JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 32, 661-676 (2016)

Improved Speciation-Based Firefly Algorithm in Dynamic and Uncertain Environments

BABAK NASIRI[1,*] AND MOHAMMAD REZA MEYBODI[2]
[1]*Department of Computer and Information Technology Engineering*
Islamic Azad University
Qazvin Branch, Qazvin, 34185-1416 Iran
E-mail: Nasiri.babak@qiau.ac.ir
[2]*Department of Computer Engineering and Information Technology*
Amirkabir University of Technology
Tehran, 15875-4413 Iran
E-mail: mmeybodi@aut.ac.ir

Many real-world optimization problems are dynamic in nature. The applied algorithms in this environment can pose serious challenges, especially when the search space is multimodal with multiple, time-varying optima. To address these challenges, this paper proposed a speciation-based firefly algorithm to maintain the population diversity in different areas of the landscape. To improve the performance of the algorithm, multiple adaptation techniques have been used such as adapting the number of species, number of fireflies in each specie and number of active fireflies in each specie. A set of experiments are conducted to study the performance of the proposed algorithm on Moving Peaks Benchmark (MPB) which is currently the most well-known benchmark for evaluating algorithm in dynamic environments. The experimental results indicate that the proposed algorithm statistically performs better than several state-of-the-art algorithms in terms of offline-error.

Keywords: firefly algorithm, multi-swarm, dynamic optimization, speciation, dynamic and uncertain environments

1. INTRODUCTION

Many real-world optimization problems are dynamic in the nature. That's why, in the past decade, there has been a growing interest in proposing new approaches for optimization in dynamic environments [1]. Generally, the goals and challenges are completely different for optimization in dynamic environments compared to the static ones. The only goal in static optimization problems is finding the global optimum as close as possible. In dynamic ones, however, tracking optima over the time is the other goal of optimization, as well. Furthermore, there are some more challenges in dynamic environments compared to the static ones which Out-dated memory and diversity loss are two of the most important ones.

Generally speaking, so far, different methods have been proposed for optimization in dynamic environments. A great category of such methods is meta-heuristic ones, more specifically evolutionary computing and swarm intelligence methods. In this category, in each iteration, the population changes to move toward to a better one. Due to this intrinsic capability, these methods are more suitable for optimization in dynamic environments and various techniques have been proposed in the last two decades, such as GA [2-5],

Received December 21, 2014; revised February 18, 2015; accepted March 26, 2015.
Communicated by Tzung-Pei Hong.

AC [6, 7], EDA [8] and PSO [9-13]. In addition, the authors proposed a speciation-based firefly algorithm in [14] by using the intrinsic capability of firefly algorithm in converging to more than one optimum at a time. Although, this approach seems ideal, it does suffer some shortcomings. First, some redundant fitness evaluations around the local optima which have not any influence on decreasing error of the optimization at all. Second, inability of the algorithm to cover a number of local optima due to the fixed number of fireflies in each speciation and also the weakness of the algorithm in global search.

In this paper, a new speciation method based on firefly algorithm has been proposed which suggests a set of adaptation techniques with the problem space for improving the performance of dynamic optimization. In this approach, all the mentioned shortcomings of the previous one are resolved completely. To evaluate the proposed method like the previous one, MPB benchmark has been used which is the best-known benchmark for evaluating dynamic optimization approaches in literature. To assess the proposed method more accurately, different scenarios and configurations of this benchmark have been considered and the efficiency of the proposed method for optimization has been evaluated and compared with state-of-the-art methods in this domain.

The rest of the paper is organized as follows: In section 2, the background about dynamic optimization and firefly algorithm will be provided. Section 3 explains the proposed method for solving optimization problems in dynamic environments. Section 4 is dedicated to experimental setup which includes benchmark function, performance metric and parameters setting. In section 5, the results and the comparison between the proposed approach and other approaches will be reviewed. The last section includes the conclusion of the paper and presents future works.

2. BACKGROUND

2.1 Dynamic Optimization

Using meta-heuristic methods for dynamic optimization has some challenges, which do not exist in static environments. Among them, outdated memory and diversity loss are two of the most important ones. The outdated memory happens when the environment changes and the fitness value of the obtained solutions will no longer correspond to the stored value in the memory. However, diversity loss occurs because of the intrinsic nature of meta-heuristic methods for convergence. The easiest way to solve these two issues is re-initialization [2]. But, it is only recommended when the changes in the environment are major and numerous. As the efficiency of the optimization process in the changed environment could be improved using the knowledge acquired from the previous environment, the re-initialization methods imply the loss of all the knowledge obtained so far from the problem space. In the following, the presented solutions for resolving these two main challenges are studied in details.

The first challenge, *i.e*, the outdated memory, is of less importance compared with the other one, and two solutions have been proposed for it, which are forgetting memory and re-evaluating memory [15]. In forgetting memory method, the position stored for each solution will be replaced by its position in the new environment. However, in re-evaluating memory method, the stored positions in the memory are re-evaluated. The second challenge, *i.e*, diversity loss, has observed many solutions, which have been clas-

sified into three main categories.

2.1.1 Presenting diversity method

In this category, the algorithms allow diversity loss to occur, and afterward they try to solve it. They are divided into two subcategories:

Mutation and Self-Adaptation: In this subcategory, the algorithms try to generate the lost diversity in the environment by performing mutation and self-adaptation [16-20].

Other Approaches: There are other methods proposed for creating diversity in the environment after discovering a change in the environment. In [21], a method called "RPSO" has been proposed for randomizing some or all of the solutions after detecting changes in the environment. In [22], another algorithm called "PBIL" (Population-Based Incremental Learning) is presented which uses a flexible probability vector for generating solutions. The vector in this algorithm is utilized for adjusting the learning rate after the change.

2.1.2 Diversity maintenance method

In this method, it is trying to keep diversity in the environment at all times (before and after the change). The proposed algorithms in this category are divided into three subcategories:

Dynamic Topology: Algorithms in this subcategory try to decrease the rate of algorithm convergence into a global optimum by limiting the existing communications among solutions, while keeping the diversity in the environment. In [23], HPSO has been proposed for preserving diversity, with its hierarchical and pseudo-tree structure. Another method which is called "CellularPSO", [24] has been suggested for dynamic optimization. In [25], the presented model in [24] has been improved by changing the roles of some particles to quantum particles just after the environment changes.

*Memory-Based Method*s: When the environment changes are periodical or recurrent, it will be very useful to store previous optimal solutions for using in the future. Memory-based methods try to store such information. The Memory-based methods can be divided into two main subclasses of implicit and explicit. In implicit methods, memory is integrated with meta-heuristic methods as a redundant representation [26, 27]. However, in explicit methods, memory is created explicitly.

Other approaches: Other methods have also been presented for maintaining the diversity in the environment after detecting the changes in the environment. In [28], a Sentinel Placement method is used for diversity maintenance. In this method, a series of sentinels distributed in the search space are utilized in order to generate a new population. In [3], Random Immigrant method has been proposed in every generation, in which a number of random solutions are added to the population to preserve diversity. In [29], a multi-objective method is proposed in which two goal functions are implemented. The first one is the main goal function, and the second one is a goal function for keeping diversity in the environment.

2.1.3 Hybrid methods

This category is a hybrid of both presenting diversity and diversity maintenance methods. Multi-population methods are the most used subcategory of hybrid methods. In this type of algorithms, a number of subpopulations are used for covering different regions of the search space. All subpopulations usually have similar tasks, but they may also have different ones. In [30] proposed 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 [31] and Differential Evolution in [32]. In [33], a population is first used to global search, and when an optimum is discovered, the population is divided into two subpopulations, one responsible for tracking optima and the other for global search. In [34] also, a regression based approach named "RSPSO" is suggested for enhancing the convergence rate using speciation based methods. In [35], a technique named "SPSO" is proposed in which every cluster is divided into two. The first cluster is responsible for exploitation and the second one is in charge of exploration. In [36], a method based on clustering is proposed for creating subpopulations, and in [37], this method has been improved and has been named "PSO-CP". In [11], two multi-population methods, named " MQSO "and "MCPSO" , are proposed. In the former one, quantum particles and in the latter one, charged particles are used to generate diversity. The number of solutions in this technique is equal for every subpopulation, and the number of subpopulations is also constant from the beginning. In [38], an approach for enhancing this method is proposed by making the number of subpopulations adaptive, and it is named "AMQSO". It has significantly improved the performance of the algorithm. Finally, in [39], a method for dynamic optimization is proposed and it is based on composite particles.

2.2 Firefly Algorithm

Firefly algorithm was introduced by Yang in 2008 [40] and inspired from flashing behaviour of fireflies in nature. In this algorithm, each firefly has a location $X=(x_1, x_2, x_3, ..., x_d)^t$ in a D-dimensional space and a light intensity $I(x)$ or attractiveness $\beta(x)$ which are proportional to the objective function $f(x)$. Each firefly can move toward another more attractive (brighter) one by Eq. (1).

$$x_i^{(t+1)} = x_i^{(t)} + \beta_0 e^{-\gamma r_{i,j}^2}(x_j - x_i) + \alpha\varepsilon, \tag{1}$$

where, $r_{i,j}$ is the distance between any two fireflies i and j, β_0 is attractiveness at $r = 0$, γ is the light absorption coefficient in the environment, α is the step size scaling factor and ε is a random number drawn from a Uniform distribution.

2.3 Speciation Based Firefly Algorithm

Firefly algorithm has a great capability to converge to more than one optimum at a time. The authors in [14] has proposed an algorithm to create the sub-swarms dynamically by using this potential capability of firefly algorithm. In their algorithm, some modifications are made on firefly algorithm as follows for better performance in dynamic environment. First, the best personal location of each firefly (*Pbest*) is added to the algo-

rithm by inspiration from particle swarm optimization [41-43]. The fireflies will move if the new position is better than the previous one in terms of fitness value. Second, the step size scaling factor of the firefly (α) is changed in an adaptive manner. The value of α increases if new position is better than the previous one, otherwise it decreases. This approach has some advantages such as (a) No need to define a mechanism for dividing fireflies into sub-populations; (b) No need to define a mechanism to make sub-populations away from each other. However, it has some disadvantages; (a) Redundant fitness evaluations near the local optima which have not any influence in reduction of the overall error for optimization. As the most performance metrics in dynamic environments calculate the error only based on the distance between the best solution found by the algorithm and the global optimum; (b) Algorithm's Inability to cover a high number of local optima due to the fixed number of fireflies in each speciation and the weakness of algorithm in global search. In the next section, a novel algorithm is suggested by the authors to remove the shortcomings of their previous algorithm in dynamic environments.

3. PROPOSED ALGORITHM

In this section, a novel algorithm based on the firefly algorithm with speciation is proposed for dynamic optimization. The proposed algorithm applies various mechanisms in order to handle the challenges in dynamic environment. In the following, the mechanisms used in the proposed algorithm will be discussed in more detail.

So far, the algorithms which have been designed for optimization in Unknown and dynamic environments have some of the properties listed below for more proper function in these environments.

(a) Covering all optima with minimum fitness evaluation.
(b) Disabling and enabling individuals for improving performance of the algorithm.
(c) Discovering or predicating Change in the environment.
(d) Appropriate local search.
(e) Appropriate global search.
(f) Appropriate trade-off between local and global search.
(g) Knowing the minimum information about the environment.

The last one (g) is a property which makes the algorithm more general than the other algorithms. In the real world, most environments are also dynamic and unknown. So, knowing minimum knowledge about the environment is a great feature. Up to now, most of the algorithms had some prior knowledge about the dynamic environment such as the number of local optima, time between two changes in environment, minimum and maximum of change severity, minimum and maximum values for each design variable and so on. However, in proposed approach minimum knowledge about the environment has been used.

The proposed approach has two types of swarm for better covering the environment. The follower swarm is responsible for local search and the discoverer swarm is responsible for global search and finds a promising area in the environment. Also, this approach has a solution for each aforementioned property which is defined as below:

(a) For covering all optima with minimum fitness evaluation, this approach changes the way that the discoverer swarm does the discovery process in [14] and has provided a better global search. The details for this will be provided later in this section.
(b) For disabling and enabling individuals, this approach uses a mechanism for activating and inactivating fireflies in the follower swarm during the run.
(c) For discovering or predicting any change in the environment, this approach uses a test point for identifying the change in the environment. The test point starts evaluation at the beginning of the algorithm. If the fitness value is different from the last evaluation, then the change happens to the environment and the algorithm performs the change process in the environment.
(d) This approach uses a cloud around the global optimum and reduces the radius of the cloud adaptively for the follower swarm during the run.
(e) For global search also this approach uses the discoverer swarm.
(f) This approach uses a mechanism to find a trade-off between local search and global search described in detail in the following.
(g) This approach does not use any knowledge about the number of local optima and also the time between two changes in the environment.

The pseudo code of the proposed algorithm is provided in Algorithm 1.

Algorithm 1: Proposed Algorithm (A-SFA)

```
1  begin
2    Initialize fireflies in discoverer swarm and follower swarm Randomly.
3    Calculate Fitness of all fireflies and set Pbest and Gbest.
4    while stop criteria is not satisfied do
5      call TestForChange()
6      if f(Gbest) > min_activating_discoverer  then
7        Activate discoverer swarm
8      end if
9      if discoverer swarm is active
10       call StartDiscovey()
11       if distance between Gbest_finder and any fireflies is less than r_excl then
12         reinitialize the discoverer_swarm
13       end if
14       if discoverer_swarm is converged then
15         add k best fireflies to the follower swarm
16       end if
17     end if
18     Move Fireflies In Follower Swarm based Eq.(1)
19     Call GbestLocalSearch()
20   end
21 end
```

In the proposed algorithm, in the first step, the fitness for all fireflies in follower and discoverer swarm is calculated. Then, a test will be done to check whether a change has happened to the environment or not. The pseudo code of the test for the change rou-

tine is provided in Algorithm 3. In the next step, the algorithm checks if discoverer swarm must be active or not. If the discoverer swarm is active then *start_discovery* routine will be called. The pseudo code of this routine is given in Algorithm 3, too. In the next step, the algorithm checks whether there is any conflict between the discoverer swarm and the follower swarm. If it finds some conflicts, the discoverer swarm will be reinitialized.

The r_{excl} parameter calculated by Eq. (2).

$$r_{excl} = 0.5 * \left(\frac{peaks\_location\_range}{number\_of\_peaks^{\frac{1}{number\_of\_dimensions}}}\right) \tag{2}$$

Next, the algorithm checks the convergence of discoverer swarm to local optima. Also, the convergence condition in this approach is that the mean of *Gbest* of the discoverer swarm in three sequential iterations is less than *num_seq_iteration*. If the discoverer swarm converges, the best *k_selected* firefly of the discoverer swarm would be added to the follower swarm. After that, the follower swarm will move according to Eq. (1) and then Gbest Local Search routine will be called. The pseudo code of Gbest local search routine is given in Algorithm 4. The algorithm will do the mentioned step until it reaches to the maximum number of fitness evaluations in the problem.

Algorithm 2: TestForChange()

```
1  begin
2      Evaluate Gbest as a test point.
3      if f(Gbest) is different from previous one then
4          Re-evaluate all the fireflies in both swarm
5          Reset Pbest and Gbest.
6          Re-initialize r_cloud and α.
7          if the fireflies number n is more than max_num_of_fireflies then
8              ClusterFirefly()
9              remove the worse sub-swarm
10         end
11     end
12 end
```

In Algorithm 2, the best firefly (*Gbest*) is considered as the test point and at the beginning of each iteration, it checks whether there are any differences between fitness value of Gbest firefly and its previous value. If there are any changes in the environment, some parameters will be adjusted for diversifying the environment. Also, if the number of fireflies in the swarm is more than *max_num_of_fireflies* parameter, the fireflies in the follower swarm will be clustered by the algorithm which is given in [44] for finding multiple species seeds and then by using these seeds, data clustering will be done. After finding the clusters, removing the worst swarm (cluster) from the follower swarm is very simple.

Algorithm 3: StartDiscovey()

```
1  begin
2    for each firefly i in discoverer swarm
3      for each firefly j in discoverer swarm
4        Move firefly i toward firefly j based Eq. (1), if f(firefly_j) is better than
5        f(firefly_i)
6      end for
7      if firefly i does not move at all
8        move firefly i randomly
9      end if
10   end for
11 end
```

In Algorithm 3, the discovery routine occurs. Discovery routine uses firefly algorithm like follower routine. But the values of parameters for $\alpha_{discoverer}$ and $\gamma_{discoverer}$ are completely different from the follower swarm. The $\gamma_{discoverer}$ for the discovery routine is set to zero. Thus, the firefly algorithm can be executed similar to particle swarm optimization. Moreover, $\alpha_{discoverer}$ decreases during the run according to Eq. (3), where f_{dec} is a predefined factor for decreasing $\alpha_{discoverer}$.

$$\alpha_{discover} = \alpha_{discover} \cdot f_{dec} \tag{3}$$

If discoverer swarm converges, k best fireflies of the discoverer swarm would be added to follower swarm. In addition, Gbest local search is recalled for improving the results as much as possible based on Algorithm 4.

In this algorithm, a quantum firefly is calculated with respect to Eq. (4).

$$Quantum\_firefly = Gbest\_position + (r_{cloud} \times R_j), \tag{4}$$

where R_j is a random number with uniform distribution in [−1, 1]. Hence, the *quantum_firefly* is located in the radius of r_{cloud} from Gbest. And r_{cloud} will decrease based on Eq. (5).

Algorithm 4: GbestLocalSearch()

```
1  begin
2    for counter =1 to try_number
3      Update Quantum_firefly based on Eq. (4).
4      if f(Quantum_firefly) is better than f(Global best) then
5        Gbest = Quantum_firefly
6      end
7    end for
8    Update r_cloud based on Eq. (5).
9  end
```

$$r_{cloud} = r_{discover} \times (w_{\min} + (rand \times (w_{\max} - w_{\min})), \tag{5}$$

where W_{min}, W_{max} are the lower and the upper limits for multiplying in r_{cloud}. In the next section, the simulation setup and the result of applying the proposed algorithm on well-known benchmark will be evaluated.

4. EXPERIMENTAL SETUP

In this section, first, the benchmark function and the performance metric are described and then, the results of applying the proposed algorithm on benchmark function is compared with some other similar algorithms.

4.1 MPB Function

We used the Moving Peaks Benchmark (MPB) originally proposed by Brank [45] as a benchmark function in our simulations. It is widely used in the literature for evaluating the performance of dynamic optimization algorithms. In this benchmark, there are some peaks in a multi-dimensional space, where height, width and position of peaks alter when a change occurs in the environment. The parameter setting for MPB function in this paper can be found in Table 1.

Table 1. MPB parameter setting.

Parameter	Value	Parameter	Value
Number of peaks, M	Variable between 1 to 200	Number of dimensions, D	5
Change frequency, CF	500,1000,2500,5000, 10000	Correlation Coefficient, λ	0
Height change, H	7.0	peaks location range	[0 – 100]
Width change, W	1.0	Peak height	[30.0 – 70.0]
Peaks shape	Cone	Peak width	[1 – 12]
Basic function	No	Initial value of peaks	50.0
Shift length, s	1.0	Number of change, NC	100

4.2 Performance Metric

The performance metric used in this paper is the offline error, which is the average of difference between the fitness of best solution found by the algorithm and the fitness of the global optimum in all fitness evaluations [45, 46].

$$offline\quad error = \frac{1}{FEs}\sum_{t=1}^{FEs}\left(f\left(gbest(t)\right) - f\left(globalOptimum(t)\right)\right) \tag{6}$$

Where FEs is the maximum fitness evaluation, and $gbest(t)$ and $globalOptimum(t)$ are the best position found by the algorithm and the global optimum at the tth fitness evaluation, respectively. It is worth mentioning that in [37, 47], offline error is calculated using a different approach which is the average of current errors exactly before each environmental change. In this paper, we calculate offline error based on Eq. (6).

4.3 Configurations of A-SFA

In A-SFA, the number of sub-swarm, number of fireflies in each sub-swarm and number of active fireflies are adaptive. However, in order to make the algorithm adaptive, some non-adaptive parameters are introduced, as well. Table 2 shows all the parameters' settings for A-SFA. Most of the values for non-adaptive parameters are selected by doing some sensitivity analysis on a simple MPB scenario (default parameter setting except CF =5000, M=10, NC=5).

The zero is considered for $\gamma_{discoverer}$ parameter. It means that a flashing firefly in discoverer swarm can be seen anywhere in the fitness landscape. Also, as the follower swarm is responsible for local search in the fitness landscape, 0.01 is considered for α parameter.

Table 2. A-SFA parameters settings.

Parameter	Value	Parameter	Value
α	0.01	*k_selected*	5
γ	1	f_{dec}	0.99
$\alpha_{discoverer}$	1	r_{cloud}	0.2 * severity
$\gamma_{discoverer}$	0	min_activating_discoverer	2
max_num_of_fireflies	100	w_{min}	0.6
num_seq_iteration	2	w_{max}	0.9

4.4 Comparison with Other State-of-the-Art Approaches

In this section, the proposed algorithm is tested on MPB with various configurations. The obtained results are the outcomes of 30 times execution of the proposed algorithm with different random seed. Also, for each environment, *t*-tests with a significance level of 0.05 have been applied and the result of the best performing algorithms is printed in bold. When the results for the best performing algorithms are not significantly different, all are printed in bold.

4.4.1 Effect of varying the number of peaks and change frequencies

Table 3 presents the comparison of proposed algorithm and all other peer algorithms on MPB function with different number of peaks and change frequencies. The results of mQSO [11], Amqso [38] and SFA [14] algorithms are obtained from their execution which implemented by us, and the results of other algorithms are obtained from their respective papers. For CPSO [37] and CPSOR [47] the results are obtained from [48] which are provided based on defined offline error in this paper. As we can see in Table 3, AmQSO is the best algorithm when the peak number is equal to 1. Also, A-SFA and SFA are in the second and third place. This superiority is due to use of only one swarm during the run in AmQSO and also using at most two swarms in A-SFA and SFA. In this way, the number of wasted fitness evaluation decreases significantly.

From Table 3, it is clear that the algorithms' performance deteriorate by increasing the number of peaks. However, in some algorithms the results are improved when the

Table 3. Comparison of the offline error (standard error) of seven algorithms on MPB function with different number of peaks and change frequencies.

Algorithm	C.F.	Number of Peaks							
		1	5	10	20	30	50	100	200
mQSO(5,5q)	500	33.67(3.42)	11.91(0.76)	9.62(0.34)	9.07(0.25)	8.80(0.21)	8.72(0.20)	8.54(0.16)	8.19(0.17)
AmQSO		**3.02(0.32)**	5.77(0.56)	5.37(0.42)	6.82(0.34)	7.10(0.39)	7.57(0.32)	7.34(0.31)	7.48(0.19)
mPSO		8.71(0.48)	6.69(0.26)	7.19(0.23)	8.01(0.19)	8.43(0.17)	8.76(0.18)	8.91(0.17)	8.88(0.14)
HmPSO		8.53(0.49)	7.40(0.31)	7.56(0.27)	7.81(0.20)	8.33(0.18)	8.83(0.17)	8.85(0.16)	8.85(0.16)
APSO		4.81(0.14)	**4.95(0.11)**	**5.16(0.11)**	**5.81(0.08)**	6.03(0.07)	**5.95(0.06)**	**6.08(0.06)**	**6.20(0.04)**
SFA		4.72(0.12)	**4.88(0.12)**	**5.11(0.14)**	**5.72(0.13)**	**5.97(0.12)**	**5.94(0.15)**	**6.15(0.08)**	**6.18(0.11)**
A-SFA		4.80(0.08)	**4.98(0.11)**	**5.09(0.10)**	**5.67(0.14)**	**5.84(0.11)**	**5.86(0.10)**	**6.04(0.11)**	**6.08(0.17)**
mQSO(5,5q)	1000	18.60(1.63)	6.56(0.38)	5.71(0.22)	5.85(0.15)	5.81(0.15)	5.87(0.13)	5.83(0.13)	5.54(0.11)
AmQSO		**2.33(0.31)**	2.90(0.32)	4.56(0.40)	5.36(0.47)	5.20(0.38)	6.06(0.14)	4.77(0.45)	5.75(0.26)
mPSO		4.44(0.24)	3.93(0.16)	4.57(0.18)	4.97(0.13)	5.15(0.12)	5.33(0.10)	5.60(0.09)	5.78(0.09)
HmPSO		4.46(0.26)	4.27(0.08)	4.61(0.07)	4.66(0.12)	4.83(0.09)	4.96(0.03)	5.14(0.08)	5.25(0.08)
APSO		2.72(0.04)	2.99(0.09)	3.87(0.08)	4.13(0.06)	4.12(0.04)	**4.11(0.03)**	**4.26(0.04)**	**4.21(0.02)**
SFA		**2.45(0.12)**	**2.71(0.06)**	3.64(0.04)	**4.01(0.07)**	**4.02(0.08)**	4.12(0.07)	4.40(0.07)	4.43(0.07)
A-SFA		2.74(0.14)	2.89(0.09)	**3.51(0.08)**	**3.92(0.12)**	**3.97(0.07)**	**4.02(0.09)**	4.34(0.09)	4.38(0.08)
mQSO(5,5q)	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		**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		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)
HmPSO		1.75(0.10)	1.92(0.11)	2.39(0.16)	2.46(0.09)	2.57(0.05)	2.65(0.05)	2.72(0.04)	2.81(0.04)
APSO		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)**
SFA		**0.85(0.06)**	1.49(0.07)	1.95(0.04)	2.28(0.05)	2.46(0.05)	2.57(0.05)	2.74(0.04)	2.76(0.04)
A-SFA		1.12(0.08)	**1.32(0.08)**	**1.77(0.07)**	**2.09(0.06)**	**2.28(0.07)**	**2.42(0.06)**	**2.64(0.05)**	**2.65(0.06)**
mQSO(5,5q)	5000	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		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)
mPSO		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)
HmPSO		0.87(0.05)	1.18 (0.04)	1.42(0.04)	1.50(0.06)	1.65(0.04)	**1.66(0.02)**	**1.68(0.03)**	**1.71(0.02)**
APSO		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)
CPSO		7.80(1.00)	4.20(0.32)	4.50(0.26)	4.00(0.16)	3.50(0.16)	3.50(0.13)	3.20(0.13)	2.50(0.091)
CPSOR		6.10(0.84)	2.90(0.34)	2.60(0.20)	2.60(0.30)	2.00(0.14)	2.40(0.12)	2.50(0.10)	2.30(0.097)
SFA		**0.42(0.03)**	**0.89(0.07)**	1.05(0.04)	1.48(0.05)	1.56(0.06)	1.87(0.05)	2.01(0.04)	1.99(0.06)
A-SFA		0.66(0.05)	**0.79(0.06)**	**0.90(0.05)**	**1.24(0.08)**	**1.41(0.06)**	**1.71(0.04)**	1.84(0.04)	1.90(0.05)
mQSO(5,5q)	10000	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		**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		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)
HmPSO		–	–	–	–	–	–	–	–
APSO		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)
SFA		0.26(0.03)	0.53(0.04)	0.72(0.02)	0.91(0.03)	0.99(0.04)	1.19(0.04)	1.44(0.04)	1.52(0.03)
A-SFA		0.35(0.03)	**0.49(0.03)**	**0.63(0.03)**	**0.77(0.03)**	**0.84(0.03)**	**0.97(0.03)**	**1.27(0.05)**	**1.31(0.03)**

number of peaks is high because the difference between local and global fitness values decreases. Furthermore, increasing the change frequency leads to more chance for the algorithms to find better values and discover the peaks which results in their improved efficiency. In effect, when the change frequency is low, it will be possible to lose the optima or never to reach them because of fast environment changes, which reduces the

algorithms efficiency. The results in Table 3 show that A-SFA is of a very good efficiency in different environments using different peaks and different high and low frequency changes.

4.4.2 Effect of varying shift severity

Table 6 presents the comparison of proposed algorithm and all other peer algorithms on MPB function with different shift severity, change frequency = 5000 and number of peaks = 10.

Table 4. Comparison of offline error (standard error) of six algorithms on MPB function with CF =5000, M=10 and different shift severity.

Algorithm	Shift length			
	1	**2**	**3**	**5**
mQSO(5,5*q*)	1.85(0.08)	2.40(0.06)	3.00(0.06)	4.24(0.10)
AmQSO	1.51(0.10)	2.09(0.08)	2.72(0.09)	3.71(0.11)
CPSO	4.50(0.26)	5.40(0.24)	6.10(0.19)	7.20(0.15)
CPSOR	2.60(0.20)	3.7(0.19)	4.50(0.24)	6.10(0.21)
SFA	1.05(0.04)	1.44(0.06)	2.06(0.07)	2.89(0.13)
A-SFA	**0.90(0.05)**	**1.31(0.07)**	**1.76(0.09)**	**2.64 (0.08)**

In Table 4, the efficiency of the proposed algorithm on MPB with different shift severities has been illustrated and has been compared with five other algorithms. MPB configuration has been done according to Table 1, and just the value of shift length parameter was changed in it. By increasing shift length, it becomes more difficult for algorithms to follow the peaks because after each environment change, the peaks move into further distances. As the results in Table 4 represented, the efficiency of all the algorithms dramatically decreases by increasing shift length, but the efficiency of the proposed A-SFA shows less degradation and more robustness in these conditions in comparison to other algorithms.

As the results of the experiments show, the efficiency of the proposed method is very high because of utilizing different mechanisms for improving its performance and eliminating the challenges in dynamic environments. The peaks are rapidly covered by the discoverer swarms in the proposed algorithm, and therefore, the algorithm can quickly discover the position of the global optimum after each environment change.

5. CONCLUSION

In this paper, a novel speciation based [14] firefly algorithm was proposed for optimization in unknown and dynamic environments, in which several mechanisms were used to conquer challenges of dynamic environments. In the proposed algorithm, two types of swarm were used, which were follower and discoverer. The discoverer swarm tries to help the main swarm for global search in the environment. On the other hand, the follower swarm is used for appropriately performing local optimization around the cov-

ered peaks and following them after an environment change. The performance of the proposed method was evaluated utilizing different configurations of Moving Peak Benchmark problem which is the most well-known benchmark in the literature and results were compared with those of the state-of-the art methods. The results showed the superiority and performance of the proposed method in terms of efficiency and accuracy.

The proposed approach already used some knowledge about the problem such as severity of change in the environment and range of designing variable for better acting in a dynamic environment. Hence, for applying the proposed algorithm for a real dynamic application, it needs to remove its sensitivity to severity and other parameters. Also, proposing a new approach with minimum number of parameters is always needed for any problems.

REFERENCES

1. T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, Vol. 6, 2012, pp. 1-24.
2. K. Krishnakumar, "Micro genetic algorithms for stationary and nonstationary function optimization," in *Proceedings of SPIE International Conference Adaptive on Systems*, 1989, pp. 289-296.
3. J. Grefenstette, "Genetic algorithms for changing environments," *Parallel Problem Solving from Nature*, Vol. 2, 1992, pp. 137-144.
4. N. Mori and H. Kita, "Genetic algorithms for adaptation to dynamic environments – a survey," in *Proceedings of IEEE International Conference on Industrial Electronics*, *Control and Instrumentation*, 2000, pp. 2947-2952.
5. L. Lili, W. Dingwei, and W. Hongfeng, "A new dual scheme for genetic algorithm in dynamic environments," in *Proceedings of Chinese Control and Decision Conference*, 2008, pp. 135-138.
6. V. Ramos, C. Fernandes, A. C. Rosa, and A. Abraham, "Computational chemotaxis in ants and bacteria over dynamic environments," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 1109-1117.
7. X. Wang, X. Z. Gao, and S. J. Ovaska, "An immune-based ant colony algorithm for static and dynamic optimization," in *Proceedings of IEEE International Conference on Systems*, *Man and Cybernetics*, 2007, pp. 1249-1255.
8. L. Xiaoxiong, W. Yan, and Y. Jimin, "An improved estimation of distribution algorithm in dynamic environments," in *Proceedings of IEEE 4th International Conference on Natural Computation*, 2008, pp. 269-272.
9. J. Branke, *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, 2001.
10. T. Blackwell and J. Branke, "Multi-swarm optimization in dynamic environments," *Applications of Evolutionary Computing*, 2004, pp. 489-500.
11. T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, Vol. 10, 2006, pp. 459-472.
12. M. Kamosi, A. Hashemi, and M. Meybodi, "A new particle swarm optimization al-

gorithm for dynamic environments," *Swarm*, *Evolutionary*, *and Memetic Computing*, 2010, pp. 129-138.
13. M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A hibernating multi-swarm optimization algorithm for dynamic environments," in *Proceedings of World Congress on Nature and Biologically Inspired Computing*, 2010, pp. 370-376.
14. B. Nasiri and M. Meybodi, "Speciation based firefly algorithm for optimization in dynamic environments," *International Journal of Artificial Intelligence*, 2012, Vol. 8, pp. 118-132.
15. A. Carlisle and G. Dozier, "Adapting particle swarm optimization to dynamic environments," in *Proceedings of International Conference on Artificial Intelligence*, Vol. 1, pp. 429-434.
16. H. G. Cobb, "An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," *NRL Memorandum Report*, Vol. 6760, 1990, pp. 523-529.
17. T. Nanayakkara, K. Watanabe, and K. Izumi, "Evolving in dynamic environments through adaptive chaotic mutation," in *Proceedings of the 4th International Symposium on Artificial Life and Robotics*, 1999, pp. 520-523.
18. F. Vavak, K. Jukes, and T. Fogarty, "Performance of a genetic algorithm with variable local search range relative to frequency of the environmental changes," in *Proceedings of the 3rd Annual Conference on Genetic Programming*, 1998, pp. 22-25.
19. E. L. Yu and P. N. Suganthan, "Evolutionary programming with ensemble of explicit memories for dynamic optimization," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2009, pp. 431-438.
20. Y. G. Woldesenbet and G. G. Yen, "Dynamic evolutionary algorithm with variable relocation," *IEEE Transactions on Evolutionary Computation*, Vol. 13, 2009, pp. 500-513.
21. X. Hu and R. C. Eberhart, "Adaptive particle swarm optimization: detection and response to dynamic systems," in *Proceedings of IEEE Congress on Evolutionary Computation*, 2002, pp. 1666-1670.
22. S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, Vol. 9, 2005, pp. 815-834.
23. S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer for dynamic optimization problems," *Applications of Evolutionary Computing*, 2004, pp. 513-524.
24. A. Hashemi and M. Meybodi, "Cellular Pso: A Pso for dynamic environments," *Advances in Computation and Intelligence*, 2009, pp. 422-433.
25. A. B. Hashemi and M. R. Meybodi, "A multi-role cellular PSO for dynamic environments," in *Proceedings of the 14th International CSI Computer Conference*, 2009, pp. 412-417.
26. Y. Shengxiang, "On the design of diploid genetic algorithms for problem optimization in dynamic environments," in *IEEE Congress on Evolutionary Computation*, 2006, pp. 1362-1369.
27. A. Uyar and A. E. Harmanci, "A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments," *Soft Computing-A Fusion of Foundations*, *Methodologies and Applications*, Vol. 9, 2005, pp. 803-814.

28. R. W. Morrison, *Designing Evolutionary Algorithms for Dynamic Environments*, Springer-Verlag, 2004.
29. L. T. Bui, H. A. Abbass, and J. Branke, "Multiobjective optimization for dynamic environments," in *IEEE Congress on Evolutionary Computation*, Vol. 3, 2005, pp. 2349-2356.
30. J. Branke, T. Kaussler, C. Schmidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," *Adaptive Computing in Design and Manufacturing*, Vol. 6, 2000.
31. H. Cheng and S. Yang, "Multi-population genetic algorithms with immigrants scheme for dynamic shortest path routing problems in mobile ad hoc networks," *Applications of Evolutionary Computation*, 2010, pp. 562-571.
32. R. I. Lung and D. Dumitrescu, "A new collaborative evolutionary-swarm optimization technique," in *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation*, 2007, pp. 2817-2820.
33. R. K. Ursem, "Multinational GAs: Multimodal optimization techniques in dynamic environments," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2000, pp. 19-26.
34. S. Bird and X. Li, "Using regression to improve local convergence," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 592-599.
35. W. Du and B. Li, "Multi-strategy ensemble particle swarm optimization for dynamic optimization," *Information Sciences*, Vol. 178, 2008, pp. 3096-3109.
36. C. Li and S. Yang, "A clustering particle swarm optimizer for dynamic optimization," in *IEEE Congress on Evolutionary Computation*, 2009, pp. 439-446.
37. S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Transactions on Evolutionary Computation*, Vol. 14, 2010, pp. 959-974.
38. T. Blackwell, J. Branke, and X. Li, "Particle swarms for dynamic optimization problems," *Swarm Intelligence*, 2008, pp. 193-217.
39. L. Liu, S. Yang, and D. Wang, "Particle swarm optimization with composite particles in dynamic environments," *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part B: Cybernetics*, Vol. 40, 2010, pp. 1634-1648.
40. X. S. Yang, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008.
41. M. Hasanzadeh, M. Meybodi, and M. Ebadzadeh, "Adaptive cooperative particle swarm optimizer," *Applied Intelligence*, 2013, pp. 1-24.
42. H. Masoud, S. Jalili, and S. Hasheminejad, "Dynamic clustering using combinatorial particle swarm optimization," *Applied Intelligence*, Vol. 38, 2013, pp. 289-314.
43. J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, 1995, pp. 1942-1948.
44. A. Petrowski, "A clearing procedure as a niching method for genetic algorithms," in *IEEE Congress on Evolutionary Computation*, 1996, pp. 798-80.
45. J. Branke, *The Moving Peaks Benchmark*, http://web.archive.org/web/20130906140931/http:/people.aifb.kit.edu/jbr/MovPeaks/, 1999.
46. J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *IEEE Congress on Evolutionary Computation*, 1999, pp. 1875-1882.
47. C. Li and S. Yang, "A general framework of multi-population methods with clustering in undetectable dynamic environments," *IEEE Transactions on Evolutionary*

Computation, Vol. 16, 2011, pp. 556-577.
48. C. Li, S. Yang, and M. Yang, "An adaptive multi-swarm optimizer for dynamic optimization problems," *Evolutionary Computation*, Vol. 22, 2014, pp. 559-594.

Babak Nasiri received his B.S. and M.S. in Computer Science in Iran, in 2002 and 2004, respectively. He has been a Ph.D. candidate in Computer Science from Qazvin Islamic Azad University, Qazvin, Iran from 2010. Prior that, he was the faculty member of Computer Engineering and IT Department at Qazvin Islamic Azad University from 2008. His research areas include soft computing, machine learning, nature-inspired algorithms, data mining and web mining.

Mohammad Reza Meybodi received his B.S. and M.S. in Economics from National University in Iran, in 1974 and 1977, respectively. He also received his M.S. and Ph.D. degree from Oklahoma University, USA, in 1980 and 1983, respectively in Computer Science. Currently, he is a Full Professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an Assistant Professor at Western Michigan University, and from 1985 to 1991 as an Associate Professor at Ohio University, USA. His research interests include soft computing, learning system, wireless networks, parallel algorithms, sensor networks, grid computing and software development.