2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Combining Multipopulation Evolutionary Algorithms with Memory for Dynamic Optimization Problems

Tao Zhu[1,2], Wenjian Luo[1,2,*] and Lihua Yue[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China
[2]Anhui Province Key Laboratory of Software Engineering in Computing and Communication in Hefei 230027, Anhui, China
Email: zhuta@mail.ustc.edu.cn; wjluo@ustc.edu.cn; llyue@ustc.edu.cn

***Abstract*—Both multipopulation and memory are widely used approaches in the field of evolutionary dynamic optimization. It would be interesting to examine the effect of the combinations of multipopulation algorithms (MPAs) and memory schemes. However, since most of the existing memory schemes are proposed with single population algorithms, straightforwardly applying them to MPAs may cause problems. By addressing the possible problems, a new memory scheme is proposed for MPAs in this paper. In the experiments, several existing memory schemes and the newly proposed scheme are combined with a MPA, i.e. the Species-based Particle Swarm Optimizer (SPSO), and these combinations are tested on cyclic and acyclic problems. The experimental results indicate that 1) straightforwardly using the existing memory schemes sometimes degrades the performance of SPSO even on cyclic problems; 2) the newly proposed memory scheme is very competitive.**

I. Introduction

In recent years, dynamic optimization problems have been attracting more and more attention in the evolutionary computation community [1]–[3]. Among the various approaches, multipopulation and memory are two widely used approaches. Generally, multipopulation algorithms (MPAs) for dynamic optimization divide the whole population into multiple subpopulations and make efforts to evolve them towards different promising regions [4]–[8]. These features of MPAs make them efficient in maintaining population diversity and tracking multiple optima in dynamic multimodal problems. By contrast, memory schemes record useful information from past and reuse it in new environments [9]–[14]. In this way, memory schemes are always efficient when new environments have close relationship with the previous ones. If the environments change randomly, their performance is much less attractive.

Most of the memory schemes were proposed with single population algorithms. There are some possible problems when combined with MPAs. Firstly, existing memory schemes constantly replace the least fit population individuals with memory individuals, which could hinder the scattering of the subpopulations towards different promising regions, and sometimes the poorly-fitted subpopulations could even be removed. This phenomenon disrupts the working mechanism of MPAs, and could harm the performance. Secondly, all the existing memory schemes have a relatively small memory capacity which can be inadequate for MPAs, because MPAs can obtain multiple optima of a multimodal environment which could be useful in the future. The dilemma is that if we enlarge the memory capacity, the increased computational overhead of memory could offset its benefit.

In order to efficiently combine memory with MPAs, in this paper a new memory scheme is proposed. The new scheme possesses a memory updating strategy that is independent of fitness. With this strategy, it does not need to reevaluate the memory frequently, so that the computational overhead is reduced and the capacity could be enlarged. When updating the population of MPAs, both space distribution and fitness of individuals is considered, so that the working mechanism of MPAs could be influenced to a slight degree.

In this paper, three existing memory schemes [9]–[11] and the new proposed memory scheme are combined with the Species-based Particle Swarm Optimizer (SPSO). Experiments are carried out to investigate the effect of these combinations. The experimental results indicate that 1) straightforwardly using the existing memory schemes sometimes degrades the performance of SPSO, even on cyclic problems; 2) the proposed memory scheme is very competitive on the test problems.

The rest of the paper is organized as follow. Section 2 briefs the related memory schemes and their combinations with SPSO. Section 3 describes the proposed memory scheme in detail. In Section 4, the experiments are carried out, and the results are presented. In Section 5, the conclusion is given.

II. Related Works

A. The Memory Schemes Used for the Comparative Study

The majority of the memory schemes employ explicit memory of solutions. Some reviews can be found in [9]. For an explicit memory scheme, three issues are often involved: 1) when and what information should be stored in the memory; 2) how to organize and update the memory; 3) how to reuse the stored information to adapt EAs to new environments. In the following, three representative explicit memory schemes are briefly introduced, which are used for comparative studies in the paper. The structure of an EA with these schemes is given in Procedure 1.

The memory immigrants (MI) is a scheme proposed in [9]. The memory of MI is randomly initialized, and is updated in a stochastic time pattern or when an environmental change is detected. When the memory is updated, the best individual in the current population replaces its closest individual (in terms of spatial distance) in the memory if it is better than the memory individual. At every generation, memory immigrants are generated by mutating the fittest memory individual to

*Corresponding author, Tel.: +86-551-63602824

take place of the worst individuals in the population. These immigrants are regarded more biased to the current environment than random immigrants [9]. The capacity of the memory is a tenth of the population, and all the memory points are reevaluated every generation.

The dynamic memory model (DMM) proposed in [10] is randomly initialized and is updated every generation. When the memory is updated, the memory individual that is closest to the best individual of the population is found firstly. Instead of being replaced, the closest memory individual is moved in the direction of the best individual. Before a memory individual is moved by a best individual, it is mutated by adding a random Gaussian number to explore the environment when the memory is updated. At every generation, the worst population individual is replaced by best memory individual. As in MI, the memory individuals are reevaluated very generation.

The variable-size memory (VM) scheme proposed in [11], [15] uses a variable-size memory. It is reported to outperform several typical memory strategies. The memory size of the VM strategy changes between the predefined limits MEM_MIN and MEM_MAX. The memory of VM is randomly initialized and updated in a stochastic time pattern as VI. When it is time to update, if there is room, the current best population individual is added to the memory. Otherwise the memory is cleaned to eliminate redundant individuals. If the cleaning fails to make room for the best population individual, the best population individual replaces the youngest memory individual. The age of a memory individual starts from zero, and increased by one at every generation. If the age reaches LIMIT_AGE, it would be reset to zero. After an environmental change is detected, the best individual of the memory is introduced into the population, and an extra increment is added to the age of this memory individual. As the above memory schemes, the memory individuals of VM are evaluated every generation.

```
Initialize population of an EA
Initialize memory
t ← 0
if MI || VM then
  T_M ← rand(5, 10)
end if
while stop criteria is not satisfied do
  A generation of the EA
  Evaluate memory
  if VM then
    if t = t_M then
      Update memory for VM
      T_M ← T_M + rand(5, 10)
    end if
    if change detected then
      Replace worst individual by best memory point
    end if
  end if
  if VI then
    if t = t_M || changedetected then
      Update memory for MI
    end if
    New individuals are generate by mutating the best memory individual
    Evaluate the new individuals
    Replace the worst population individuals with the new individuals
  end if
  if DMM then
    Update memory for DMM
    Replace worst individual with best memory point
  end if
end while
```

Procedure 1: The structure of an EA with the memory schemes

B. SPSO

Many MPAs are based on Particle Swarm Optimizer (PSO) [6]–[8]. PSO is a stochastic optimization technique introduced by Eberhart and Kennedy for the first time in 1995 [16]. PSO optimizes problems by steering a swarm of particles across the search space, and each particle moves and learns from the best solution that it finds and the best solution that its neighbor particles find. Since PSO was proposed, many versions have been developed and applied to many academic and real world problems successfully. Among them, a convergence guaranteed one is proposed by Clerc in [17]. The mathematic description of the movement of each particle is thus

$$\vec{v}_{(i,t+1)} = \chi(\vec{v}_{(i,t)} + \phi_1(\vec{p}_{(i,t)} - \vec{x}_{(i,t)}) + \phi_2(\vec{p}_{(g,t)} - \vec{x}_{(i,t)})) \quad (1)$$

$$\vec{x}_{(i,t+1)} = \vec{x}_{(i,t)} + \vec{v}_{(i,t+1)} \quad (2)$$

where:

$$\phi_1 = c_1 r_1, \phi_2 = c_2 r_2$$
$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|}$$

In the above equations, $\vec{x}_{(i,t)}$ denotes the location of the ith particle at step t, updated by Equation (2); $\vec{v}_{(i,t)}$ represents the velocity of the ith particle and is updated by Equation (1). $\vec{p}_{(i,t)}$ is the best solution that the ith particle has found, and $\vec{p}_{(g,t)}$ is the best solution that its neighbor particles have found by step t. χ is a constriction factor to allow the swarm to converge. r_1 and r_2 are uniform random numbers between 0 and 1. $c = c_1 + c_2$, where c_1 and c_2 are constant numbers, usually set to 2.05.

```
seeds ← ∅
sort all particles in swarm in decreasing fitness order
while Not reaching the end of swarm do
  get the best unprocessed particle p
  found ← false
  for each seed ∈ seeds do
    if the distance from seed to p is less than r_s then
      found ← true
      break
    end if
  end for
  if not found then
    seeds ← seeds ∪ {p}
  end if
end while
```

Procedure 2: Determine species seeds for $swarm$

The speciation-based Particle Swarm Optimizer (SPSO) is a neat and elegant MPA for dynamic optimization. It is introduced in [7], [18]. SPSO divides the population into multiple species (subpopulations). The division starts from the best particle, all the particles within its radius are assigned to the same species. Then for the remained particles, recursively execute the same division until every particle is assigned to a species. The best particle of each species is called seed and plays the role of $\vec{p}_{(g,t)}$ for all the particles in the same species. The mechanism to determine seeds for the swarm is given in Procedure 2. In [7], a parameter $Pmax$ is introduced to confine the species size. Only the fittest $Pmax$ particles will be preserved in a species, and the lower fitness particles in excess will be reinitialized to a random position and not allocated to any species until the next generation. In this manner, the population is prevented from concentrating on too few peaks and driven to explore new regions. This $Pmax$ version is employed in this paper to test the memory schemes.

III. The proposed memory Scheme

A. Considerations

MPAs are always used in multimodal environments. Local optima are important futures of dynamic environments and could evolve to global optima [8]. Therefore, it is useful to store local optima. However, the memory capacities of the existing memory schemes are relatively small. Storing multiple solutions for the current environment would result in losing information of the previous environments. Therefore, it is necessary to enlarge memory capacity. But there is a problem that the updating strategies of the existing memory schemes are dependent on fitness. Since the environment changes constantly, in order to maintain the correctness of the fitness of memory individuals, the existing memory schemes reevaluate memory individuals frequently, and thus heavy computational overhead is brought in. In consequence, in order to enlarge memory capacity and avoid heavy computational overhead meanwhile, a memory updating strategy that is independent of the fitness of memory individuals is required.

As having stated previous, existing memory schemes constantly replace the least fit population individuals with memory individuals, which could hinder the scattering of the subpopulations towards different promising regions, and sometimes the poorly-fitted subpopulations could even be removed. This phenomenon disrupts the work mechanism of MPAs, and could harm the performance. Therefore, when updating the population with memory, not only should the fitness of the population individuals be considered, their space distribution should be taken into account but also.

B. Details of the Proposed Large Memory Scheme

According to the above considerations, a new memory scheme with a large memory capacity is introduced to enhance the performance of MPAs for dynamic optimization problems. The structure of a MPA with the large memory scheme is presented in Procedure 3.

After an environmental change is detected, the memory updating procedure, described in Procedure 4, is executed. At the beginning, mark every subpopulation that is regarded as having located an optimum. For complex dynamic problems, it is possible that very few subpopulations would locate an optimum. In this case, other subpopulations are marked until the number of marked subpopulations reaches 5 or all the subpopulations are marked. For each marked subpopulation, if the memory is not full, copy its best individual p into the memory, otherwise, calculate the distance between p and its closest memory individual cm. If the distance is less than $UpdateDistance$, cm is replaced. Otherwise, replace the eldest memory individual with probability $2/3$. Every memory individual has an age to record how many generations it stays in the memory without being used. Note that some schemes also use distance and age to update memory [11], [14], but they are dependent on fitness.

The memory updating strategy is independent of fitness. The reason underlying this strategy is as follows. p is from the last environment, while cm is from a previous environment. Therefore, making cm compete with p in terms of the fitness in the last environment, just as many existing memory schemes do, is not fair. If cm is far away from p, replacing cm may

```
Initialize population of a MPA
Initialize memory
while stop criteria is not satisfied do
  A generation of the MPA
  if change is detected then
    Update memory
    Evaluate memory
    Update the population
  end if
end while
```

Procedure 3: The structure of a MPA with the large memory scheme

```
Mark every subpopulation that has located a local optimum.
while the number of the marked subpopulations are less than
5 && there is any unmarked subpopulation do
  Mark the subpopulation that has the fittest individual
  among the unmarked subopulations
end while
for each marked subpopulation do
  Let p denote the best individual of the subpopulation
  if memory is not full then
    Copy p into memory
  else
    Find the memory individual closet to p, and name it
    cm
    if the distance from cm to p is less than
    UpdateDistance then
      Replace cm with p
    else
      Find the eldest memory individual, and substitute p
      for it with a probability of 2/3
    end if
  end if
end for
The ages of all new memory individuals are set to zero
Increase the ages of all the memory individuals by 1
```

Procedure 4: The procedure of updating memory for the large memory scheme

$m \leftarrow n \ mod \ ns$, $l \leftarrow \lfloor n/ns \rfloor$, where ns is the number of subpopulations, and n equals to a tenth of the population size
For the worst m subpopulations, remove the $l+1$ least fit individuals. For the other subpopulations, remove the l least fit individuals
if Using LM1 **then**
Add the best n memory individuals into the population
end if
if Using LM2 **then**
Determine the specie seeds for memory individual by Procedure 2
Add the best n memory seeds into the population
end if
Set the age of all the selected memory individuals to zero

Procedure 5: Two versions of updating population for the large memory scheme

result in loss of information of another environment, and it is better to replace a rarely used memory individual with a certain probability. If cm is close to p, replace cm with p, and so the information loss is possibly slight and the information of the last environment stored

In Procedure 5, two versions of population updating are proposed, which are denoted as LM1 and LM2, respectively. According to the considerations above, we take both fitness and space distribution into account when insert memory individuals. The individuals to be replaced by memory individuals are evenly selected from all subpopulations. The "worst" subpopulation means its best individual has the worst fitness. As for which memory individuals are selected to update the population, LM1 and LM2 act differently. For LM1, the fittest n memory individuals are selected. For LM2 the fittest n memory seeds are selected. In the later strategy, both fitness and space distribution of memory individuals are considered.

Detecting environmental changes is important for an algorithm to effectively optimize dynamic problems, and in some cases, it is a difficult task. However, it is not a key concern in this study. In this study, we reevaluate the 5 random points every generation just as in [19], if any of them changes, an environmental change is detected.

IV. Dynamic Testing Problems Used in this Paper

The moving peaks benchmark (MPB) problem has been widely applied to simulate dynamic environment in the literature [8]. In this paper this convention is followed. However, since the performance of memories is heavily dependent on whether the dynamic problems in process are cyclic or not, experiments are also carried out on the cyclic MPB (CMPB) problem which was introduced in [20].

A. The Moving Peaks Benchmark Problem

The MPB problem is composed of a number of component problems with peak-like fitness landscape. It has the form as below [4]:

$$F(\vec{x},t) = \max_{i=1,\ldots,p} \frac{H_i(t)}{1 + W_i(t) \sum_{j=1}^{D} \left(x_j(t) - X_{ij}(t)\right)^2}$$

The peaks change independently with three features, i.e. the peak width W, the peak height H and the peak location $\vec{X}$. Formally the change of peak i can be described as

$$\sigma \in N(0,1)$$
$$W_i(t) = W_i(t-1) + width\_severity \cdot \sigma$$
$$H_i(t) = H_i(t-1) + height\_severity \cdot \sigma$$
$$\vec{X}_i(t) = \vec{X}_i(t-1) + \vec{v}$$

where the direction of $\vec{v}$ is random, and its length is called shift severity.

B. The Cyclic Moving Peaks Benchmark Problem

The CMPB problem is constructed by modifying the MPB problem with a cyclical benchmark problem generator [20]. It is easy to tune independently the change severity, the number of peaks and the cycle length of the CMPB problem.

1) A Cyclical Benchmark Problem Generator in Real Space [20]: Given the changing severity s, the cycle length l and the number of dimensions d, the following procedure generates a $d \times d$ rotation matrix $\mathbf{R}$ and a d-dimensional vector set $V = \{\vec{v}_i : i \geq 0\}$. $\mathbf{R}$ and $\mathbf{V}$ satisfy the two following properties.

Property 1: $\mathbf{R}\vec{v}_i = \vec{v}_{(i+1) \pmod l}, (\vec{v}_i \in \mathbf{V})$

Property 2: $\|\mathbf{R}\vec{v}_i - \vec{v}_i\| = s, (\vec{v}_i \in \mathbf{V})$

The details of the generating procedure are described as below. Note that the first two steps are very similar to the procedure of generating a rotation matrix described in [21].

Step 1: Randomly choose k (k is an even number, i.e. d or $d-1$) dimensions $d_1, d_2, \ldots, d_k$ out of the d dimensions to compose $k/2$ planes $p_1, p_2, \ldots, p_{k/2}$, with p_i composed of d_{2i-1} and d_{2i}, $i \leq i \leq k/2$.

Step 2: Construct the rotation matrix $\mathbf{R}$.

A rotation matrix $R_{i,j}(\theta)$ described in [22] rotates the projection of a vector $\vec{v}$ in the plane i-j by an angle θ from the ith axis to the jth axis. $\mathbf{R}$ is obtained by multiplying these rotations matrixes.

$$\mathbf{R} = R_{d_1,d_2}(\frac{2\pi}{l}) \times R_{d_3,d_4}(\frac{2\pi}{l}) \times \cdots \times R_{d_{k-1},d_k}(\frac{2\pi}{l})$$

Step 3: Construct $\vec{v}_0$.

Randomly generate $k/2$ positive real numbers $\lambda_1, \ldots, \lambda_{k/2}$ satisfying $\sum_{i=1}^{k/2} {\lambda_i}^2 = s^2$

1) For each λ_i, calculate a real number γ_i

$$\gamma_i = \frac{\lambda_i}{2\sin(\pi/l)}$$

2) For each γ_i, randomly generate two real numbers μ_{2i-1}, μ_{2i} satisfying

$${\mu_{2i-1}}^2 + {\mu_{2i}}^2 = {\gamma_i}^2$$

3) The ith dimension of $\vec{v}_0$ is set to μ_i, if i is one of the k randomly chosen dimensions. Otherwise, set it to a random number.

TABLE I. DEFAULT SETTINGS FOR THE MPB AND CMPB PROBLEMS

Parameter	Value
Change of frequency	5000
Number of change	100
Height severity	7.0
Width severity	1.0
Peak shape	Cone
Number of dimensions D	5
S	[0,100]
H	[30.0,70.0]
W	[1,12]
I	50

Step 4: Construct $\mathbf{V} = \{\vec{v}_i : 0 \leq i\}$.

$$\vec{v}_i = \mathbf{R}^i \vec{v}_0, i > 0$$

With the above steps and given a function in real space, it is easy to construct a cyclic dynamic version. Suppose $F(\vec{x})$ is a function in real space, and the optimal solution is $\vec{O}$. Construct a vector set $\mathbf{V}$ for $F(\vec{x})$ with the above procedure, $F(\vec{x})$ can be transformed into a cyclically changing problem $F'(\vec{x})$ easily as below:

$$F'(\vec{x}, t) = F(\vec{x} - \vec{v}_t + \vec{O})$$

2) The Cyclic MPB (CMPB) Problem: The cyclic MPB problem can be constructed easily. For each peak position $\vec{X}_i$, given the cycle length and shift severity, a rotation matrix $\mathbf{R}$ and the corresponding vector set $\mathbf{V} = \{\vec{v}_j^i : 0 \leq j\}$ is generated using the procedure proposed above. The new benchmark problem changes the peak height and width in the same manners as MBP, but moves the peak location in the following manner [20].

$$\vec{X}_i(0) = \vec{v}_0^i$$

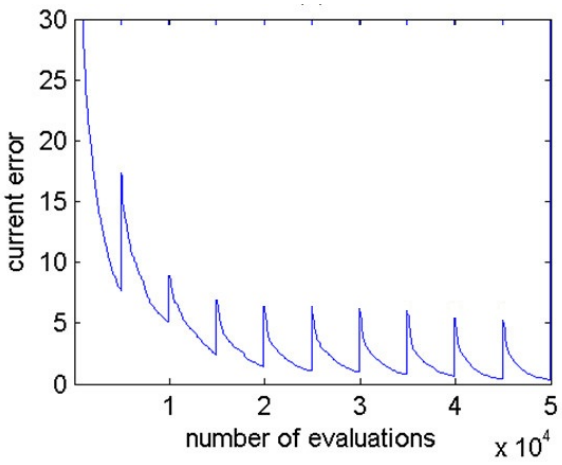
$$\vec{X}_i(t) = \mathbf{R}\vec{X}_i(t-1) = \vec{v}_t^i, \ t > 0$$

V. EXPERIMENTS

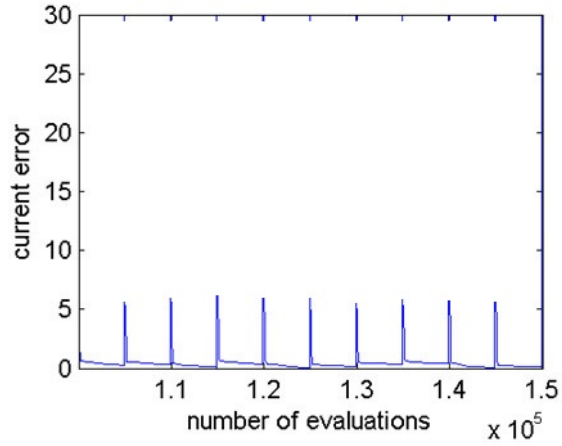
A. Experimental Settings

Experiments are carried out both on MPB and CMPB problems, and some parameters are fixed for all experiments. These parameters are listed in TABLE I as in [8]. S confines the range of allele values, and I denotes the initial peaks height for all peaks. The height and width of the peaks shift randomly in the range of H and W respectively. Besides these parameters, the shift severity, the number of peaks and the cycle length are varied to generate test problems with different degrees of complexity. For the sake of convenience, in this paper, a CMPB problem is denoted as CMPB(*s,l,n*) to mean that the shift severity, the cycle length and the number of peaks are s, l and n respectively.

In the remainder part of the paper, SPSO with VM, MI and DMM are denoted as VMSPSO, MISPSO, DMMSPSO, respectively. According to the version of population updating, the combinations of the large memory scheme and SPSO are denoted as LM1SPSO and LM2SPSO, respectively. Some parameters of the algorithms are fixed in the experiments. Because the settings of MBP problems used here is the same as in [7], the same settings of SPSO are used in this paper. For LM1SPSO and LM2SPSO, the population size is 100, $UpdateDistance = 0.8$. Inspired by [23], if the best individual of a subpopulation has not been improved for 5 generations, this subpopulation is regarded as having found a local optimum. For DMMSPSO and MISPSO, the population size is 100, including 10 memory individuals, just the same as in [9].



(a) Current error for the first ten environments



(b) Current error for the third ten environments

Fig. 1. The current error of LM1SPSO on CMPB(1,10,10)

The widely used Mühlenbein mutation [24] is employed to generate immigrants for MISPSO, whose mutation rule is shown below:

$$x_i^{'} = x_i \pm range_i \cdot \gamma$$

The sign are selected randomly with probability of 0.5. $range_i$ confines the mutation range and normally set to a tenth of the whole search range. γ is defined as:

$$\gamma = \sum_{k=0}^{15} a_k 2^{-k}.$$

a_k is set to 0 or 1 randomly with $p(a_k = 1) = \frac{1}{16}$. At every generation, 5 memory immigrants are generated with Mühlenbein mutation.

To make VM strategy effective for working with SPSO, we tuned its parameter settings by means of trial and error. MEM_MIN is set to 8 and MEM_MAX is set to 15. We fix the population sizes of SPSO to 90 in the experiments. As

TABLE II. OFFLINE ERROR OF ALGORITHMS ON THE CMPB PROBLEMS

s ,l, n	1, 5, 5	1, 5, 10	1, 5, 20	1, 5, 40	1, 20, 5	1, 20, 10	1, 20, 20	1, 20, 40
LM1SPSO	0.08 ± 0.03	$\mathbf{0.14 \pm 0.06}$	$\mathbf{0.77 \pm 0.19}$	$\mathbf{1.15 \pm 0.16}$	$\mathbf{0.14 \pm 0.19}$	$\mathbf{0.24 \pm 0.14}$	$\mathbf{0.99 \pm 0.06}$	$\mathbf{1.56 \pm 0.07}$
LM2SPSO	$\mathbf{0.07 \pm 0.02}$	0.17 ± 0.1	0.81 ± 0.18	1.37 ± 0.19	$\mathbf{0.14 \pm 0.13}$	0.31 ± 0.09	1.07 ± 0.14	1.63 ± 0.17
SPSO	0.65 ± 0.3	0.91 ± 0.36	1.85 ± 0.33	2.64 ± 0.33	0.73 ± 0.37	0.9 ± 0.26	2.11 ± 0.33	2.88 ± 0.46
VMSPSO	0.20 ± 0.18	0.47 ± 0.27	1.77 ± 0.33	2.25 ± 0.37	0.26 ± 0.29	0.61 ± 0.24	1.75 ± 0.28	2.41 ± 0.36
DMMSPSO	0.93 ± 0.35	0.97 ± 0.43	1.94 ± 0.28	2.41 ± 0.30	1.09 ± 0.47	1.02 ± 0.37	2.04 ± 0.31	2.58 ± 0.24
MISPSO	3.56 ± 0.47	3.88 ± 0.77	5.58 ± 1.11	6 ± 0.51	9.29 ± 1.08	8.62 ± 0.83	9.89 ± 0.7	10.51 ± 0.78
s, l, n	2, 5, 5	2, 5, 10	2, 5, 20	2, 5, 40	2, 20, 5	2, 20, 10	2, 20, 20	2, 20, 40
LM1SPSO	$\mathbf{0.08 \pm 0.01}$	$\mathbf{0.19 \pm 0.07}$	$\mathbf{0.79 \pm 0.15}$	$\mathbf{1.17 \pm 0.13}$	$\mathbf{0.20 \pm 0.14}$	0.42 ± 0.12	$\mathbf{1.13 \pm 0.14}$	$\mathbf{1.71 \pm 0.17}$
LM2SPSO	0.09 ± 0.03	0.21 ± 0.12	0.86 ± 0.14	1.32 ± 0.14	0.24 ± 0.07	$\mathbf{0.36 \pm 0.19}$	1.22 ± 0.11	1.79 ± 0.18
SPSO	0.94 ± 0.29	1.22 ± 0.29	2.46 ± 0.33	3.23 ± 0.51	0.99 ± 0.29	1.47 ± 0.34	2.59 ± 0.36	3.39 ± 0.34
VMSPSO	0.41 ± 0.13	0.96 ± 0.31	2.12 ± 0.28	2.75 ± 0.33	0.64 ± 0.32	1.01 ± 0.29	2.31 ± 0.26	2.8 ± 0.25
DMMSPSO	1.33 ± 0.45	1.17 ± 0.30	2.23 ± 0.34	2.66 ± 0.30	1.39 ± 0.61	1.48 ± 0.38	2.27 ± 0.30	2.83 ± 0.28
MISPSO	6.11 ± 0.92	5.58 ± 0.55	7 ± 0.6	7.84 ± 0.78	14.88 ± 1.28	13.02 ± 1.07	13.57 ± 1.02	13.34 ± 1.1
s, l, n	5.5, 5	5, 5, 10	5, 5, 20	5, 5, 40	5, 20, 5	5, 20, 10	5, 20, 20	5, 20, 40
LM1SPSO	$\mathbf{0.10 \pm 0.03}$	$\mathbf{0.26 \pm 0.08}$	$\mathbf{0.65 \pm 0.09}$	$\mathbf{1.12 \pm 0.10}$	0.46 ± 0.09	0.84 ± 0.08	1.51 ± 0.29	$\mathbf{2.05 \pm 0.15}$
LM2SPSO	0.14 ± 0.07	$\mathbf{0.26 \pm 0.09}$	0.77 ± 0.10	1.16 ± 0.11	$\mathbf{0.41 \pm 0.08}$	$\mathbf{0.77 \pm 0.09}$	$\mathbf{1.49 \pm 0.15}$	2.09 ± 0.12
SPSO	2.08 ± 0.61	2.49 ± 0.36	3.75 ± 0.39	4.59 ± 0.41	2.25 ± 0.39	3.02 ± 0.48	4.15 ± 0.38	4.51 ± 0.37
VMSPSO	1.09 ± 0.53	1.59 ± 0.37	2.66 ± 0.37	3.17 ± 0.32	1.29 ± 0.55	2.37 ± 0.52	2.98 ± 0.33	3.24 ± 0.34
DMMSPSO	1.90 ± 0.37	2.15 ± 0.50	2.63 ± 0.33	3.04 ± 0.34	2.31 ± 0.55	2.59 ± 0.55	3.17 ± 0.41	3.19 ± 0.32
MISPSO	10.27 ± 1.34	9.24 ± 0.98	10.64 ± 1.33	10.82 ± 0.83	18.6 ± 1.75	19.28 ± 1.18	18.16 ± 1.02	16.73 ± 1.02
s, l, n	10, 5, 5	10, 5, 10	10, 5, 20	10, 5, 40	10, 20, 5	10, 20, 10	10, 20, 20	10, 20, 40
LM1SPSO	$\mathbf{0.15 \pm 0.04}$	$\mathbf{0.33 \pm 0.06}$	0.74 ± 0.10	$\mathbf{1.17 \pm 0.10}$	0.86 ± 0.09	$\mathbf{1.51 \pm 0.08}$	2.35 ± 0.28	2.78 ± 0.24
LM2SPSO	$\mathbf{0.15 \pm 0.04}$	$\mathbf{0.33 \pm 0.09}$	$\mathbf{0.73 \pm 0.09}$	$\mathbf{1.17 \pm 0.09}$	$\mathbf{0.82 \pm 0.11}$	1.52 ± 0.10	$\mathbf{2.31 \pm 0.24}$	$\mathbf{2.77 \pm 0.19}$
SPSO	3.79 ± 0.59	4.71 ± 0.5	5.25 ± 0.43	5.34 ± 0.41	4.62 ± 0.63	5.54 ± 0.52	5.79 ± 0.4	5.37 ± 0.41
VMSPSO	2.05 ± 0.64	2.80 ± 0.54	3.37 ± 0.50	3.48 ± 0.33	3.11 ± 0.72	3.91 ± 0.72	4.21 ± 0.49	3.85 ± 0.40
DMMSPSO	3.24 ± 0.74	3.57 ± 0.75	3.63 ± 0.43	3.73 ± 0.43	4.15 ± 0.88	4.54 ± 0.59	4.54 ± 0.50	3.93 ± 0.36
MISPSO	14.89 ± 1.84	12.7 ± 1.2	12.89 ± 1.22	13.05 ± 1.15	23.48 ± 1.9	21.94 ± 1.48	20.13 ± 1.28	17.94 ± 0.84

regards the updating strategy, we adopt the gen strategy, which is reportedly best in [15].

The offline error is used in this paper to evaluate the performance of the algorithms, which is defined as follows [8]

$$e_{offline} = \frac{1}{T}\sum_{i=1}^{T}(optimum_i - fbest_i)$$

here T is the total number of environments, $optimum_i$ is the theoretic optimal value of the ith environment, and $fbest_i$ is the fitness of the best solution found in the ith environment.

B. The Online Performance of the Large Memory Scheme

As an example, Fig. 1 depicts the average current error of LM1SPSO over 30 runs. The test problem is CMPB(1,10,10). The current error refers to the difference between the theoretic global optimum and the fittest particle found since the last environmental change. The effect of the LM1 scheme is obvious by comparing the results for the first ten and the third ten environments. Since the cycle length is 10, after the first ten environments, all the environments are recorded in the memory. In Fig 1(a), the current error decreases gradually, while in Fig 1(b), the current error drops to a valve near 0 quickly. This indicates that the effect of the large memory scheme is very good.

C. Comparison on Cyclical Problems

In this group of experiments, we compare the performance of the combinations on the CMPB problems. For these problems, the shifting severity is chosen from {1, 3, 5, 10}, the cycle length from {5, 20}, and the number of peaks from {5, 10, 20, 40}. The results are proposed in TABLE II. Every single result is averaged over 30 independent runs, and the standard deviations are attached.

The results indicate that LM1SPSO and LM2SPSO outperform all the other four algorithms in all the test cases; VMSPSO outperforms SPSO in all the test cases; DMMSPSO outperforms SPSO in some test cases and is defeated by SPSO in the other cases; MISPSO is worst in all the test cases. This indicates that in this set of experiments, the large memory scheme dramatically enhances the performance of SPSO, but not all the memory schemes can enhance SPSO, even in cyclic environments.

The reason could be as follows. SPSO adaptively distributes the subpopulations over different ranges in the search space to locate and track optima. With memory schemes, although memory information is added to population, the population distribution and search is disturbed, which could bring negative effect. If the disturbance is too severe, the benefit of memory scheme would be offset. VM replaces one population individual every detected change. The disturbance is slight and the negative effect is slight consequently. Therefore VM could enhance SPSO in all the cases. DMM replaces one population individual every generation. The disturbance is medium, and therefore, the benefit of DMM can sometimes overcome its negative effect. MI replaces more than one population individuals with memory immigrants every generation. In consequence, the formation and movement of the subpopulation is disturbed severely, and thus the performance of SPSO gets much worse. LM1 and LM2 replace a tenth of population individuals with memory individuals once an environmental change is detected. Because the replaced individuals are evenly chosen from all the subpopulations, the influence on the distribution of subpopulations is relieved.

The shift severity greatly affects the offline error of SPSO, because there is a long distance for the particles to fly to the new optima. However, the shift severity affects LM1SPSO and LM2SPSO much more slightly. This is because after the environment changes, highly fit memory individuals of the new environment is added to the population at once. By contrast, the cycle length hardly affects the performance of SPSO, but it affects the performance of LM1SPSO and LM2SPSO. This is because long cycle length means there is long time for the

TABLE III. OFFLINE ERROR OF ALGORITHMS ON THE STANDARD MPB PROBLEMS WITH DIFFERENT NUMBER OF PEAKS

severity	LM1SPSO	LM2SPSO	SPSO	MISPSO	VMSPSO	DMMSPSO
1	0.70 ± 0.15	$\mathbf{0.66 \pm 0.14}$	1.07 ± 0.32	11.81 ± 1.3	2.22 ± 0.43	2.83 ± 0.83
2	1.23 ± 0.13	$\mathbf{1.19 \pm 0.15}$	1.45 ± 0.3	18.3 ± 1.88	3.22 ± 0.54	4.10 ± 0.76
3	1.62 ± 0.18	$\mathbf{1.60 \pm 0.16}$	1.9 ± 0.34	22.46 ± 2.27	4.69 ± 0.68	5.16 ± 0.72
4	2.05 ± 0.17	$\mathbf{1.99 \pm 0.13}$	2.38 ± 0.45	25.24 ± 2.11	5.02 ± 0.75	5.85 ± 0.81
5	$\mathbf{2.45 \pm 0.19}$	2.47 ± 0.18	2.65 ± 0.38	27.45 ± 2.28	6.17 ± 0.81	6.48 ± 0.89
6	2.90 ± 0.18	$\mathbf{2.86 \pm 0.18}$	3.09 ± 0.41	28.39 ± 2.16	6.90 ± 0.96	7.36 ± 1.06

TABLE IV. OFFLINE ERROR OF ALGORITHMS ON THE STANDARD MPB PROBLEMS WITH DIFFERENT SHIFT SEVERITY

peaks	LM1SPSO	LM2SPSO	SPSO	MISPSO	VMSPSO	DMMSPSO
1	0.54 ± 0.05	0.57 ± 0.06	$\mathbf{0.5 \pm 0.09}$	17.4 ± 4.55	2.59 ± 0.73	7.05 ± 2.34
2	0.58 ± 0.05	$\mathbf{0.56 \pm 0.10}$	0.71 ± 1.05	14.94 ± 2.74	2.11 ± 0.50	4.80 ± 1.02
5	$\mathbf{0.58 \pm 0.11}$	0.60 ± 0.17	0.67 ± 0.44	13.29 ± 1.34	1.88 ± 0.37	3.10 ± 0.43
7	$\mathbf{0.61 \pm 0.13}$	0.63 ± 0.17	0.7 ± 0.34	12.19 ± 1.2	1.85 ± 0.31	3.00 ± 0.52
10	0.70 ± 0.14	$\mathbf{0.66 \pm 0.14}$	1.07 ± 0.32	11.81 ± 1.3	2.22 ± 0.43	2.83 ± 0.83
20	1.78 ± 0.14	$\mathbf{1.73 \pm 0.12}$	2.39 ± 0.35	12.98 ± 1.26	3.33 ± 0.60	3.56 ± 0.68
30	$\mathbf{2.23 \pm 0.13}$	2.25 ± 0.17	2.81 ± 0.35	13.45 ± 1.08	3.54 ± 0.59	3.49 ± 0.60
40	2.49 ± 0.15	$\mathbf{2.44 \pm 0.17}$	3.03 ± 0.38	13.52 ± 0.98	3.74 ± 0.57	3.52 ± 0.57
50	2.61 ± 0.12	$\mathbf{2.55 \pm 0.15}$	3.26 ± 0.44	13.86 ± 1.29	3.72 ± 0.45	3.53 ± 0.57
100	2.86 ± 0.17	$\mathbf{2.81 \pm 0.12}$	3.59 ± 0.41	13.46 ± 0.94	4.07 ± 0.57	3.31 ± 0.35

memory to collect useful information of all the environments, and before the useful information is obtained, the effect of memory is week. It is difficult to tell the influence of the number of peaks, because it affects SPSO and LMSPSO currently.

D. Comparison on Acyclic Problems

In this group of experiments, we compare the performance of the combinations on the standard MPB problems with different settings.

1) Effect of the shift severity: In this group of experiments, the performances of the combinations are compared on the standard MPB problems with different shift severities. The number of peaks is set to 10. The offline error and related standard deviation averaged over 30 independent runs are presented in TABLE III.

It can be seen from TABLE III that LM1SPSO and LM2SPSO outperform all the others, while MISPSO achieves the worst in all the test cases. In all the test cases, SPSO is better than VMSPSO and DMMSPSO, and VMSPSO is better than DMMSPSO. Since memory is less efficient in acyclic environments than in cyclic environments, for MI, DMM and VM, the memory benefit is overcome by the negative effect. These results also show that, on standard MPB problems, the enhancing effect of the large memory scheme is competitive with the changing of shift severity.

2) Effect of number of peaks: In this group of experiments, the performances of the combinations are compared on the standard MPB problems with different numbers of peaks. The shift severity is set to 1.0. The offline error and the related standard deviation averaged over 30 independent runs are presented in TABLE IV.

From TABLE IV we can observe that LM1SPSO and LM2SPSO are better than the others except when the number of peaks is 1, while MISPSO, MISPSO and VMSPSO are beaten by SPSO in all the cases. In some cases, VMSPSO is worse than DMMSPSO, and in the other cases DMMSPSO is worse than VMSPSO. The offline errors of SPSO and LM1SPSO and LM2SPSO increase with the number of peaks, while the offline errors of the other algorithms are irregular. The results show that the large memory scheme can enhance the performance of SPSO for optimizing dynamic multimodal problems.

E. Comparison between LM1 and LM2

LM1 selects the fittest memory individuals into the population, while LM2 takes into account both fitness and space distribution of the memory individuals. According to the results above, the performances LM1SPSO and LM2SPSO are close in all the test cases. On CMPB, LM1SPSO is better when $l = 5$ and s is small. While $l = 20$ and s is great, LM2SPSO is better. On MPB, LM2SPSO is better in most cases.

The reason could be as follows. In relatively simple environments, it is easy for the memory to store all the positions that the global optimum could appear. Therefore, when the environment changes, the high-fitted memory individuals selected by LM1 could help to locate the new global optimum quickly. In relatively complex environments, it is less likely to find the new global optimum directly from the memory. In these cases, the memory seeds can help to find high-fitted solution from different regions of the search space, which is helpful for SPSO to locate the global optimum.

VI. CONCLUSION

By addressing the possible problems of applying existing memory to MPAs, the large memory scheme is proposed in this paper specially for MPAs. The scheme updates the memory with a strategy independent of the fitness. In this way, it does not need to reevaluate the memory individuals frequently as the exiting memory schemes do, and thus is able to enlarge the memory capacity without consuming too much computational resource. In order not to disrupt the work mechanism of MPAs, when updating population with memory, the proposed scheme takes into account the fitness and space distribution of individuals at the same time.

Experimental study is carried out to investigate the effect of combining MPAs with memory schemes. Three existing memory schemes and the proposed memory scheme are compared

by combining with SPSO and being tested on the cycle and acyclic problems. The experimental results show that proposed memory scheme is able to enhance the performance of SPSO effectively in almost all the test cases. The results also show that straightforwardly using existing memory scheme does not always help and could harm the performance of MPAs even in cyclic environments.

ACKNOWLEDGEMENTS

This work is partly supported by the 2006-2007 Excellent Young and Middle-aged Academic Leader Development Program of Anhui Province Research Experiment Bases.

REFERENCES

[1] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments - a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[2] C. Cruz, J. Gonzlez, and D. Pelta, "Optimization in dynamic environments: a survey on problems, methods and measures," *Soft Computing*, vol. 15, no. 7, pp. 1427–1448, 2011.

[3] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[4] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *the 1999 IEEE Congress on Evolutionary Computation (CEC'99)*, Washington, DC, USA, 1999, pp. 1875–1882.

[5] J. Branke, T. Kauler, C. Smidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Evolutionary Design and Manufacture*. Springer, 2000, pp. 299–307.

[6] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 459–472, 2006.

[7] D. Parrott and X. Li, "Locating and tracking multiple dynamic optima by a particle swarm model using speciation," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 440–458, 2006.

[8] S. Yang and C. Li, "A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 959–974, 2010.

[9] S. Yang, "Genetic algorithms with memory- and elitism-based immigrants in dynamic environments," *Evolutionary Computation*, vol. 16, no. 3, pp. 385–416, 2008.

[10] C. N. Bendtsen and T. Krink, "Dynamic memory model for non-stationary optimization," in *the 2002 IEEE Congress on Evolutionary Computation*, 2002, pp. 145 – 150.

[11] A. Simöes and E. Costa, "Variable-size memory evolutionary algorithm to deal with dynamic environments," in *Applications of Evolutionary Computing*. Springer Berlin Heidelberg, 2007, vol. 4448, pp. 617–626.

[12] Y. Cao and W. Luo, "Novel associative memory retrieving strategies for evolutionary algorithms in dynamic environments," in *Proceedings of the 4th International Symposium on Advances in Computation and Intelligence*, 2009, pp. 258–268.

[13] ——, "A novel updating strategy for associative memory scheme in cyclic dynamic environments," in *Proceedings of the Third International Workshop on Advanced Computational Intelligence (IWACI2010)*, Suzhou, China, 2010.

[14] T. Zhu, W. Luo, and Z. Li, "An adaptive strategy for updating the memory in evolutionary algorithms for dynamic optimization," in *Proceedings of the 2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments*, Paris, France, 2011, p. 2011.

[15] A. Simöes and E. Costa, "Improving memory's usage in evolutionary algorithms for changing environments," in *the 2007 IEEE Congress on Evolutionary Computation*, Singapore, 2007, pp. 276–283.

[16] R. Eberhart and J. Kennedy, "New optimizer using particle swarm theory," in *Proceedings of the 1995 6th International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39–43.

[17] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[18] X. Li, "Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization," in *Proceedings of Genetic and Evolutionary Computation Conference*, 2004, pp. 105–116.

[19] S. Bird and X. Li, "Using regression to improve local convergence," in *Evolutionary Computation, 2007. IEEE Congress on*, 2007, pp. 592–599.

[20] T. Zhu, W. Luo, and L. Yue, "Dynamic optimization facilitated by the memory tree," *Soft Computing*, 2014, DOI:10.1007/s00500-014-1273-1.

[21] C. Li, S. Yang, T. T. Nguyen *et al.*, "Benchmark generator for CEC 2009 competition on dynamic optimization," Department of Computer Science, University of Leicester, U.K., Tech. Rep., 2008.

[22] R. Salomon, "Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms," *Biosystems*, vol. 39, no. 3, pp. 263–278, 1996.

[23] S. Bird and L. Xiaodong, "Enhancing the robustness of a speciation-based pso," in *IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, 2006, pp. 843–850.

[24] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm i. continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25–49, 1993.