Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 7, No. 2, May 2016), Pages: 53-68
www.jacr.iausari.ac.ir

Genetic Algorithm Based on Explicit Memory for Solving Dynamic Problems

Majid Mohammadpour[1] and Hamid Parvin✉[2]

1) Young Researchers and Elite Club, Yasooj Branch, Islamic Azad University, Yasooj, Iran
2) Department of Computer Engineering, Yasooj Branch, Islamic Azad University, Yasooj, Iran

m.mohammadpour@iauyasooj.ac.ir; parvin@iust.ac.ir

Received: 2015/07/04; Accepted: 2015/10/04

Abstract

Nowadays, it is common to find optimal point of the dynamic problem; dynamic problems whose optimal point changes over time require algorithms which dynamically adapt the search space instability. In the most of them, the exploitation of some information from the past allows to quickly adapt after an environmental change (some optimal points change). This is the idea underlining the use of memory in the field, which involves key design issues concerning the memory content, the process of memory update, and the process of memory retrieval. With use of the Aging Best Solution and Keeping Diversity in Population, the speed convergence of algorithm can be increased. This article presents a genetic algorithm based on memory for dealing with dynamic optimization problems and focuses on explicit placement of memory schemes, and performs a comprehensive analysis on current design of Moving Peaks Benchmark (MPB) problem. The MPB problem is the most proper benchmark for simulation of dynamic environments. The experimental study show the efficiency of the proposed approach for solving dynamic optimization problems in comparison with other algorithms presented in the literature.

Keywords: *Dynamic Optimization, Genetic Algorithm, Explicit Memory, Offline Error*

1. Introduction

Nowadays engineering intervenes with issues with which we are faced. These problems will be solved (optimized) if they are minimized/maximized. In some of real-world problems the purpose of the optimization is to reduce expenses; here optimization process is seen as a minimization one. In some others of real-world problems the purpose of the optimization is to increase profit making; here optimization process is seen as a maximizing one. Optimization stands for the process of exploration of parameter space for finding the best parameters that maximize/minimize an objective function. Any instantiation of parameters is considered as a possible solution to the problem. The instantiation of parameters which leads the objective function (i.e. problem) to its best value is considered the best parameters.

There are two major categories for real-world problems: dynamic problems and static problems. In static problems, local optima parameters (including their positions) are stationary while the time pasts. In the static problems tracking the positions of local optima is an easy task. In dynamic problems, the parameters of local optima are not

stationary when the time is ticking. In the dynamic problems tracking the positions of local optima is a really hard task.

Evolutionary algorithms may be with good efficiency in static environments, but these algorithms are weak to solve dynamic optimization and in the literature it has been shown that they can't have a good performance. Therefore to solve dynamic optimization problems some strong heuristics are needed. There are some important challenges in the dynamic problems that should be handled before any algorithm can be selected from the literature or even any algorithm can be proposed.

Some of these challenges may include:(1) difficulty in updating of the memory after occurring each environmental change, (2) preserving diversity in the population of some possible solutions (provided that we intend to use a population based optimization), (3) detecting memory capacity, (4) identifying the environmental change times, (5) difficulty in updating of the population after deciding to update population. There are many methods in the literature dealing with dynamic optimization that we will discuss them in the remaining of the current section. Each of these methods manages the above mentioned challenges in a different mechanism. This paper has presented a combinational method for solving dynamic optimization problems. Proposed method uses the explicit memory mechanism in the Genetic Algorithm (GA). Novelty in the proposed method is a new strategy for updating memory. The traditional methods update only one individual at each updating time (only one individual is exchanged in memory and population). In the proposed method instead of updating only one individual from memory we have used a new strategy where it updates more than one individual at each updating time.

The paper is organized as follows. Section 1 introduces the dynamic optimization problems, dynamic environments, MPB problem, Offline Error (the measure of effectiveness of evolutionary algorithms in dynamic environments), genetic algorithm and memory. Section 2 presents related work. The proposed algorithm is presented in section 3. Section 4 presents the experimental study and discussions. Finally, conclusions are given in section 5.

1.1 Dynamic Optimization Problems

In most real-world problems, the objective function can change over time and therefore optimal points in these problems may be due to changes in the environmental change time. These problems are referred to as dynamic optimization problems.

1.2 Dynamic Environments

Those real world environments in that the optimal points are not static and are changing over the time are referred to as dynamic environments. The effect of environmental changes makes it a non-easy task to achieve desired global optima in these environments.

1.3 Moment of Change in the Environment

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.

1.4 Types of Environmental Changes

- ***Cyclic:*** in a cyclic environment, situations from the past reappear in the future in a cyclic manner. In this type of environments the length of a cycle can determine the difficulty of the problem. We say that an environment is cyclic if a set of environmental states always reappears in the same order (A-B-C-A-B-C…) [1].
- ***Cyclic with noise:*** in a noisy cyclic environment, environments from past reappear but with small differences introduced by a noise factor. We say that an environment is cyclic with noise if a set of environmental states always reappears in the same order but with small differences introduced by a noise factor (*A-B-C-A′-B′-C′*- . . .) [1].
- ***Probabilistic:*** when the transition between environmental states are governed by some distribution probability [1].
- ***Random:*** when the environmental changes occur from a state to another completely different state without any correlation with the past [1].

1.5 Severity of Change

The severity of change measures the modification strength in an environmental change. The environment can change to a completely different state or to a similar one.

1.6 Benchmark Problem

Benchmark problems are used to examination the performance of the evolutionary algorithms in dynamic environments.

1.7 Moving Peaks Benchmark

Moving Peaks Benchmark is a problem for simulating the dynamic environments. It consists of n peaks, in an m-dimensional space. It can take into consideration all actual parameters in a real world problem. The width, height and position of peaks can be changed over time. The MPB problem has been presented in the reference [2].

1.8 Measure the Effectiveness of Evolutionary Algorithms

To measure the effectiveness of evolutionary algorithms in dynamic environment Offline Error should be used. The Offline Error properly shows the effectiveness of an evolutionary algorithm in a dynamic environment. The Offline Error is calculated according to equation (1) [3].

$$Offlin\ Error = \frac{1}{FE_s}\sum_{t=1}^{FE_s}(h(t)-f(t)) \quad (1)$$

Regarding Eq. (1),the Offline Error is equal to the average of fitness of the best found position by the algorithm at the end of all environments.

1.9 Genetic Algorithm

Genetic Algorithm (GA) is based on learning method of biological evolution [4]. A GA to solve a problem produces a very large set of possible solutions (a population). Each of these solutions is evaluated by a metric to find out how it is good. Then some of the best solutions incorporate in production of new solutions (new population). This work helped to evolve the solutions (population). The evolution of GA is in such a direction that the desirable solutions will remain. With the combination of people within the current population, GA in each stage tries to generate a developed generation that is

better than the current generation. GA for the combination operator uses the name of the crossover operator that includes a single point and a multipoint crossover. The GA to probably improve the generation produced by crossover operator uses a new operator named mutation. For any iteration of the genetic algorithm each individual in the population is evaluated using a fitness function. A number of individuals in the population form a new population. It means that a number of these people (the current population and the offspring population) have chosen to be used as the next population.

Figure 1 shows the pseudo code of the GA.

Algorithm1: 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 t rue
```

Figure 1. Pseudo code of the standard genetic algorithm

1.9.1 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 be set to different values (or alleles).

1.9.2 Representation

The choice for the representation of a chromosome is made according to the type of the problem that should be solved. The representation defines how the individuals of the population will be encoded.

1.9.3 Fitness Function

The fitness function is used to measure the quality of an individual of the population. To measure it, a decoding process is needed to obtain the individual's phenotype (the concept of individual). The fitness of an individual is a real value obtained by applying the fitness function to the phenotype of that individual.

1.9.4 Selection

The selection method is used to choose a new population from parent population (previous population) and offspring population (created population) based on the fitness values of the individuals available in them. Solutions with higher fitness values have more probability to be chosen for mating and participating in the next population.

1.10 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, this type of the memory is used to store all information in an arbitrary chromosome (each chromosome might have two or more alleles). The implicit memory is divided into two categories: A dualism memory and a diploid memory. The diploid implicit memory functions are presented in [5]. 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 two types of direct memory and associative memory [6]. 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 into memory. Then they are reused in new environments [7]. In associative memory environmental information are stored in addition to good solutions; among the data stored in the memory there are lists of the problem space states or the likelihood of good solutions in the problem space [8]. They are then reused in the new future environment.

1.10.1 Memory Retrieval

The information stored in memory should be used for tracking of the new optima. So the best time to retrieve data from memory is the moment when the environment has been changed. Several strategies can be employed to retrieve memory. One of the methods for memory retrieval includes substituting the best individual in the memory with the worst individual in the population [9, 10].

1.10.2 Memory Update Strategy

The information of population should be used to be placed in the memory for future exploiting. The first time to store one or more individuals from population into the memory is at a random time. But the next time is the previous time plus a random value in a fixed range. There are some strategies in the literature to update the memory. We will explain some them here.

Strategy 1: In this strategy, a pair of individuals of the memory with the least distance (or the closest two individuals from the memory) are nominated as replacement candidates. The winner is someone who is less efficient. For example assume *i*-th and *j*-th individuals are the replacement candidates. If $fit(i)$ (efficiency of *i*-th individual) is less than $fit(j)$, i.e. $fit(i) < fit(j)$**,** the best person to be replaced by an individual from the population is *i*-th individual.

Strategy 2: In this strategy, the individual from the memory selected to be replaced was compared with the new individual to store. Again, suppose that individuals *i* and *j* were chosen:

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 strategy, a pair of individuals including one in the memory and one in population with the least distance (or the closest two individuals where one of them is in the memory and the other one is in the population), selected substituting candidates. Indeed in this strategy the best individual present in population will be replaced with the most similar one in memory (provided that the individual of the population has more efficiency than the individual of the memory). Euclidean distance can be employed for similarity measure.

2. Related Work

Among the methods proposed to solve dynamic optimization problems, the methods that use combinational strategies are superior to others. The memory usage mechanism and diversity preserving mechanism that are two of the most versatile mechanisms have simultaneously and successfully been used in [11-14].

Yang et al, proposed CPSOR method, that in their method, particles are divided to into clusters as the smallest particles existing in each cluster are to search local cluster in practice. Every particle that is near to the average position of that cluster is the cluster center. In this method if the particles of a cluster converged or be very sparse, then some of the particles will be done by random immigration to the area more privacy. Another criterion for preventing the convergence is used in the name of degree variety. The variety after every change is computed in the environment [3].

In CPSO algorithm to cluster particles are particles that divided the existing cluster in each to search in local Pay cluster. Each particle is the average of the best locations visited by the particles in the cluster that is in move-to. In this algorithm is called the degree of the measure on particle failure used to be that if the search space in an area of bustle creates a random particle came as some of the particles in this region have created overcrowding and together have fallen more into the area of the search space of random immigration retreats. Achieving optimal solution for multi-peak function in a dynamic environment optimization using evolutionary algorithms, particle swarm is change. To achieve this, a speciation model that allows the parallel development of the following, and for clustering using k-mean method used in this population [15].

CGAR algorithm presented by Yang et. al [3]. In their algorithm a standard genetic algorithm with a simple crossover operator and a simple mutation operator is employed. They also benefits from the k-means clustering.

SOS begins with a number of probably good solutions. When a peak (a good solution) found population is divided into two subsections. Song population must be able to trace the other couriers, while the search for a new peak population [16].

Particle swarm optimization algorithm based on quantum particles is named mQSO algorithm. In mQSO the population is divided into several groups. Three quantum particles is also employed called functional diversity. Pluralism is an anti-convergence mechanism employed by mQSO [17]. Quantum particles [17] are placed in random positions to maintain the diversity of groups. If you find a real function overlap between the two groups, the worse group will be re-initialized. Anti-convergence is activated when all groups are converged. In anti-convergence operation the groups will be re-initialized [17]. In algorithm Adaptive mQSO [18], the number of groups from the very beginning is not determined; when the change in environment occurs or the new peaks emerged the number of new groups will be increased. So the operator against convergence is activated when all groups converges. In memory/search population, the number of the individuals can be divided into two populations. Indeed they have claimed that the population consists of 'memory' individuals (named 'memory' population) and population individuals (named 'search' population). The first population is based on memory and is notation for the possible solutions of the old. The second population that is search population wants to explore new peaks and to introduce them to the used memory. The second population is randomly initialized after each change [18].

The method of FMSO [19] is parent of a group. Each group wants to identify promising areas, i.e. a group of children is used for exploration of a local search space. Each child has its own search area. In their method there is a trade-off between local search and global search. The method of FMSO searches an area to form a circle centered in the best particle in each group. Each particle that has a shorter distance than the radius of the circle (and is closer to the center of the particle) belongs 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.

FSABC presented in [22]. An information sharing Artificial Bee Colony (ABC) algorithm has been proposed for locating and tracking multiple peaks in non-stationary environments. The niching method has been adapted by hybridizing two techniques. A modified variant of the fitness sharing has been used for detecting multiple peaks simultaneously and a speciation based technique is employed to keep the better individuals of the previous generation. The base algorithm used here is a modified variant of ABC that helps to synchronize the employer and onlooker forager swarms by synergizing the local information. The main crux of our algorithm is its independency of the problem dependent control parameters, like niche radius, and the absence of any hard-partitioning technique that leads to high computational burden [22].

ICATS method proposed in [23]. The first stage of the ICATS is to solve the TSP by the imperialist competitive algorithm (ICA), and then the TS is used for improving solutions. This process avoids the premature convergence and makes better solutions. The traveling salesman problem (TSP) is the problem of finding the shortest tour through all the nodes that a salesman has to visit. The TSP is probably the most famous and extensively studied problem in the field of combinatorial optimization. Because this problem is an NP-hard problem, practical large-scale instances cannot be solved by exact algorithms within acceptable computational times [23].

3. Proposed Algorithm

In this article we have proposed a different approach for dealing dynamic environments. This approach uses Genetic Algorithm with Explicit Memory.

It is of course true that there are many algorithmic systems that could be explored to solve the types of optimization problems. For instance, neural networks, agent-based modeling systems, classifier systems, reinforcement learning techniques, and Bayesian based systems could all provide potential solutions to the problem of learning within a dynamic environment. So the natural question is: **Why Genetic Algorithms?** The answer for me is that since nature uses an evolutionary process in a dynamic environment, it would be interesting to investigate a similar process in dynamic environments that are of particular interest to engineers and scientists. The goal is not to design a perfect general search algorithm, but to describe the behavior of the genetic algorithm in dynamic environments so that we can make recommendations as to types of problems where it would be appropriate to use the GA. Most importantly, we are interested in investigating the effects of Genetic Algorithms in various settings regardless of their actual efficacy. The exploration of the Genetic Algorithm as a tool is a worthwhile goal, regardless of whether it is the best search technique for a dynamic fitness landscape.

Explicit memory used to maintain the old appropriate solutions. It maintains the old appropriate solutions so as to increase the efficiency of the algorithm. In this article first the population and the memory was initialized at random. The memory size is equal with $0.1\times N$ where N is the size of the population. The memory was updated in random time each t steps where $t \in (5,10)$. It means that after being updated, the algorithm decided the next time the memory should be updated in 5-th to 10-th next generation. If the memory is updated at generation t, the next updating generation will occur at generation $t + rand\,(5,10)$. The pseudo code for proposed algorithm is shown in Figure 2.

Algorithm2: Proposed Algorithm

```
Input:
MCN: Number of cycles that algorithm is permitted to go
US: Size of updating memory or updating population
Output: BEST Solution, BEST Fitness, Offline Error
Begin
    initialize random x and mem  % x is population and mem is memory
    f_i = fit(x_i)% f_i is fitness for i-th individual of population
    m_i = fit(mem_i)% m_i is fitness for i-th individual of memory
    UMflag = 1% Update memory flag
cycle = 1
Repeat:
  If(UMflag == 1)
    update_mem = 0
    mem = update_mem function(mem, x, US)
    update_time = rand(5,10) + cycle
  If(∃i: ((fit(mem_i) ≠ m_i) ∨ (fit(x_i) ≠ f_i))) ChangeFlag = 1 % Change detected
  If(ChangeFlag == 1) % We should reuse memory
    ChangeFlag = 0
    m_i = fit(mem_i), f_i = fit(x_i)
    j = arg min_i m_i % j is the best individual in memory
    d = arg max_i f_i % d is the worst individual in population
    x_d = mem_j% the worst individual in population is replaced with the best individual in memory
  If (update_time ≥ cycle) UMflag = 1
  cycle = cycle + 1
  Compute: OfflineError_cycle
  x = next(x)
  f_i = fit(x_i)
Until cycle == MCN
End
```

Figure 2. Pseudo code for the proposed algorithm

3.1 Memory Update

In the proposed algorithm we used a new strategy for updating memory. The strategy used in the proposed algorithm is as follows:

The function *sortedindex* (f_m) also returns an array of indices *I*, with the size of f_m. If $f_m(I)$ stands for $f_m^{'}$, then $f_m^{'}$ is sorted version of array f_m. *j*th feature of *i*th is denoted by $Pop(i)(j)$. Pop_Size shows the size of population. Mem_Size shows the size of memory. d is the number of dimensions of the problem. $Mem(i)$ stands for *i*th individual in the memory. $Pop(i)$ stands for *i*th individual in the population. *Threshold* is a parameter of replacement algorithm; It is very important for accepting a

replacement. k is the number of replacements that should be done. The pseudo-code for the updating memory is shown in Figure 3.

Algorithm3: Updating Memory

Inputs: k: Number of Memory Update
Inputs: Mem: Memory
01. $\forall i \in \{1,2,\dots,Mem\_Size\}: f_m(i) = fitnessfunction(Mem(i))$;
02. $c = 0; Q = [\ ]; I = sortedindex(f_m)$;
03. $for\ i = 1\ to\ k$
04. $lab1$: $c = c + 1$;
05. $Q(i) = s$;
06. $temp = Mem(I(c))$;
07. $\forall j \in \{1,2,\dots,Pop\_Size\}: dis(j) = \sum_{p=1}^{d}(Pop(i)(p) - temp(p))$;
08. $s = \arg_{j \in \{1,2,\dots,Pop\_Size\}-Q} \min dis(j)$;
09. $if(\max_{j \in Q} dis(j) \leq Threshold)$
10. $Q(i) = s$;
11. $else$
12. $goto\ lab1$;
13. end;
14. end;

Figure3. The pseudo code of proposed algorithm for updating population from memory

The general flowchart of this model is shown in Figure 4.

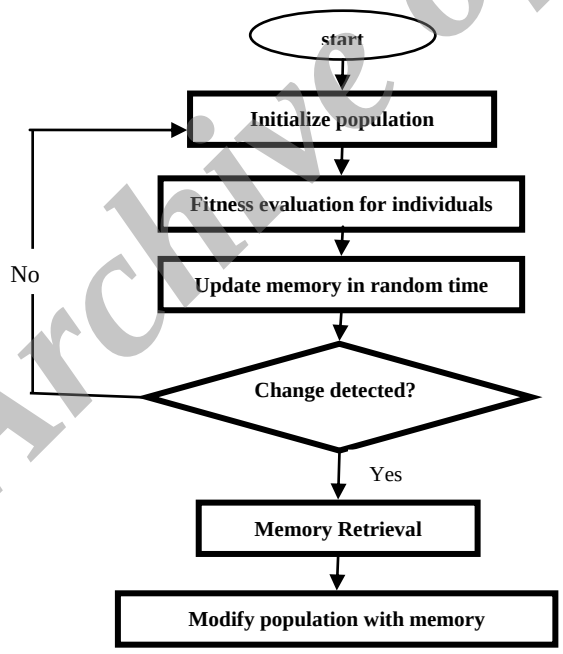

Figure 4. Flowchart of the proposed algorithm

4. Experimental Settings

In experimental section, the experiments are conducted on moving peaks benchmark (MPB), with the same parameters as proposed by Branke (2001). MPB were performed in order to test the behavior of all methods with a fix set of parameter values through all

the paper (having a set of default values) except when it is explicitly noted that a parameter has a different value. The default settings and definition of the benchmark used in the experiments of this paper can be found in Table 1. Setting of parameters for the proposed algorithm has been presented in Table 2.

Table 1. The standard configuration parameters for the dynamic peaks

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

Table 2. The proposed algorithm parameters

Parameter	Value
Lower bound	0
Upper bound	100
$Totalpopulation$	100
$MemorySize$	10
The probability of crossover	0.6
The probability of mutation	0.2
k (memory update size)	5

4.1 Varying Shift Severity

The shift severity parameter of the MPB controls the severity of the change in height, width and position of peaks. From Table (3), it can be seen that the results achieved by the proposed algorithm are much better than the results of the other 9 algorithms on the MPB problems with different shift severity, i.e. 1, 2 and 3. As it is expected, the peaks are more and more difficult to be tracked with increment of the shift severity value. Of course, the performance of all algorithms degrades when the shift severity value increases. However, the Offline Error of proposed algorithm is better than the other 9 algorithms while changing shift severity value from 1 to 2 to 3. It is also observed that, even in shift severity 5 and 6, the proposed algorithm is better than all other algorithms except ESCA algorithm.

Table 3. Average Offline Errors for Different Algorithms and proposed algorithm on the MPB Problem with Different Shift Severities

Algorithm	Shift Severity (*s*)						
	0	**1**	**2**	**3**	**4**	**5**	**6**
Proposed algorithm	1.18	**1.19**	**1.24**	**1.35**	2.67	2.20	3.19
mQSO [18]	1.17	1.75	2.40	3.0	3.59	4.24	4.79
rSPSO [34]	0.74	1.50	1.87	2.4	2.90	3.25	3.87
ESCA [19]	1.72	1.53	1.57	1.67	**1.72**	**1.78**	**1.79**
CESO [20]	**0.58**	1.38	1.78	2.03	2.23	2.52	2.74
mCPSO [18]	1.18	2.05	2.80	3.57	4.18	4.89	5.53
SPSO [27]	0.95	2.51	3.78	4.96	2.56	6.76	7.68
CGAR [3]	1.48	2.62	2.76	2.96	3.16	3.46	3.8
CDER [3]	2.56	2.52	7.47	8.62	9.81	10.7	11.4
PSO − CP [30]	0.87	1.31	1.98	2.21	2.61	3.20	3.93

4.2 Varying Number of Peaks

Table (4) presents the experimental results in terms of the Offline Error of 19 algorithms, where the results of the other 18 algorithms are provided by the corresponding papers with their optima configuration that enables them to achieve their best performances. In Table (4), mCPSO∗[17] and mQSO∗[17] denote mCPSO [17] without anti-convergence and mQSO without anti-convergence, respectively. From Table (4), it can be seen that the performance of proposed algorithm is not influenced too much when the number of peaks is increased. Usually, increasing the number of peaks makes it harder for algorithms to track the optima. However, the Offline Error decreases when the number of peaks is larger than 50 for the proposed algorithm. Figure 5 shows trend of the convergence individuals in 4 steps at the optimum peaks during algorithm run. Figure 6 shows Offline Error for the proposed algorithm on MPB problem with change frequency 5000, 10000 and number of peaks 10, 50.

Table 4. Average Offline Errors for Different Algorithms on the MPB Problem with Different Numbers of Peaks, Where the Suggested Configuration for the Framework and the Default Settings for the MPB Problem in Table 1

Algorithm	Number of peaks									
	1	**2**	**5**	**7**	**10**	**20**	**30**	**50**	**100**	**200**
Proposed algorithm	1.09	1.11	1.16	**1.17**	**1.19**	2.09	2.50	2.65	2.41	2.34
mCPSO[17]	4.93	3.36	2.07	2.11	2.08	2.64	2.63	2.65	2.49	2.44
mQSO[17]	5.07	3.47	1.81	1.77	1.80	2.42	2.48	2.50	2.36	2.26
mCPSO∗[17]	4.93	3.36	2.07	2.11	2.05	2.95	3.38	3.68	4.07	3.97
mQSO∗[17]	5.07	3.47	1.81	1.77	1.75	2.74	3.27	3.65	3.93	3.86
CESO[21]	1.04	-	-	-	1.38	1.72	**1.24**	**1.45**	**1.28**	-
rSPSO[26]	1.42	**1.10**	1.04	1.21	1.50	2.20	2.62	2.72	2.93	2.79
SPSO[27]	2.64	2.31	2.15	1.98	2.51	3.21	3.64	3.86	4.01	3.82
ESCA[20]	0.98	-	-	-	1.54	1.89	1.52	1.67	1.61	-
HmSO[28]	**0.87**	-	1.18	-	1.42	**1.5**	1.65	1.66	1.68	**1.71**
FMSO[19]	3.44	-	2.94	-	3.11	3.36	3.28	3.22	3.06	2.84
Cellular PSO[29]	2.55	-	1.68	-	1.78	2.60	2.93	3.26	3.41	3.40
rPSO[25]	0.56	-	12.58	-	12.98	12.79	12.35	11.34	9.73	8.90
Adaptive mQSO [18]	0.51	-	**1.01**	-	1.51	2.00	2.19	2.43	2.68	2.62

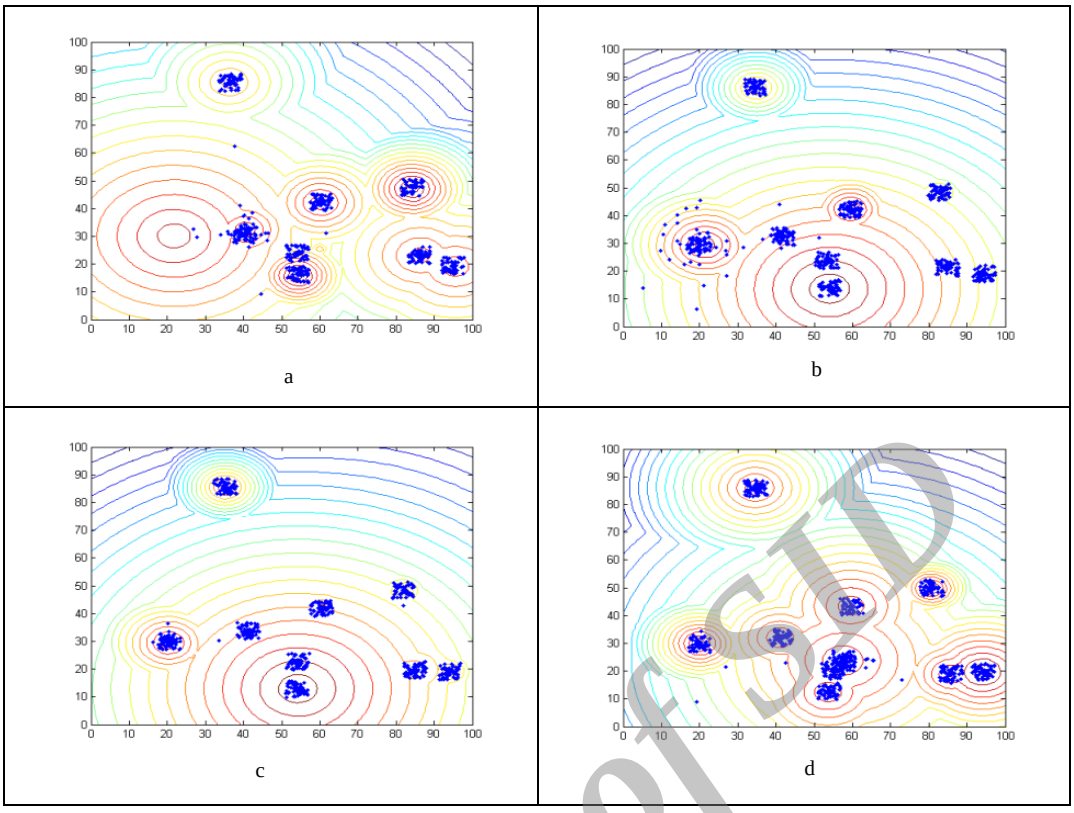

Figure 5. trend of convergence individual at the optimum peaks in during run algorithm

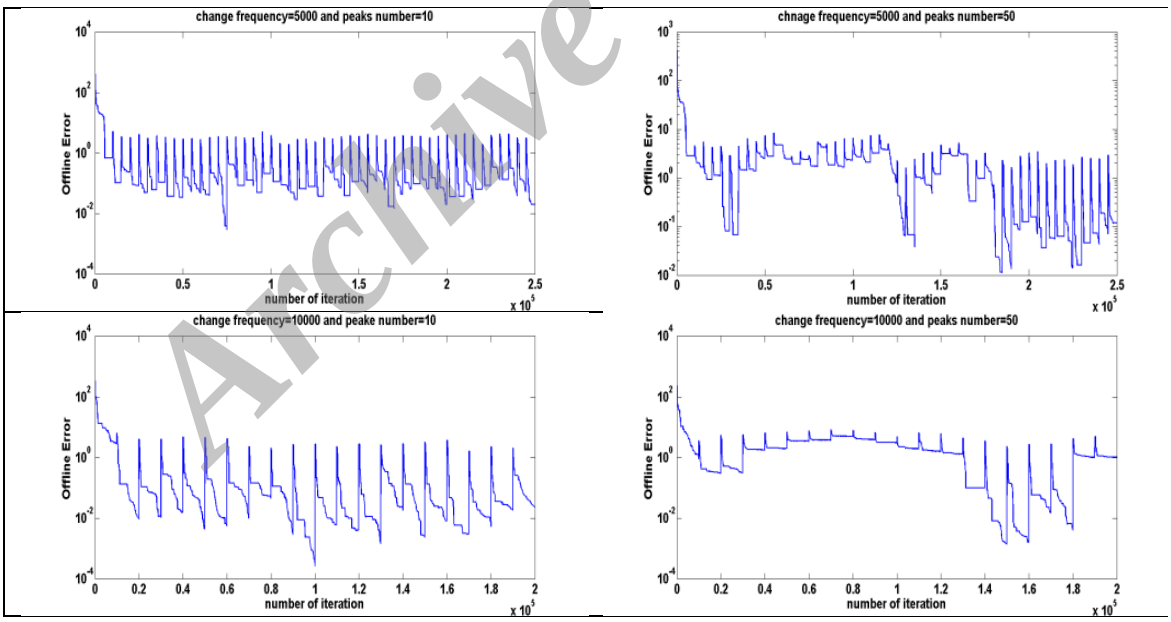

Figure 6. Offline Error for the proposed algorithm on the MPB problem with change frequency 5000, 10000 and number of peaks 10, 50.

From Figure 6, we can see that the Offline Error in the most of fitness landscape is almost zero, which is because the peaks have the same initial heights in the fitness landscape, which enables the algorithm to easily find one or more of the peaks.

4.3 Effect Dimension Number

Table (5) shows the result of the proposed method with different dimensions, in addition to those of mQSO [17], Adaptive mQSO [18], rPSO [25] and mPSO [24]. Here MPB involves 10 peaks, 5000 frequency of change and 1 shift severity. The results existed in Table (7) confirms that with increasing the number of MPB dimensions, the performance of the proposed algorithm become better and better in comparison with other algorithms. Figure 7 shows Offline Error for the proposed algorithm in the 10 and 20 dimensions.

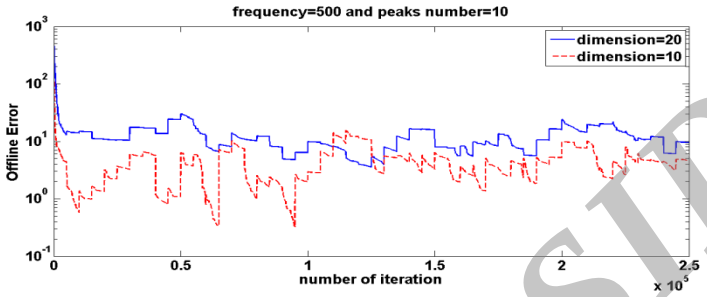

Figure 7. Shows Offline Error for proposed algorithm on the MPB problem and dimension is 10 and 20, change frequency is 500 and number of peaks is 10.

Table 5. Result of the proposed method with different dimensions involving peaks number is 10, frequency of change is 5000 and shift severity is 1, in comparison with mQSO, Adaptive mQSO, rPSO and mPSO.

Algorithm	Dimension						
	2	3	4	5	10	15	20
proposed method	0.96	1.18	1.35	**1.19**	**3.26**	**4.16**	**5.65**
Adaptive mQSO[18]	**0.71**	**1.16**	**1.33**	1.51	3.37	4.91	5.83
mQSO[17]	1.01	1.49	1.47	1.85	4.22	6.50	8.88
rPSO[25]	2.62	6.61	10.43	12.98	16.87	18.48	18.48
mPSO[24]	1.24	1.42	1.35	1.51	4.32	7.07	10.77

4.4 Effect Memory Size

The results obtained with the proposed algorithm for MPB problem, using different values for the memory size, are shown in Figure 8.

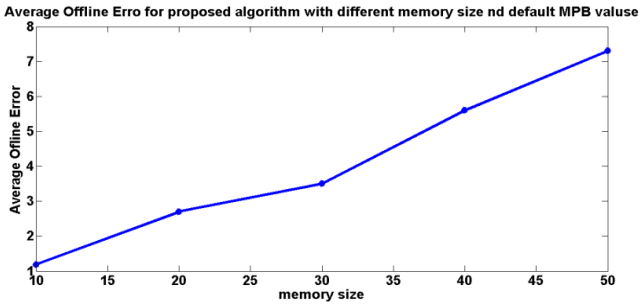

Figure 8. Effect of various values of the memory size in the proposed algorithm

4.5 Effect Update Size

The results obtained with the proposed algorithm for MPB problem, using different values for the Update Size, are shown in Figure 9.

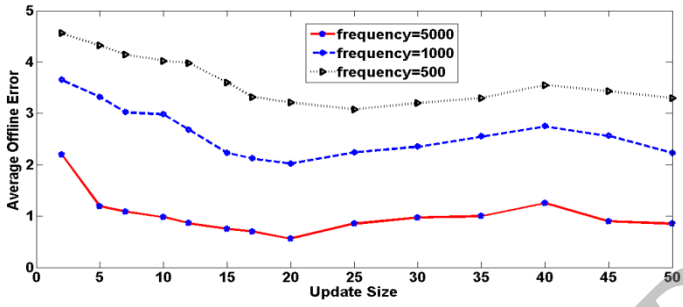

Figure 9. Effect of various values of the Update Size in the proposed algorithm

4.6 Concluding Results

Figure 10 shows the average Offline Error of the proposed method compares with the standard genetic algorithm on the different change frequencies and default parameters of the MPB problem.

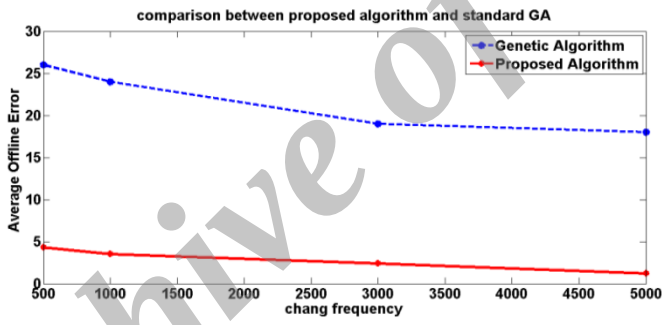

Figure 10. Shows average Offline Error for proposed algorithm compares standard GA on the MPB problem with different change frequency

5. Conclusion

Genetic Algorithm based on Explicit Memory work by storing good solutions of the current population has been proposed in the paper where the stored information can be reused later in new environments. When the environment changes, old solutions in the memory that are suitable for the new environment are reactivated allowing the genetic algorithm to readapt to the new environment. In this article a new strategy is proposed for updating memory. This strategy can update the memory more than once at a time. Usage of the aging best solution and diversity in environments helps to speed-up the convergence of the proposed algorithm. Using suitable clustering in proposed algorithm is good idea for future work in this literature.

References

[1] A. Simoes and E. Costa. "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), pages 168-174. Springer-Verlag, 2003.

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

[3] S. Yang. C .Li., "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," IEEE Trans. Vol 16, no. 4, Aug 2012.

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

[5] C. Ryan, "Diploidy without dominance," in Nordic Workshop on Genetic Algorithms, pp. 45–52, 1997.

[6] S. Yang. "Genetic algorithms with elitism-based immigrants for changing optimization problems," in Applications of Evolutionary Computing, Lecture Notes in Computer Science 4448, pages 627–636, 2007.

[7] C. Ramsey, ,J. Grefenstette., "Case-based initialization of genetic algorithms," in S. Forrest, editor, Proceedings of the Fifth International Conference on Genetic Algorithms, pages 84-91. Morgan Kaufmann, 1993.

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

[9] J. Branke. "Memory enhanced evolutionary algorithms for changing optimization problems," in Congress on Evolutionary Computation, pp. 1875–1882, 1999.

[10] C. N. Bendtsen and T. Krink. "Dynamic memory model for non-stationary optimization," In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), pages 145{150. IEEE Press, 2007.

[11] S. Yang. "Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems," In Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), volume 3, pages 2560-2567. IEEE Press, 2005.

[12] S. J. Louis and Z. Xu. "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), pages 99-102, 1996.

[13] N. Mori, H. Kita, and Y. Nishikawa. "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, pages 513-522, 1996.

[14] N. Mori, H. Kita, and Y. Nishikawa. "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), pages 299-306. Morgan Kaufmann, 1997.

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

[16] J. Branke, T. Kauler, and C. Schmidt., "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. Springer-Verlag. 2000.

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

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

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

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

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

[22] S. Biswas, S. Das, S. G.R. Kundu Patra., "Utilizing time linkage property in DOPs: An information sharing based Artificial Bee Colony algorithm for tracking multiple optima in uncertain environments," Soft Comput 18:1199–1212. 2014.

[23] M. Ahmadvand, M. Yousefikhoshbakht, N. Mahmoodi Darani., "Solving the Traveling Salesman Problem by an Efficient Hybrid Metaheuristic Algorithm," Journal of Advances in Computer Research. Vol. 3, No. 3, Pages: 75-84. , August 2012.

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

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

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

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

[28] M. Kamosi, A. B. Hashemi, and M. R. Meybodi, "A hibernating multi swarm optimization algorithm for dynamic environments," in Proc. World Congr. NaBIC, pp. 363–369, 2010.

[29] B. Hashemi and M. R Meybodi, "Cellular PSO: A PSO for Dynamic Environments," in Advances in Computation and Intelligence, Lecture Notes in Computer Science, vol. 5821, pp. 422-433, 2009.

[30] L. Liu, S. Yang, and Wang, D., "Particle swarm optimization with composite particles in dynamic environments," IEEE Trans. Syst. Man Cybern. B Cybern., vol. 40, no. 6, pp. 1634–1648, Dec. 2010.