Associative Memory Scheme for Genetic Algorithms in Dynamic Environments

Shengxiang Yang

Department of Computer Science, University of Leicester,
University Road, Leicester LE1 7RH, United Kingdom
s.yang@mcs.le.ac.uk

Abstract. In recent years dynamic optimization problems have attracted a growing interest from the community of genetic algorithms with several approaches developed to address these problems, of which the memory scheme is a major one. In this paper an associative memory scheme is proposed for genetic algorithms to enhance their performance in dynamic environments. In this memory scheme, the environmental information is also stored and associated with current best individual of the population in the memory. When the environment changes the stored environmental information that is associated with the best re-evaluated memory solution is extracted to create new individuals into the population. Based on a series of systematically constructed dynamic test environments, experiments are carried out to validate the proposed associative memory scheme. The environmental results show the efficiency of the associative memory scheme for genetic algorithms in dynamic environments.

1 Introduction

Genetic algorithms (GAs) have been applied to solve many optimization problems with promising results. Traditionally, the research and application of GAs have been focused on stationary problems. However, many real world optimization problems are actually dynamic optimization problems (DOPs) [4]. For DOPs, the fitness function, design variables, and/or environmental conditions may change over time due to many reasons. Hence, the aim of an optimization algorithm is now no longer to locate a stationary optimal solution but to track the moving optima with time. This challenges traditional GAs seriously since they cannot adapt well to the changing environment once converged.

In recent years, there has been a growing interest in investigating GAs for DOPs. Several approaches have been developed into GAs to address DOPs, such as diversity maintaining and increasing schemes [5, 7, 11], memory schemes [2, 14, 17], and multi-population approaches [3]. Among the approaches developed for GAs in dynamic environments, memory schemes have proved to be beneficial for many DOPs. Memory schemes work by storing useful information, either implicitly [6, 9, 12] or explicitly, from the current environment and reusing it later in new environments. In [17, 19], a memory scheme was proposed into population-based incremental learning (PBIL) [1] algorithms for DOPs, where

F. Rothlauf et al. (Eds.): EvoWorkshops 2006, LNCS 3907, pp. 788–799, 2006.

the working probability vector is also stored and associated with the best sample it creates in the memory. When the environment changes, the stored probability vector can be reused in the new environment.

In this paper, the idea in [17] is extended and an *associative memory scheme* is proposed for GAs in dynamic environments. For this associative memory scheme, when the best solution of the population is stored into the memory, the current environmental information, the *allele distribution vector*, is also stored in the memory and associated with the best solution. When the environment changes, the stored environmental information associated with the best re-evaluated memory solution is used to create new individuals into the population. Based on the dynamic problem generator proposed in [16, 18], a series of DOPs with different environmental dynamics are constructed as the test bed and experiments are carried out to compare the performance of the proposed associative memory scheme with traditional direct memory scheme for GAs in dynamic environments. Based on the experimental results we analyze the strength and weakness of the associative memory over direct memory for GAs in dynamic environments.

2 Overview of Memory Schemes

The standard GA, denoted *SGA* in this paper, maintains and evolves a population of candidate solutions through selection and variation. New populations are generated by first probabilistically selecting relatively fitter individuals from the current population and then performing crossover and mutation on them to create new off-springs. This process continues until some stop condition becomes true, e.g., the maximum allowable number of generations t_{max} is reached.

Usually, with the iteration of SGA, individuals in the population will eventually converge to the optimal solution(s) in stationary environments due to the pressure of selection. Convergence at a proper pace, instead of pre-mature, may be beneficial and is expected for GAs to locate expected solutions for stationary optimization problems. However, convergence becomes a big problem for GAs in dynamic environments because it deprives the population of genetic diversity. Consequently, when change occurs, it is hard for GAs to escape from the optimal solution of the old environment. Hence, additional approaches, e.g., memory schemes, are required to adapt GAs to the new environment.

The basic principle of memory schemes is to, implicitly or explicitly, store useful information from the current environment and reuse it later in new environments. Implicit memory schemes for GAs in dynamic environments depend on redundant representations to store useful information for GAs to exploit during the run [6, 9, 12]. In contrast, explicit memory schemes make use of precise representations but split an extra storage space where useful information from current generation can be explicitly stored and reused later [2, 10, 15].

For explicit memory there are three technical considerations: what to store in the memory, how to update the memory, and how to retrieve the memory. For the first aspect, usually good solutions are stored in the memory and reused directly when change occurs. This is called *direct memory scheme*. It is also

interesting to store environmental information as well as good solutions in the memory and reuse the environmental information when change occurs [8, 13, 17]. This is called *associative memory scheme*, see Section 3 for more information. For the second consideration, since the memory space is limited, it is necessary to update memory solutions to make room for new ones. A general strategy is to select one memory point to be replaced by the best individual from the population. As to which memory point should be updated, there are several strategies [2]. For example, the most similar strategy replaces the memory point that is the closest to the best individual from the population. For the memory retrieval, a natural strategy is to use the best individual(s) in the memory to replace the worst individual(s) in the population. This can be done periodically or only when the environment change is detected.

The GA with the direct memory scheme studied in this paper is called *direct memory GA* (DMGA). DMGA (and other memory based GAs in this study) uses a randomly initialized memory of size $m = 0.1 * n$ (n is the total population size). When the memory is due to update, if any of the randomly initialized points still exists in the memory, the best individual of the population will replace one of them randomly; otherwise, it will replace the closest memory point if it is better (the most similar memory updating strategy). Instead of updating the memory in a fixed time interval, the memory in DMGA is updated in a stochastic time pattern as follows. Suppose the memory is updated at generation t, the next memory updating time t_M is given by: $t_M = t + rand(5, 10)$. This dynamic time pattern can smooth away the potential effect that the environmental change period coincides with the memory updating period (e.g., the memory is updated whenever the environment changes).

The memory in DMGA is re-evaluated every generation to detect environmental changes. The environment is detected as changed if at least one individual in the memory has been detected changed its fitness. If an environment change is detected, the memory is merged with the old population and the best $n - m$ individuals are selected as an interim population to undergo standard genetic operations for a new population while the memory remains unchanged.

3 Associative Memory for Genetic Algorithms

As mentioned before, direct memory schemes only store good solutions in the memory and directly reuse the solutions (e.g., combining them with the current population) when change occurs. In fact, in addition to good solutions we can also store current environmental information in the memory. For example, Ramsey and Greffenstette [13] studied a GA for robot control problem, where good candidate solutions are stored in a permanent memory together with information about the current environment the robot is in. When the robot incurs a new environment that is similar to a stored environment instance, the associated stored controller solution is re-activated. This scheme was reported to significantly improve the robot's performance in dynamic environments. In [17, 19], a memory scheme was proposed into PBIL algorithms for DOPs, where the working probability vector is also stored and associated with the best sample it creates in

the memory. When the environment is detected changed, the stored probability vector associated with the best re-evaluated memory sample is extracted to compete with the current working probability vector to become the future working probability vector for creating new samples.

The idea in [17, 19] can be extended to GAs for DOPs. That is, we can store environmental information together with good solutions in the memory for later reuse. Here, the key thing is how to represent current environment. As mentioned before, given a problem in certain environment the individuals in the population of a GA will eventually converge toward the optimum of the environment when the GA progress its searching. The convergence information, i.e., *allele distribution* in the population, can be taken as the natural representation of current environment. Each time when the best individual of the population is stored in the memory, the statistics information on the allele distribution for each locus, the *allele distribution vector*, can also be stored in the memory and associated with the best individual.

The pseudo-code for the GA with the associative memory, called *associative memory GA* (AMGA), is shown in Fig. 1. Within AMGA, a memory of size $m = 0.1 * n$ is used to store solutions and environmental information. Now each memory point consists of a pair $< S, \boldsymbol{D} >$, where S is the stored solution and $\boldsymbol{D}$ is the associated allele distribution vector. For binary encoding (as per this paper), the frequency of ones over the population in a gene locus can be taken as the allele distribution for that locus.

As in DMGA, the memory in AMGA is re-evaluated every generation. If an environmental change is detected, the allele distribution vector of the best memory point $< S_M(t), \boldsymbol{D}_M(t) >$, i.e., the memory point with its solution $S_M(t)$ having the highest re-evaluated fitness, is extracted. And a set of $\alpha * (n - m)$ new individuals are created from this allele distribution vector $\boldsymbol{D}_M(t)$ and randomly swapped into the population. Here, the parameter $\alpha \in [0.0, 1.0]$, called *associative factor*, determines the number of new individuals and hence the impact of the associative memory to the current population. Just as sampling a probability vector in PBIL algorithms [1], a new individual $S = \{s_1, \cdots, s_l\}$ is created by $\boldsymbol{D}_M(t) = \{d_1^M, \cdots, d_l^M\}$ (l is the encoding length) as follows:

$$s_i = \begin{cases} 1, & \text{if } rand(0.0, 1.0) < d_i^M \\ 0, & otherwise \end{cases} \tag{1}$$

The memory replacement strategy in AMGA is similar to that in DMGA. When the memory is due to update, if there are still any randomly initialized memory points in the memory, a random one will be replaced by $<S_P(t), \boldsymbol{D}_P(t)>$, where $S_P(t)$ and $\boldsymbol{D}_P(t)$ are the best individual and allele distribution vector of the current population respectively; otherwise, we first find the memory point $< S_M^c(t), \boldsymbol{D}_M^c >$ with its solution $S_M^c(t)$ closest to $S_P(t)$. If $S_P(t)$ is fitter than $S_M^c(t)$, i.e., $f(S_P(t)) > f(S_M^c(t))$, the memory point is replaced by $< S_P(t), \boldsymbol{D}_P(t) >$.

The aforementioned direct and associative memory can be combined into GAs. The GA with hybrid direct and associative memory schemes, denoted *DAMGA*,

```
t := 0 and t_M := rand(5, 10)
initialize P(0) randomly and empty memory M(0)
evaluate population P(0)
repeat
   evaluate memory M(t)
   if environmental change detected then
      denote the best memory point < S_M(t), D_M(t) >
      I(t) := create α * (n − m) individuals from D_M(t)
      P'(t) := swap individuals in I(t) into P(t) randomly
      if direct memory combined then    // for DAMGA
         P'(t) := retrieveBestMembersFrom(P'(t), M(t))
   else P'(t) := P(t)

   if t = t_M then t_M := t + rand(5, 10)  // time to update memory
      denote the best individual in P'(t) by S_P(t)
      extract the allele distribution vector D_P(t) from P'(t)
      if still any random point in memory then
         replace a random one by < S_P(t), D_P(t) >
      else find memory point < S^c_M(t), D^c_M(t) > closest to < S_P(t), D_P(t) >
         if f(S_P(t)) > f(S^c_M(t)) then < S^c_M(t), D^c_M > :=< S_P(t), D_P(t) >

   // standard genetic operations
   P'(t) := selectForReproduction(P'(t))
   crossover(P'(t), p_c) // p_c is the crossover probability
   mutate(P'(t), p_m)  // p_m is the mutation probability
   replace elite from P(t − 1) into P'(t) randomly
   evaluate the interim population P'(t)
until terminated = true          // e.g., t > t_max
```

Fig. 1. Pseudo-code for the AMGA and DAMGA

is also shown in Fig. 1. DAMGA differs from AMGA only as follows. After new individuals have been created and swapped into the population, the original memory solutions $M(t)$ are merged with the population to select $n - m$ best ones as the interim population to go though standard genetic operations.

4 Dynamic Test Environments

The DOP generator proposed in [16, 18] can construct *random* dynamic environments from any binary-encoded stationary function $f(\boldsymbol{x})$ ($\boldsymbol{x} \in \{0,1\}^l$) by a bitwise exclusive-or (XOR) operator. We suppose the environment changes every τ generations. For each environmental period k, an XORing mask $\boldsymbol{M}(k)$ is incrementally generated as follows:

$$\boldsymbol{M}(k) = \boldsymbol{M}(k-1) \oplus \boldsymbol{T}(k), \tag{2}$$

where "$\oplus$" is the XOR operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$) and $\boldsymbol{T}(k)$ is an intermediate binary template randomly created with $\rho \times l$ ones (ρ is

a parameter) for environmental period k. For the first period $k = 1$, $\boldsymbol{M}(1)$ is set to a zero vector. Then, the population at generation t is evaluated as below:

$$f(\boldsymbol{x}, t) = f(\boldsymbol{x} \oplus \boldsymbol{M}(k)), \tag{3}$$

where $k = \lceil t/\tau \rceil$ is the environmental period index. With this generator, the parameter τ controls the change speed while $\rho \in (0.0, 1.0)$ controls the severity of environmental changes. Bigger ρ means severer environmental change.

The above generator can be extended to construct *cyclic dynamic environments*[1], see [19], as follows. First, we can generate $2K$ XORing masks $\boldsymbol{M}(0), \cdots,$ $\boldsymbol{M}(2K-1)$ as the *base states* in the search space randomly. Then, the environment can cycle among them in a fixed logical ring. Suppose the environment changes every τ generations, then the individuals at generation t is evaluated as:

$$f(\boldsymbol{x}, t) = f(\boldsymbol{x} \oplus \boldsymbol{M}(I_t)) = f(\boldsymbol{x} \oplus \boldsymbol{M}(k\%(2K))), \tag{4}$$

where $k = \lfloor t/\tau \rfloor$ is the index of current environmental period and $I_t = k\%(2K)$ is the index of the base state the environment is in at generation t.

The $2K$ XORing masks can be generated as follows. First, we construct K binary templates $\boldsymbol{T}(0), \cdots, \boldsymbol{T}(K-1)$ that form a random partition of the search space with each template containing $\rho \times l = l/K$ bits of ones[2]. Let $\boldsymbol{M}(0) = \boldsymbol{0}$ denote the initial state, the other XORing masks are generated iteratively as:

$$\boldsymbol{M}(i+1) = \boldsymbol{M}(i) \oplus \boldsymbol{T}(i\%K), i = 0, \cdots, 2K-1 \tag{5}$$

The templates $\boldsymbol{T}(0), \cdots, \boldsymbol{T}(K-1)$ are first used to create K masks till $\boldsymbol{M}(K) = \boldsymbol{1}$ and then orderly reused to construct another K XORing masks till $\boldsymbol{M}(2K) = \boldsymbol{M}(0) = \boldsymbol{0}$. The Hamming distance between two neighbour XORing masks is the same and equals $\rho \times l$. Here, $\rho \in [1/l, 1.0]$ is the distance factor, determining the number of base states.

We can further construct *cyclic dynamic environments with noise* [19] as follows. Each time the environment is about to move to a next base state $\boldsymbol{M}(i)$, noise is applied to $\boldsymbol{M}(i)$ by flipping it bitwise with a small probability p_n.

In this paper, the 100-bit $OneMax$ function is selected as the base stationary function to construct dynamic test environments. $OneMax$ function aims to maximize the number of ones in a binary string. Three kinds of dynamic environments, cyclic, cyclic with noise, and random, are constructed from the base function using the aforementioned dynamic problem generator. For cyclic environments with noise, the parameter p_n is set to 0.05. For each dynamic environment, the landscape is periodically changed every τ generations during the

[1] For the convenience of description, we differentiate the environmental changing periodicality in time and space by wording *periodical* and *cyclic* respectively. The environment is said to be *periodically* changing if it changes in a fixed time interval, e.g., every certain GA generations, and is said to be *cyclicly* changing if it visits several fixed states in the search space in certain order repeatedly.

[2] In the partition each template $\boldsymbol{T}(i)$ $(i = 0, \cdots, K-1)$ has randomly but exclusively selected $\rho \times l$ bits set to 1 while other bits set to 0. For example, $\boldsymbol{T}(0) = 0101$ and $\boldsymbol{T}(1) = 1010$ form a partition of the 4-bit search space.

run of an algorithm. In order to compare the performance of algorithms in different dynamic environments, the parameters τ is set to 10, 25 and 50 and ρ is set to 0.1, 0.2, 0.5, and 1.0 respectively. Totally, a series of 36 DOPs, 3 values of τ combined with 4 values of ρ under three kinds of dynamic environments, are constructed from the stationary $OneMax$ function.

5 Experimental Study

5.1 Experimental Design

Experiments were carried out to compare the performance of GAs on the dynamic test environments. All GAs have the following generator and parameter settings: tournament selection with tournament size 2, uniform crossover with $p_c = 0.6$, bit flip mutation with $p_m = 0.01$, elitism of size 1, and population size $n = 100$ (including memory size $m = 10$ if used). In order to test the effect of the associative factor α on the performance of AMGA and DAMGA, α is set to 0.2, 0.6, and 1.0 respectively. And the corresponding GAs are reported as α-AMGA and α-DAMGA in the experimental results respectively.

For each experiment of a GA on a DOP, 50 independent runs were executed with the same set of random seeds. For each run 5000 generations were allowed, which are equivalent to 500, 200 and 100 environmental changes for $\tau = 10$, 25 and 50 respectively. For each run the best-of-generation fitness was recorded every generation. The overall performance of a GA on a problem is defined as:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}F_{BOG_{ij}}), \tag{6}$$

where $G = 5000$ is the total number of generations for a run, $N = 50$ is the total number of runs, and $F_{BOG_{ij}}$ is the best-of-generation fitness of generation i of run j. The off-line performance $\overline{F}_{BOG}$ is the best-of-generation fitness averaged over 50 runs and then averaged over the data gathering period.

5.2 Experimental Results and Analysis

Experiments were first carried out to compare the performance of SGA, DMGA and α-AMGAs under different dynamic environments. The experimental results regarding SGA, DMGA and α-AMGAs are plotted in Fig. 2. The major statistical results of comparing GAs by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table 1. In Table 1, the t-test result regarding $Alg.\ 1 - Alg.\ 2$ is shown as "+", "−", "s+" and "s−" when $Alg.\ 1$ is insignificantly better than, insignificantly worse than, significantly better than, and significantly worse than $Alg.\ 2$ respectively. From Fig. 2 and Table 1 several results can be observed.

First, both DMGA and AMGAs perform significantly better than SGA on most dynamic problems. This result validates the efficiency of introducing memory schemes into GAs in dynamic environments. Viewing from left to right in

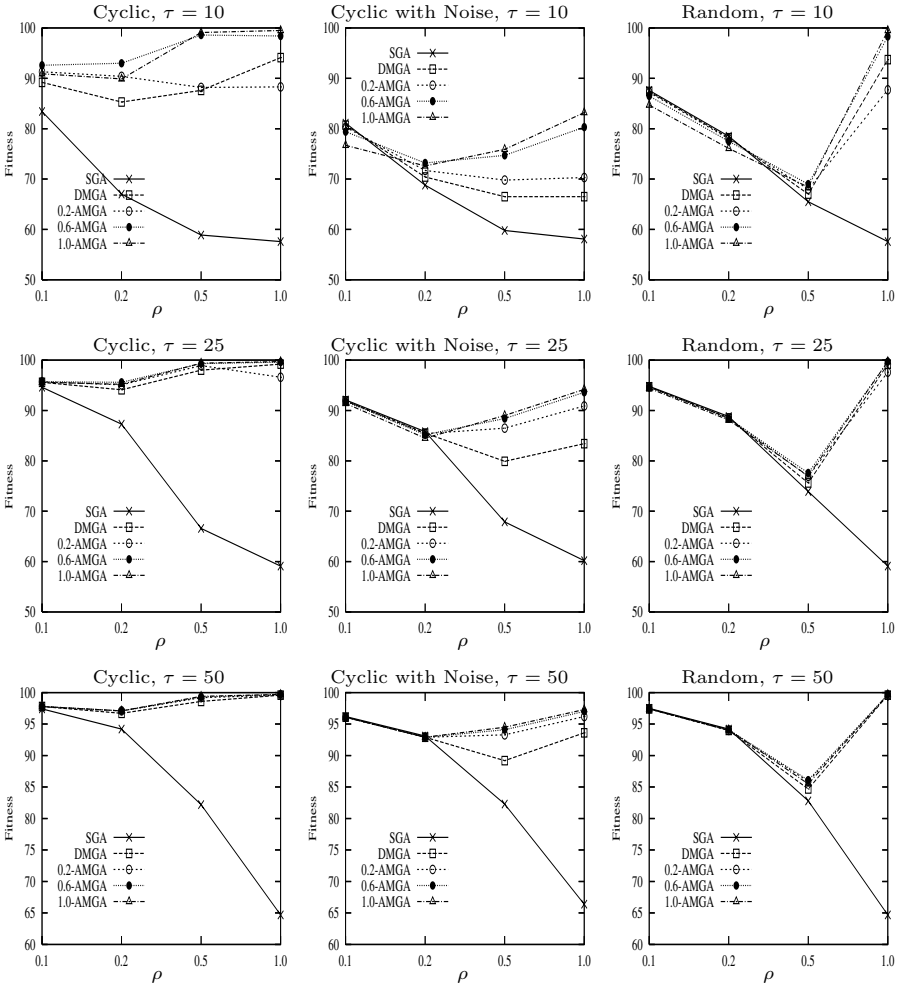

Fig. 2. Experimental results of SGA, DMGA, and α-AMGAs

Fig. 2, it can be seen that both DMGA and AMGAs achieve the largest performance improvement over SGA in cyclic environments. For example, when $\tau = 10$ and $\rho = 0.5$, the performance difference of DMGA over SGA, $\overline{F}_{BOG}(DMGA) - \overline{F}_{BOG}(SGA)$, is $87.6 - 58.9 = 28.7$, $66.5 - 59.8 = 6.7$, and $67.0 - 65.5 = 1.5$ under cyclic, cyclic with noise, and random environments respectively. This result indicates that the effect of memory schemes depends on the cyclicity of dynamic environments. When the environment changes randomly and slightly (i.e., ρ is small), both DMGA and AMGAs are beaten by SGA. This is because under these conditions, the environment is unlikely to return to a previous state that is memorized by the memory scheme. And hence inserting stored solutions or creating new ones according to the stored allele distribution vector may mislead or slow down the progress of the GAs.

Table 1. The t-test results of comparing SAG, DMGA and α-AMGAs

t-test Result	Cyclic				Cyclic with Noise				Random			
$\tau = 10$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMGA − SGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.2-AMGA − DMGA	$s+$	$s+$	$+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.6-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
1.0-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.6-AMGA − 0.2-AMGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
1.0-AMGA − 0.6-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$	$s+$
$\tau = 25$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMGA − SGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.2-AMGA − DMGA	$s+$	$s+$	$s+$	$s-$	$-$	$s-$	$s+$	$s+$	$-$	$-$	$s+$	$s-$
0.6-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
1.0-AMGA − DMGA	$-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.6-AMGA − 0.2-AMGA	$-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
1.0-AMGA − 0.6-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$	$s+$
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMGA − SGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.2-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$+$	$+$	$s+$	$s+$	$-$	$-$	$s+$	$s+$
0.6-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$-$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
1.0-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$-$	$-$	$s+$	$s+$	$-$	$+$	$s+$	$s+$
0.6-AMGA − 0.2-AMGA	$s+$	$+$	$s+$	$s+$	$-$	$-$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
1.0-AMGA − 0.6-AMGA	$-$	$-$	$s+$	$+$	$+$	$-$	$s+$	$s+$	$-$	$s-$	$s-$	$s+$

Second, comparing AMGAs over DMGA, it can be seen that AMGAs outperform DMGA on many DOPs, especially under cyclic environments. This happens because the extracted memory allele distribution vector is much stronger than the stored memory solutions in adapting the GA to the new environment. However, when ρ is small and the environment changes randomly, AMGAs are beaten by DMGA for most cases, see the t-test results regarding α-AMGA – DMGA. This is because under these environments the negative effect of the associative memory in AMGAs may weigh over the direct memory in DMGA.

In order to better understand the performance of GAs, the dynamic behaviour of GAs regarding best-of-generation fitness against generations on dynamic $OneMax$ functions with $\tau = 10$ and $\rho = 0.5$ under different cyclicity of dynamic environments is plotted in Fig. 3. In Fig. 3, the first and last 10 environmental changes (i.e., 100 generations) are shown and the data were averaged over 50 runs. From Fig. 3, it can be seen that, under cyclic and cyclic with noise environments, after several early stage environmental changes, the memory schemes start to take effect to maintain the performance of DMGA and AMGAs at a much higher fitness level than SGA. And the associative memory in AMGAs works better than the direct memory in DMGA, which can be seen in the late stage behaviour of GAs. Under random environments the effect of memory schemes is greatly deduced where all GAs behave almost the same and there is no clear view of the memory schemes in DMGA and AMGAs.

Third, when examining the effect of α on AMGA's performance, it can be seen that 0.6-AMGA outperforms 0.2-AMGA on most dynamic problems, see the t-test results regarding 0.6-AMGA – 0.2-AMGA. This is because increasing the value of α enhances the effect of associative memory for AMGA. However, 1.0-AMGA is beaten by 0.6-AMGA on many cases, especially when ρ is small, see the t-test results regarding 1.0-AMGA – 0.6-AMGA. When $\alpha = 1.0$, all the

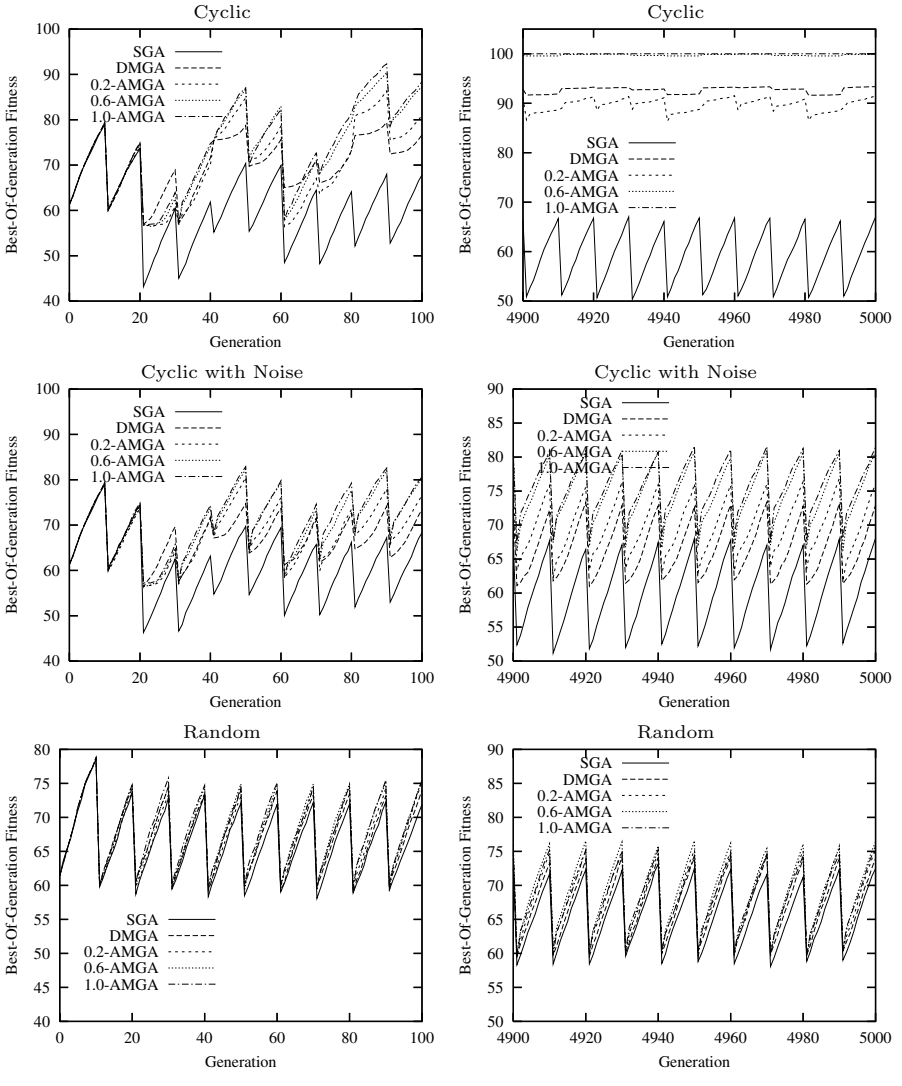

Fig. 3. Dynamic behaviour of GAs during the (*Left Column*) early and (*Right Column*) late stages on dynamic $OneMax$ functions with $\tau = 10$ and $\rho = 0.5$

individuals in the population are replaced by the new individuals created by the re-activated memory allele distribution vector when change occurs. This may be disadvantageous. Especially, when ρ is small, the environment changes slightly and good solutions of previous environment are likely also good for the new one. It is better to keep some of them instead of discarding them all.

In order to test the effect of combining direct memory with associative memory into GAs for DOPs, experiments were further carried out to compare the performance of DAMGAs over AMGAs. The relevant t-test results are presented

in Table 2, from which it can be seen that DAMGAs outperform AMGAs under most dynamic environments. However, the experiments (not shown here) indicate the performance improvement of α-DAMGA over α-AMGA is relatively small in comparison with the performance improvement of α-AMGA over SGA.

Table 2. The t-test results of comparing α-AMGAs and α-DAMGAs

t-test Result	Cyclic				Cyclic with Noise				Random			
$\tau = 10$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
0.2-DAMGA − 0.2-AMGA	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$-$	$+$	$s+$	$s+$
0.6-DAMGA − 0.6-AMGA	$s+$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$+$	$+$	$s+$	$s+$
1.0-DAMGA − 1.0-AMGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
$\tau = 25$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
0.2-DAMGA − 0.2-AMGA	$s+$	$s+$	$s+$	$s+$	$-$	$+$	$s+$	$s+$	$+$	$-$	$s+$	$s+$
0.6-DAMGA − 0.6-AMGA	$s+$	$+$	$s+$	$s+$	$+$	$-$	$+$	$s+$	$+$	$-$	$s+$	$s+$
1.0-DAMGA − 1.0-AMGA	$+$	$s+$	$s+$	$s+$	$+$	$s+$	$+$	$+$	$s+$	$+$	$s+$	$s+$
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
0.2-DAMGA − 0.2-AMGA	$+$	$+$	$-$	$s+$	$+$	$-$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
0.6-DAMGA − 0.6-AMGA	$+$	$+$	$s+$	$s+$	$-$	$+$	$+$	$s+$	$-$	$-$	$s+$	$s+$
1.0-DAMGA − 1.0-AMGA	$+$	$+$	$+$	$s+$	$-$	$-$	$-$	$+$	$+$	$-$	$s+$	$s+$

6 Conclusions and Discussions

This paper investigates the introduction of an associative memory scheme into GAs for dynamic optimization problems. Within this memory scheme, the allele distribution information is taken as the representation of the current environment that GAs have searched. The allele distribution vector is stored together with the best member of the current population in the memory. When the environmental change is detected, the stored allele distribution vector that is associated with the best re-evaluated memory solution is extracted to create new individuals into the population. A series of dynamic problems were systematically constructed, featuring three kinds of dynamic environments: cyclic, cyclic with noise, and random. Based on this test platform experimental study was carried out to test the proposed associative memory scheme.

From the experimental results, the following conclusions can be drawn on the dynamic test environments. First, memory schemes are efficient to improve the performance of GAs in dynamic environments and the cyclicity of dynamic environments greatly affect the performance of memory schemes for GAs in dynamic environments. Second, generally speaking the proposed associative memory scheme outperforms traditional direct memory scheme for GAs in dynamic environments. Third, the associative factor has an important impact on the performance of AMGAs. Setting α to a medium value, e.g., 0.6, seems a good choice for AMGAs. Fourth, combining the direct scheme with the associative memory scheme may further improve GA's performance in dynamic environments.

For future work, comparing the memory scheme investigated with implicit memory schemes is now under investigation. And it is also interesting to further investigate the interactions between the associative memory scheme and other approaches, such as multi-population and diversity approaches, for GAs in dynamic environments.

References

1. S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *Tech. Report CMU-CS-94-163*, Carnegie Mellon University, 1994.
2. J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. *Proc. of the 1999 Congr. on Evol. Comput.*, vol. 3, pp. 1875-1882, 1999.
3. J. Branke, T. Kaußler, C. Schmidth, and H. Schmeck. A multi-population approach to dynamic optimization problems. *Proc. of the Adaptive Computing in Design and Manufacturing*, pp. 299-308, 2000.
4. J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2002.
5. H. G. Cobb and J. J. Grefenstette. Genetic algorithms for tracking changing environments. *Proc. of the 5th Int. Conf. on Genetic Algorithms*, pp. 523-530, 1993.
6. D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pp. 59-68, 1987.
7. J. J. Grefenstette. Genetic algorithms for changing environments. *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, pp. 137-144, 1992.
8. A. Karaman, S. Uyar, and G. Eryigit. The memory indexing evolutionary algorithm for dynamic environments. *EvoWorkshops 2005*, LNCS 3449, pp. 563-573, 2005.
9. E. H. J. Lewis and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature*, pp. 139-148, 1998.
10. N. Mori, H. Kita and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. *Proc. of the 7th Int. Conf. on Genetic Algorithms*, pp. 299-306, 1997.
11. R. W. Morrison and K. A. De Jong. Triggered hypermutation revisited. *Proc. of the 2000 Congress on Evol. Comput.*, pp. 1025-1032, 2000.
12. K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1997.
13. C. L. Ramsey and J. J. Greffenstette. Case-based initializtion of genetic algorithms. *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 1993.
14. A. Simões and E. Costa. An immune system-based genetic algorithm to deal with dynamic environments: diversity and memory. *Proc. of the 6th Int. Conf. on Neural Networks and Genetic Algorithms*, pp. 168-174, 2003.
15. K. Trojanowski and Z. Michalewicz. Searching for optima in non-stationary environments. *Proc. of the 1999 Congress on Evol. Comput.*, pp. 1843-1850, 1999.
16. S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm. *Proc. of the 2003 IEEE Congress on Evol. Comput.*, vol. 3, pp. 2246-2253, 2003.
17. S. Yang. Population-based incremental learning with memory scheme for changing environments. *Proc. of the 2005 Genetic and Evol. Comput. Conference*, vol. 1, pp. 711-718, 2005.
18. S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, vol. 9, no. 11, pp. 815-834, 2005.
19. S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. Submitted to *IEEE Trans. on Evol. Comput.*, 2005.