*Journal of **AI** and **D**ata **M**ining*
Vol 6, No 1, 2018, 191-205

Chaotic Genetic Algorithm based on Explicit Memory with a new Strategy for Updating and Retrieval of Memory in Dynamic Environments

M. Mohammadpour[1], H. Parvin[2,3*] and M. Sina[2]

1. Young Researchers and Elite Club, Yasooj Branch, Islamic Azad University, Yasooj, Iran.
2. Department of Computer Engineering, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran.
3. Young Researchers and Elite Club, Nourabad Mamasani Branch, Islamic Azad University, Nourabad Mamasani, Iran.

Received 21 July 2015; Revised 21 July 2016; Accepted 26 August 2016
**Corresponding author: parvin@alumni.iust.ac.ir (H. Parvin).*

Abstract
Many problems considered in the optimization and learning processes assume that solutions change dynamically. Hence, the algorithms are required that dynamically adapt with the new conditions of the problem through searching new conditions. Mostly, utilization of information from the past allows to quickly adapting changes right after they occur in the environment. This is the idea underlining the use of memory in this field, what involves the key design issues concerning the memory content, update process, and retrieval process. In this work, we use the chaotic genetic algorithm (GA) with memory for solving dynamic optimization problems. A chaotic system has a much more accurate prediction of the future compared with a random system. The proposed method uses a new memory with diversity maximization. Here, we propose a new strategy for updating memory and memory retrieval. An experimental study is conducted based on the moving peaks benchmark (MPB) in order to test the performance of the developed method in comparison with several state-of-the-art algorithms from the literature. The experimental results obtained show the superiority and more effectiveness of the proposed algorithm in dynamic environments.

Keywords: *Dynamic Environments, Explicit Memory, Moving Peaks Benchmark, Offline Error, Chaos.*

1. Introduction

Nowadays in engineering problems, we are faced with what should be optimal in some of these issues with the purpose of optimization to reduce expenses (minimizing), or to increase quality (maximizing). In any of possible optimal solution, suitable quantities of all parameters are the best way to solve the problem. In optimization expression we may intend to find goal minima in dynamic or static environment. If the optima of the problem change during the evolution, we can assume the environment as a dynamic one; otherwise, we should assume the environment as a static one. Evolutionary algorithms may perform in static environments with good efficiency but these algorithms alone are not able to solve dynamic optimization with a good performance. Therefore, to solve optimization problems, dynamic optimization problems need to strong heuristics. In these environments, dynamic changes are the main reason for occurring the challenges. Some of these challenges may include the difficulty in updating the memory after each change in the population, preserve diversity in a convergence environment to optimize the desired particle size and limited memory capacity, and identify the changes in their environment. To solve the optimization problems in the dynamic environment, the various dynamic methods are presented where each of them solves one or many of these challenges. Among the methods proposed to solve the dynamic optimization problems, methods that use a combination of strategies are of special significance. The combination memory and diversity have been used in [1-9].
Many researchers have used a composition of memory and evolutionary algorithms to solve dynamic optimization problems but our proposed method has a main difference with the other methods. We used a proper memory with a novel storage and retrieval strategy. The main goal of this work was to provide maximum diversity for the GA evolutionary algorithm. We focused on

setting a memory that can be embedded with evolutionary algorithms and improved them to solve dynamic optimization problems (DOPs). The proposed method can be considered as an application or an algorithm.

2. Related work

In this section, we introduce different solutions to DOPs, each of which are able to overcome some of the DOP challenges.

a. Methods based on memory for dynamic environments

In [10], Mohammadpour and Parvin have presented a new method based on memory for the optimization problems. This paper presents a GA-based memory to deal with DOPs, and focuses on explicit placement of memory schemes.

The Memory Immigrant Genetic Algorithm (MIGA) [11], introduced by Yang et al., uses a combination of GA and memory-based immigrants. The random immigrant method aims to improve the GA performance in dynamic environments through maintaining the population diversity level with random immigrants, and the memory approach aims to move the GA directly into an old environment that is similar to the new one through reusing the better old solutions. In the memory/search method, the total population size can be divided into two populations including the "memory" and "search" populations. The first population is based upon the memory applied to memorize old and well solutions. The second one is based upon the search used to explore and introduce new peaks and their introduction to memory. The second population is randomly initialized after each change [12]. The MEGA method of memory combined with GA is used. The main populations are the randomly initialized memory and the memory used to store the previous solutions [11]. Mohammadpour and Parvin, in [13], have presented a new method for solving the optimization problems with the aid of memory and clustering in addition to the chaos theory for population creation.

b. Methods based on clustering for dynamic environments

Yang et al. have proposed the CPSOR (Clustering Particle Swarm Optimization), in which the particles to cluster are divided as the smallest particles existing in each cluster to search for the local cluster in practice. The PSO algorithm with the $gbest$ model is used in the CPSOR algorithm where each particle’s neighborhood is defined as the whole swarm. In order to speed up the local search within the PSO algorithm, we introduce a learning method for the $gbest$ particle used in CPSO. When a particle i in a sub-population finds a better position, we iteratively check each dimension of the $gbest$ particle: replace the dimension with the corresponding dimensional value of particle i if the $gbest$ particle is improved by doing so. [14]. The clustering method in CPSO will assign the particles that are close to each other into a sub-swarm, which is the same as the neighborhood defined in PSO_{lbest}. However, CPSO has several major advantages in comparison with PSO_{lbest}. First, CPSO can track multiple optimal in dynamic environments. In CPSO, if more than one sub-swarms cover a same peak, they will finally be combined with each other into one sub-swarm by the overlapping check function. Second, CPSO can control overcrowding in a single peak [15]. Yang and et al. [15] have proposed the CGAR algorithm. In this algorithm, the standard GA with simple crossover and mutation and also the k-means clustering are used.

c. Methods based on multi-Populations

Self-organizing scouts (SOS) is a state-of-the-art multi-population evolutionary algorithm approach designed to overcome this limitation of a standard memory [16]. SOS begins with some number of base subpopulations searching for good solutions. When a peak (region with good solutions) has been found, the population splits. A scout population is formed to keep track of the peak, while the base population resumes searching for other peaks [16]. In particle swarm algorithm based on quantum particles that are known in this algorithm (i.e. mQSO algorithm), the population is divided into several groups and three quantum particles called functional diversity, pluralism is anti-convergence disposal. Quantum particles are placed in random positions to maintain the diversity of groups. If you find a real function that overlaps between the two groups, the group will reinitialize worse. Anti-convergence operation will be taken when all groups are converging, i.e. the group will re-initialize worse [17]. In [18], an approach is proposed by making the number of subpopulations adaptive, and it is named “AMQSO”. The method of FMSO has been used parent of a group as a group foundation for identifying promising areas and group of children for local search. Each child has its own search area. In this way there is a kind of balance between local search and global search. The method of FMSO search area to form a circle centered in the best particle group is considered.

Each particle that has a shorter distance than the radius of the circle is (closer to the center of the particle) by virtue of belonging to the group of children [19]. In the ESCA [20] and CESO [21] methods, the populations are divided into different categories, where each group uses unlike search approaches. The generalized methods based on multi-population have been presented in [32-38].

3. Dynamic environments

Most of the problems considered in optimization and learning assume that solutions exist in a static unchanging environment. If the environment does change, one may simply treat the new environment as a completely new version of the problem that can be solved as before. When a problem changes infrequently or only in small amounts, this can be a reasonable method. However, this assumption tends to break down when the environment undergoes frequent discontinuous changes. When this occurs, a search process may be slow to react; hurting performance in the time it takes to find a new solution. Instead of focusing only on finding the best solution to a dynamic problem, one must often balance the quality of solutions with the speed required to find good solutions.

a. Moment changes in environment:

When a dynamic event occurs, the fitness landscape changes. The term change is typically used to mean a change in the fitness landscape. The term environment is used to refer to the problem formulation and constraints at a given time. Typically, a change in the environment leads to a change in the fitness landscape. A dynamic environment changes cyclically over time. The cycle in which the optimization function should be changed, is considered as change frequency. The cycle at which a change occurs is referred to as *a moment of change* in the environment.

b. Response of changes the environments:

Many real world optimization problems are actually dynamic, and the optimization methods capable of continuously adapting the solution to a changing environment are required.
Most of the research works in evolutionary computation focus on optimization in dynamic environments, where changes for environments are small and algorithms must track those changes quickly. When changes to the environment are much more severe, a very different approach is necessary. After a change in the environment occurs, the location of the global optimum may change drastically. For discontinuous problems, search must be able to find areas containing good solutions in addition to refining those solutions to find the best solutions possible. Search algorithms that are able to explore widely across the search space after a change will have an advantage over those that search only locally. If search is population-based, introducing diversity into the search may help explore the search space after a change. For many problems, changes in the search space, though discontinuous, are not completely random.

3.1. Groups of dynamic environments

Dynamic environments can be taxonomy in different approaches. Branke [22] have categorized the dynamic environments into several parameters: the frequency of changes, severity of changes, predictability of changes, detectability of changes, and influence of search on the environment.
In the frequency of change, changes may be rare, while in others, changes may occur constantly. Problems are also not limited to one frequency of changes. Some problems may have small frequent changes, while others have large infrequent changes. The most important aspect of a frequency change is how long a learning or optimization algorithm has to find a solution both before it has an effect on performance and before another change occurs.
As mentioned earlier, the severity of changes also defines a dynamic problem. Some problems may have changes that are small enough to be easily tracked, while others have large discontinuous changes. Small changes may not have a large effect on the fitness landscape, while large changes may completely change the landscape.
The predictability expresses that, in some problems, changes follow a particular prototype. In others, changes are completely random. A problem with small predictable changes needs a very diverse algorithm than the ones with severe unpredictable changes.
The detectability of the change express in some problems, changes to the fitness landscape are easy to detect; one knows exactly when a change has occurred. In others, it may take some time before it is clear that the environment has changed. This can have a large effect on how an algorithm solves a problem.
Influence of search on the environment expresses that although every problem with dynamic environments suffers changes autonomous of the

results of search, the search can make extra changes in the environment. For some problems, solutions have no effect on the environment. For many others, though, the particular solution changes the environment.

3.2. Benchmark Problems for Dynamic Environments

Benchmark problems are used to examine the performance of the evolutionary algorithm in dynamic environments. The benchmark problems can simulate evolutionary algorithms in dynamic environments.

3.2.1. Moving Peaks Benchmark (MPB) for simulation of Dynamic Environments

MPB is a multimodal, multidimensional dynamic problem, proposed by Branke [23]. In Moving Peaks, the landscape is composed of m peaks in an n dimensional real-valued space.

At each point, the fitness is defined as the maximum over all m peak functions. This fitness can be formulated as follows:

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

where, $P(\vec{x}, h_i, w_i, \vec{p}_i)$ is a function describing the fitness of a given point ($\vec{x}$) for a peak described by height (h), width (w), and peak position ($\vec{p}$).

Every Δe evaluations, the height, width, and position are changed for each peak, changing the state of the environment. The height and width of each peak are changed by the addition of $Gaussian\,random$ variables scaled by height severity (hs) and width severity (ws) parameters. The position is shifted using a shift length s and a correlation factor λ. The shift length controls how far the peak moves, while the correlation factor determines how random a peak's motion will be. If $\lambda = 0.0$, the motion of a peak will be completely random, but if $\lambda = 1.0$, the peak will always move in the same direction until it reaches a boundary of the coordinate space where its path reflects like a ray of light. At the time of a change in the environment, the changes in a single peak can be described as

$$h_i\left(t\right) = h_i\left(t-1\right)+height_{severity}\,.\,\sigma \tag{2}$$

$$w_i\left(t\right) = w_i\left(t-1\right)+width_{severity}\,.\sigma \tag{3}$$

$$\vec{p}_i\left(t\right) = \vec{p}_i\left(t-1\right)+\vec{v}_i\left(t\right) \tag{4}$$

$$\sigma\in N\left(0,1\right) \tag{5}$$

The shift vector $\vec{v}_i(t)$ combines a random vector $\vec{r}$ with the previous shift vector $\vec{v}_i(t-1)$. The random vector is created by drawing uniformly from $[0;1]$ for each dimension, and then scaling the vector to have length s. $\vec{v}_i(t)$ is formulated by (6).

$$\vec{v}_i\left(t\right)=\frac{s}{\left|\vec{r}+\vec{v}_i\left(t-1\right)\right|}\left(\left(1-\lambda\right)\vec{r}+\lambda\vec{v}_i\left(t-1\right)\right) \tag{6}$$

Peak function for h, w, and p of each peak can be calculated as follows:

$$P\left(\vec{x},h\left(t\right),w\left(t\right),\vec{p}\left(t\right)\right)=h\left(t\right)-w\left(t\right).\sqrt{\sum_{j=1..n}\left(x_j-p_j\right)^2} \tag{7}$$

Part of the radical, the distance between the point exist and the position of each peak is expressed [23]. Figure 1 shows the trend of the changes in the peaks.

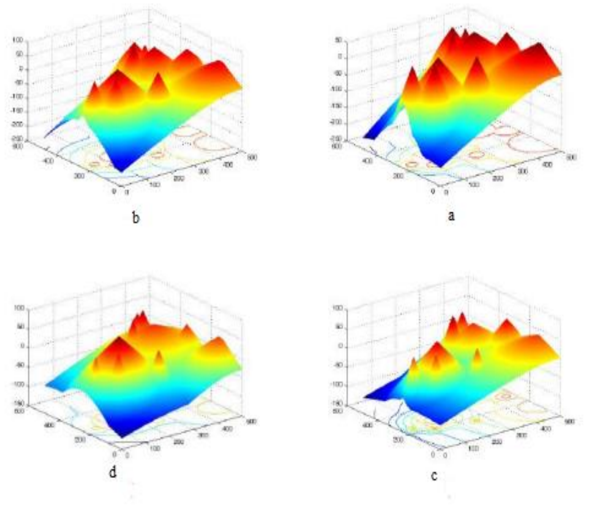


Figure 1. Trend of changes in peaks in MPB.

3.3. *Offline Error*

The performance measure used is the *Offline Error* , which is defined as follows:

$$Offline\ Error=\frac{1}{M}\sum_{t=1}^{M}\left(h\left(t\right)-f\left(t\right)\right) \tag{8}$$

where, $f(t)$ is the best solution obtained by an algorithm just before the M-th environmental change, $h(t)$ is the optimum value for the M-th environment, and M is the total number of evaluations.

4. Genetic algorithm

Genetic algorithm (GA) is based on learning method of biological evolution [24]. For a genetic algorithm to solve a particular problem, a set of candidate solutions to that problem is randomly created. This set of solutions is called population. Then, the quality of each of these potential solutions is measured and the best ones are selected as parents. The chosen individuals reproduce and undergo a variation process by means of genetic operators, e.g., recombination and mutation, defining a new offspring population. Finally, the next generation of the population is formed by combining parent and offspring populations. This process is repeated until a certain stop condition, e.g., number of generations, is attained. Figure 2 shows the pseudo code of a GA.

```
Algorithm 1: Genetic Algorithm

1.  Initialize population
2.  Evaluate population
3.  repeat
4.      Select parents
5.      Recombine pairs of parents
6.      Mutate the offspring
7.      Evaluate the offspring
8.  Create new population from parents and offspring
9.  until stop_condition is true
```

Figure 2. Pseudo-Code of GA.

4.1. Population

A population is formed by a set of individuals, also called chromosomes, typically of a fixed size. Each individual represents a possible solution to the problem, and consists of a sequence of smaller components, called genes. Each gene may assume different values, or alleles.

4.2. Representation

The choice for the representation of the individuals is made according to the type of problem to solve. The representation defines how the population individuals are encoded.

4.3. Fitness function

The fitness function is used to measure the quality of the population individuals. To measure it, a decoding process is needed to obtain the individual phenotype. The fitness is a real value obtained by applying the fitness function to the phenotype.

4.4. Selection

The selection method is used to choose a pool of parents based on their fitness. The solutions with higher fitness values have more probabilities to be chosen for mating.

4.5. Genetic operators

The role of genetic operators is to create variations among the population individuals. Genetic operators can be divided in two main categories: recombination (or crossover) and mutation. Recombination is applied using two (or more) selected parents and mixing their genetic content. Mutation is applied to the individual genes by making a small change in their corresponding alleles.

5. Memory

Generally, memory is divided into two categories: explicit memory and implicit memory. The implicit memory is used to store all information (including additional information). In fact, the memory is used to store all information in one chromosome (each chromosome has two or more alleles). The convergence data, i.e. distribution of alleles can be used as the normal view of the current environment. The diploid implicit memory functions have been presented in [25]. The implicit memory is divided into two categories: a dualism memory and a diploid memory. Explicit memory is used to store useful information about the environment, and unlike implicit memory that stores additional data, it only stores useful information. Explicit memory involves the two types of direct memory and associative memory [26]. In direct memory, good solutions obtained by each individual (local information) or solutions obtained by all members of the population (general information) are directly stored in memory and reused in new environments [27]. In associative memory, environmental information are stored as well as good solutions; among the data stored in the memory, are lists of the states of problem space or the likelihood of a good solution in problem space [28] and reused in the new environment.

5.1. Memory retrieval

The information stored in memory should be used for new tracking of the optimum. Thus the best time to retrieve data from memory is the moment when the environment is changed. Several strategies can be adopted to retrieve memory. One of the memory retrieval methods is the replacing the best person in the memory instead of the worst person in the memory [29, 30].

5.2. Update strategy for memory

The previously described memory approaches used different replacing methods [29].
In general, the remaining approaches used the method called similar proposed by Branke [29]. Branke investigated and compared the following replacing schemes:

Strategy 1: This strategy analyzed the two individuals in memory with the minimum distance between them and replaced the worst with the best individual in the population. For example, suppose that individuals i and j were chosen:
- $if\ fit(i) < fit(j)$, replaced the individual j by the current best.
- $if\ fit(i) > fit(j)$, replaced the individual i by the current best.

Strategy 2:
- If $fit(j) \times \frac{d_{ij}}{d_{max}} \le fit(new)$ replaced the individual j by the current best
- otherwise, replaced the individual i by the current best

Where $d_{ij}$was the distance between the individuals i and j and d_{max}
Was the maximal possible distance between the individuals i and j.

Strategy 3: In this *similarity* method, the current best individual of the population replaced the most similar individual stored in memory as long as it was a better solution. The similarity measure depended on the used representation. For binary encodings, the similarity between to individuals was measured using the Hamming distance.

6. Chaos theory

The chaos theory [27] refers to chaotic dynamical systems. Chaotic systems are non-linear dynamical systems that are very sensitive to initial conditions. The behavior of chaotic systems is apparently random, although these behaviors are not random. The random element is not necessary in creating a chaotic behavior. A famous example of such a system is the logistic map model. The features of the chaos theory can be, self-organization (adapting to environmental conditions) in dynamic environments, self-similarity (each part of the system has the features of the general form and it is similar to that form) and sensitivity to initial conditions. Figure 3 presents the sensitivity of chaotic systems to the initial conditions.

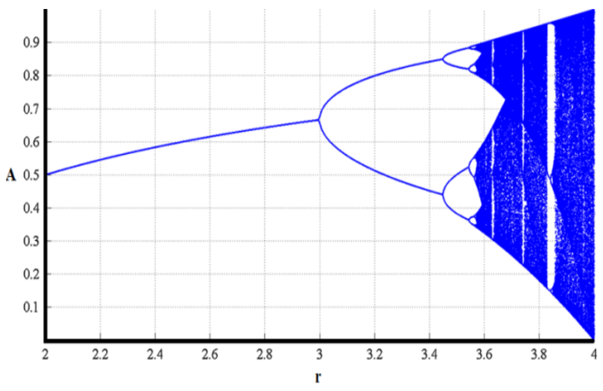


Figure 3. Sensitivity of chaotic systems to initial conditions.

In this figure, it is obvious that, in chaotic systems, not only a small initial divergence does not remain small, but also it grows exponentially. Also according to the figure, it can be clearly said that if we start moving from the end point, we will converge to the start point, which is one of the properties of a chaotic system.

In a random system, the current state of the system is independent from its previous state. In a random model, it is not possible to have an accurate prediction about the output of the model even within a short time period. A famous example is the formal logistic map as equation (9).

$$r_{n+1} = Ar_n(1 - r_n) \qquad (9)$$

In this equation, r_n is a a real number in the range of [0, 1], and parameter A, which is known as the logistic factor, causes unique features in this function. In this work, the amount of parameter A was equal to 4.

7. Proposed algorithm

In this work, we proposed a chaotic genetic algorithm based on the explicit memory to maintain the appropriate solutions to increase the efficiency of the algorithm. First the population and the memory were initialized by means of the chaos theory. If the memory was updated at generation t, the next update would happen at generation $t + rand(5;\ 10)$. The memory size equal to $0.1 \times N$. N is the size of the population. The memory was updated as follows. Every individual in memory had a feature that specified its age. When the memory was started, all individuals had an age equal to zero. In every generation, the age of all the memory individuals was computed using a linear combination of its actual age and a contribution of its fitness. This contribution was set using a parameter called μ.

More formally, the age of an individual was calculated using (10).

$$age_i = age_i + 1 + \mu \times fit_i \quad (10)$$
$$\forall i = 1 \ldots m, \forall \mu \grave{o} \ (0,1)$$

In this equation, m is the memory size. In this strategy, the individual age was not set to zero, so the older individuals were not penalized. When the memory was full and it was necessary to start replacing memory individuals, again the youngest was selected to be deleted and replaced by a new member if this individual was better. The pseudo-code for the memory update is shown in Algorithm 2 in figure 4.

```
Algorithm 2: MEM Function(MEM)
    Input: m is memory size; μϵ (0,1).
    Output: ∅

Initialize memory based chaos theory (logistic map)
$age_i$= 0
Every Generation:
    $age_i = age_i + 1 + \mu \times fit_i$
IF it is time to Update Memory
    $Best_{pop}$i s the best individual of the population
    Select memory individual $Memory_{sel}$with the lowest age
        IF $fitness(Memory_{sel}) < fitness\ (Best_{pop})$
          memory individual $Memory_{sel}$is replaced by $Best_{pop}$
    END of IF
END of IF
```

Figure 4. Pseudo-code for update memory.

The memory was retrieved as follows. In order to store the most relevant information to an environment in the memory, each time an environmental change was detected, the memory was also retrieved. When the memory was retrieved, the current best individual of the memory or the elite from the previous memory was stored, replacing the worst individual of the population. After retrieval of memory individuals, the GA readapts easier to the new environment. The memory was also used to detect environmental changes: the change in the environment was analyzed in a way that if the suitability of one of the members is changed in the re-evaluate, the proposed algorithm finds that the environment is changed. At this moment, a new set of individuals was formed by merging the memory and the main population. Then these individuals were evaluated in the context of the new environment, and the best population individuals were selected to become the new search population, which evolved through the selection, crossover, and mutation. Through this process, the memory remained unchanged. The best individual from the previous population was preserved and transferred to the next population, replacing the worst individual. The pseudo-code for the proposed algorithm is shown in figure 5.

For more explanations, we shall explain the related stages of the pseudo-code proposed method. In Step 2, the memory and population is initialized using the logistic map function. Unlike the standard GA that uses random numbers for creating initial population; in the proposed algorithm we use the chaos theory for creating initial population. A chaotic system has a precise prediction of the future compared with a random system. Thus the chaos theory can help speed convergence of the algorithm.

```
Algorithm3: Proposed Algorithm
    Input: $N, D, MCN, x_j^{min}, x_j^{max}, MemorySize$
    output: BEST Solution, BEST Fit, Error

    1.  Begin
    2.  Initialize POP and MEM with chaos thory
    3.  $FitPOP_i = fitness(POP_{:i})$
    4.  $FitMEM_i = fitness(MEM_i)$ $Update\_MEM = 1$
                $Update\ Time = rand(5,10)$
    5.  $Cycle = 1$
    Repeat:
    6.  $Chang_{flag}$:
    7.  $Update\_MEM$:
    8.  $Update\_MEM = 0$
    9.  $MEM = Update\_MEM\ Function(MEM, POP)$
    10. $Update\ Time = rand(5,10) + Cycle$
    11. $POPc = Update\_POPFunction(POP)$
     % POPc is current  POP and POPFunction is genetic algorithm
    12. $FCPOP_i = Fitness(POP_{:i})$ % Fitness for Current POP
    13. IF $FCPOP_i \neq FPOP_i$ then $Chang_{flag} = 1$
        % Change Detected
    14. $Chang_{flag}$is active:  % Reuse Memory
          1. $FM_i = Fitness(MEM_{:i}),\ FPOP_i = Fitness(POP_{:i})$
          2. Select $j_r \in MEM_r$ % $j_r$ is best individual in memory
                 subject to  $MEM_r \leq FM_l$
          3. Select $d_r \in POP_r$ % $d_r$ is worst individual in population
                 subject to  $x_r \geq x_l$
          4. $POP_r = MEM_r$%best individual in memory replacement instead of worse individual in population
    15. IF  $Update\_Time \geq Cycle$
                then  $update\_mem = 1$
    16. $Cycle = Cycle + 1$
    17. $Until\ cycle = MCN$
    18. End
```

Figure 5. Pseudo-code for proposed algorithm.

In the proposed algorithm, we used a logistic map function for the chaos theory. In this algorithm, instead of using random behaviors for individuals, we used the chaotic behaviors for each individual in the main population and memory population. For example, to create initial population, we used (11).

$$POP = LB_i + Ar_n(1 - r_n)(UB_i - LB_i) \quad (11)$$
$$\forall\ r_n \grave{o} [0,1], j \in (1,2,\ldots,SN), \forall i \in (1,2,\ldots,N)$$

In this equation, LB_i is the lower band, UB_i is the upper band, and pop is the population. In this equation, instead of a random value, we used a chaotic value.
In steps 3 and 4, as a function of efficiency, the competence for each individual memory of the main population and the memory population was calculated. If the memory update time occurs in cycle iteration then the next update for memory is in $Rand(5,10) + cycle$. The algorithm cycle begins at stage 5. Steps 6 and 7 state the function of updating for the individual and memory population. In step 8 the reassess is performed to calculate the efficiency of the individual and if the efficiency is changed even for a single individual, we understand that the environment has changed. In step 12, evaluation to calculate the fitness of individuals is done if fitness for even an individual be changed in that case alone in that environment has changed. Step 14 based on the changes in the environment, the data stored in the memory should be applied foe the new environment which is done in 4 various phases as follows:

Phase 1- Fitness is calculated for the memory population.

Phase 2- The best individual of the memory is selected; in fact, the best individual out of this memory is the individual that has the more fitness.

Phase 3- The worst individual of the population is selected; in fact, the worst individual out of this population is the individual that has the lowest fitness.

Phase 4- A replacement strategy chooses whether to replace the best individual of the memory instead of the worst individual of the main population.

Explicit memory maintains diversity for the proposed approach throughout the run. In this work, the worst individuals of the population were replaced by the new ones.
One of the approaches for increasing efficiency of the proposed algorithm is to maintain diversity throughout the run. If a population always remains diverse, then convergence may be avoided at all times, and optimization may be more adaptive to changes. The proposed approach maintains the diversity of the population by inserting chaotic initialized individuals into the population at every generation.
Step 15 explains that "if time update cycle of the algorithm is larger than a threshold, the algorithm will activate an update memory flag; otherwise, *if* finally reaches the end".

The general flowchart of this model is shown in figure 6.

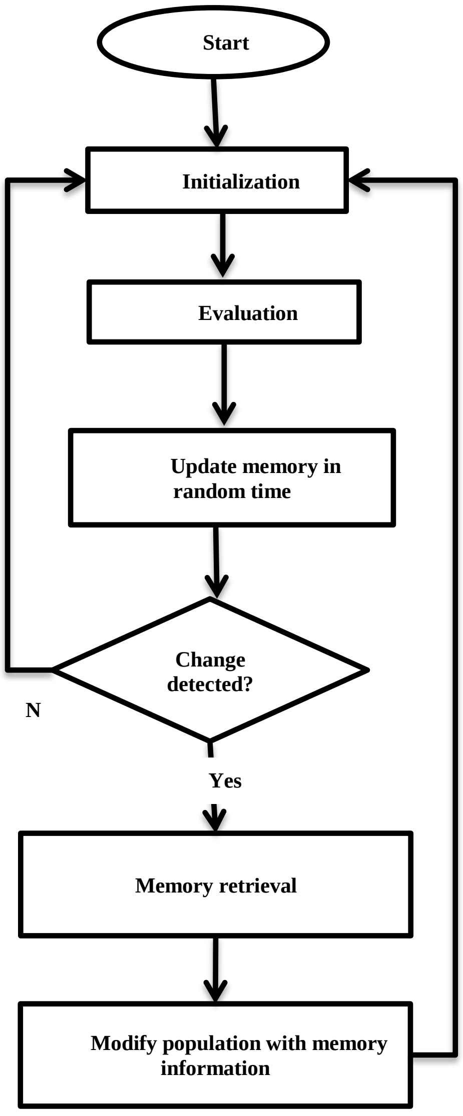

Start
Initialization
Evaluation
Update memory in random time
Change detected?
N
Yes
Memory retrieval
Modify population with memory information

Figure 6. Flowchart of proposed algorithm.

8. Experimental settings

Branke [12] has introduced a dynamic benchmark problem called the moving peaks benchmark (MPB) problem. Numerical experiments concerning MPB, scenario 2, as proposed by Branke [12], were performed in order to test the behavior of the proposed method. The default settings and definition of the benchmark used in the experiments of this paper can be found in table 1.
The default parameter setting of MPB used in the experiments is presented in table 1 [12].
The default parameter setting of proposed method is presented in table 2. Giving different values for the parameters, different results can be obtained, through which we can reach the proper (optimal) values for the parameters used in the proposed method.

To demonstrate the dynamics of the proposed algorithm, we gave the contour-plot of the population distribution at different instances of the sample run (as shown in Figure 7). A *2D* MPB problem landscape was considered with 10 optima. Figure 7 shows that in the proposed algorithm, a very high percentage of the peaks was covered in each assessment. We implemented all algorithms using Matlab 2012, and run all algorithms on a PC with a core i5 processor and a 8 G Bytes RAM memory. The proposed algorithm was compared with AmQSO, mQSO, FMSO, cellular PSO, rPSO, and CPSO. For mQSO, we adapted the configuration 10 (5+5q), which created 10 swarms with 5 neutral (standard) particles and 5 quantum particles with $rcloud = 0.5$ and $rexcl = rconv = 31.5$, as suggested in [17].

For FMSO, there are at most 10 child swarms; each has a radius of 25.0. The size of the parent and the child swarms were set to 100 and 10 particles, respectively [19]. For FMSO, there are at most 10 child swarms each has a radius of 25.0. The size of the parent and the child swarms are set to 100 and 10 particles, respectively [16].

For cellular PSO, a 5-dimensional cellular automaton with 105 cells and Moore neighborhood with radius of two cells is embedded into the search space. The maximum velocity of particles is set to the neighborhood radius of the cellular automaton and the radius for the random local search (r) is set to 0.5 for all experiments. The cell capacity θ is set to 10 particles for every cell [31].

In CPSO, each particle learns from its own historical best position and the historical best position of its nearest neighbor other than the global best position, as in the basic PSO algorithm. Using a hierarchical clustering method, the whole swarm in CPSO can be divided into sub-swarms that cover different local regions. In order to accelerate the local search, a learning strategy for the global best particle was also introduced in CPSO [15]. Hu and Eberhart proposed re-randomization PSO (RPSO) for optimization in dynamic environments [29] in which some particles randomly are relocated after a change is detected or when the diversity is lost, to prevent losing the diversity.

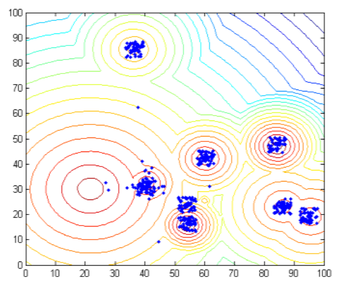
Blackwell et al. [18] introduced compound particle swarm optimization (AmQSO) utilizing a new type of particles which helps explore the search space more comprehensively after a change occurred in the environment. For all algorithms, we reported the average offline error and 95% confidence interval for 100 runs.

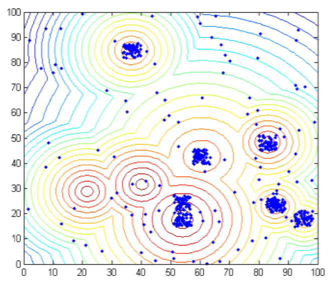
Table 1. Standard configuration parameters for MPB problem [22].

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

Table 2. Parameter values for proposed algorithm.

Parameter	Value
Lower bound	0
Upper bound	100
Total population	100
Memory Size	10
Probability of Crossover	0.6
Probability of Mutation	0.2
Logistic factor (A)	4


a


b

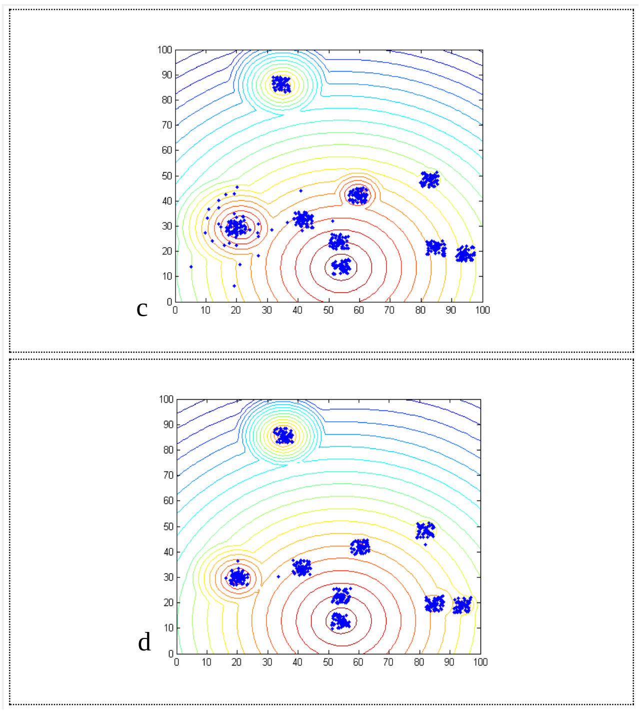


Figure 7. Trend of convergence individuals for peaks in through the run of the algorithm with population size = 100, peaks number = 10, frequency of change = 5000, and shift length = 1.

8.1. Influence of different parameter values on proposed algorithm

Table 3 shows the average offline error for the proposed method and the other methods of frequency change of 500 and different numbers of peaks.

From the results of the table 3, it can be easily seen that the proposed method outperforms all the other peer algorithms when the number of peaks is larger than one. Table 4 shows the mean offline error for the proposed method and the other methods in the frequency change of 1000 and varying the number of peaks.

Table 5 shows the mean offline error for the proposed method and the other methods in the frequency change of 5000 and varying the number of peaks. It can be seen in table 5 that the performance of the proposed algorithm was not influenced too much when the number of peaks increased. Generally, increasing the number of peaks makes it harder for algorithms to track the optima. However, the offline error decreases when the number of peaks is larger than 50 for the proposed algorithm. Figures 8, 9, 10, and 11 show the offline error for the proposed algorithm, respectively, with frequency changes of 500 and 5000 and 10 peaks and 50 peaks. Table 6 shows the results of the proposed method with different dimensions involving peaks number 10, frequency change of 5000, and shift length of 1, in addition to those of mQSO, adaptive mQSO, rPSO, and mPSO [31]. Result of exist in table 6 shows with dimension 3, 4, 5, 10, 15, 20 of the landscape space; the performance of the proposed algorithm was better than the other algorithms. Table 7 shows the offline error for the proposed algorithm with frequency change of 500 and different dimension and different numbers of peaks. Table 8 shows offline error for the proposed algorithm with different severity of change and different number of peaks. Figure 12 shows the offline error for the proposed algorithm with shift lengths of 5 and 7. Figure 13 shows that the percentage cover of peaks for the proposed algorithm with population size is 300 and high frequency change applying as well. Increase of population size help at the speed convergence of the proposed algorithm. Figure 14 shows comparison proposed algorithm with AmQSO algorithm with different correlation coefficients, frequency of change of 500, number of peaks of 10, and shift length of 1. Figure 15 shows the average offline error for the proposed algorithm with different memory sizes and default values for MPB problem.

Table 3. Average offline error for different algorithms on MPB problem with different numbers of peaks and frequency 500.

Peak number	Proposed algorithm	mQSO	rPSO	FMSO	Cellular PSO[32]	AmQSO	CPSO
1	**2.85(0.22)**	33.67(3.4)	4.27(-)	7.58(0.9)	13.46(0.3)	3.02(0.32)	14.25(-)
5	**3.57(0.25)**	11.91(0.7)	16.19(-)	9.45(0.4)	9.63(0.49)	5.77(0.56)	36.40(-)
10	**3.96(0.21)**	9.62(0.34)	17.34(-)	18.26(0.3)	9.35(0.37)	5.37(0.42)	20.91(-)
20	**4.05(0.18)**	9.07(0.25)	17.06(-)	17.34(0.3)	8.84(0.28)	6.82(0.34)	13.11(-)
30	**4.67(0.20)**	8.80(0.21)	16.98(-)	16.39(0.4)	8.81(0.24)	7.10(0.39)	10.83(-)
40	**4.95(0.15)**	8.55(0.21)	16.64(-)	15.34(0.4)	8.94(0.24)	7.05(0.41)	10.12(-)
50	**5.23(0.17)**	8.72(0.20)	15.77(-)	5.54(0.2)	8.62(0.23)	8.97(0.32)	9.28(-)
100	**5.06(0.16)**	8.54(0.16)	14.55(-)	2.87(0.6)	8.54(0.21)	7.34(0.31)	7.77(-)
200	**4.81(0.13)**	8.19(0.17)	13.40(-)	11.52(0.6)	8.28(0.18)	7.48(0.19)	6.83(-)

Table 4. Average offline errors for different algorithms on MPB problem with different numbers of peaks and frequency 1000.

Peak number	Proposed algorithm	mQSO	rPSO	FMSO	Cellular PSO	AmQSO	CPSO
1	**1.10(0.10)**	18.60(1.3)	1.94(-)	14.42(0.9)	6.77(0.38)	2.33(0.31)	8.93(-)
5	**1.12(0.11)**	6.56(0.38)	13.77(-)	10.59(0.4)	5.30(0.32)	2.90(0.32)	8.62(-)
10	**1.28(0.13)**	5.71(0.22)	15.55(-)	10.40(0.3)	5.15(0.19)	4.56(0.40)	7.48(-)
20	**1.76(0.9)**	5.85(0.15)	15.54(-)	10.33(0.3)	5.23(0.18)	5.36(0.47)	6.10(-)
30	**2.01(0.14)**	5.81(0.15)	14.38(-)	10.06(0.4)	5.33(0.16)	5.20(0.38)	5.44(-)
40	**2.23(0.16)**	5.70(0.14)	14.11(-)	9.85(0.4)	5.61(0.16)	5.25(0.37)	5.57(-)
50	**2.56(0.10)**	5.87(0.13)	13.75(-)	9.54(0.2)	5.55(0.14)	6.06(0.14)	5.17(-)
100	**2.42(0.14)**	5.83(0.13)	12.27(-)	8.77(0.6)	5.57(0.12)	4.77(0.45)	4.26(-)
200	**2.20(0.11)**	5.54(0.11)	11.32(-)	8.06(0.6)	5.50(0.12)	5.75(0.26)	3.74(-)

Table 5. Average offline errors for different algorithms on MPB Problem with different numbers of peaks and frequency 5000.

Peak number	Proposed algorithm	mQSO	rPSO	FMSO	Cellular PSO	AmQSO	CPSO
1	0.92(0.09)	3.82(0.35)	0.56(0.04)	3.44(0.1)	2.54(0.1)	0.51(0.0)	**0.14(0.11)**
5	1.06(0.7)	1.90(0.08)	12.58(0.76)	2.94(0.0)	1.72(0.1)	1.01(0.0)	**0.72(0.72)**
10	1.15(0.10)	1.91(0.08)	12.98(0.48)	3.11(0.0)	1.76(0.1)	1.51(0.1)	**1.05(0.24)**
20	**1.18(0.06)**	2.56(0.10)	12.79(0.06)	3.36(0.0)	2.59(0.1)	2.00(0.1)	1.59(0.22)
30	**1.35(0.05)**	2.68(0.10)	12.35(0.54)	3.28(0.0)	2.95(0.1)	2.19(0.1)	1.58(0.17)
40	1.53(0.09)	2.65(0.08)	11.23(0.62)	3.26(0.0)	3.11(0.1)	2.28(0.1)	**1.51(0.12)**
50	1.65(0.07)	2.63(0.08)	11.34(0.29)	3.22(0.0)	3.22(0.1)	2.43(0.1)	**1.54(0.12)**
100	1.80(0.06)	2.52(0.06)	9.73(0.28)	3.06(0.0)	3.39(0.1)	2.68(0.1)	**1.41(0.08)**
200	1.71(0.05)	2.30(0.05)	8.90(0.19)	2.84(0.0)	3.36(0.0)	2.62(0.1)	**1.24(0.06)**

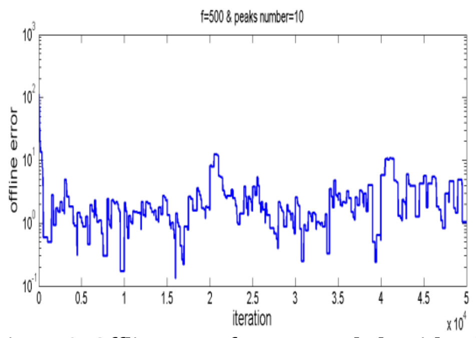


Figure 8. Offline error for proposed algorithm in frequency = 500 and peaks number = 10.

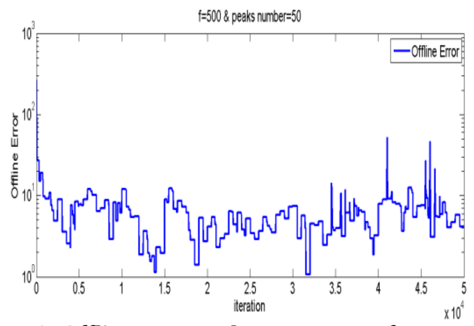


Figure 9. Offline error and current error for proposed algorithm in frequency = 500 and peak number = 50.

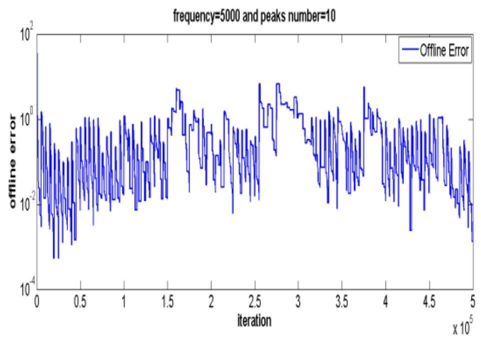


Figure 10. Offline error and current error for proposed algorithm in frequency = 5000 and peak number = 50

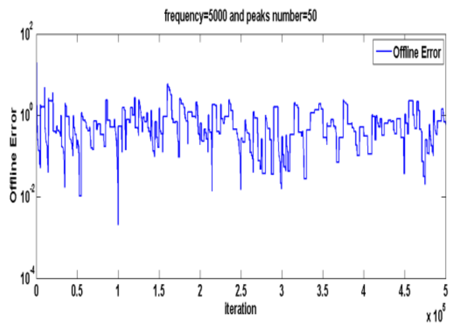


Figure 11. Offline error for proposed algorithm in frequency = 5000 and peak number = 10

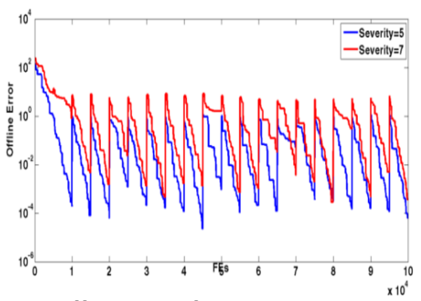


Figure 12. Offline error for proposed algorithm with shift lengths of 5 and 7 with frequency of 5000, and peaks number of 10.

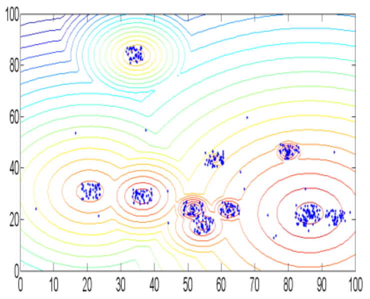


Figure 13. Cover of peaks with iteration = 500000, frequency change = 10000, peak number = 10, population size = 300, shift length = 1, and other default parameter values for MPB benchmark.

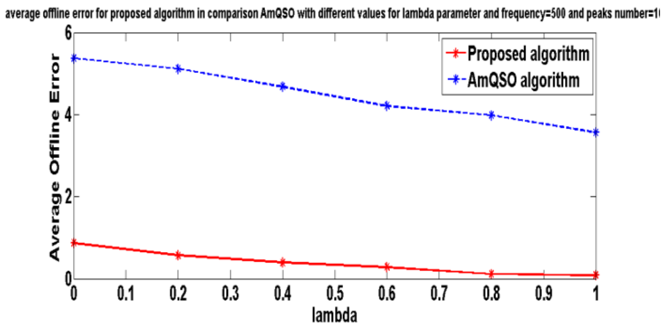


Figure 14. Comparison between proposed algorithm with AmQSO in frequency change = 500, peak number = 10, and different correlation coefficients (λ).

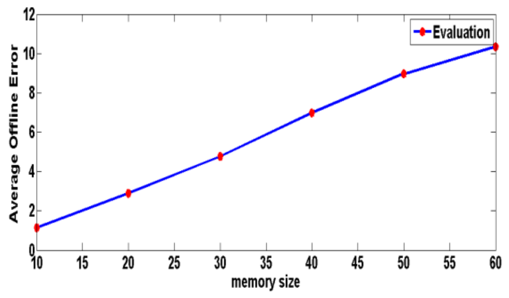


Figure 15. Average offline error for proposed algorithm with different memory sizes and default values for MPB problem.

Table 6. Result of proposed method with different dimensions involving peak number of 10, frequency change of 5000, and shift length of 1, in comparison with mQSO, AmQSO, rPSO, and mPSO.

Algorithm	Dimension						
	2	3	4	5	10	15	20
Proposed method	0.96(0.09)	**1.08(0.11)**	**1.11(0.10)**	**1.15(0.13)**	2.45(0.28)	**3.85(0.31)**	**4.25(0.35)**
Adaptive mQSO	**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)
mQSO	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)
rPSO	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)
mPSO	1.24(0.07)	1.42(0.10)	1.35(0.09)	1.51(0.12)	4.32(0.26)	7.07(0.25)	10.77(0.40)

Table 7. Offline error for proposed algorithm with frequency change = 5000 and different peaks and different dimensions and shift length = 1.

peaks	Dimension				
	5	7	10	15	20
1	0.92(0.08)	1.11(0.14)	2.09(0.17)	2.80(0.20)	3.15(0.41)
5	1.06(0.09)	1.23(0.13)	2.25(0.21)	2.95(0.25)	3.26(0.40)
10	1.15(0.13)	2.09(0.16)	2.45(0.23)	3.85(0.28)	4.25(0.39)
20	1.18(0.09)	2.45(0.11)	3.56(0.25)	4.23(0.27)	6.85(0.39)
30	1.35(0.10)	2.66(0.16)	3.84(0.19)	5.25(0.29)	6.92(0.46)
40	1.53(0.12)	2.90(0.12)	3.96(0.18)	5.63(0.31)	7.12(0.48)
50	1.85(0.08)	3.56(0.10)	4.25(0.21)	5.91(0.33)	7.93(0.45)
100	1.80(0.13)	3.42(0.14)	4.15(0.21)	5.64(0.37)	7.76(0.53)
200	1.71(0.15)	3.36(0.11)	4.03(0.20)	5.45(0.30)	7.60(0.51)

Table 8. Offline error for proposed algorithm with frequency change = 5000 and different peaks and different dimensions and shift length = 1.

peaks	shift length				
	2	3	4	5	6
1	1.89(0.13)	3.02(0.19)	5.09(0.32)	6.12(0.43)	8.09(0.56)
5	2.06(0.10)	3.16(0.22)	5.23(0.30)	6.26(0.40)	8.12(0.56)
10	2.19(0.15)	3.31(0.25)	5.35(0.35)	6.45(0.45)	8.15(0.68)
20	2.36(0.17)	3.65(0.20)	5.41(0.35)	6.52(0.42)	8.53(0.70)
30	2.52(0.14)	3.74(0.22)	5.49(0.37)	6.58(0.42)	8.65(0.50)
40	2.70(0.17)	3.86(0.25)	5.62(0.33)	6.69(0.45)	8.69(0.55)
50	2.92(0.16)	3.95(0.27)	5.87(0.36)	6.87(0.47)	8.74(0.52)
100	2.80(0.14)	3.90(0.25)	5.96(0.35)	6.80(0.49)	8.79(0.60)
200	2.71(0.14)	3.86(0.26)	5.90(0.30)	6.96(0.46)	8.98(0.53)

Table 9 shows Min, Max and Std (standard divation) error for the proposed algorithm with frequency change = 5000 and different peaks and different dimension and shift length = 1.

Table 9. Min, Max, and std error for proposed algorithm with frequency change = 5000 and different peaks and different dimensions and shift length = 1.

peaks	Min error	Max error	Std error
1	0.62	0.11	0.09
5	1.00	1.21	0.7
10	1.02	1.27	0.10
20	1.07	1.29	0.06
50	1.10	1.35	0.05
100	1.12	1.41	0.09
200	1.16	1.62	0.07

9. Conclusion

In dynamic problems, storing and maintaining the memory has been one of the largest problems examined in the prior works. First, it must be decided how often to update the memory. Secondly, we should decide what should continue to be stored in the memory as one tries to add a new environment.

Memory usage information to remember environment and good storage solutions that are not too old can increase the efficiency of the algorithm. The solutions maintained from the past can be tracked for future exploration. The novel strategy designed for updating memory at the proposed algorithm maintains diversity among population through the run. The retrieval of memorized individuals, which usually occurs after a change, takes place before the re-adaptation of the proposed algorithm to the new environment.

A chaotic system has a precise prediction of the future in comparison with a random system. Using the chaos theory for initializing population helps speed up the convergence of individuals in the proposed algorithm. However, it is worth mentioning that in the dynamic environment, the diversity and convergence dilemma is foundation of local search and global search dilemma where it is gained by keeping old solutions in a memory. Therefore, the mechanism design and new algorithm change proposal to solve the issues with dynamic optimization qualities and the challenges of different options can be suitable for future works.

References

[1] Ramsey, C. L. & Grefenstette, J. J. (1993). Case-based initialization of genetic algorithms. In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 84-91. Morgan Kaufmann.

[2] Louis, S. J. & Xu, Z. (1996). Genetic algorithms for open shop scheduling and re- scheduling. In M. E. Cohen and D. L. Hudson, editors, Proceedings of the Eleventh International Conference on Computers and their Applications (ISCA), pp. 99-102.

[3] Mori, N., Kita, H. & Nishikawa, Y. (1996). Adaptation to a changing environment by means of the thermo dynamical genetic algorithm. In H.-M. Voigt, editor, Parallel Problem Solving from Nature (PPSN IV), volume 1141 of Lecture Notes in Computer Science, pp. 513-52.

[4] Mori, N., Kita, H. & Nishikawa, Y. (1997). Adaptation to changing environments by means of the memory-based thermo dynamical genetic algorithm. In I. Back, editor, Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA 1997), pp. 299-306. Morgan Kaufmann.

[5] Mori, N., Kita, H. & Nishikawa, Y. (1998). Adaptation to a changing environment by means of the feedback thermo dynamical genetic algorithm. In Parallel Problem Solving from Nature (PPSN V), vol. 1498 of Lecture Notes in Computer Science, pp. 149-158.

[6] Uyar, A. S. & Harmanci, A. E. (2002). Preserving diversity in changing environments through diploidy with adaptive dominance. In W. B. Langdon and et al., editors, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), page 679. Morgan Kaufmann.

[7] Simoes, A & Costa, E. (2003). An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory. In D. W. Pearson, N. C. Steele, and R. Albrecht, editors, Proceedings of the 6th International Conference on Artificial Neural Networks (ICANNGA 2003), pp. 168-174. Springer-Verlag.

[8] Yang, S. (2006). A comparative study of immune system based genetic algorithms in dynamic environments. In M. Keijzer and et al., editors, Proceedings of the Eighth International Genetic and Evolutionary Computation. Conference (GECCO 2006), pp. 1377-1384. ACM Press.

[9] Liu, L., Wang, D. & Yang, S. (2009). An immune system based genetic algorithm using permutation-based dualism for dynamic traveling salesman problems. In M. Giacobini and et al., editors, Evo Workshops 2009: Applications of Evolutionary Computing (EVOSTOC 2009), vol. 5484 of Lecture Notes on Computer Science, pp. 725-734. Springer.

[10] Mohammadpour, M. & Parvin, H. (2016). "Genetic Algorithm Based on Explicit Memory for Solving Dynamic Problems", In Journal of Advances in Computer Research Sari Branch Islamic Azad University. vol. 7, no. 2, pp. 53-68.

[11] Yang, S. (2005). Memory-based immigrants for genetic algorithms in dynamic environments. In H.-G. Beyer, editor, Proceedings of the Seventh International Genetic and Evolutionary Computation Conference (GECCO2005), vol. 2, pp. 1115-1122. ACM Press.

[12] Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1875-1882. IEEE Press.

[13] Mohammadpour, M. & Parvin, H. (2016). Chaotic genetic algorithm based on clustering and memory for solving dynamic optimization problem. Tabriz Journal of Electrical Engineering, vol. 46, no. 3 (Persian Journal).

[14] Yang, S. & Li, C. (2012). A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Trans. vol. 16, no. 4. pp. 959-974.

[15] Yang, S. & Li, C. (2009). A clustering particle swarm optimizer for dynamic optimization. in Proc. Congr. Evol. Comput., pp. 439–446.

[16] Branke, J., Kauler, T. & Schmidt, C. (2000). A multi-population approach to dynamic optimization problems. In I. Parmee, editor, Proceedings of Adaptsim03ive Computing in Design and Manufacture (ACDM 2000), pp. 299-308. Spriger-Verlag.

[17] Blackwell, T. & Branke, J. (2006). Multi-Swarms, Exclusion, and Anti-Convergence in Dynamic Environments. IEEE Transactions on Evolutionary Computation 10, pp. 459–472.

[18] Blackwell, T., Branke, J. & Li, X. (2008). Particle swarms for dynamic optimization problems. Swarm Intelligence. Springer Berlin Heidelberg,. pp. 193-217.

[19] Yang, S. & Li, C. (2008). Fast Multi-Swarm Optimization for Dynamic Optimization Problems. Proc, Int'l Conf. Natural Computation, vol. 7, no. 3, pp. 624-628.

[20] Lung, R. I. & Dumitrescu, D. (2010). Evolutionary swarm cooperative optimization in dynamic environments. Natural Comput., vol. 9, no. 1, pp. 83–94.

[21] Lung, R. I. & Dumitrescu, D. (2007). A collaborative model for tracking optima in dynamic environments. In Proc. Congr. Evol. Comput, pp. 564–567.

[22] Branke, J. (2002). Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers.

[23] Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1875-1882. IEEE Press.

[24] Holland, J. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI.

[25] Ryan, C. (1997). Diploidy without dominance. In Nordic Workshop on Genetic Algorithms, pp. 45–52.

[26] Yang, S. (2007). Explicit memory schemes for evolutionary algorithms in dynamic environments. In S. Yang, Y.-S. Ong, and Y. Jin, editors, Evolutionary Computation in Dynamic and Uncertain Environments, volume 51 of Studies in Computational Intelligence, pp. 3-28. Springer-Verlag.

[27] Ramsey, C. & Grefenstette, J. (1993). Case-based initialization of genetic algorithms. In S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 84-91. Morgan Kaufmann.

[28] Trojanowski, K. & Michalewicz, Z. (1999) Searching for optima in non-stationary environments. in Proc of the IEEE Congress on Evolutionary Computation (CEC 1999), pp. 1843-1850. IEEE Press.

[29] Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In Congress on Evolutionary Computation, pp. 1875–1882.

[30] Wang, H. & Yang, S. (2012). Ip WH, Wang D, A memetic particle swarm optimization algorithm for dynamic multi modal optimization problems. Int J Syst Sci, vol. 43, no. 7, pp. 1268-1283.

[31] Hashemi, B. & Meybodi, M. R. (2009). Cellular PSO: A PSO for Dynamic Environments. In Advances in Computation and ntelligence, Lecture Notes in Computer Science, vol. 5821, pp. 422-433.

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

[33] Ozsoydan, F. B. & Baykasoglu, A., (2015). A multi-population firefly algorithm for dynamic optimization problems, Evolving and Adaptive Intelligent Systems (EAIS), 2015 IEEE International Conference. pp. 1-7.

[34] Sadeghi, S., Parvin, H. & Rad, F. (2015). Particle Swarm Optimization for Dynamic Environments, Springer International Publishing, 14th Mexican International Conference on Artificial intelligence, MICAI 2015, pp. 260-269, October 2015.

[35] Nguyen, T. T., (2013). Solving dynamic optimization problems by combining Evolutionary Algorithms with KD-Tree, Soft Computing and Pttern Recogonition (SoCPaR), International Conference, pp. 247-25.

[36] Yildiz, A., Lekesiz, H. & Yi;diz A. R. (2016). "Structural design of vehicle components using gravitational search and charged system search algorithm, Material Testing, vol. 58, no, 1, pp. 79-91.

[37] Kiani, M. & Yildiz, A. R, (2015). A Comparative Study of Non-traditional Methods for Vehicle Crashworthiness and NVH Optimization, In J Crashworthiness, pp. 1-12, doi: 10.1007/s11831-015-9155-y.

[38] Motameni, H. (2016). PSO for multi-objective problems: Criteria for leader selection and uniformity distribution. Journal of Artificial Intelligence and Data Mining, vol. 4, no. 1, pp. 67-76. doi: 10.5829/idosi.JAIDM.2016.04.01.08.

الگوریتم ژنتیک آشوب‌گونه مبتنی بر حافظه صریح با راه‌کاری جدید برای به‌روزرسانی و بازیابی از حافظه در محیط‌های پویا

مجید محمدپور[1,*]، حمید پروین[2] و مجید سینا[3]

[1] دانشکده مهندسی برق و کامپیوتر، دانشگاه آزاد اسلامی واحد علوم تحقیقات یاسوج، یاسوج، ایران.

[2] دانشکده مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد نورآباد ممسنی، فارس، ایران.

[3] دانشکده مهندسی برق و کامپیوتر، دانشگاه آزاد اسلامی واحد علوم تحقیقات یاسوج، یاسوج، ایران.

ارسال ۲۰۱۵/۰۷/۲۱؛ بازنگری ۲۰۱۶/۰۷/۲۱؛ پذیرش ۲۰۱۶/۰۸/۲۶

چکیده:

اکثر مسائل موجود در فرآیند بهینه‌سازی ویادگیری دارای ماهیت پویا هستند. بنابراین برای حل این مسائل، الگوریتم‌هایی نیاز است که به‌صورت پویا با شرایط این مسائل سازگاری یافته و شرایط جدید را برای این مسائل جستجو نمایند. اغلب اوقات استفاده از اطلاعات گذشته باعث می‌شود که الگوریتم با شرایط تغییریافته به‌سرعت سازگاری پیدا کند. یکی از ایده‌های ارائه شده در این زمینه استفاده از حافظه می‌باشد که شامل فرآیندهای به‌روزرسانی و بازیابی می‌باشد. در این مقاله ما یک الگوریتم ژنتیک آشوب‌گونه با حافظه برای حل مسائل بهینه‌سازی پویا ارائه نموده‌ایم. یک سیستم آشوب‌گونه پیش‌بینی دقیق‌تری از آینده نسبت به یک سیستم تصادفی دارد. در روش پیشنهادی ما یک حافظه جدید با حداکثر تنوع پیشنهاد داده‌ایم. برای حافظه پیشنهادی از راه‌کار جدیدی برای به‌روزرسانی و بازیابی اطلاعات استفاده شده است. برای آزمایش کارآیی روش پیشنهادی از مسئله محک قله‌های متحرک استفاده شده که رفتاری شبیه به مسائل پویا در دنیای واقعی را شبیه‌سازی می‌کند. نتایج آزمایش‌ها برتری روش پیشنهادی را در مقایسه با دیگر روش‌ها برای محیط‌های پویا نشان می‌دهد.

کلمات کلیدی: محیط‌های پویا، حافظه صریح، محک قله‌های متحرک، خطای برون‌خطی، آشوب.