1

Explicit Memory Schemes for Evolutionary 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

Summary. Problem optimization in dynamic environments has atrracted a growing interest from the evolutionary computation community in reccent years due to its importance in real world optimization problems. Several approaches have been developed to enhance the performance of evolutionary algorithms for dynamic optimization problems, of which the memory scheme is a major one. This chapter investigates the application of explicit memory schemes for evolutionary algorithms in dynamic environments. Two kinds of explicit memory schemes: direct memory and associative memory, are studied within two classes of evolutionary algorithms: genetic algorithms and univariate marginal distribution algorithms for dynamic optimization problems. Based on a series of systematically constructed dynamic test environments, experiments are carried out to investigate these explicit memory schemes and the performance of direct and associative memory schemes are campared and analysed. The experimental results show the efficiency of the memory schemes for evolutionary algorithms in dynamic environments, especially when the environment changes cyclically. The experimental results also indicate that the effect of the memory schemes depends not only on the dynamic problems and dynamic environments but also on the evolutionary algorithm used.

1.1 Introduction

Evolutionary algorithms (EAs) have been widely applied to solve stationary optimization problems. However, many real world problems are actually dynamic optimization problems (DOPs) . For DOPs, the fitness function, design variables, and/or environmental conditions may change over time due to many reasons, e.g., machine breakdown and financial factors. Hence, for DOPs the aim of an optimization algorithm is no longer to locate an optimal solution but to track the moving optima with time. This challenges traditional EAs seriously since they cannot adapt well to the changing environment once converged. However, traditional EAs with proper enhancements are still good tools of choice for optimization problems in dynamic environments. This is

S. Yang: *Explicit Memory Schemes for Evolutionary Algorithms in Dynamic Environments*, Studies in Computational Intelligence (SCI) **51**, 3–28 (2007)
www.springerlink.com © Springer-Verlag Berlin Heidelberg 2007

because EAs are basically inspired by principles of natural evolution, which has been taking place in the ever-changing dynamic environments in nature.

In recent years, there has been a growing interest in investigating EAs for DOPs. This trend reflects the importance of the practical application of EAs for real world optimization problems, many of which are DOPs [4]. Several approaches have been developed into EAs to address DOPs, such as maintaining diversity during the run via random immigrants [8, 23, 25], increasing diversity after a change [6, 7], using memory schemes to store and reuse useful information [3, 26], and multi-population approaches [12].

Among the approaches developed for EAs in dynamic environments, memory schemes have proved to be beneficial for many DOPs. Memory schemes work by storing useful information from the current environment and reusing it later in new environments. The useful information may be stored in two mechanisms: by implicit memory or by explicit memory. For implicit memory schemes, EAs use genotype representations that contain redundant information to store good (partial) solutions to be reused later. Typical examples are genetic algorithms (GAs) based on multiploidy representations [9, 9, 10, 12], structured encoding [7], or dualism mechanisms [24, 29]. Explicit memory schemes use precise representations but split an extra memory space to explicitly store useful information, e.g., good solutions [2, 3, 13, 22] and/or environmental information [21, 25], from the current generation for reuse in later generations or environments.

In this chapter, we focus on studying explicit memory schemes for EAs in dynamic environments. Two kinds of explicit memory schemes, *direct memroy* and *associative memory*, are investigated within two classes of EAs, GAs and univariate marginal distribution algorithms (UMDAs), for DOPs. For the direct memory scheme good solutions are stored in the memory and reused in new environments. For the associative memory scheme, the environmental information as well as good solutoins are stored and associated in the memory. When a change occurs, the stored environmental information associated with the best re-evaluated memory solution is used to create new individuals into the population. Using the dynamic problem generator proposed in [24, 29, 30], a series of dynamic test problems are constructed from a set of stationary functions and experiments are carried out to compare the performance of investigated GAs and UMDAs, with and without explicit memory schemes. The experimental results validates the efficiency of the memory scheme for GAs and UMDAs in dynamic environments.

The outline of this chapter is given as follows. The next section briefly reviews explicit memory schemes developed for EAs in dynamic environments. Section 1.3 describes the memory enhanced GAs investigated in this study while Section 1.4 describes the memory enhanced UMDAs investigated in this study. Section 1.5 presents the dynamic test environments for this study. The experimental results and relevant analysis are presented in Section 1.6. Section 1.7 concludes this paper with discussions on relevant future work.

1.2 Explicit Memory for EAs in Dynamic Environments

The application of memory schemes has proved to be able to enhance EA's performance in dynamic environments, especially when the environment changes cyclically in the search space[1]. In these environments, with time going an old environment will reappear exactly and the associated solution in the memory, which exactly remembers the old environment, will instantaneously move EAs to the reappeared environment.

As mentioned before, 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 EAs in dynamic environments depend on redundant representations to store useful information for EAs to exploit during the run. On the contrast, explicit memory schemes make use of precise representation but split an extra storage space where useful information from the current generation can be explicitly stored and reused in later generations or environments. For explicit memory schemes there are three major technical considerations: what to store in the memory, how to update the memory, and how to retrieve the memory.

For the first aspect, a natural choice is to store good solutions and reuse them when the environment change is detected. This is called *direct memory scheme*. For example, Louis and Xu [13] studied the open shop re-scheduling problem. Whenever a change (in a known pattern) occurs, the GA is restarted from a population with partial (5-10%) individuals inherited from the old run while the rest are randomly initialized. The authors reported a significant improvement of their GA over the GA with totally random restart scheme. Instead of storing good solutions only, the environmental information can also be stored and associated with good solutions in the memory. When the environment changes, the stored environemntal information can be used to associate with certain stored good solutions and reuse them more efficiently or used to create new individuals into the population. This memory scheme is called *associative memory scheme*. For example, Ramsey and Greffenstette [21] studied a GA for robot control problem, where good candidate solutions are stored in a permanent memory together with information about the robot current environment. 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 yield significant improvements. In Yang [26] and Yang and Yao [30], an associative memory scheme was introduced into population-based incremental learning (PBIL) algorithms [1]. In this memory scheme, the best sample in the population together with the propability vector, which represents the current environment, is stored in the memory.

[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 EA generations, and is said to be *cyclically* changing if it visits several fixed states in the search space in a certain order repeatedly.

When a change is detected, the probability vector associated with the best re-evaluated memory sample is used to create new samples. The associative memory greatly improves PBIL's performance in dynamic environments.

The memory space is usually limited (and fixed) for the efficiency of computation and searching. This leads to the second consideration of explicit memory schemes: memory organization and updating mechanisms. As to the memory organization, there exist two mechanisms: *local mechanism* where the memory is individual-oriented and *global mechanism* where the memory is population-oriented. Trojanowski and Michalewicz [22] introduced a local memory approach, where for each individual the memory stores a number of its ancestors. When the environment changes, the current individual and its ancestors are re-evaluated and compete together with the best becoming the active individual while the others stored in the memory. The global memory mechanism is more natural and popular. In the global memory mechanism, the best individual of the population is replaced into the memory every certain or random generations according to a certain replacement policy, see [3, 3].

As to the memory updating mechanism, a general principle is to select one memory individual to be removed for or updated by the best individual from the population in order to make the stored individuals to be of above average fitness, not too old, and distributed across several promising areas of the search space. Branke [3] has discussed several memory replacement strategies: 1). replacing the least important one with the importance value of individuals being the linear combination of age, contribution to diversity, and fitness; 2). replacing the one with least contribution to memory variance; 3). replacing the most similar one if the new individual is better; and 4). replacing the less fit of a pair of memory individuals that has the minimum distance among all pairs. The third strategy seems the most practical one due to its simplicity and will be applied in the memory enhanced EAs studied in this chapter. Bendtsen and Krink [2] proposed a different memory updating scheme where the memory individual closest to the best population individual is moved toward the best population individual, instead of being replaced from the memory by the best population individual.

For the third concern regarding how to retrieve the memory, a natural idea is to retrieve the best memory individual(s) to replace the least fit individual(s) in the population. This can be done every generation or only when the environment changes. The memory retrieval is sort of coupled with the above two concerns. For example, for the direct memory scheme the whole memory individuals may enter the new population as in [13] or compete with the population individuals for the new population as in [3], while for the associative memory scheme only the associated memory individual(s) [21] or new individuals created by the associated environmental information [26, 30] may enter the new population. And for the local memory organization scheme the best ancestor of an active individual competes with it to become active in the population [22], while for the global memory scheme the best memory individual(s) may compete with all individuals in the population.

```
t := 0 and initialize population P(0) randomly
repeat
   evaluate(P(t))
   replace the worst individual in P(t) by elite from P(t − 1)
   P′(t) := selectForReproduction(P(t))
   crossover(P′(t), p_c)
   mutate(P′(t), p_m)
until termination condition holds        // e.g., t > t_max
```

Fig. 1.1. Pseudo-code for the standard GA (SGA) where elitism of size 1 is used

In the following two sections we will respectively describe the GAs and UMDAs with direct and associative memory schemes, which are investigated in this chapter.

1.3 Description of Investigated GAs

1.3.1 The Standard GA

The standard GA 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-spring. This process continues until some termination condition becomes true, e.g., the maximum allowable number of generations t_{max} is reached. The pseudo-code for the standard GA (SGA) investigated in this chapter is shown in Fig. 1.1, where p_c and p_m are the crossover and mutation probabilities respectively and the elitism is used.

Usually, with the iteration of SGA, individuals in the population will eventually converge to the optimam or near optimum solution(s) in stationary environments due to the pressure of selection. Convergence at a proper pace, instead of pre-mature, may be beneficial and, in fact, is expected in many optimization problems for GAs to locate expected solutions in stationary environments. However, convergence becomes a big problem for GAs in dynamic environments. In fact, it is the main reason why traditional GAs do not perform well in dynamic environments. Convergence deprives the population of genetic diversity. Consequently, when a change occurs, it is hard for GAs to adpat to the new environment. Hence, in dynamic environments additional approaches are required to maintain the population diversity by random immigrants or adapt the GA directly to the new environment by memory schemes. The next two sub-sections describe respectively GAs with direct memory and associative memory enhancements, which are the main concern of this chapter.

$t := 0$ and initialize population $P(0)$ randomly
$t_M := rand(5, 10)$ and initialize memory $M(0)$ randomly
repeat
evaluate($P(t), M(t)$)
replace the worst individual in $P(t)$ by elite from $P(t-1)$

if *environmental change detected* **then**
if *DMGA* **then** $P'(t)$:= retrieveBestMembers($P(t), M(t)$)
else // *for AMGA and HMGA*
denote the best memory point $< B_M(t), \mathbf{D}_M(t) >$
$I(t)$:= create $\alpha * (n-m)$ individuals from $\mathbf{D}_M(t)$
$P'(t)$:= replace the worst individuals in $P(t)$ by ones in $I(t)$
if *HMGA* **then** $P'(t)$:= retrieveBestMembers($P'(t), M(t)$)
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 $B_P(t)$
if *not DMGA* **then** $\mathbf{D}_P(t)$:= allele distribution vector in $P'(t)$

if *still any random point in* $M(t)$ **then**
if *DMGA* **then** replace a random memory point by $B_P(t)$
else replace a random memory point by $< B_P(t), \mathbf{D}_P(t) >$
else // *memory is full*
if *DMGA* **then** $S^c_M(t)$:= the memory point closest to $B_P(t)$
if $f(B_P(t)) \geq f(S^c_M(t))$ **then** $S^c_M(t) := B_P(t)$
else $< S^c_M(t), \mathbf{D}^c_M(t) >$:= the memory point closest to $< B_P(t), \mathbf{D}_P(t) >$
if $f(B_P(t)) \geq f(S^c_M(t))$ **then** $< S^c_M(t), \mathbf{D}^c_M(t) > := < B_P(t), \mathbf{D}_P(t) >$

// standard genetic operations
$P'(t)$:= selectForReproduction($P'(t)$)
crossover($P'(t), p_c$)
mutate($P'(t), p_m$)
until *termination condition holds* // *e.g.,* $t > t_{max}$

Fig. 1.2. Pseudo-code for the memory enhanced GAs: GA with direct memory (DMGA), GA with associative memory (AMGA), and GA with hybrid memory (HMGA)

1.3.2 GA with Direct Memory

The pseudo-code for the GA with direct memory, denoted *DMGA* in this chapter, is shown in Fig. 1.2, where $f(\cdot)$ is the fitness function. DMGA (and other memory enhanced EAs in this study) uses a memory of size $m = 0.1 * n$, which is randomly initialized. 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. After a memory updating at generation t, the next memory updating time t_M is given by: $t_M = t + rand(5, 10)$. This way, the potential effect that the environmental change period coincides with the memory updating period (e.g., the memory is updated whenever the environment changes) can be smoothed away.

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 is detected having changed its fitness. If a change is detected, the memory is merged with the current 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.

1.3.3 GA with Associative Memory

In [26, 30], an associative memory has been developed for PBILs in dynamic environments. The idea can be extended to GAs for DOPs [28]. That is, we can store the environmental information together with good solutions in the memory for later reuses. Here, the key thing is how to represent the current environment. As mentioned before, given a problem in a certain environment the population of a GA will eventually converge toward the optimum or near optimum of the environment when the GA progresses its searching. The convergence information, i.e., the *allele distribution* in the population, can be taken as the natural representation of the 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, called 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, denoted $AMGA$, is also shown in Fig. 1.2. Within AMGA, the memory is used to store solutions and associated environmental information. That is, each memory point consists of a pair $< S, \mathbf{D} >$, where S is the stored solution and $\mathbf{D}$ is the associated allele distribution vector. For binary encoding (as per this study), 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 $< B_M(t), \mathbf{D}_M(t) >$, i.e., the memory point with its solution $B_M(t)$ having the highest re-evaluated fitness, is extracted. And a set of $\alpha * (n - m)$ new individuals are created from this allele distribution vector $\mathbf{D}_M(t)$ and swapped into the population by replacing the worst individuals. Here, the parameter $\alpha \in [0.0, 1.0]$, called *associative factor*, determines the number of new individuals to be generated and hence the impact of the associative memory to the current population. A new individual $S = \{s_1, \cdots, s_l\}$ is created by $\mathbf{D}_M(t) = \{d_1, \cdots, d_l\}$ (l is the encoding length) as follows:

$$s_i = \begin{cases} 1, & \text{if } rand(0.0, 1.0) < d_i \\ 0, & otherwise \end{cases} \tag{1.1}$$

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 $< B_P(t),\ \mathbf{D}_P(t) >$, where $B_P(t)$ and $\mathbf{D}_P(t)$ are the best individual and allele distribution vector of the current population respectively; otherwise, we first find the memory point $< S_M^c(t), \mathbf{D}_M^c >$ with its solution $S_M^c(t)$ closest to $B_P(t)$. If $B_P(t)$ is fitter than $S_M^c(t)$, i.e., $f(B_P(t)) > f(S_M^c(t))$, the memory point is replaced by $< B_P(t), \mathbf{D}_P(t) >$.

The aforementioned direct and associative memory can be combined into GAs. The resulted GA is called the *hybrid memory based GA* (HMGA in short). The pseudo-code of HMGA is also shown in Fig. 1.2. HMGA differs from AMGA only as follows. When a change is detected, new individuals are created from the allele distribution vector of the best memory point and swapped into the population. Then, the original memory solutions $M(t)$ are merged with the main population to select $n - m$ best ones as the interim population to go though standard genetic operations.

1.4 Description of Investigated UMDAs

1.4.1 The Standard UMDA

Mühlenbein [19] introduced the UMDA as the simplest version of estimation of distribution algorithms (EDAs) [18]. Thereafter, there have been several modifications of UMDAs [14] and UMDAs have been applied to many optimization problems [11]. In the binary search space, UMDAs evolve a probability vector $\mathbf{p}(t) = (p(1,t), \ldots, p(l,t))$ where all the variables are assumed to be independent of each other. The pseudo-code for the standard UMDA (SUMDA) studied in this chapter is shown in Fig. 1.3, where the mechanisms of mutation and elitism are used.

SUMDA starts from the *central probability vector* that has a value of 0.5 for each locus and falls in the central point of the search space. Sampling this probability vector creates random solutions because the probability of creating a 1 or 0 on each locus is equal[2]. At iteration t, a population $S(t)$ of n individuals are sampled from the probability vector $\mathbf{p}(t)$. The samples are evaluated and an interim population $D(t)$ is formed by selecting μ ($\mu < n$) best individuals, denoted $\mathbf{x}_1(t), \cdots, \mathbf{x}_\mu(t)$, from $S(t)$. Then, the probability vector is updated by extracting statistics information from $D(t)$ as follows:

[2] Without loss of generality, a binary-encoded solution $\mathbf{x} = (x_1, \ldots, x_l) \in \{0,1\}^l$ is sampled from a probability vector $\mathbf{p}(t)$ as follows: for each locus i, if a randomly created number $r = rand(0.0, 1.0) < p(i,t)$, its allele x_i is set to 1; otherwise, x_i is set to 0.

```
t := 0 and initialize the probability vector p(0) := 0.5
repeat
   sample a population S(t) of individuals by p(t)
   evaluate(S(t))
   replace the worst individual in S(t) by elite from S(t − 1)
   select the best μ individuals from S(t) to form D(t)
   build p′(t) according to D(t) by Eqn. (1.2)
   mutate p′(t) by Eqn. (1.3)
until termination condition holds        // e.g., t > t_max
```

Fig. 1.3. Pseudo-code of the standard UMDA (SUMDA) with mutation and elitism

$$\mathbf{p}'(t) := \frac{1}{\mu}\sum_{k=1}^{k=\mu}\mathbf{x}_k(t) \tag{1.2}$$

After the probability vector is updated according to $D(t)$, in order to keep the diversity of generated samples in dynamic environments, a bitwise mutation is applied in SUMDA. The mutation operation always changes the probability vector toward the central probability vector as follows. For each locus $i = \{1, \ldots, l\}$, if a random number $r = rand(0.0, 1.0) < p_m$ (p_m is the mutation probability), then mutate $p(i,t)$ using the following formula:

$$p'(i,t) = \begin{cases} p(i,t) * (1.0 - \delta_m), & p(i,t) > 0.5 \\ p(i,t), & p(i,t) = 0.5 \\ p(i,t) * (1.0 - \delta_m) + \delta_m, & p(i,t) < 0.5, \end{cases} \tag{1.3}$$

where δ_m is the mutation shift that controls the amount a mutation operation alters the value in each bit position. After the mutation operation, a new set of samples is generated by the new probability vector and this cycle is repeated.

As the search progresses, the elements in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0, representing smaples of high fitness. The search stops when some termination condition holds, e.g., the maximum allowable number of iterations t_{max} is reached.

1.4.2 UMDA with Direct Memory

The direct memory scheme for GAs can be easily extended to UMDAs for DOPs. The pseudo-code for the investigated UMDA with the direct memory, denoted *DMUMDA*, is shown in Fig. 1.4. In Fig. 1.4, n is the number of evaluations per iteration including the memory samples and $f(\mathbf{x})$ denotes the fitness of individual $\mathbf{x}$.

As in DMGA, DMUMDA uses a memory to store best samples from the population. And the memory in DMUMDA is updated using the same stochastic time pattern as in DMGA: after a memory update at time t, the next

$t := 0$ and initialize $\mathbf{p}(0) := \mathbf{0.5}$
$t_M := rand(5, 10)$ and initialize memory $M(0)$ randomly
repeat
  sample a population $S(t)$ of individuals by $\mathbf{p}(t)$
  evaluate($S(t), M(t)$)
  replace the worst individual in $S(t)$ by elite from $S(t-1)$

  if *environmental change detected* **then**
    if *DMUMDA* **then** $S'(t) :=$ retrieveBestMembers($S(t), M(t)$)
    else // *for AMUMDA and HMUMDA*
      denote the best memory point $<B_M(t), \mathbf{p}_M(t)>$
      $I(t) :=$ create $\alpha * (n - m)$ individuals from $\mathbf{p}_M(t)$
      $S'(t) :=$ replace the worst individuals in $S(t)$ by ones in $I(t)$
      if *HMUMDA* **then** $S'(t) :=$ retrieveBestMembers($S'(t), M(t)$)
  else $S'(t) := S(t)$

  if $t = t_M$ **then** $t_M := t + rand(5, 10)$ // *time to update memory*
    denote the best individual in $S'(t)$ by $B_S(t)$
    if *still any random point in* $M(t)$ **then**
      if *DMUMDA* **then** replace a random memory point by $B_S(t)$
      else replace a random memory point by $<B_S(t), \mathbf{p}(t)>$
    else // *memory is full*
      if *DMUMDA* **then** $S_M^c(t) :=$ the memory point closest to $B_S(t)$
        if $f(B_S(t)) \geq f(S_M^c(t))$ **then** $S_M^c(t) := B_S(t)$
      else $<S_M^c(t), \mathbf{p}_M^c(t)> :=$ the memory point closest to $<B_S(t), \mathbf{p}(t)>$
        if $f(B_S(t)) \geq f(S_M^c(t))$ **then** $<S_M^c(t), \mathbf{p}_M^c(t)> := <B_S(t), \mathbf{p}(t)>$

  select the best μ individuals from $S'(t)$ to form $D(t)$
  build $\mathbf{p}'(t)$ according to $D(t)$ by Eqn. (1.2)
  mutate $\mathbf{p}'(t)$ by Eqn. (1.3)
until *termination condition holds* // *e.g.,* $t > t_{max}$

Fig. 1.4. Pseudo-code for the memory enhanced UMDAs: UMDA with direct memory (DMUMDA), UMDA with associative memory (AMUMDA), and UMDA with hybrid memory (HMUMDA)

memory updating time is $t_M = t + rand(5, 10)$. When the memory is due to update, we first find the memory point closest to the best population sample in terms of Hamming distance. If the best population sample has higher fitness than this memory sample, it is replaced by the best population sample; otherwise, the memory stays unchanged.

The memory in DMUMDA is re-evaluated every iteration. If any memory sample has its fitness changed, the environment is detected to be changed. Then, the memory will be merged with the current population to form an intermit population. If no environmental change is detected, DMUMDA progresses just as the standard UMDA does.

1.4.3 UMDA with Associative Memory

Using associative memory for UMDAs is more straightforward than for GAs because the probability vector that is evolved within UMDAs can be directly taken as the environmental information without any cost of further calculation. Each time when the best sample of the population is stored in the memory, the probability vector is also stored in the memory and associated with the sample. The pseudo-code of the UMDA with the associative memory, denoted *AMUMDA*, is also shown in Fig. 1.4.

The memory in AMUMDA has $m = 0.1*n$ points, each consisting of a pair $<S, \mathbf{p}>$, where S is a stored sample and $\mathbf{p}$ is the associated probability vector. The memory is re-evaluated every generation. If an environmental change is detected, the probability vector of the best memory point $<B_M(t), \mathbf{p}_M(t)>$ is extracted to create a set of $\alpha*(n-m)$ new samples to replace the worst ones in the population. Here, the parameter $\alpha \in [0.0, 1.0]$ is the associative factor. The memory in AMUDMA is updated similarly as in AMGA. When the memory is due to update, if there are still any randomly initialized memory points in the memory, a random one is replaced by $<B_S(t), \mathbf{p}(t)>$, where $B_S(t)$ is the best sample in the population; otherwise, the memory point $<S^c_M(t), \mathbf{p}^c_M(t)>$ closest to $B_S(t)$ is replaced by $<B_S(t), \mathbf{p}(t)>$ if $B_S(t)$ is fitter than $S^c_M(t)$.

Similarly, the above direct and associative memory can be combined into UMDAs. The pseudo-code of the UMDA with a hybrid direct and associative memory, denoted *HMUMDA*, is also shown in Fig. 1.4. In HMUMDA, when a change is detected, after integrating the individuals that are sampled from the best memory probability vector into the population, the memory samples $M(t)$ are also merged with the population to select $n - m$ best ones as the interim population to build a new model.

1.5 Dynamic Test Environments

The dynamic problem generator proposed in [24, 29] can construct *random dynamic environments* from any binary-encoded stationary function $f(\mathbf{x})$ ($\mathbf{x} \in \{0, 1\}^l$) by a bitwise exclusive-or (XOR) operator. Suppose the environment changes every τ generations. For each environmental period k, an XORing mask $\mathbf{M}(k)$ is incrementally generated as follows:

$$\mathbf{M}(k) = \mathbf{M}(k-1) \oplus \mathbf{T}(k), \tag{1.4}$$

where "$\oplus$" is the XOR operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$) and $\mathbf{T}(k)$ is an intermediate binary template randomly created with $\rho \times l$ ones for environmental period k. For the first period $k = 1$, $\mathbf{M}(1)$ is set to a zero vector. Then, the population at generation t is evaluated as below:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(k)), \tag{1.5}$$

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.

Recently, the XOR dynamic problem generator has been extended to construct *cyclic dynamic environments* in [27] and *cyclic dynamic environments with noise* further in [30]. With the XOR generator, cyclic dynamic environments are constructed as follows. First, we can generate $2K$ XORing masks $\mathbf{M}(0), \mathbf{M}(1), \cdots, \mathbf{M}(2K-1)$ as the *base states* in the search space randomly. Then, the environment can cycle among these base states in a fixed logical ring. Suppose the environment changes every τ generations, then the individuals at generation t is evaluated as follows:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(I_t)) = f(\mathbf{x} \oplus \mathbf{M}(k\%(2K))), \tag{1.6}$$

where $k = \lfloor t/\tau \rfloor$ is the index of the 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 in the following way. First, we construct K binary templates $\mathbf{T}(0), \cdots, \mathbf{T}(K-1)$ that form a random partition of the search space with each template containing $\rho \times l = l/K$ bits of ones[3]. Let $\mathbf{M}(0) = \mathbf{0}$ denote the initial state. Then, the other XORing masks are generated iteratively as follows:

$$\mathbf{M}(i+1) = \mathbf{M}(i) \oplus \mathbf{T}(i\%K), i = 0, \cdots, 2K-1 \tag{1.7}$$

The templates $\mathbf{T}(0), \cdots, \mathbf{T}(K-1)$ are first used to create K masks till $\mathbf{M}(K) = \mathbf{1}$ and then orderly reused to construct another K XORing masks till $\mathbf{M}(2K) = \mathbf{M}(0) = \mathbf{0}$. 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.

From the XOR generator, we can further construct cyclic dynamic environments with noise as follows. We can construct a set of base states and let the environment cycles among the base states just as above. However, each time the environment is about to move to a next base state $\mathbf{M}(i)$, $\mathbf{M}(i)$ is bitwise flipped with a small probability, denoted p_n in this chapter.

In this experimmental study, three 100-bit binary functions, denoted *OneMax*, *Royal Road* and *Deceptive* respectively, are selected as the base stationary functions to construct dynamic test environments. They all consist of 25 contiguous 4-bit building blocks and have an optimum fitness of 100. As shown in Fig. 1.5, the building block for each function is defined based on the unitation function, i.e., the number of ones inside the building block. The building block for *OneMax* is just a *OneMax* sub-function, which aims to maximize the number of ones in a chromosome. The building block for *Royal Road* contributes 4 to the total fitness if its unitation is 4; otherwise, it contributes 0. The building block for *Deceptive* is fully deceptive. These three stationary functions have increasing difficulty for EAs in the order from *OneMax* to *Royal Road* to *Deceptive*.

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

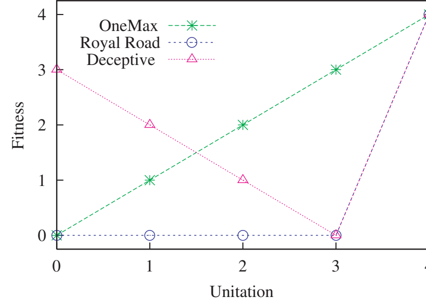

Fig. 1.5. Building block of the three stationary functions

Three kinds of dynamic environments, cyclic, cyclic with noise and random, are constructed from each base function using the XOR DOP generator. For each kind of dynamic environments, the landscape is periodically changed every τ generations during the run of an EA. In order to compare the performance of EAs in different dynamic environments, the parameters τ is set to 10 and 25 and ρ is set to 0.1, 0.2, 0.5, and 1.0 respectively. For cyclic dynamic problems with noise, the noise probability p_n is set to 0.05. Totally, a series of 24 DOPs, 2 values of τ combined with 4 values of ρ under three kinds of dynamic environments, are constructed from each stationary function.

1.6 Experimental Study

1.6.1 Experimental Design

Experiments were carried out to compare the performance of investigated EAs on the dynamic test environments. For all EAs, the parameters are set as follows: the total population size is set to $n = 100$, including memory size $m = 0.1 * n = 10$ if used, and the elitism size is set to 1. For all GAs, parameters are set as: standard uniform crossover with the crossover probability $p_c = 0.6$, bit flip mutation with the mutation probability $p_m = 0.01$. For all UMDAs, the mutation probability $p_m = 0.02$ with the mutation shift $\delta_m = 0.05$, μ is set to $0.5 * n$ for SUMDA or $0.5 * (n - m)$ for memory enhanced UMDAs. For AMGAs and AMUMDAs, in order to test the effect of the associative factor α on their performance, α is set to 0.1, 0.5, and 1.0 respectively. For HMGAs and HMUMDAs, the associative factor α is set to 0.5. And an EA with associative memory will be reported as α-AMGA, α-HMGA, α-AMUMDA, or α-HMUMDA respectively in the experimental results.

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

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}F_{BOG_{ij}}), \tag{1.8}$$

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 offline performance $\overline{F}_{BOG}$ is the best-of-generation fitness averaged over 50 runs and then averaged over the data gathering period.

1.6.2 Experimental Results and Analysis of GAs on DOPs

The experimental results of GAs on the dynamic test problems under cyclic, cyclic with noise, and random dynamic environments are plotted in Fig. 1.6 to Fig. 1.8 respectively. The corresponding 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 1.1 to Table 1.3 respectively. In Table 1.1 to Table 1.3, the t-test result regarding $Alg.\ 1 - Alg.\ 2$ is shown as "=", "+", "−", "$s+$" or "$s-$" if $Alg.$ 1 is statistically equivalent to, insignificantly better than, insignificantly worse than, significantly better than, or significantly worse than $Alg.$ 2 respectively. From the figures and tables several results can be observed.

First, both DMGA and AMGAs perform significantly better than SGA on most dynamic problems, especially in cyclic environments. This result validates the efficiency of introducing memory schemes, either direct or associative, into GAs in dynamic environments. Viewing across Fig. 1.6 to Fig. 1.8, 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 $94.1 - 58.9 = 35.2$, $67.2 - 59.8 = 7.4$, and $67.2 - 65.6 = 1.6$ 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.

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.

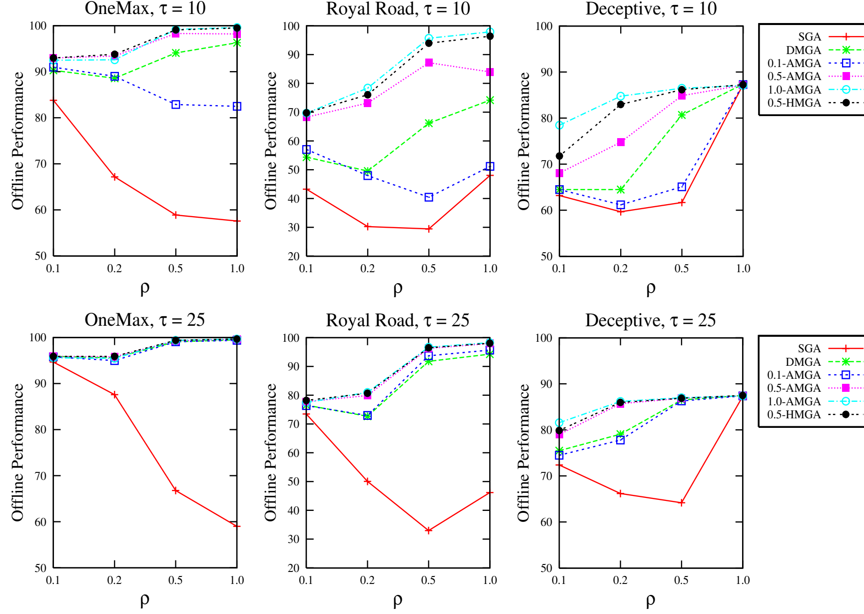

Fig. 1.6. Experimental results of GAs on cyclic DOPs

Table 1.1. The t-test results of comparing GAs on cyclic DOPs

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	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	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
0.1-AMGA − DMGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$+$	$s-$	$s-$	$s-$
0.5-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
1.0-AMGA − DMGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
0.5-AMGA − 0.1-AMGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
0.5-HMGA − 0.5-AMGA	$-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
$\tau = 25,\ \rho \Rightarrow$	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	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMGA − DMGA	$s-$	$s-$	$s+$	$-$	$-$	$+$	$s+$	$s+$	$s-$	$-$	$-$	$-$
0.5-AMGA − DMGA	$-$	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
1.0-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
0.5-AMGA − 0.1-AMGA	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s+$	$s+$	$-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
0.5-HMGA − 0.5-AMGA	$s+$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$+$	$s+$

In order to better understand the performance of GAs, the dynamic performance 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. 1.9. In Fig. 1.9, the first and last 10 environmental changes (i.e., 100 generations) are shown and the data were

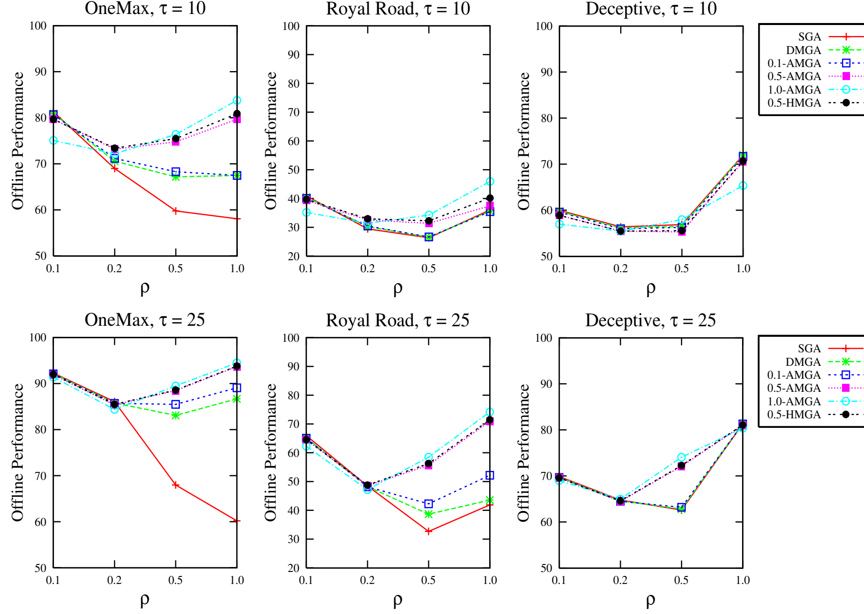

Fig. 1.7. Experimental results of GAs on cyclic DOPs with noise

Table 1.2. The t-test results of comparing GAs on cyclic DOPs with noise

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	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	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$-$	$-$	$-$	$-$
0.5-AMGA − SGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
0.1-AMGA − DMGA	$s-$	$s+$	$s+$	$-$	$s-$	$s+$	$+$	$s-$	$s-$	$-$	$-$	$s-$
0.5-AMGA − DMGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
1.0-AMGA − DMGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.5-AMGA − 0.1-AMGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$=$	$s+$	$s-$
0.5-HMGA − 0.5-AMGA	$-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$+$	$s+$	$+$
$\tau = 25,\ \rho \Rightarrow$	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	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$-$	$-$	$+$	$+$
0.5-AMGA − SGA	$s-$	$s-$	$s+$	$s+$	$s-$	$+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.1-AMGA − DMGA	$-$	$s-$	$s+$	$s+$	$-$	$-$	$s+$	$s+$	$-$	$-$	$s+$	$s-$
0.5-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$-$	$s+$	$s-$
1.0-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
0.5-AMGA − 0.1-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$-$	$s+$	$s-$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
0.5-HMGA − 0.5-AMGA	$+$	$+$	$+$	$s+$	$+$	$s+$	$s+$	$s+$	$+$	$+$	$s+$	$+$

averaged over 50 runs. From Fig. 1.9, 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

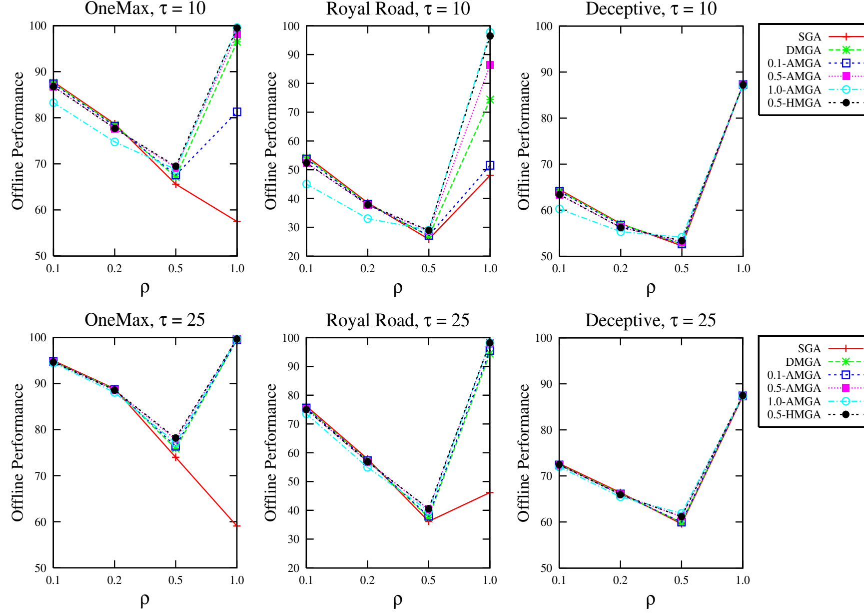

Fig. 1.8. Experimental results of GAs on random DOPs

Table 1.3. The t-test results of comparing GAs on random DOPs

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	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	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.5-AMGA − SGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$-$
0.1-AMGA − DMGA	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$
0.5-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
1.0-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.5-AMGA − 0.1-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.5-HMGA − 0.5-AMGA	$+$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$+$	$+$	$s+$	$s+$
$\tau = 25,\ \rho \Rightarrow$	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	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$-$	$s-$	$s+$	$s+$
0.5-AMGA − SGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
0.1-AMGA − DMGA	$-$	$s-$	$s+$	$+$	$-$	$-$	$s+$	$s+$	$-$	$s-$	$s+$	$s-$
0.5-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$-$	$s-$	$s+$	$s-$
1.0-AMGA − DMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$
0.5-AMGA − 0.1-AMGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$+$	$s-$	$s+$	$+$
1.0-AMGA − 0.5-AMGA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$	$-$
0.5-HMGA − 0.5-AMGA	$-$	$-$	$s+$	$s+$	$-$	$+$	$s+$	$+$	$+$	$+$	$+$	$s+$

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 vision regarding the effect of the memory schemes on the performance of DMGA and AMGAs.

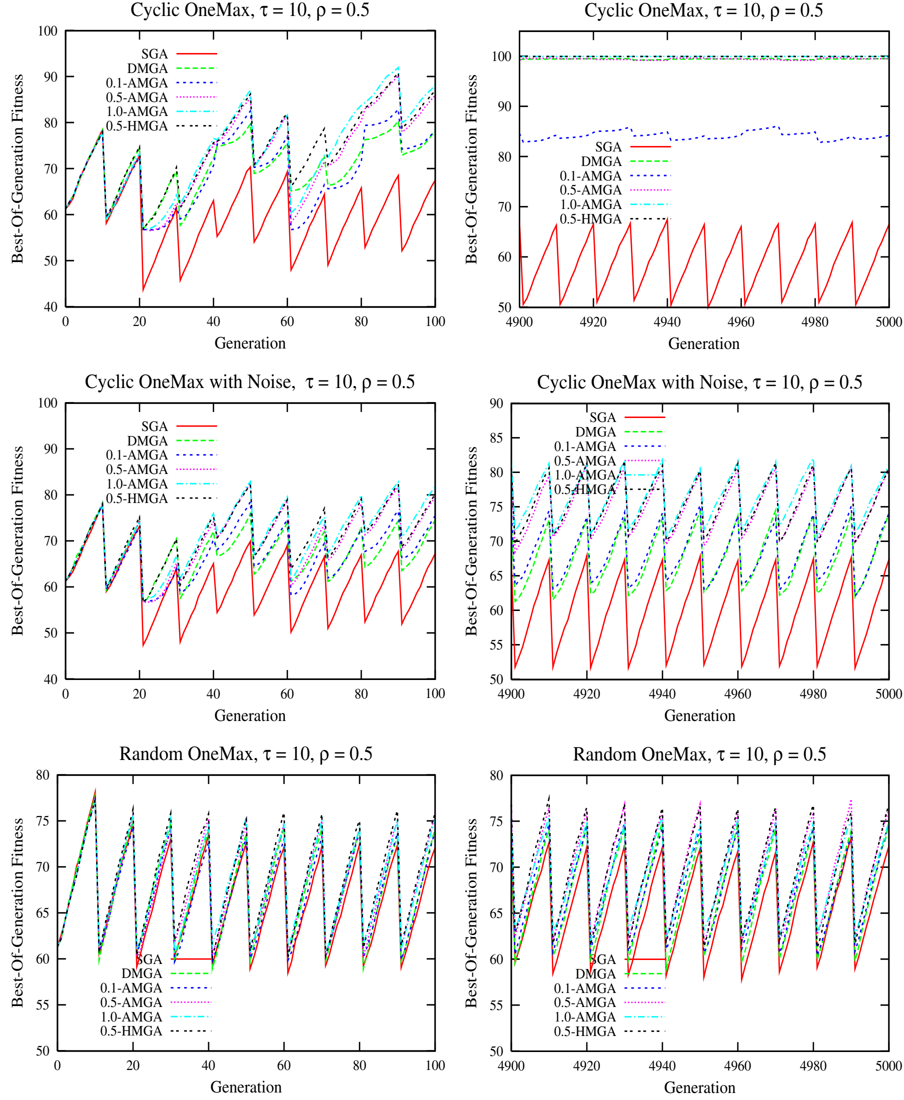

Fig. 1.9. Dynamic performance of GAs on the dynamic *OneMax* problems

Third, when examining the effect of α on AMGA's performance, it can be seen that 0.5-AMGA outperforms 0.1-AMGA on most dynamic problems, see the t-test results regarding 0.5-AMGA – 0.1-AMGA. This is because increasing the value of α enhances the effect of associative memory for AMGA. However, 1.0-AMGA is beaten by 0.5-AMGA on many cases, especially when ρ is small, see the t-test results regarding 1.0-AMGA – 0.5-AMGA. When

$\alpha = 1.0$, all individuals in the population are replaced by the new individuals created by the re-activated memory allele distribution vector when a change occurs. This may be disadvantageous. Especially, when ρ is small, the environment changes slightly and good solutions of the previous environment are likely also good for the new one. It is better to keep some of them instead of discarding them all.

Finally, comparing the performance of HMGA over AMGAs for DOPs, it can be seen that HMGA outperforms AMGAs for most dynamic problems, see the t-test results regarding 0.5-HMGA − 0.5-AMGA. For example, on the cyclic dynamic *Royal Road* function with $\tau = 10$ and $\rho = 0.5$, the performance of 0.5-HMGA is $\overline{F}_{BOG}(0.5\text{-}HMGA) = 94.0$, which is significantly better than the performance of 0.5-AMGA with $\overline{F}_{BOG}(0.5\text{-}AMGA) = 87.2$. However, the performance improvement of α-HMGA over α-AMGA is relatively small in comparison with the performance improvement of α-AMGA over SGA.

1.6.3 Experimental Results and Analysis of UMDAs on DOPs

The experimental results of UMDAs on the dynamic test problems under cyclic, cyclic with noise, and random dynamic environments are plotted in Fig. 1.10 to Fig. 1.12 respectively. The corresponding statistical results of comparing UMDAs by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table 1.4 to Table 1.6 respectively. And the dynamic performance of UMDAs with respect to best-of-generation fitness against generations on the dynamic $OneMax$ problems with $\tau = 10$ and $\rho = 0.5$ is plotted in Fig. 1.13, where the first and last 10 environmental changes are shown and the data were averaged over 50 runs. From the tables and figures, several results can be observed. The observations are similar to the previous observations regarding the experimental results of GAs and will be briefly recapped below. The main concern will be focused on the differences between the performance of UMDAs and GAs.

First, both direct and associative memory schemes significantly improve the performance of UMDAs on most DOPs, see the t-test results regarding DMUMDA − SUMDA and 0.5-AMUMDA − SUMDA. And it seems the memory schemes have a more consistent positive effect on the performance of UMDAs on all DOPs than on the performance of GAs. For example, 0.5-AMUMDA significantly outperforms SUMDA not only on all cyclic DOPs but also on almost all noisy and random DOPs. For example, from the dynamic behaviour of UMDAs shown in Fig. 1.13, it can be see that memory enhanced UMDAs maintain a much higher fitness level than SUMDA not only on cyclic $OneMax$ problem but also on the random $OneMax$ problem. This result indicates that the effect of memory schemes depends not only on the dynamic problems and environments but also on the EA used.

Second, the associative factor α has the similar effect on the performance of AMUMDAs as on the performance of AMGAs. Increasing the value of α

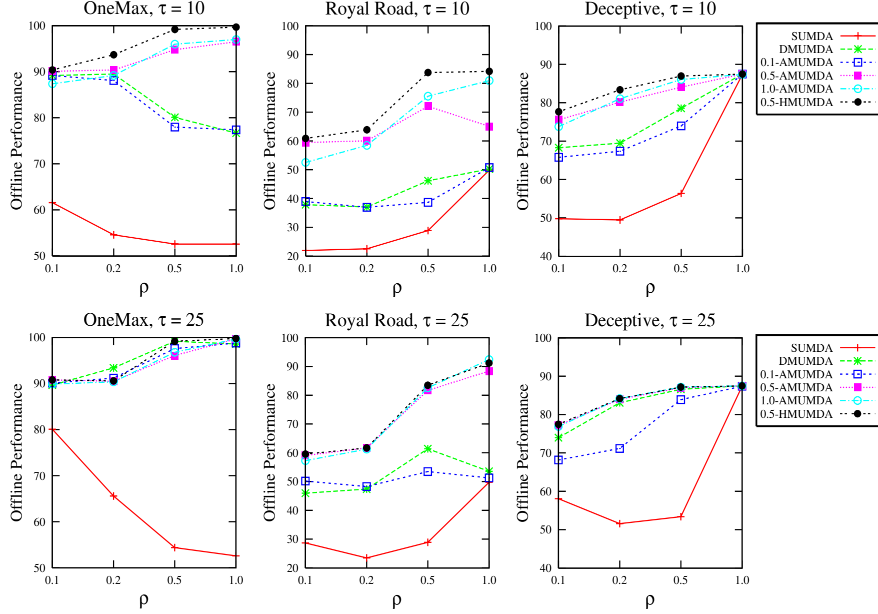

Fig. 1.10. Experimental results of UMDAs on cyclic DOPs

Table 1.4. The t-test results of comparing UMDAs on cyclic DOPs

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	$-$	$s-$	$s-$	$+$	$s+$	$-$	$s-$	$s+$	$s-$	$s-$	$s-$	$=$
0.5-AMUMDA − DMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
1.0-AMUMDA − DMUMDA	$s-$	$-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
0.5-AMUMDA − 0.1-AMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$s-$	$s+$	$+$	$s-$	$s-$	$s+$	$s+$	$s-$	$+$	$s+$	$-$
0.5-HMUMDA − 0.5-AMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
$\tau = 25,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	$s+$	$s-$	$s-$	$s+$	$s+$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.5-AMUMDA − DMUMDA	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
1.0-AMUMDA − DMUMDA	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
0.5-AMUMDA − 0.1-AMUMDA	$s+$	$-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$-$	$+$	$+$	$s-$	$-$	$+$	$s+$	$-$	$+$	$s+$	$+$
0.5-HMUMDA − 0.5-AMUMDA	$+$	$+$	$s+$	$s+$	$+$	$+$	$s+$	$s+$	$+$	$+$	$s+$	$+$

from 0.1 to 0.5 improves the performance of AMUMDA while further raising the value of α to 1.0 degrades the performance of AMUMDA, see the t-test results regarding 0.5-AMUMDA – 0.1-AMUMDA and 1.0-AMUMDA – 0.5-AMUMDA in Table1.4 to Table1.6. This result can also be seen in their dynamic performance shown in Fig. 1.13, where 1.0-AMUMDA achieves a

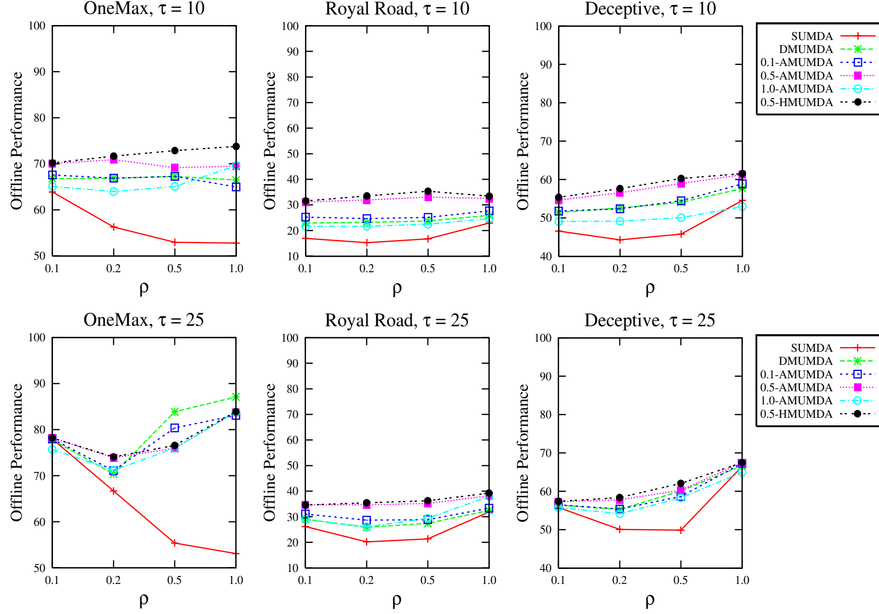

Fig. 1.11. Experimental results of UMDAs on cyclic DOPs with noise

Table 1.5. The t-test results of comparing UMDAs on cyclic DOPs with noise

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	$s+$	$+$	$-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$
0.5-AMUMDA − DMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
1.0-AMUMDA − DMUMDA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.5-AMUMDA − 0.1-AMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$s-$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.5-HMUMDA − 0.5-AMUMDA	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$\tau = 25,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
0.5-AMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s-$	$s+$
0.5-AMUMDA − DMUMDA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$
1.0-AMUMDA − DMUMDA	$s-$	$s+$	$s-$	$s-$	$-$	$+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
0.5-AMUMDA − 0.1-AMUMDA	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$s-$	$-$	$+$	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$	$s-$
0.5-HMUMDA − 0.5-AMUMDA	$-$	$s+$	$s+$	$+$	$-$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$+$

much lower fitness level than 0.5-AMUMDA during each environmental period, especially under cyclic with noise and random environments.

Third, combining the direct memory with the associative memory further improves the performance of AMUMDAs, see the t-test results with respect to 0.5-HMUMDA − 0.5-HMUMDA. This result can be further seen in the

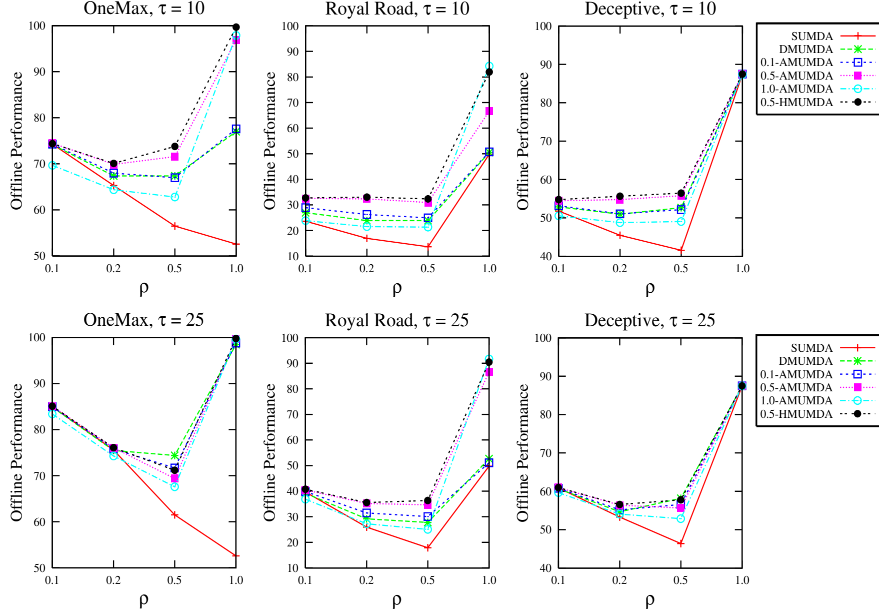

Fig. 1.12. Experimental results of UMDAs on random DOPs

Table 1.6. The t-test results of comparing UMDAs on random DOPs

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
$\tau = 10,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	−	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.5-AMUMDA − SUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	$s+$	$s+$	$s-$	+	$s+$	$s+$	$s+$	+	$s+$	$s+$	$s-$	−
0.5-AMUMDA − DMUMDA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	+
1.0-AMUMDA − DMUMDA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	−
0.5-AMUMDA − 0.1-AMUMDA	+	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	+
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$s-$	$s-$	+	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$
0.5-HMUMDA − 0.5-AMUMDA	−	$s+$	$s+$	$s+$	+	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	−
$\tau = 25,\ \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
DMUMDA − SUMDA	$s-$	−	$s+$	$s+$	−	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$
0.5-AMUMDA − SUMDA	+	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
0.1-AMUMDA − DMUMDA	+	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	+
0.5-AMUMDA − DMUMDA	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	−
1.0-AMUMDA − DMUMDA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	+
0.5-AMUMDA − 0.1-AMUMDA	+	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	−
1.0-AMUMDA − 0.5-AMUMDA	$s-$	$s-$	$s-$	+	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$
0.5-HMUMDA − 0.5-AMUMDA	=	+	$s+$	$s+$	+	$s+$	$s+$	$s+$	+	$s+$	$s+$	−

dynamic performance of 0.5-HMUMDA shown in Fig. 1.13, where 0.5-HMUMDA maintains the highest level of fitness during each environmental period under cyclic, noisy and random environments.

Finally, let's compare the performance of investigated UMDAs and GAs on DOPs. The t-test results of comparing UMDAs and GAs on the dynamic

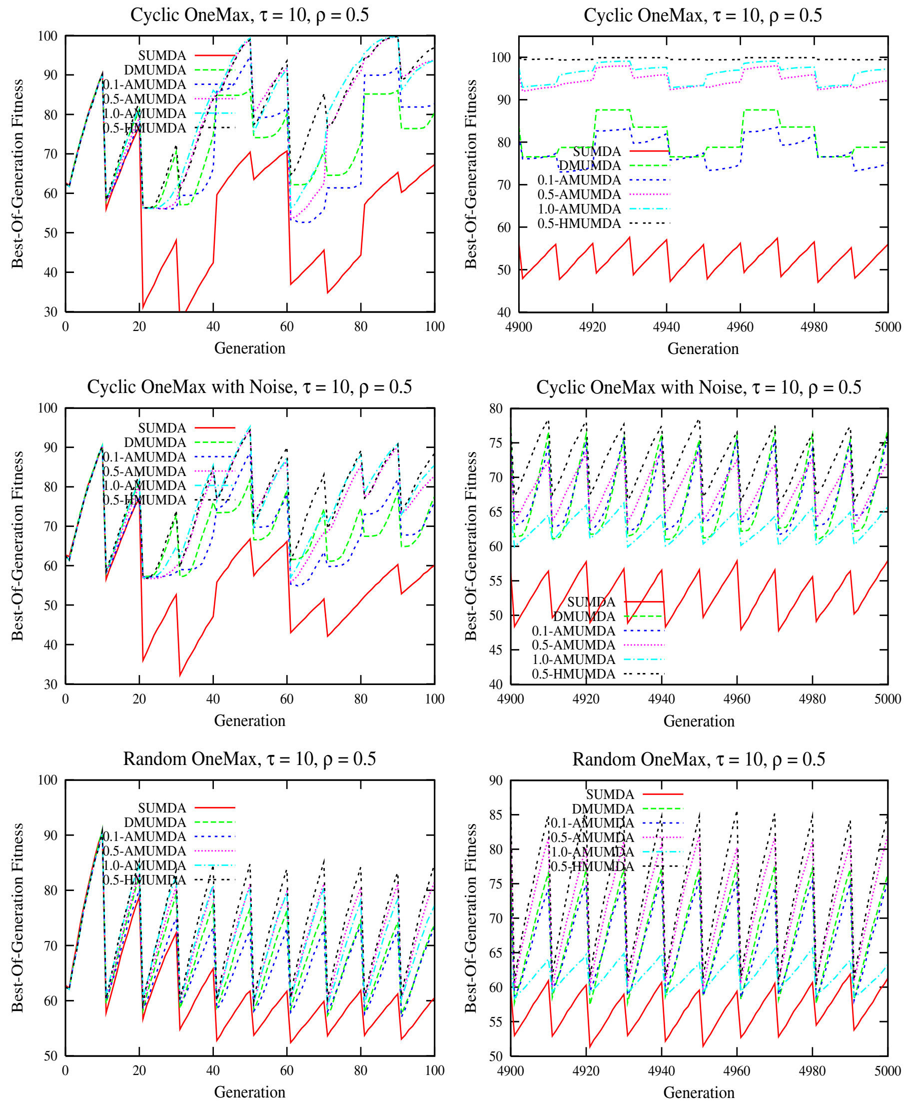

Fig. 1.13. Dynamic performance of UMDAs on the dynamic *OneMax* problems

test problems with $\tau = 10$ are given in Table 1.7. From Table 1.7, it can be seen that GAs outperform corresponding UMDAs on most dynamic test problems.

Table 1.7. The t-test results of comparing UMDAs and GAs on DOPs with $\tau = 10$

t-test Result	*OneMax*				*Royal Road*				*Deceptive*			
Cyclic, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
SUMDA − SGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$
DMUMDA − DMGA	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$
0.1-AMUMDA − 0.1-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s+$	$s+$	$s+$	$s+$
0.5-AMUMDA − 0.5-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$
1.0-AMUMDA − 1.0-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$-$	$s+$
0.5-HMUMDA − 0.5-HMGA	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$
Cyclic with Noise, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
SUMDA − SGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
DMUMDA − DMGA	$s-$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.1-AMUMDA − 0.1-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.5-AMUMDA − 0.5-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$
1.0-AMUMDA − 1.0-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.5-HMUMDA − 0.5-HMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$
Random, $\rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
SUMDA − SGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$
DMUMDA − DMGA	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$+$	$s+$
0.1-AMUMDA − 0.1-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$
0.5-AMUMDA − 0.5-AMGA	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s+$
1.0-AMUMDA − 1.0-AMGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$
0.5-HMUMDA − 0.5-HMGA	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$

1.7 Conclusions

This chapter investigates the application of explicit memory schemes for EAs in dynamic environments. Two kinds of explicit memory schemes, i.e., direct memory and associative memory, are applied into two kinds of EAs, i.e., GAs and UMDAs, to address dynamic optimization problems. The direct memory scheme just stores and reuses best solutions in the memory. In the contrast, in the associative memory scheme, best solutions together with the current environmental information, (the allele distribution vector for GAs or working probability vector for UMDAs) are stored in the memory. When an environmental change is detected, the stored allele distribution vector (for GAs) or probability vector (for UMDAs) that is associated with the best re-evaluated memory solution is extracted to create new individuals into the population.

Based on the XOR dynamic problem generator, a series of dynamic test problems were systematically constructed, featuring three kinds of dynamic environments: cyclic, cyclic with noise, and random. Based on this dynamic test problems, experimental study was carried out to test the memory schemes for GAs and UMDAs. 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 and UMDAs in dynamic environments and the cyclicity of dynamic environments greatly affect the performance of memory schemes for GAs and UMDAs in dynamic environments. Second, generally speaking the associative memory scheme outperforms traditional direct memory scheme for GAs and UMDAs in dynamic environments. Third, the associative factor has an important impact on the performance of AMGAs and AMUMDAs. Setting α to 0.5 seems a good choice for AMGAs

and AMUMDAs. Fourth, combining direct memory with associative memory may further improve the performance of GAs and UMDAs in dynamic environments. The hybrid memory scheme is a good approach for EAs for DOPs.

The work studied in this chapter can be extended in several ways. Developing other memory management and retrieval mechanisms would be an interesting future work for memory-based UMDAs and other estimation of distribution algorithms [1, 18] in dynamic environments. Comparing the investigated explicit memory schemes with implicit memory schemes is another future work. And it is also an interesting work to further investigate the integration of the memory schemes with other approaches, such as multi-population, diversity approaches, and adaptive operators, for EAs in dynamic environments.

References

1. S. Baluja (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *Technical Report CMU-CS-94-163*, Carnegie Mellon University, USA.
2. C. N. Bendtsen and T. Krink (2002). Dynamic memory model for non-stationary optimization. *Proc. of the 2002 Congress on Evol. Comput.*, pp. 145-150.
3. J. Branke (1999). Memory enhanced evolutionary algorithms for changing optimization problems. *Proc. of the 1999 Congress on Evolutionary Computation*, vol. 3, pp. 1875-1882.
4. J. Branke, T. Kaußler, C. Schmidth, and H. Schmeck (2000). A multi-population approach to dynamic optimization problems. *Proc. of the 4th Int. Conf. on Adaptive Computing in Design and Manufacturing*, pp. 299-308.
5. J. Branke (2002). *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers.
6. H. G. Cobb and J. J. Grefenstette (1993). Genetic algorithms for tracking changing environments. *Proc. of the 5th Int. Conf. on Genetic Algorithms*, pp. 523-530.
7. D. Dasgupta and D. McGregor (1992). Nonstationary function optimization using the structured genetic algorithm. *PPSN II*, pp. 145-154.
8. D. E. Goldberg and R. E. Smith (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pp. 59-68.
9. D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.
10. J. J. Grefenstette (1992). Genetic algorithms for changing environments. *Parallel Problem Solving from Nature II*, pp. 137-144.
11. P. Larrañaga and J. A. Lozano (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer Academic Publishers.
12. E. H. J. Lewis and G. Ritchie (1998). 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.
13. S. J. Louis and Z. Xu (1996). Genetic algorithms for open shop scheduling and re-scheduling. *Proc. of the 11th ISCA Int. Conf. on Computers and their Applications*, pp. 99-102.

14. T. Mahnig and H. Mühlenbein (2000). Mathematical analysis of optimization methods using search distributions. *Proc. of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pp. 205-208.
15. N. Mori, H. Kita and Y. Nishikawa (1997). 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.
16. R. W. Morrison and K. A. De Jong (1999). A test problem generator for non-stationary environments. In *Proc. of the 1999 Congress on Evolutionary Computation*, vol. 3, pp. 2047-2053.
17. R. W. Morrison and K. A. De Jong (2000). Triggered hypermutation revisited. In *Proc. of the 2000 Congress on Evolutionary Computation*, pp. 1025-1032.
18. H. Mühlenbein and G. Paaß(1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Proc. of the 4th Int. Conf. on Parallel Problem Solving from Nature*, pp. 178-187.
19. H. Mühlenbein (1998). The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5: 303–346.
20. K. P. Ng and K. C. Wong (1997). A new diploid scheme and dominance change mechanism for non-stationary function optimisation. *Proc. of the 6th Int. Conf. on Genetic Algorithms*.
21. C. L. Ramsey and J. J. Greffenstette (1993). Case-based initializtion of genetic algorithms. *Proc. of the 5th Int. Conf. on Genetic Algorithms*.
22. K. Trojanowski and Z. Michalewicz (1999). Searching for optima in non-stationary environments. *Proc. of the 1999 Congress on Evolutionary Computation*, pp. 1843-1850.
23. F. Vavak and T. C. Fogarty (1996). A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. *AISB Workshop on Evolutionary Computing, LNCS 1143*, pp. 297-304.
24. S. Yang (2003). Non-stationary problem optimization using the primal-dual genetic algorithm. *Proc. of the 2003 Congress on Evolutionary Computation*, vol. 3, pp. 2246-2253.
25. S. Yang (2005). Memory-based immigrants for genetic algorithms in dynamic environments. *Proc. of the 2005 Genetic and Evolutionary Computation Conference*, vol. 2, pp. 1115-1122.
26. S. Yang (2005). Population-based incremental learning with memory scheme for changing environments. *Proc. of the 2005 Genetic and Evolutionary Computation Conference*, vol. 1, pp. 711-718.
27. S. Yang (2005). Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. *Proc. of the 2005 Congress on Evolutionary Computation*, vol. 3, pp. 2560-2567.
28. S. Yang (2006). Associative memory scheme for genetic algorithms in dynamic environments. *Applications of Evolutionary Computing, LNCS 3907*, pp. 788-799.
29. S. Yang and X. Yao (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems, *Soft Computing*, 9(11): 815–834.
30. S. Yang and X. Yao (2006). Population-based incremental learning with associative memory for dynamic environments, submitted to *IEEE Transactions on Evolutionary Computation*.