it.

In the underwater world, a fish can find areas that have more food, which is done through individual or swarm search by the fish. According to this characteristic, an artificial fish (AF) model is represented by four mentioned behaviors. AFs search the problem space by these behaviors. The environment, where an AF lives, is essentially the solution space and other AFs domain. Food consistency degree in the water area is AFSA objective function. Finally, AFs reach to a point where its food consistency degree is maxima (global optimum).

As it is observed in Fig. 1, AF perceives external concepts with a sense of sight. The current position of an AF is shown by vector $X=(x_1, x_2, \dots, x_n)$. The *Visual* is equal to sight field of the AF and X_v is a position in *Visual* where the AF wants to go. Then if X_v has better food consistency than the current position of AF, it goes one step toward X_v which causes change in the AF position from X to X_{next}, but if the current position of AF is better than X_v, it continues searching in its *Visual* area. Food consistency in position X is fitness value of this position and is shown with $f(X)$. The *Step* is equal to the maximum length of the movement. The distance between two AFs which are in X_i and X_j positions is shown by $Dis_{i,j}=\|X_i-X_j\|$ (Euclidean distance) that is computed by

$$Dis_{i,j}=\sqrt[2]{\sum_{d=1}^{D}(X_{j,d}-X_{i,d})^2} \tag{1}$$

Several parameters have been involved in AFSA. These parameters are X (position vector of AF), *Step* (maximum length of step by which an AF can be moved), *Visual* (the range in which an AF can explore), *try_number* (the maximum number of tries that an AF explore around itself for finding better positions), bulletin (a memory for keeping the best ever found position of AFs) and crowd factor δ $(0<\delta<1)$ (a control parameter to control AFs' crowd around a position).

In each step of the optimization process, AF looks for locations with better fitness values in the problem search space by performing these behaviors based on the algorithm procedure. In the following, the behaviors of AFSA will be discussed.

3.1.1. Free-move behavior

In AFSA, when an AF cannot move toward a place with more food, it moves a random step in the problem space by

$$X_{i,d}(t+1)=X_{i,d}(t)+Step\times Rand_d(-1,\ 1) \tag{2}$$

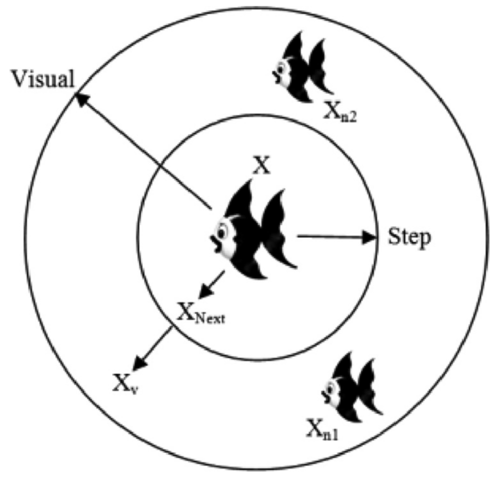

Fig. 1. Artificial fish and the environment around it.

$X_{i,d}$ is the component d of AF i's position in the D-dimensional space and $1 \le d \le D$. Rand function generates a random number with a uniform distribution in [−1, 1].

3.1.2. Prey behavior

If X_i is the current position of AF i, we choose position X_j in the *Visual* of AF i randomly. $f(X)$ is the food consistency in position X or its fitness value. Position X_j is given by

$$X_{j,d} = X_{i,d} + Visual \times Rand_d(-1,\ 1) \quad (3)$$

Then food density in X_i is compared with the that of the current position, if $f(X_i) \ge f(X_j)$, AF i moves forward a step from its current position to X_j, which is done by

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \frac{\vec{X}_j - \vec{X}_i(t)}{Dis_{i,j}} \times Step \times Rand(0,\ 1) \quad (4)$$

X_i is a D-dimensional vector and $Dis_{i,j}$ is the Euclidean distance between vectors X_i and X_j. X_j–X_i generates a transfer vector from X_i to X_j and when divided by $Dis_{i,j}$, a vector with unit length is created from X_i toward X_j. Here, Rand function generates a random number which causes AF i to move as much as a random percent of Step toward position X_j. Nevertheless, if $f(X_i) < f(X_j)$, we choose another position X_i by Eq. (3) and evaluate its food density to understand whether forward condition is satisfied or not. If after *try_number* times AF does not succeed in satisfying forward condition, the concerned AF performs free-move behavior and moves a step in the problem space randomly.

3.1.3. Swarm behavior

In AFSA, in order to keep swarm generality, in each of the iterations, AFs try to move toward a central position. A central position of swarm is given by

$$X_{\mathrm{Center},d} = \frac{1}{N}\sum_{i=1}^{N} X_{i,d} \quad (5)$$

As it is seen in Eq. (5), component d of X_{center} vector is the arithmetic average of component d of all swarm AFs. Let N be the population size, and n_c be the number of AF in *Visual* field around X_{center}. If $f(X_{\mathrm{center}}) \le f(X_i)$ and $\delta > (n_c/N)$, that is the central position has a better food consistency than the current position and population density in its neighborhood is not much, so AF i moves toward the central position by

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \frac{\vec{X}_{\mathrm{Center}} - \vec{X}_i(t)}{Dis_{i,\mathrm{Center}}} \times Step \times Rand(0,\ 1) \quad (6)$$

If $n_c = 0$ or the condition of moving toward the central position is not satisfied, prey behavior is performed for AF i.

3.1.4. Follow behavior

If X_i is the current position of AF i, it checks neighbor X_n, if n_n is the number of AFs in the *Visual* of AF n, if $f(X_n) \le f(X_i)$ and $\delta > (n_n/N)$, i.e. position X_n has a better food consistency than the current position of AF i and population density in its neighborhood is not much, therefore AF i moves one step toward AF by

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \frac{\vec{X}_n - \vec{X}_i(t)}{Dis_{i,n}} \times Step \times Rand(0,\ 1) \quad (7)$$

If AF i has no neighbors or none of its neighbors satisfy following condition, prey behavior would be performed for AF i.

AFSA performs the optimization process iteratively by means of functions described as its behaviors, and the algorithm factors or AFs try to get closer to our objectives by executing these functions.

For AFs, prey and free-move behaviors are the individual ones and follow and swarm behaviors are group behaviors. Prey behavior would be performed if an AF cannot move to a better position by executing follow behavior and/or swarm behavior, and free-move behavior is performed if an AF is not successful in finding a better position by performing prey behavior. Indeed, prey and free-move behaviors are not performed independently by AFs and are only performed when an AF cannot move to a better position by follow and swarm behaviors. In AFSA, in each step of the optimization process, all AFs pass the same procedure in parallel.

Let the position of AF i at time t be $X_i(t)$. AF i performs follow behavior at position $X_i(t)$ once, which leads to obtain position $X_{i,\mathrm{Follow}}$. After executing the follow behavior, AF i performs swarm behavior from that position $X_i(t)$ and it leads to obtain position $X_{i,\mathrm{Swarm}}$. After performing both follow and swarm behaviors that were done with respect to X_i(t), the next position of AF i ($X_i(t+1)$) is obtained by

$$X_i(t+1) = \begin{cases} X_{i,\mathrm{follow}} & \text{if } f(X_{i,\mathrm{follow}}) \le f(X_{i,\mathrm{swarm}}) \\ X_{i,\mathrm{swarm}} & \text{if } f(X_{i,\mathrm{follow}}) > f(X_{i,\mathrm{swarm}}) \end{cases} \quad (8)$$

In AFSA, a bulletin is used for recording the best position that has been found so far by all swarm members. In each of the iterations, after performing AFSA's behaviors by all AFs and transferring them to new positions, the fitness value of the best AF is compared with the recorded position on the bulletin. If fitness value of the best AF is better than the recorded position on the bulletin, the position of the best AF is recorded as the best found position so far. Standard AFSA pseudo code is shown in Fig. 2. In this pseudo code, prey and free-move behaviors were considered as a part of swarm and follow behaviors. That is, if an AF could not perform successfully a swarm or follow behavior, it would perform prey behavior and if could not reach a better position by executing this behavior, it would perform free move behavior.

3.2. Comparison between AFSA and PSO

As it was mentioned, AFSA belongs to swarm intelligence algorithms. Among swarm intelligence algorithms, there are other algorithms which resemble AFSA. Particle swarm optimization is one of the most straight-forward and popular swarm intelligence algorithms [13] whose applications could be the same as those of AFSA. Many versions of this algorithm have been represented by researchers so far [14]. In the following, similarities and differences of AFSA and PSO will be studied. It should be noted that the contents of this section are due to various researches and experiments done on different versions of these two algorithms.

Since both algorithms belong to swarm intelligence algorithms, there is a population of agents in them that compose together a

```
Standard AFSA
foreach AF i
  initialize $x_i$
endfor
$bulletin = arg\ \min_{X_i} f(X_i)$
Repeat
  foreach AF i
    Perform swarm behavior on $X_i$(t) and obtain $X_{i,swarm}$
    Perform Follow behavior on $X_i$(t) and obtain $X_{i,follow}$
    if f($X_{i,swarm}$) ≥ f($X_{i,follow}$) then
      $X_i$(t+1)= $X_{i,swarm}$;
    else
      $X_i$(t+1)= $X_{i,follow}$;
    endif
  endfor
  if f($X_{Best-AF}$) ≤ f(bulletin) then
    bulletin = $X_{Best-AF}$;
  endif
until stopping criterion is met
```

Fig. 2. Pseudo code of AFSA algorithm.

swarm. Members of these swarms cooperate with each other to reach a common objective and their interactions with each other lead to creating a general behavior among them. Both PSO and AFSA algorithms work based on random search and probability rules. In fact particles or AFs are distributed in the problem space randomly and after that their motions are also completely dependent on numbers generated randomly. First, it may be seen that these two algorithms are similar to each other or AFSA is one of PSO versions, but the instruction of these two algorithms and the type of motion of swarm members in them are fundamentally different. In a different manner of AFSA, various versions of PSO have been presented until now, but in all of them, the framework of PSO has been kept and has not been changed.

According to various studies we have done on PSO and AFSA algorithms, the biggest difference between them is the motion difference of particles and AFs. In PSO, particles are completely dependent to their previous experience and to the swarm previous experience for the next motion. On the basis of PSO procedure, to determine a particle's next position, every particle needs (a) the best position that has been experienced so far, (b) the best position that all members of the swarm have experienced so far, (c) the previous velocity that was the displacement vector of its previous motion and (d) its current place, and it does not need the current position of other swarm particles. However, in AFSA, AFs do not use their previous experiences and other swarm members' experiences, and also their previous motion for determining their next position at all. Indeed, according to Eqs. (2)–(7), for determining the next position, AFs depend on their current position and other swarm members.

3.3. *Weak points of standard AFSA*

In this section, on the basis of various executions of experiments on AFSA, we will study the weak points of standard AFSA:

3.3.1. *Not using previous experiences of AFs in the optimization process*

In standard AFSA, the best position that has been found so far is always recorded on the bulletin. The reason of using the bulletin in AFSA is that even an AF that has found the best position so far, can lose its position in the next iteration by performing free-move behavior and be involved in a worse position. Therefore, AFs lose their concentration on the current best found position. Consequently, AFs are not capable of doing an acceptable local search in the surroundings of the currently best found position to find better positions. Therefore, the local search ability of AFs decreases. AFs in AFSA do not use the best recorded position on the bulletin for their next motions. This is a considerable weak point in AFSA because the best result found by the swarm so far is not applied for improving swarm member positions. Therefore, the convergence rate of AFSA decreases.

3.3.2. *Non existence of balance between global search and local search abilities*

In AFSA, AFs search the problem space by performing AFSA's behaviors in their *Visual* space. Also, AFs move toward their goal based on a random percentage of *Step* in each iteration. Setting the initial value for *Visual* and *Step* parameters has a considerable effect on the quality of the final result, because the values of these two parameters remain fixed and are equal to their initial value until the end of the algorithm execution. If we consider the large initial value for these two parameters, AFs can move faster toward the goal since they can search a larger space around themselves and move with larger *Step* lengths in each iteration. In this condition, largeness of *Visual* parameter leads to increasing in the search boundary of AFs and they can search further distances that makes them search spaces out of local optimums that they are trapped in and consequently get out of there.

However, large values of *Visual* and *Step* parameters have some disadvantages. In this case, the local search ability of AFs decreases. In fact, the algorithm performs the global search well but after reaching near the goal, AFs are unable to do an acceptable local search, because in this situation *Visual* parameter value is larger than the space where AF should search. Therefore the probability of finding positions with a better fitness decreases. Also because of a large *Step*, AFs may pass the global optimum after reaching near it and even go further than that which leads to a decreased accuracy and stability of the algorithm.

If we consider small values for these two parameters, the algorithm can perform an acceptable local search, but in this case, AFs move slowly toward the goal and their ability to pass local optimums decreases. Therefore, the global search ability of the algorithm would decrease. As a result, by adjusting the initial values of *Visual* and *Step* parameters, AFs are able to do just one of the global or local search tasks acceptably.

3.3.3. *Wasting computations, high complexity*

In AFSA optimization process, as discussed in Section 2, in each of the iterations, swarm and follow behaviors are performed for each AF independently. After that, at the end of each iteration, the new position of an AF is determined regarding one of these two behaviors, which has a better result. But considerable computational load is consumed for executing both behaviors and just one of them influences on determining the new position of AFs. Therefore, consumed computational load for performing one of these behaviors has no effect on the motion of AF and it is wasted. One of the most important criteria for comparing the optimization algorithms is comparing the achieved results on the basis of the number of fitness evaluations, which is equal to the number of times that the optimization algorithm has evaluated the fitness function value in the optimization process. Since calculating cost functions in real-world problems is the most complex stage in optimization process, the number of fitness evaluations required by the algorithm for convergence is one of the prominent metric to evaluate the speed of optimization algorithms.

In term of this criterion, AFSA cannot compete with other similar algorithms. AFSA has high computational complexity. All neighbors of AFs have to be specified for follow behavior execution. For this reason, Euclidean distances between all AFs have to be computed. Therefore, a huge computational load occurs in each of the iterations. Also, in the best situation, every AF performs two fitness evaluations for follow and swarm behaviors in a single iteration and this number, in the worst case, could reach to $2\times(try\_number+1)$ times.

3.3.4. *Weak points of AFSA's parameters*

In AFSA, AFs in each of their motions displace maximally as long as their *Step*. Parameter value of *Step* is equal or less than the *Visual* value. In prey and follow behaviors, AFs move toward a position in their *Visual* as much as a random percentage of *Step* parameter by Eqs. (4) and (7). But in this situation, AFs may go to a position where it is unwanted. For example, suppose the Euclidean distance between the destination position and the current position of AF to be 2, and the parameter value of *Step* to be 10. The random generated number by Rand function in Eqs. (4) and (7) is, however, larger than 2, meaning that the AF gets further from the goal position.

Crowd factor parameter is for controlling the swarm diversity. Smaller values of this parameter cause maintaining a distance between AFs and bigger values of it cause the distance between them to decrease. Note that the distance between AFs or swarm density has considerable influence on global and local search abilities. Although the adjustment of global search and local search ability highly depend on determining the *Visual* and *Step* parameters value, crowd factor value is also effective on that. In general, specifying an appropriate initial value for crowd factor, *Visual* and *Step* parameters is very difficult for various applications and the final result is highly dependent on these parameters values. According to these conditions, solving problems requiring both high global search ability and high local search ability with standard AFSA is very difficult or even impossible.

4. Modifying AFSA for dynamic environments

In this section, first, an improved version of AFSA algorithm is presented. Then a novel algorithm based on the previous one is proposed for optimization in dynamic environments.

4.1. New AFSA algorithm (NAFSA)

In this section, a new algorithm based on artificial fish swarm algorithm is presented for optimization in dynamic environments. In new AFSA algorithm, parameters, behaviors and AFSA procedure are modified to be appropriate for optimization in dynamic environments. In this algorithm, several fundamental modifications have been performed on *prey*, *follow* and *swarm* behaviors. The reason to perform these changes is to adapt AFSA for working in dynamic environments and remove its weaknesses. In the following, the behaviors of NAFSA will be discussed.

4.1.1. New_prey behavior

This behavior is an individual behavior and each AF performs it without considering other swarm members. Along performing this behavior, each AF does a local search around itself. By performing this behavior each AF attempts *try_number* times to be replaced to a new position with a better fitness. Let suppose AF *i* is in position X_i and wants to execute prey behavior. Following steps are done in prey behavior:

(a) AF *i* considers a target position in *Visual* by Eq. (9), then evaluates its fitness value. *d* is the dimension number and Rand generates a random number with a uniform distribution in [−1, 1]:

$$X_{T,d} = X_{i,d} + Visual \times Rand_d(-1,\ 1) \tag{9}$$

(b) If the fitness value of position X_T is better than the current position of AF *i*, this position will be updated by

$$\vec{X}_i = \vec{X}_T \tag{10}$$

Steps (a) and (b) are performed *try_number* times. By executing the above steps, in the best case, an AF can update its position at most *try_number* times and move toward better positions. In the worst case, none of an AF attempts to find a better position will be successful. Therefore, after performing new_prey behavior in this situation, there will be no replacement at all.

4.1.2. New_swarm behavior

This function is a group behavior as well and is performed globally among members of the swarm. In new_swarm behavior, first of all, the central position of the swarm is calculated in terms of arithmetic average of all swarm members position in every dimension. The central position of the swarm X_{center} is obtained by Eq. (5).

As it is observed, component d of vector X_{center} is the arithmetic mean of component d of all AFs of the swarm. For AF *i*, the move condition toward the central position is checked, i.e. $f(X_{\text{Center}}) \ge f(X_i)$ and if this condition is satisfied, the next position of AF *i* is obtained by

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \frac{\vec{X}_{\text{Center}} - \vec{X}_i(t)}{Dis_{i,\text{Center}}} \times \left[\vec{V}isual \times Rand(0,\ 1)\right] \tag{11}$$

Eq. (5) is used for all AFs that have worse positions than the central position so they move toward X_{Center}. But for the best AF locating in X_{Best}, if the fitness value of X_{Center} is better than X_{Best}, the next position of the best AF is obtained by

$$\vec{X}_{\text{Best}} = \vec{X}_{\text{Center}} \tag{12}$$

In NAFSA algorithm, Eq. (11) is not used for relocating the best AF. This issue is due to the possibility of existing worse positions between source and destination in Eq. (11), based on the structure of the problem. In this situation, it is possible that the AF which performs Eq. (11) moves to another position with worst fitness value. Thus, to retain the best position, Eq. (12) is utilized for relocating the best AF. Therefore, it may cause all members of the swarm to lose the best position found so far. This problem is removed by using Eq. (12) for the best AF. The reason for not using of Eq. (12) for all AFs is that changing the position of swarm fish to a similar position leads to an extreme decreasing diversity of the swarm and a considerable decrease of convergence rate.

4.1.3. New_follow behavior

In case of lack of finding better positions in performing standard prey behavior, AFs in Standard AFSA move one step randomly and lose their previous position. But, as it was discussed, if an AF is not able to move to better positions in new_prey behavior in NAFSA, it would not move at all and will keep its previous position. This causes the best AF (according to fitness value) of swarm to locate in the best found position by the swarm member so far. The reason is that in prey behavior in NAFSA, an AF displaces if only it moves to a better position. In follow behavior, each AF moves one step toward the best AF in the swarm using

$$\vec{X}_i(t+1) = \vec{X}_i(t) + \frac{\vec{X}_{\text{Best}} - \vec{X}_i(t)}{Dis_{i,\text{Best}}} \times \left[Visual \times Rand(0,\ 1)\right] \tag{13}$$

X_i is the position vector of AF *i* which performs *new_follow* behavior and X_{best} is the position vector of the best AF in the swarm.Therefore, AF *i* moves at most to the *Visual* extent in each dimension toward the best AF in the swarm. In fact, after finding more food by a fish, other swarm members follow it to reach more food. Following the best AF in the swarm causes the convergence rate to increase and helps to keep integrity of AFs in a swarm. This behavior is a group behavior and interactions between swarm members are done globally. In NAFSA algorithm, by performing *new_prey*, *new_follow* and *new_swarm* behaviors, AFs try to find better positions.

4.1.4. NAFSA procedure

In NAFSA, for every AF, *new_prey*, *new_follow* and *new_swarm* behaviors are performed in each iteration. In Standard AFSA, executing one of the standard swarm and standard follow behaviors did not affect AFs movement and a huge amount of computations were wasted. In NAFSA, in a different manner of standard AFSA, all three behaviors influence the movement of AFs and the swarm moves toward better positions. In NAFSA, first all AFs perform *new_prey* behavior and their positions are updated based on this behavior procedure. By executing this behavior, every AF can displace up to *try_number* times. Subsequently, all AFs perform *new_follow* behavior, based on their new positions and the best AF of the swarm. Then each AF performs *new_swarm* behavior. By performing *new_swarm* behavior, AFs which are apart from swarm and are located in worse positions than other swarm members, can join the swarm faster. In fact, this behavior results in a faster movement of AFs which are in worse positions. As a result of performing this behavior, the convergence rate would increase and unlike *new_follow* behavior, it can cause position improvement of the best AF.

At the end of each iteration of performing NAFSA algorithm, *Visual* value is updated for AFs. According to various experiments done on AFSA, it is concluded that big value of *Visual* parameter leads to increasing the convergence rate, since AFs move with larger steps. But when the swarm converges, AFs are not able to do an acceptable local search, due to this fact that after convergence, the space which has to be searched for better positions becomes smaller and AFs with a large *Visual* would have less chance to reach better positions by performing *new_prey* behavior. On the other hand, if *Visual* is small, the local search ability increases but because of small move steps, the convergence rate decreases drastically. Thus, it can be concluded that the best condition is that at first *Visual* should be large so AFs converge fast to their goals. Along with converging of the swarm to the goal, *Visual* value has to decrease gradually so at last AFs with a small *Visual* could reach acceptable results by doing an appropriate local search. For this reason, *Visual* value has to be multiplied by a number less than one in each iteration. We can use various mechanisms to determine this number. In this paper, a specified random number generator function is used as following:

$$Visual(t+1) = Visual(t) \times (L_{Low} + (Rand \times (1 - L_{Low}))) \tag{14}$$

In Eq. (14), in each iteration, *Visual* is obtained randomly based on its value at the previous iteration. L_{Low} is the lower limit of *Visual* change percentage in comparison with the previous iteration. *Rand* is the random number generator function with a uniform distribution in [0, 1]. So, *Visual* value in each iteration is randomly in $[Visual(t-1) \times L_{Low}, Visual(t-1)]$.

NAFSA pseudo code is represented in Fig. 3. Here we suppose the optimization problem is a maximizing problem.

```
NAFSA:

foreach Artificial Fish i ϵ [1 .. N]
    initialize $x_i$
endfor
repeat:
    //New_prey Behavior
Foreach AF i
for counter=1totry_number
Obtain $\vec{X}_T$ with Eq. (1) and Calculate f($\vec{X}_T$)
if $f(\vec{X}_T) \geq f(\vec{X}_i)$ then
$\vec{X}_i = \vec{X}_T$
        end if
endfor
endfor
    // New_Follow Behavior
for each AF i
Apply Eq. (3)
endfor
    // New_Swarm Behavior
Calculate Central Position $\vec{X}_{Center}$ by Eq. (4)
foreach AF i
        if $f(\vec{X}_{Center}) \geq f(\vec{X}_i)$ then
            if $\vec{X}_i$ is Best AF
                Apply Eq. (6) for AF i
            else
                Apply Eq. (5)for AF i
            endif
        end if
endfor
Update Visual according to Eq. (7)
Until stopping criterion is met
```

Fig. 3. NAFSA pseudo code.

4.1.5. Comparison between AFSA and NAFSA

In this section, standard AFSA and NAFSA algorithms are compared and their differences are investigated. First, these two algorithms are compared from the perspective of computational load and complexity. As it was mentioned before, for performing *follow* behavior in standard AFSA, Euclidean distance of all AFs from each other must be calculated which leads to high computational complexity, where *new_follow* behavior is NAFSA has not such complexity. Similarly, there is great deal of difference between computational complexity of *swarm* behavior in standard AFSA and *new_swarm* behavior is NAFSA. In order to perform *swarm* behavior in standard AFSA, Euclidean distance of all AFs from the central point must be calculated in order to measure crowd factor around central position which leads to high computational complexity.

For comparing *prey* and *new_pray* behaviors, it must be noted that in AFSA both Eqs. (3) and (4) are performed one time in the best case and Eqs. (3) and (4) are performed *try_number* times and one time, respectively, in the worst case. On the other hand, *new_pray* structure is completely different than *prey*, where an AF can improve its position *try_number* times. However, it seems that comparing complexity of *prey* and *new_pray* behaviors is related to dimensions and fitness function complexity in different problems, so such comparison cannot be generally done, without considering the type of the problem.

From different viewpoint, AFSA and NAFSA have a major difference in their iterations. In AFSA, *follow* and *swarm* behaviors are performed simultaneously, and *prey* behavior is performed in case of failure any of them. Finally, after calculating result, the obtained position is accepted by either *follow* or *swarm* behaviors. On the other hand, *new_follow*, *new_swarm* and *new_prey* behaviors are respectively performed in NAFSA by all AFs in order to improve their positions.

In addition, in NAFSA, a number of parameters i.e. *crowd factor*, *step* and *free-move* are eliminated, compared to standard AFSA. As it was mentioned in Section 3.3.4, *step* parameter has some disadvantages. In NAFSA, Eq. (10) is used for relocation in *new_-pray* behavior. Also, since the goals in *new_follow* and *new_swarm* behaviors are out of *visual* of AFs, *visual* parameter is used as the maximum step in Eqs. (11) and (13). By eliminating *crowd factor* and the related conditions in NAFSA, local search ability is improved and computational complexity is reduced. However, the global search ability of the algorithm is reduced, where this problem is solved by proposing the multi-swarm algorithm.

4.2. Multi-swarm NAFSA (mNAFSA)

In most dynamic problems, when there are more than one peak in the problem space, each peak can be modified into global optima after the environment change. Hence, all peaks have to be covered by AFs as much as possible so that if each of them transforms into global optima, the algorithm could find it fast. Thus, one swarm should be located at each peak and cover it. So, multi-swarm has to be used in mNAFSA and each swarm acts independently from other swarms and does according to NAFSA procedure. In this paper, at the beginning, there is only one swarm in the problem space. If the swarm converges to a peak, another swarm is generated in the problem space. A swarm has converged when the best AF position is almost constant after a number of iterations in the problem space. It means Euclidean distance of the current position of the best AF from its position in some previous iterations will be less than a specified amount. In mNAFSA, whenever all swarms have converged, a new swarm is generated in the problem space and starts searching. If the recently generated swarm converges to a peak which is covered by another swarm, a race condition occurs. In this condition, the swarm which has a better best AF, according to the fitness value, continues its activity and the loser will be expelled and re-initialized.

The new generated swarm would converge to a covered peak by another swarm when Euclidean distance of the best AF position of the newly generated swarm from the best AF position in one of the other swarms in the problem space is less than a specified value r_{excl}. Thereafter, any newly generated swarm has a chance to converge to a peak where no other swarms have resided yet. But if it converges to a covered peak, with a high probability it will be reinitialized. It is because the probability of being placed in a better position than a previously residing swarm is very low. If the new swarm converges to an empty peak which is not covered by any swarms yet, another swarm will be generated in the problem space. Therefore, the number of available swarms in the problem space is always one more than the number of found peaks in the problem space. This mechanism of generating new swarms is similar to [10] which was presented for PSO, but with some modifications in it.

To discover change in the environment, the best artificial fish of each swarm must be evaluated at the end of each iteration. In case that any changes in the obtained values, compared to the stored ones, the environment has been changed [9,10].

After detecting change in the environment, the problem of diversity reduction has to be resolved first. Indeed, when a swarm converges to a point, the distance of AFs decreases considerably from each other and the diversity in swarm decreases too much, as well. As a result, after the environment change, AFs cannot take advantage of follow and swarm behaviors because the distance of artificial fish from each other has decreased too much.

To resolve this issue, after detecting change in the environment, the best artificial fish position of the swarm is kept for converged swarms and other AFs of that swarm are distributed randomly in a d-dimensional ball around the best AF position of the swarm. Radius of this d-dimensional ball in each dimension is determined in terms of *Shift Severity* of peaks in MPB since it is expected that a new peak position will be placed in a d-dimensional ball with the center of the previous peak position before the environment change. Also, *Visual* has decreased with respect to the space where the swarm has to search for better positions by using Eq. (9). Thus, after detecting an environment change, *Visual* value should be determined in terms of *Shift Severity*. Consequently, the new peak position would be in the searching neighborhood of AFs and they can move fast toward it. For a swarm that has not converged, there is no need to change the position of AFs because diversity has not been lost but its *Visual* value must adjust equal to its initial value to avoid reducing the convergence rate.

The value of *Shift Severity* is the maximum of the amount of peaks' movements in the problem space. Since there is no prior knowledge about this value in real-world problems, the algorithm needs to calculate it during the optimization process. For this reason, we could use the movement's position of the best AF in its swarm in two consecutive environment changes. It is worth to note that a default value is used for the first change in the environment (1 seems to be sufficient in simulations) and so, *Shift Severity* value can be calculated after the second change.

After adjusting *Visual* and determining AFs positions, their fitness values are reevaluated to remove outdated memory. Then, the algorithm continues searching in the new environment. In mNAFSA, there is no global relation among swarms and swarms do not use other swarms information and only there exists a local relation across swarms whose Euclidean distance between their best artificial fish is less than r_{excl}.

In order to improve the efficiency of the algorithm, a novel sleeping–awakening mechanism has been added to the algorithm. After the environment change, every swarm moves toward the peak where it is placed by performing a exploitation. Among the swarms, this task is much more important for the swarm placed near the highest peak. In fact, the value of the *Current Error* is determined according the results of the swarm whose *Best_AF* fitness value is better than the rest of the swarms. Therefore, other swarms do not contribute to determining the result in the current environment, but performing a local search is important for them as well. If these swarms do not perform a local search, their distances from their peaks may increase after several environment changes and peak relocations, and they may even lose it. As a result, performing a local search is essential for all swarms after the environment change.

To improve performance and to give more chance for the swarm nearest to the highest peak, a *sleeping–awakening* mechanism is used for swarms in the proposed algorithm. In this mechanism, the swarms can have two different states of *sleeping* and *awakening*. The *awake* swarm is the swarm whose AFs exist in the problem space and evaluate fitness in the search process, and whose particles relocate in every iteration of the algorithm execution. The *asleep* swarm is the swarm whose AFs exist in the problem space, but they do not move and do not perform fitness evaluation.

In this mechanism, all swarms are awakened after each environment change, and they perform the optimization process. As discussed earlier, the active swarms placed at non-optimum peaks should perform a local search after each environment change so as not to increase their distance from their covered peaks. However, continuing the local search for improving accuracy, after drawing near their peaks, is not helpful for these swarms and has no effect on the algorithm outcome. Therefore, as soon as a swarm residing at a non-optimum peak gets close to its goal, it is asleep.

By applying this mechanism, after active *tracker* swarms placed at non-optimum peaks in the current environment have reached their goals, they are asleep and they stop fitness evaluation until the next environment change. So a considerable number of ineffective fitness evaluations are eliminated and this time could be used to give more chances to the swarm placed at the global optimum peak. Therefore, it leads to performing more local searches in the neighborhood of the global optimum position, which results in an improvement in performance and accuracy of the algorithm.

In order to determine the time of sleeping for swarms in the problem space, r_{sleep} parameter has been used. For this reason, if the Euclidean distance of all artificial fishes in a swarm is less than this parameter, the swarm will be slept.

Pseudo code of the proposed algorithm mNAFSA is shown in Fig. 4.

5. Experimental study

In this section, the effect of different values of the parameters of the proposed algorithm is studied on the algorithm efficiency. The main objective in this section is to determine appropriate parameter values by performing various experiments. The experiments in this section are executed in Scenario 2 of MPB, which is

```
Multiswarm NAFSA(mNAFSA):
  //Initializing first swarm
  foreach AF j in first_swarm
      initialize $X_{1,j}$ randomly
  endfor
  initialize Test_Point randomly
  repeat
    foreach swarm
        Execute NAFSA procedure
    endfor
    if All swarm are converged then
        Create new_swarm and initialize it
    end if
  //Exclusion
    foreach swarm i
      if distance($bestAF_i$ and $bestAF_{new\_swarm}$) <$r_{excl}$ then
        if f($bestAF_i$) ≤ f($bestAF_{new\_swarm}$) then
            reinitialize Swarm i
        else
            reinitialize new_Swarm
        endif
      end if
    endfor
  //sleeping-awakening
  for each non best swarm i
      if distance($bestAF_i$ and $bestAF_{new\_swarm}$) <$r_{sleep}$ then
          Sleep[i]=True
      endif
  endfor
  //Test for Change
    Evaluate Test_Point
      if new value is different from last iteration then
        Wake up all asleep swarm
        foreach swarm i
          if $swarm_i$ is converged then
              keep $best\_AF_i$ and randomize others around it based on Shift_length
          end if
          Set $Visual_i$ based on Shift_ length
        endfor
      end if
  Until stopping criterion is met
```

Fig. 4. Pseudo code of multi-swarm NAFSA.

considered as one of the best-known benchmarks. MPB parameters setting for this scenario, also called standard setting for this benchmark [11,12], is illustrated in Table 1.

In order to measure the efficiency of the algorithms, *Offline Error* that is the moving average of the difference between fitness of the best solution found by the algorithm and fitness of the global optimum, is used.

$$offline\_error_{\text{1st Type}} = \frac{1}{FEs}\sum_{t=1}^{FEs}(fitness(gbest(t)) - fitness(globalOptimum(t))) \quad (15)$$

where FEs is the maximum fitness evaluations, and $gbest(t)$ and $globalOptimum(t)$ are the best position found by the algorithm and the global optimum at the tth fitness evaluation, respectively. In other word, the amount of *Offline Error* equals to the average of all *Current Errors* which is defined in time t as deviance between the best found position by the algorithm in time t in the current environment and the position of the global optimum in the current environments.

In some papers, another type of *Offline Error* is used [17,30,42] which is shown in the following equation:

$$offline\_error_{\text{2nd Type}} = \frac{1}{K}\sum_{k=1}^{K}(h_k - f_k) \quad (16)$$

where K is the number of environments, f_k is the best found solution by the algorithm in the kth environment and h_k is equal to the optimum value in the kth environment. Regarding Eq. (16), the second type of offline error is equal to the average of fitness of the best found position by the algorithm at the end of all environments. In fact, contrary to the first type of offline error, only the final result in each environment are considered and there is no consideration about the obtained result during the execution of each environment.

Since most of environment changes in real world dynamic optimization problems are occurred in form of continues in time and the environments are continuously changing, so the first type of *Offline Error* (Eq. (15)) would be more suitable for evaluating the

Table 1
Standard parameters setting for MPB.

Parameter	Value
Number of peaks, M	10
Change frequency	5000
Height change	7.0
Width change	1.0
Peaks shape	Cone
Basic function	No
Shift Severity	1.0
Number of dimensions, D	5
Correlation coefficient, λ	0
Peaks location range	[0–100]
Peak height	[30.0–70.0]
Peak width	[1–12]
Initial value of peaks	50.0

performance of the algorithms in this domain. Thus, the first type of offline error has been used in this paper and the proposed algorithm has been designed based on this type of offline error. It could be concluded from the course of experiments that by changing the involved parameters, the designed algorithm could be adapted for improving the second type of offline error.

It is worth noting that since the best obtained results in each environment are used for calculating the second type of offline error, the value of the second type of offline error is always better than that of the first type and it does not shows any superiority.

5.1. Parameter settings

In this section, first the effect of various values of parameters on the efficiency of NAFSA algorithm is studied in MPB problem with one peak. Then the effect of various values of mNAFSA parameters will be studied. Finally, it will be applied on different configurations of dynamic environments and its efficiency will be compared with some well-known algorithms.

The main metric for evaluating the performance of algorithms in this domain is *Offline Error* which is indicates the average of the best found position's fitness using algorithms during the running optimization process. In other word, the value of *Offline Error* is the average of *Current Errors. Current Error* at time t is the deviance of the best found position using algorithm at time t in the current environment and the position of global optimum in the current environment. The value of *Offline Error* is equal or greater than zero, where zero indicates the ideal situation.

5.1.1. Studying the efficiency of NAFSA with different configurations

Tests were performed in order to study the effects of different values of NAFSA parameters on its efficiency on MPB with one peak, 2500 fitness evaluations, without changing the environment and in a 5-dimensional space. All tests were repeated 50 times and the average results along with the standard error after 2500 evaluations were presented in related tables.

a) Effect of various values of algorithm population size and *try_number* value:
Population size and *try_number* value are two parameters which have correlations and are studied with each other. These two parameters determine the degree of performed fitness evaluation at each iteration and searching strategy. *Try_number* value specifies the amount of local search around each AF and population size shows the number of positions around which search should be performed. Experiments show the effect of various values of population size and *try_number*. Here, by default, *Visual* value was taken 20 and L_{low} was taken 0.8. Table 2 demonstrates obtained results for different configurations of NAFSA.
According to Table 2, the best result was obtained by configuration (2, 4) for population size and *try_number*. When *try_number* is 2, local search around every AF is considerably low to obtain acceptable results for any number of AFs. By increasing *try_number* to 4, the performance of the algorithm is enhanced. Actually, the best results were obtained with different population sizes and *try_number* equal to 4. Values greater than 4 for *try_number* result degrade the performance, since the number of performed fitness evaluations increase at each iteration and the algorithm is performed for fewer numbers of times. In this case, the cost of the number of performed evaluations of algorithm is more than its advancement at each iteration. This problem also exists for larger population sizes. According to results in Table 2, algorithm efficiency was decreased by increasing population size. Therefore, population size and *try_number* are taken 2 and 4, respectively.

b) Effect of various *Visual* values along with $L_{\min}$
In following, various values of *Visual* along with $L_{\min}$ are studied. These two parameters, in a similar manner of population size and *try_number*, have dependency to each other. In fact, *Visual* and $L_{\min}$ together can determine the primary ability of global search and its conversion rate into local search. As discussed in Section 4.1.4, first, a large value is considered for *Visual*. Then, it is decreased by Eq. (14), and $L_{\min}$ specifies its reduction rate. Table 3 shows the effect of various primary values of *Visual* and $L_{\min}$ on NAFSA performance with population size 2 and *try_number* 3.

As results demonstrated in Table 3, the performance of NAFSA with *Visual*=25 and $L_{\min}$=0.75 outperforms that of with other values for these two parameters. When *Visual* is considered smaller than 10, effectiveness decreases considerably. In this case, *Visual* is too small to do an acceptable global search. So it reaches too late to its objective and the convergence rate is too low. The reason for this convergence rate reduction is that the search space around every AF is small in Eq. (9) and *Step* length is very small in Eqs. (11) and (13), as well. If $L_{\min}$ is also considered small, algorithm efficiency becomes unacceptable for it that causes their *Step* length becomes extremely small and nearly to stop before reaching the objective. By increasing *Visual* and taking $L_{\min}$ about 0.8, the algorithm reaches fine results. In fact, local search is balanced well with global search and the algorithm can perform a good global search and proper local search.

5.1.2. Studying the efficiency of mNAFSA with different parameter value

In this section, like as Section 4.2, NAFSA is used as multi swarm (mNAFSA) for optimizing dynamic environments. Parameters are divided into two parts in mNAFSA, i.e. parameters of NAFSA swarms and parameters related to multi swarm mechanisms described in Section 4.2. In the previous section, the effect of different parameters of NAFSA was surveyed on its efficacy and proper values for all of its parameters were obtained. In this part, the effect of various values of related parameters to mNAFSA are discussed which were explained in Section 2.2.

a) The effect of various values of *Visual* after finding out the environment change
As explained in Section 4.2, because one of the local optima

Table 2
Obtained results using various values of population size (N) and *try_number* parameters.

(N, *Try_number*)	Off_err ± Std_error	(N, *Try_number*)	Off_err ± Std_err
(2, 2)	5.8275 ± 2.6851	(4, 2)	1.6048 ± 0.2905
(2, 3)	2.4130 ± 1.3661	(4, 3)	3.64e–05 ± 3.88e–06
(2, 4)	**2.42e–09 ± 3.21e–10**	(4, 4)	0.0003 ± 4.98e–05
(2, 5)	9.32e–08 ± 1.71e–08	(4, 5)	0.0023 ± 0.0002
(2, 7)	1.14e–05 ± 2.13e–06	(4, 7)	0.0267 ± 0.0036
(2, 10)	0.0006 ± 7.72e–05	(4, 10)	0.1709 ± 0.0186
(2, 15)	0.0143 ± 0.0013	(4, 15)	1.0893 ± 0.1108
(2, 20)	0.1174 ± 0.0110	(4, 20)	2.2069 ± 0.1807
(3, 2)	7.7404 ± 4.0610	(5, 2)	0.1043 ± 0.1042
(3, 3)	1.90e–07 ± 2.18e–08	(5, 3)	0.0045 ± 0.0005
(3, 4)	9.45e–06 ± 1.69e–06	(5, 4)	0.0004 ± 5.76e–05
(3, 5)	0.0001 ± 1.84–05	(5, 5)	0.0197 ± 0.0024
(3, 7)	0.0020 ± 0.0002	(5, 7)	0.1098 ± 0.0142
(3, 10)	0.0306 ± 0.0036	(5, 10)	0.4235 ± 0.0382
(3, 15)	0.2683 ± 0.0295	(5, 15)	1.9447 ± 0.1864
(3, 20)	0.8447 ± 0.0811	(5, 20)	3.6547 ± 0.3659

Table 3
Obtained results using various primary values of *Visual* (V) and $L_{\min}$.

V	$L_{\min}$	Off_err ± Std_error	V	$L_{\min}$	Off_err ± Std_error	V	$L_{\min}$	Off_err ± Std_error
5	0.95	0.03 ± 0.003	10	0.95	0.0760 ± 0.0061	15	0.95	0.1235 ± 0.0114
	0.9	5.9594 ± 4.2391		0.9	0.0003 ± 2.54e-05		0.9	0.0004 ± 4.14e-005
	0.8	222.8022 ± 28.0101		0.8	17.3333 ± 6.0815		0.8	0.0232 ± 0.0232
	0.75	232.43 ± 28.9920		0.75	83.7342 ± 13.7900		0.75	19.0188 ± 5.5565
	0.7	301.9345 ± 28.7206		0.7	140.1543 ± 22.2515		0.7	50.8755 ± 12.3201
	0.6	341.1569 ± 30.7649		0.6	205.7751 ± 25.1709		0.6	137.5714 ± 21.9111
20	0.95	0.2004 ± 0.0174	25	0.95	0.2157 ± 0.0197	30	0.95	0.2462 ± 0.0249
	0.9	0.0005 ± 6.56e–05		0.9	0.0007 ± 9.99e–05		0.9	0.0009 ± 8.13e–05
	0.8	2.81e–09 ± 6.39e–010		0.8	3.50e–09 ± 6.25e–10		0.8	6.57e–009 ± 1.36e–09
	0.75	11.9557 ± 6.3010		**0.75**	**2.56e–11 ± 5.48e–12**		0.75	5.88e–11 ± 1.22e–12
	0.7	13.1628 ± 4.3364		0.7	6.8349 ± 4.6837		0.7	0.0375 ± 0.0375
	0.6	109.0391 ± 18.9038		0.6	49.5295 ± 12.5796		0.6	19.6831 ± 5.2877

Table 4
The efficiency of the method using different values of P and *Severity*(S).

P	S	Off_err ± Std_err	P	S	Off_err ± Std_err	P	S	Off_err ± Std_err
0.1	1	2.45 ± 0.69	0.5	1	0.21 ± 0.01	0.9	1	0.30 ± 0.03
	2	3.27 ± 0.38		2	0.37 ± 0.04		2	0.36 ± 0.04
	5	8.41 ± 1.32		5	0.74 ± 0.10		5	0.90 ± 0.11
0.2	1	0.24 ± 0.01	0.6	1	0.22 ± 0.02	1	1	0.30 ± 0.02
	2	0.38 ± 0.04		2	0.36 ± 0.05		2	0.42 ± 0.06
	5	0.95 ± 0.09		5	0.76 ± 0.08		5	0.79 ± 0.12
0.3	1	0.22 ± 0.01	0.7	1	0.22 ± 0.01	2	1	0.41 ± 0.03
	2	0.28 ± 0.04		2	0.37 ± 0.04		2	0.65 ± 0.04
	5	0.73 ± 0.07		5	0.94 ± 0.10		5	1.61 ± 0.16
0.4	1	**0.19 ± 0.01**	0.8	1	0.22 ± 0.01	5	1	0.58 ± 0.07
	2	**0.27 ± 0.03**		2	0.38 ± 0.05		2	0.96 ± 0.11
	5	**0.67 ± 0.10**		5	1.00 ± 0.10		5	2.16 ± 0.20

may become the global optimum after each environment change, each existing peak in the problem space must be covered by a swarm. After the environment change, *Visual* value of swarms is reinitialized so that artificial fish are able to find new positions of their covered peaks. As discussed, the best AF position is maintained and other AFs in the swarm are distributed randomly in a cloud with a radius length of *Severity* around the best AF. Since the distance of the new peak position from its previous position before the environment change has changed at most as much as *Severity* in each dimension, therefore it could be concluded that *Visual* value has to be specified according to *Severity* ($Visual = P \times Severity$). Here, the effect of different values for *Visual* after the environment change is studied for AFs. Experiments were done on MPB with one peak and the environment change frequency 2500 and *Severity* values 1, 2 and 5. They were repeated 50 times and every time they were continued until 100 environment changes. Experimental results in this regards are tabulated in Table 4.

As it can be seen, when *Visual* value is considered equal to 0.4 times *Severity* ($P = 0.4$) after each environment change, the algorithm efficiency will increase. In fact, in this situation AFs move faster toward the new position of the peak. When *Visual* value is considered small, it is possible that before AFs can approach the goal, *Visual* value becomes so small by Eq. (14) that they can no longer move toward the goal and as a result the algorithm efficiency will decrease. Also, when *Visual* value is considered high, *Visual* search space gets larger than the space required by an AF to reach better positions in its search, and as a result, it decreases the possibility of finding better positions in prey behavior. Consequently, AFs progress will be slow while *Visual* value is getting more proper, which leads to a decrease in the algorithm efficiency.

b) Effect of different values of r_{conv} and k for determining the convergence of a swarm

At the beginning of algorithm execution in mNAFSA, there is only one swarm. Later when the swarm converges to a peak, a new swarm is initialized in the problem space. When the new swarm also converges to an uncovered peak, another new swarm is initialized in the problem space and in this way every peak in the problem space is covered by one swarm. Every newly initialized swarm is either converged to an uncovered peak where it resides, or converged to a previously covered peak and thus is removed by exclusion. In both cases, another new swarm is created in the problem space. This process continues until all peaks are finally covered. Even with all peaks covered, there is always one swarm in the problem space looking for an uncovered peak, and since all peaks are already covered, this swarm is removed after converging to a new peak, and then another swarm is initialized, which has the same ending. Therefore, after all peaks are covered, there will still be one extra swarm in the problem space.

The question, however, is when a swarm is converged. In order to determine convergence in mNAFSA, the Euclidean distance of the best AF position in its m_{th} iteration and in its $(m+k)_{\text{th}}$ iteration is measured. If this Euclidean position is less than r_{conv}, it means that the swarm has converged.

The experiments in this section have been performed on MPB with 10 peaks, *Severity* equal to 1, and a change frequency of 2500. The experiments in this section have also been repeated 50 times and each time with 100 environment changes. The values of NAFSA parameters have been according to the

Table 5
Effect of different values of r_{conv} and k on the efficiency of the proposed method.

r_{conv}	k	Off_err ± Std_err	r_{conv}	k	Off_err ± Std_err	r_{conv}	k	Off_err ± Std_err
0.01	1	2.51 ± 0.14	0.1	1	2.40 ± 011	1	1	2.40 ± 0.13
	2	2.27 ± 0.14		2	2.13 ± 010		2	2.09 ± 0.12
	3	2.12 ± 0.10		3	2.07 ± 0.08		3	2.00 ± 0.10
	4	2.22 ± 0.09		4	2.11 ± 0.09		4	2.09 ± 0.10
	5	2.33 ± 0.10		5	2.20 ± 0.10		5	2.20 ± 0.10
0.05	1	2.48 ± 0.11	0.5	1	2.34 ± 0.10	2	1	2.49 ± 0.12
	2	2.14 ± 0.11		2	2.05 ± 0.10		2	2.15 ± 0.12
	3	2.14 ± 0.09		3	**1.95 ± 0.08**		3	2.11 ± 0.11
	4	2.10 ± 0.10		4	2.02 ± 0.08		4	2.17 ± 0.11
	5	2.25 ± 0.11		5	2.13 ± 0.08		5	2.33 ± 0.12

experiments in previous sections. Since the changes in MPB are global, a test point has been used in order to discover changes in the environment. Table 5 shows the effect of different values for k and r_{conv}.
According to the results in Table 5, when the Euclidean distance of the best AF are calculated in iterations m and $m+3$ to determine a swarm convergence, the best results are obtained. Also, when r_{conv} is considered 0.5, the algorithm efficiency is improved. When the best positions of an AF are compared in two consecutive iterations to determine the swarm convergence, it cannot be determined correctly and it is possible that a swarm which has not converged yet to be wrongly considered convergent. This is due to this fact that the AF could not find a better position in one repetition before the swarm convergence. The algorithm efficiency is enhanced by increasing k up to 3, but values larger than 3 decrease efficiency because more repetitions will be needed to specify whether a swarm has converged and creating new swarms to find other uncovered peaks is postponed which causes degradation of efficiency.

c) Effect of different values of r_{sleep}
In Table 6, the results obtained with different values of r_{sleep} are shown. The results are obtained on MBP with 10 peaks, change frequency of 2500 and *Shift Severity* of 1.

Table 6
Effect of different values of r_{sleep} on the efficiency of the proposed method.

r_{sleep}	Off_err ± Std_err
0	1.99 ± 0.10
0.01	1.96 ± 0.10
0.1	1.91 ± 0.09
0.2	1.91 ± 0.10
0.3	1.87 ± 0.08
0.4	**1.83 ± 0.08**
0.5	**1.97 ± 0.08**
1	2.10 ± 0.11
2	2.30 ± 0.12

As we can see, introducing the sleeping–awakening mechanism with a proper value of r_{sleep} can improve the algorithm efficiency. When r_{sleep} is considered greater than or equal to 0.5, the algorithm efficiency declines. The reason for such degradation is that in this case, the swarms which are placed at local optimums in the current environment will be asleep too early, before they can find the opportunity to sufficiently approach the peak. Therefore, some swarms may get farther from their covered peaks. In this case, if one of these peaks becomes the global optimum after the environment change, it needs more time for reducing error, since the distance of the covering swarm is high. Thus, the efficiency is degraded.

On the other hand, considerable reduction in the value of r_{sleep} deteriorates the efficiency of sleeping–awakening mechanism as well, because it will take a long time for those swarms, which are awake in non-optimum peaks, to be asleep. Therefore, the number of fitness evaluations performed by the best swarm will decrease, and consequently the algorithm efficiency will also decrease. Among the values of r_{sleep}, the algorithm efficiency is improved, when $r_{sleep}=0.4$. Henceforth, the value of r_{sleep} will be considered 0.4.

5.2. Comparison with peer algorithms

5.2.1. Comparison of NAFSA with peer algorithms

In this section, NAFSA performance is compared with Standard AFSA, PSO [13,14] with *global star* and linear decreasing inertia weight (GPSO) [15], PSO with random inertia weight (RIWPSO) [16] and PSO with constriction factor (CF_PSO) [16]. The goal of these comparisons is to show that why standard AFSA cannot be used to construct an optimization algorithm in dynamic environments and why using NAFSA is appropriate for optimization in these environments. Population size and *try_number* in AFSA were considered the same as NAFSA. Crowd factor was taken 0.5, *Visual* and *Step* were taken 10 and 5, respectively [17]. For PSO algorithms, population size was 5 [10] and other parameters were considered equal to their references. Obtained results are tabulated in Table 7.

Table 7
Comparison between proposed method and other comparative studies.

Algorithm	Off_err ± Std_err
AFSA [1]	36.5331 ± 3.2439
RIW-PSO [16]	1.58e−04 ± 5.92e−04
GPSO [15]	1.23e−05 ± 4.61e−10
CF-PSO [16]	9.88e−09 ± 2.72e−08
NAFSA	**2.56e−11 ± 5.48e−12**

Fig. 5 illustrates the convergence behavior related to the five algorithms for optimizing MPB with one peak and up to 2500 fitness evaluations. As it is observed, NAFSA involves high convergence rate which makes it suitable to be applied in dynamic environments, since proposed algorithms in these environments, because of the environment change, have to be of high convergence rate. Among PSO algorithms, CF-PSO has a higher convergence rate and has reached better results which leads to utilizing this version for dynamic environments.

Standard AFSA has received the worst result. Considering weaknesses of AFSA stated in Section 2.2, it cannot reach acceptable results and has a low convergence rate too. Consequently, using this algorithm to design optimization algorithms for dynamic environments is not proper at all. In this paper, after solving AFSA drawbacks and improving it, NAFSA has been presented which, with respect to its high convergence rate and good obtained results, has been applied as mNAFSA for optimization in dynamic environments.

5.2.2. Comparison of mNAFSA with peer algorithms

In this section, the efficacy of the proposed algorithm is compared with peer algorithms for optimizing different configurations of MPB. Table 8 represents parameters values of mNAFSA.

The experiments using configuration presented in Table 1 on MPB is done and the performance of the proposed algorithm is compared with four other algorithms with change frequencies of 2500 and 10,000 in Table 9. In Table 10, the result of change frequencies of 5000 is presented. The experiments are repeated 50 times and each repeat is done until 100 environment changes. Other results of Tables 9 and 10 are extracted from the related works.

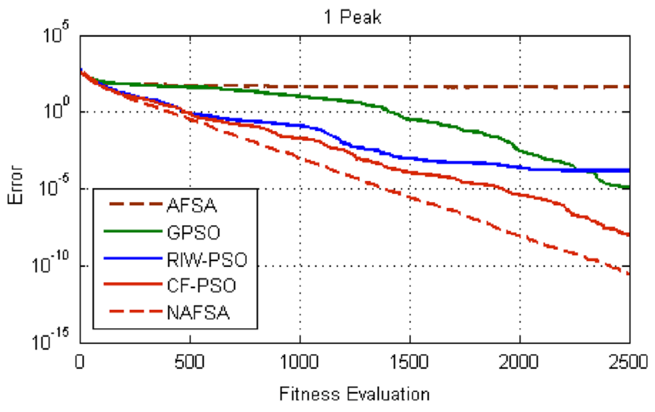

Fig. 5. Convergence behavior of the proposed algorithm and four comparative algorithms for optimizing MPB with one peak and up to 2500 fitness evaluations.

Table 8
Value configuration of mNAFSA parameters.

Parameter	Value
Population size	2
try_number	4
Initial *Visual*	25
After change *Visual*	0.4 × *Severity*
$L_{\min}$	0.75
r_{conv}	0.5
r_{excl}	d_{boa}(Blackwell and Branke [9])
K	3

With respect to Tables 9 and 10, the performance of mNAFSA is better than the other algorithms indifferent conditions of MPB with various numbers of peaks and change frequencies. By increasing the environment change frequency, algorithms efficiencies are enhanced, thanks to this fact that there is a more time to find better positions before the environment change.

Among different frequencies of the environment change in MPB, the one with the value 5000 is more considerable and more well-known amongst others. As it is observed in Table 10, the proposed algorithm has a better efficiency than other algorithms. In the proposed algorithm, since the number of swarms is based on the number of found peaks, the algorithm efficiency is significantly high both when the number of peaks is low and when the number of peaks is high.

Table 11 illustrates the performance of the proposed algorithm on MPB using 10 peaks, change frequency of 5000 and the *Severities* of 1, 2, 3, 4 and 5. As can be seen, by increasing the *Severity* value, the performance of the algorithm is degraded, due to this fact that the distance of the new positions of the peaks are far away compared to that of previous ones after environment change. Thus, the value of the current value is grater after environment changes. On the other hand, because of increasing the distance of the peaks, the algorithms must perform additional fitness evaluation processes. Fig. 6 shows the *Offline Error* and *Current Error* curves of the proposed algorithm in MPB with 5000 change frequencies, 10 picks and the *Severities* of 1 and 5 up to 100 environment changes. These curves are obtained using 50 iterations. As can be seen, the value of *Current Error* is higher using *Severity* of 5, compared to *Severity* of 1, after each environment change.

Table 12 shows the results of the proposed method with different dimensions involving 10 peaks, change frequency of 5000 and *Shift Severity* of 1, in addition to those of mQSO, AmQSO, RPSO and mPSO. As can be seen, by increasing dimension of the problem space, the performance of the proposed method is less degraded, compared to other algorithms. *Offline Error* and standard error curves of the proposed algorithm in two dimensional and 20 dimensional spaces are shown in Fig. 7.

5.2.3. Statistical analysis

To compare the efficiency of the algorithms, only the average results of the algorithms are not sufficient. So that, in order to perform a more precise analysis of the proposed method and prove its efficiency, the obtained results of the proposed algorithm have been compared with those of other comparative methods by performing a statistical analysis. The analysis which has been done in this paper is one sample *t*-test approach [45]. H_0 and H_1

Table 9
Offline Error (standard error) of the proposed method and 4 comparative algorithms on MPB problem with different number of peaks and change frequencies of 2500 and 10,000.

ALG.	C-F	Number of peaks							
		1	5	10	20	30	50	100	200
mQSO(5,5^q) [9]	2500	7.64(0.64)	3.26(0.21)	3.12(0.14)	3.58(0.13)	3.63(0.10)	3.63(0.10)	3.58(0.08)	3.30(0.06)
AmQSO [10]		0.87(0.11)	2.16(0.19)	2.49(0.10)	2.73(0.11)	3.24(0.18)	3.68(0.15)	3.53(0.14)	3.07(0.12)
mPSO [23]		1.79(0.10)	2.04(0.12)	2.66(0.16)	3.07(0.11)	3.15(0.08)	3.26(0.07)	3.31(0.05)	3.36(0.05)
APSO [27]		1.06(0.03)	1.55(0.05)	2.17(0.07)	2.51(0.05)	2.61(0.02)	2.66(0.02)	2.62(0.02)	2.64(0.01)
mNAFSA		**0.45(0.06)**	**1.30(0.05)**	**1.83(0.08)**	**2.36(0.09)**	**2.44(0.09)**	**2.49(0.08)**	**2.60(0.07)**	**2.61(0.06)**
mQSO(5,5^q) [9]	10,000	1.90(0.18)	1.03(0.06)	1.10(0.07)	1.84(0.08)	2.00(0.09)	1.99(0.07)	1.85(0.05)	1.71(0.04)
AmQSO [10]		**0.19(0.02)**	0.45(0.04)	0.76(0.06)	1.28(0.12)	1.78(0.09)	1.55(0.08)	1.89(0.14)	2.52(0.10)
mPSO [23]		0.27(0.02)	0.70(0.10)	0.97(0.04)	1.34(0.08)	1.43(0.05)	1.47(0.04)	1.50(0.03)	1.48(0.02)
APSO [27]		0.25(0.01)	0.57(0.03)	0.82(0.02)	1.23(0.02)	1.39(0.02)	1.46(0.01)	1.38(0.01)	1.36(0.01)
mNAFSA		**0.22(0.03)**	**0.39(0.03)**	**0.45(0.02)**	**0.62(0.03)**	**0.90 (0.03)**	**1.13 (0.04)**	**1.23(0.04)**	**1.31(0.04)**

Table 10
Offline Error (standard error) of the proposed method and 15 comparative algorithms on MPB problem with different number of peaks and change frequency 5000.

ALG.	Number of peaks							
	1	**5**	**10**	**20**	**30**	**50**	**100**	**200**
mQSO(5,5^q) [9]	4.92(0.21)	1.82(0.08)	1.85(0.08)	2.48(0.09)	2.51(0.10)	2.53(0.08)	2.35(0.06)	2.24(0.05)
AmQSO [10]	0.51(0.04)	1.01(0.09)	1.51(0.10)	2.00(0.15)	2.19(0.17)	2.43(0.13)	2.68(0.12)	2.62(0.10)
CLPSO [18]	2.55(0.12)	1.68(0.11)	1.78(0.05)	2.61(0.07)	2.93(0.08)	3.26(0.08)	3.41(0.07)	3.40(0.06)
FMSO [19]	3.44(0.11)	2.94(0.07)	3.11(0.06)	3.36(0.06)	3.28(0.05)	3.22(0.05)	3.06(0.04)	2.84(0.03)
RPSO [20]	0.56(0.04)	12.22(0.76)	12.98(0.48)	12.79(0.54)	12.35(0.62)	11.34(0.29)	9.73(0.28)	8.90(0.19)
mCPSO [9]	4.93(0.17)	2.07(0.08)	2.08(0.07)	2.64(0.07)	2.63(0.08)	2.65(0.06)	2.49(0.04)	2.44(0.04)
SPSO [21]	2.64(0.10)	2.15(0.07)	2.51(0.09)	3.21(0.07)	3.64(0.07)	3.86(0.08)	4.01(0.07)	3.82(0.05)
rSPSO [22]	1.42(0.06)	1.04(0.03)	1.50(0.08)	2.20(0.07)	2.62(0.07)	2.72(0.08)	2.93(0.06)	2.79(0.05)
mPSO [23]	0.90(0.05)	1.21(0.12)	1.61(0.12)	2.05(0.08)	2.18(0.06)	2.34(0.06)	2.32(0.04)	2.34(0.03)
PSO-CP [24]	3.41(0.06)	–	1.31(0.06)	–	2.02(0.07)	–	2.14(0.08)	2.04(0.07)
RVDEA [25]	1.02(–)	–	3.54(–)	3.87(–)	3.92(–)	3.78(–)	3.37(–)	3.54(–)
SFA [26]	0.42(0.07)	0.89(0.09)	1.05(0.04)	1.48(0.05)	1.56(0.06)	1.87(0.05)	2.01(0.04)	1.99(0.06)
APSO [27]	0.53(0.01)	1.05(0.06)	1.31(0.03)	1.69(0.05)	1.78(0.02)	1.95(0.02)	1.95(0.01)	1.90(0.01)
CLDE [28]	1.53(0.07)	1.50(0.04)	1.64(0.03)	2.46(0.05)	2.62(0.05)	2.75(0.05)	2.73(0.03)	2.61(0.02)
DynPopDE [29]	–	1.03(0.13)	1.39(0.07)	–	–	2.10(0.06)	2.34(0.05)	2.44(0.05)
DMAFSA [31]	0.59(0.06)	0.66(0.05)	0.94(0.04)	1.29(0.05)	1.60(0.06)	1.81(0.06)	1.92(0.05)	1.97(0.05)
mNAFSA	**0.38(0.06)**	**0.55(0.04)**	**0.90 (0.03)**	**1.25 (0.06)**	**1.47(0.05)**	**1.65(0.05)**	**1.83(0.05)**	**1.84(0.05)**

Table 11
Offline Error (standard error) of the proposed method and 8 comparative algorithms on MPB using 10 peaks, change frequency of 5000 and the *Severities* of 1–5.

Algorithm	*Shift Severity*				
	1	2	3	4	5
mQSO(5,5^q) [9]	1.85(0.08)	2.40(0.06)	3.00(0.06)	3.59(0.10)	4.24(0.10)
mCPSO [9]	2.08(0.07)	2.80(0.07)	3.57(0.08)	4.18(0.09)	4.89(0.11)
AmQSO [10]	1.51(0.10)	2.09(0.08)	2.75(0.09)	3.36(0.11)	3.71(0.11)
SPSO [21]	2.51(0.09)	3.78(0.09)	4.28(0.10)	4.96(0.12)	6.76(0.15)
rSPSO [22]	1.50(0.08)	1.87(0.05)	2.40(0.08)	2.90(0.08)	3.25(0.09)
mPSO [23]	1.61(0.12)	2.54(0.11)	3.46(0.11)	4.51(0.15)	5.32(0.09)
CLPSO [18]	1.78(0.05)	2.72(0.10)	3.70(0.11)	4.71(0.14)	5.68(0.18)
SFA [26]	1.05(0.04)	1.44(0.06)	2.06(0.07)	2.45(0.09)	2.89(0.13)
mNAFSA	**0.90(0.03)**	**1.41(0.06)**	**1.77(0.07)**	**2.15(0.09)**	**2.58(0.10)**

hypotheses have been defined as follows for performing the test:

$$H_0 : \mu_j = a_j^*, \qquad (17)$$

$$H_1 : \mu_j \neq a_j^* \qquad (18)$$

$$a_i = \arg\min\ (\mu_{i,j})^* \qquad (19)$$

*i ∈ all algorithms, j ∈ all scenarios

In which a_j is the average result of the best algorithm performed on j scenario and μ_j is that of the proposed method performed on j scenario. Since, the number of samples (the number of performing algorithm) is more than 30 ($n=50>30$), test of normal distribution of the samples is not needed, based on the central limit theory. So that *t*-test could be utilized for evaluating the results. *t*-Test approach in form of two-tailed test is performed with 49 degrees of freedom, where alpha is 0.01. The obtained results by *t*-test approach show that in 30 cases out of 36 cases presented in Tables 9–12, H_0 is rejected with 99% confidence level (1% significant level). This means that the average results obtained from the proposed algorithm is better than those obtained from the best comparative algorithm in these 30 cases with 99% confidence level. In two cases presented in Tables 9 ($f=10{,}000$, $m=1$) and Table 10 ($f=5000$, $m=200$), the obtained results of the proposed method is not the best one. In four other cases, the test fail to reject H_0 with $\alpha=0.01$ significant level. Three out of these four cases are presented in Table 9 ($f=2500$, $m=100$), ($f=10{,}000$, $m=200$) and ($f=2500$, $m=200$) and one of them is presented in Table 10 ($f=5000$, $m=200$). As can be seen, most of these cases are the cases in which the number of peaks are considerable ($m=100$ and 200). In fact, the efficiency of the proposed algorithm has been deteriorated, because of time consuming process for finding peaks and existing considerable number of swarms in the problem space. In this situation, there are a few algorithms particularly the proposed algorithms which their efficiency is not noticeably degraded.

6. Conclusion

In this paper, for the first time, a method was proposed for optimization in dynamic environments based on artificial fish swarm algorithm. Designed algorithm for optimization in static environment could be utilized as the base algorithm in order to propose dynamic optimization algorithms. Regarding environment changes in dynamic environment, convergence speed is one of the most prominent characteristics of the base algorithms. Artificial fish swarm is a population based algorithm in which each individual could execute Prey behavior for performing a significant local search. Combination of individual Prey behavior and group behaviors leads to an appropriate convergence speed in this algorithm. In this paper, a novel model of AFSA, so called NAFSA, has been presented that involves an appropriate local search ability and considerable convergence speed. This algorithm was employed as the base algorithm for optimization in dynamic environments. Subsequently after designing NAFSA, it was extended to conquer particular challenges of dynamic environment. This algorithm, so called mNAFSA, was a multi-swarm algorithm based on NAFSA in which several mechanisms were involved to conquer the challenges. Modified multi-swarm mechanism was used for finding and covering potential optimum peaks and new diversity increase mechanism was applied after detecting an environment change.

Results of the proposed algorithm were evaluated on moving peak benchmark and were compared with those of several state-of-the-art methods in this domain. Experimental results showed the high efficiency of the proposed algorithm, in terms of locating and tracking optimums, convergence speed and a local search ability.

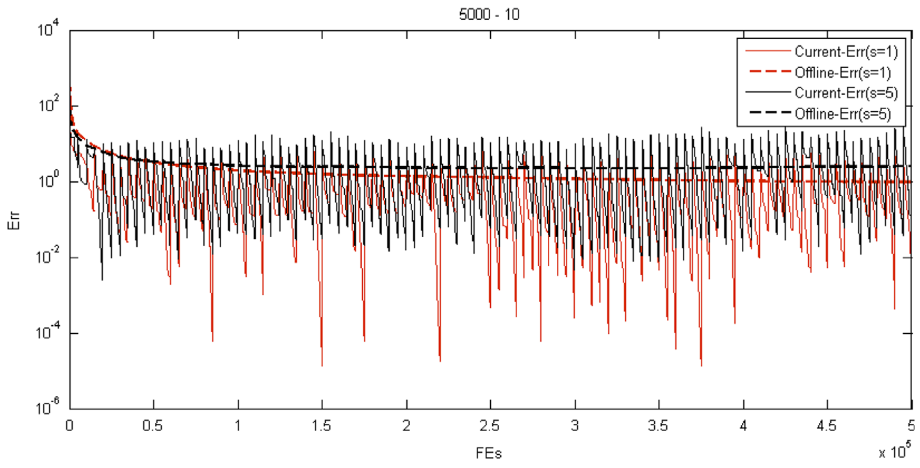

Fig. 6. *Offline Error* and *Current Error* curves of the proposed algorithm on MPB using the *Severities* of 1 and 5.

Table 12
Offline Error (standard error) of the proposed method and 8 comparative algorithms on MPB with different dimensions, change frequency of 5000, *Shift Severity* of 1 and 10 peaks.

Algorithm	Dimension						
	2	3	4	5	10	15	20
mQSO [9]	1.01(0.04)	1.49(0.09)	1.47(0.08)	1.85(0.08)	4.22(0.20)	6.50(0.33)	8.88(0.34)
AmQSO [10]	0.71(0.05)	1.16(0.10)	1.33(0.08)	1.51(0.10)	3.37(0.22)	4.91(0.31)	5.83(0.29)
mPSO [23]	1.24(0.07)	1.42(0.10)	1.35(0.09)	1.61(0.12)	4.32(0.26)	7.07(0.25)	10.77(0.40)
RPSO [20]	2.62(0.08)	6.61(0.33)	10.43(0.54)	12.98(0.48)	16.87(0.83)	18.48(0.97)	18.48(0.94)
mNAFSA	**0.43(0.04)**	**0.60(0.06)**	**0.75(0.05)**	0.90(0.03)	**2.36(0.28)**	**3.56(0.31)**	**4.30(0.35)**

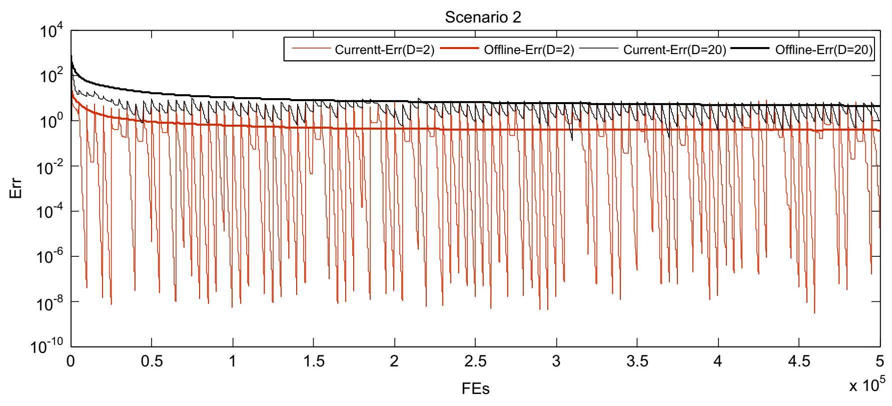

Fig. 7. *Offline Error* and *Current Error* curves of the proposed algorithm on MPB with dimensions of 2 and 20.

The proposed algorithm was designed for optimizing those dynamic environments that have similar behaviors and environment properties to moving peak benchmark. Different dynamic environments may represent different dynamic behaviors. Therefore, algorithms that are designed to optimize them have to meet their various requirements and conquer challenges. As a result, applied mechanisms in these algorithms are designed with respect to the dynamic type of these environments. Therefore, proposed algorithms for specific types of dynamic environments cannot be applicable for other types of dynamic environments because, firstly, applied mechanisms in some types of algorithms may not resolve challenges of other types. Secondly, some of these mechanisms result in algorithm efficiency reduction in other dynamic environments. Consequently, the proposed algorithm cannot be used for optimizing other types of dynamic environments with completely different behaviors.

Although a number of swarm and evolutionary approaches have been developed for solving dynamic optimization problems, new efficient approaches [44] are still greatly needed to address different types of dynamic optimization problems. Hence, it is also worthy to further develop and investigate new methods for dynamic optimization in the future.

The problems which could be pursued as future work of this research are as follow: The modification of the algorithm based on other types of dynamic environment could be the first line of future work. Second, the proposed algorithm could be applied to real-world dynamic problems and its efficiency can be evaluated. Using learning, adjusting and self-adjusting parameters in the structure of algorithm can be useful in order to decrease the number of parameters and the efficiency of the algorithm which could be pursued as the next line. Finally, changing the involved parameters of the algorithm will be considered for improving the second type of offline error.

References

[1] L.X. Lei, S.Z. Jiang, Q.J. Xin, An optimizing method based on autonomous animals: fish swarm algorithm, Syst. Eng.: Theory Pract. 22 (2002) 32–38.

[2] Y. Luo, J. Zhang, X. Li, The optimization of PID controller parameters based on artificial fish swarm algorithm, in: Proceedings of the IEEE International Conference on Automation and Logistics, 2007, pp. 1058–1062.

[3] M. Jiang, K. Zhu, Multiobjective optimization by artificial fish swarm algorithm, in: Proceedings of the IEEE International Conference on Computer Science and Automation Engineering, vol. 3, 2011, pp. 506–511.

[4] A.M.A.C. Rocha, T.F.M.C. Martins, E.M.G.P. Fernandes, An augmented Lagrangian fish swarm based method for global optimization, J. Comput. Appl. Math. 235 (16) (2011) 4611–4620.

[5] H.C. Tsai, Y.H. Lin, Modification of the fish swarm algorithm with particle swarm optimization formulation and communication behavior, Appl. Soft Comput. 11 (8) (2011) 5367–5374, http://dx.doi.org/10.1016/j.asoc.2011.05.022.

[6] S. He, N. Belacel, H. Hamam, Y. Bouslimani, Fuzzy clustering with improved artificial fish swarm algorithm, in: Proceedings of the International Joint Conference on Computational Sciences and Optimization, vol. 2, 2009, pp. 317–321.

[7] D. Yazdani, H. Nabizadeh, E.M. Kosari, A.D. Toosi, Color quantization using modified artificial fish swarm algorithm, Lect. Notes Comput. Sci. 7106 (2011) 382–391, http://dx.doi.org/10.1007/978-3-642-25832-9_39.

[8] Y. Jin, J. Branke, Evolutionary optimization in uncertain environments—a survey, IEEE Trans. Evol. Comput. 9 (3) (2005) 303–317.

[9] T. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, IEEE Trans. Evol. Comput. 10 (4) (2006) 459–472.

[10] T. Blackwell, J. Branke, X. Li, Particle swarms for dynamic optimization problems, Swarm Intelligence-Introduction and Applications, Book Part: 2, Natural Computing Series, Springer, (2008) 193–217, http://dx.doi.org/10.1007/978-3-540-74089-6_6.

[11] J. Branke, Memory enhanced evolutionary algorithms for changing optimization problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 3, 1999, pp. 1875–1882.

[12] J. Branke, The Moving Peaks Benchmark, 〈http://people.aifb.kit.edu/jbr/MovPeaks/〉.

[13] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.

[14] Z.H. Zhan, J. Zhang, Y. Li, H.S.H. Chung, Adaptive particle swarm optimization, IEEE Trans. Syst., Man, Cybern., Part B: Cybern. 39 (6) (2009) 1362–1381.

[15] Y. Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the IEEE International Conference on Evolutionary Computation, CEC98, 1998, pp. 69–73.

[16] R.C. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: Proceedings of the Congerss on Evolutionary Computation, 2000, pp. 84–88.

[17] C. Li, S. Yang, A general framework of multipopulation methods with clustering in undetectable dynamic environments, IEEE Trans. Evol. Comput. 16 (2011) 556–577.

[18] A.B. Hashemi, M.R. Meybodi, Cellular PSO: a pso for dynamic environment, in: Proceedings of the 4th International Conference on Intelligence Computation and Applications (ISICA 2009), Lecture Notes in Computer, 2009.

[19] C. Li, S. Yang, Fast multi-swarm optimization for dynamic optimization problems, in: Proceedings of the ICNC '08, 4th International Conference on Natural Computation, vol. 7, 2008, pp. 624–628.

[20] X. Hu, R.C. Eberhart, Adaptive particle swarm optimization: detection and response to dynamic systems, in: Proceedings of the IEEE Congeress on Evolutionary Computation, vol. 2, 2002, pp. 1666–1670.

[21] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Inf. Sci. 178 (15) (2008) 3096–3109, http://dx.doi.org/10.1016/j.ins.2008.01.020.

[22] S. Bird, X. Li, Using regression to improve local convergence, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 2007, pp. 592–599.

[23] M. Kamosi, A.B. Hashemi, M.R. Meybodi, A new particle swarm optimization algorithm for dynamic environment. Swarm, Evolutionary, and Memetic Computing, SEMCO 2010, Lect. Notes in Comput. Sci. 6466 (2010) 129–138.

[24] L. Liu, S. Yang, D. Wang, Particle swarm optimization with composite particles in dynamic environments, IEEE Trans. Syst., Man, Cybern., Part B: Cybern. 40 (6) (2010) 1634–1648.

[25] Y.G. Woldesenbet, G.G. Yen, Dynamic evolutionary algorithm with variable relocation, IEEE Trans. Evol. Comput. 13 (3) (2009) 500–513.

[26] B. Nasiri, M.R. Meybodi, Speciation based firefly algorithm for optimization in dynamic environments, Int. J. Artif. Intell. 8 (S12) (2012) 118–132.

[27] I. Rezazadeh, M.R. Meybodi, A. Naebi, Adaptive particle swarm optimization algorithm for dynamic environments, Lect. Notes Comput. Sci. 6728 (2011) 120–129, http://dx.doi.org/10..1007/978.3.642-21515-5_15.

[28] V. Noroozi, A.B. Hashemi, M.R. Meybodi, CellularDE: a cellular based optimization algorithm for dynamic environments, in: Proceedings of the 14th International Conference on Genetic and Evolutionary Computation (GECCO 2012), 2012, pp. 1519–1520.

[29] M.C. Plessis, A.P. Engelbrecht, Differential evolution for dynamic environments with unknown numbers of optima, J. Glob. Optim. 55 (1) (2012) 73–99, http://dx.doi.org/10.1007/s10898-012-9864-9.

[30] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Trans. Evol. Comput. 4 (6) (2010) 959–974.

[31] D. Yazdani, M.R. Akbarzadeh-T, B. Nasiri,M.R. Meybodi, A new artificial fish swarm algorithm for dynamic optimization problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC2012, Australia, 2012.

[32] T.T. Nguyen, Continuous dynamic optimisation using evolutionary algorithms (Ph.D. thesis), The University of Birmingham, 2010.

[33] F. Oppacher, M. Wineberg, The shifting balance genetic algorithm: Improving the GA in a dynamic environment, in: Proceedings of the Genetic and Evolutionary Computation Conference, 1999, pp. 504–510.

[34] J. Branke, H. Kaußler, C. Schmidt, H. Schmeck, A multi-population approach to dynamic optimization problems, Adapt. Comput. Design Manuf. 6 (2000) 299–307, http://dx.doi.org/10.1007/978-1-4471-0519-0_24.

[35] H. Cheng, S. Yang, Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks, Appl. Evol. Comput. (2010) 562–571.

[36] R.I. Lung, D. Dumitrescu, A new collaborative evolutionary-swarm optimization technique, in: Proceedings of the Companion on Genetic and Evolutionary Computation GECCO, 2007, pp. 2817–2820.

[37] R.K. Ursem, Multinational GAs: multimodal optimization techniques in dynamic environments, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2000, pp. 19–26.

[38] X. Li, J. Branke, T. Blackwell, Particle swarm with speciation and adaptation in a dynamic environment, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006, pp. 51–58.

[39] D. Parrott, X. Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, IEEE Trans. Evol. Comput. 10 (2006) 440–458.

[40] D. Parrott, X. Li, A particle swarm model for tracking multiple peaks in a dynamic environment using speciation, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, CEC2004, 2004, pp. 98–103.

[41] C. Li, S. Yang, A clustering particle swarm optimizer for dynamic optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC '09, 2009, pp. 439–446.

[42] I. Moser, R. Chiong, Dynamic function optimisation with hybridised extremal dynamics, Memet. Comput. 2 (2) (2010) 137–148.

[43] T.T. Nguyen, S. Yang, J. Branke, Evolutionary dynamic optimization: a survey of the state of the art, Swarm Evol. Comput. 6 (2012) 1–24.

[44] R. Kumar, R.A. Gupta, A.K. Bansal, Economic analysis and power management of a stand-alone wind/photovoltaic hybrid energy system using biogeography based optimization algorithm, Swarm Evol. Comput. 8 (2013) 33–43.

[45] S.J. Coakes, L.G. Steed, SPSS: Analysis Without Anguish: Versions 7.0, 7.5, 8.0 for Windows, John Wiley & Sons, Brisbane; Chichester, 1999.