Memory-Enhanced Univariate Marginal Distribution Algorithms for Dynamic Optimization Problems

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

Abstract- Several approaches have been developed into evolutionary algorithms to deal with dynamic optimization problems, of which memory and random immigrants are two major schemes. This paper investigates the application of a direct memory scheme for univariate marginal distribution algorithms (UMDAs), a class of evolutionary algorithms, for dynamic optimization problems. The interaction between memory and random immigrants for UMDAs in dynamic environments is also investigated. Experimental study shows that the memory scheme is efficient for UMDAs in dynamic environments and that the interactive effect between memory and random immigrants for UMDAs in dynamic environments depends on the dynamic environments.

1 Introduction

Evolutionary algorithms (EAs) have been widely applied to solve stationary optimization problems. In recent years, there has been a growing interest in investigating EAs for dynamic optimization problems. This trend reflects the fact that many real world optimization problems are actually dynamic [1]. For dynamic optimization problems (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. For dynamic problems, the aim of an 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.

In recent years, several approaches have been developed into EAs to address dynamic problems [5], of which memory and random immigrants are two major approaches. The random immigrants approach [7] aims to maintain the population diversity by immigrating random individuals into the population to adapt EAs to dynamic environments.

The basic principle of memory schemes is to store useful information from the current environment and reuse it later in new environments. As reviewed in [4], the 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 of implicit memory schemes are genetic algorithms (GAs) based on multiploidy representations [6, 9, 15] or dualism mechanisms [18, 19]. Explicit memory schemes use precise representations but split an extra memory space to explicitly store useful information, e.g., good solutions, from current generation for reuse in later generations or environments [3, 10, 12, 17].

In this paper, the memory scheme is investigated for the Univariate Marginal Distribution Algorithm (UMDA), which was first introduced by Mühlenbein [14] as a class of EAs, for dynamic optimization problems. An explicit memory scheme is used to improve UMDA's adaptability in dynamic environments. Within this memory scheme, the best samples created by the probability vector are stored in the memory in certain time and space pattern. When the environmental change is detected, the memory points are merged with the current population members in UMDA for further iterations. This paper also investigates the relationship between memory and random immigrants schemes for UMDAs in dynamic environments.

Using the dynamic problem generator proposed in [18, 19, 20], a series of dynamic test problems are constructed from three stationary functions and experiments are carried out to compare the performance of investigated UMDAs and a peer memory-enhanced GA. The experimental results validates the efficiency of the memory scheme for UMDAs in dynamic environments. The experimental results also indicates that the random immigrants scheme has different interactions on the memory scheme for UMDAs in different dynamic environments.

The outline of this paper is given as follows. The next section describes the UMDAs with and without memory schemes and the memory-enhanced GA investigated in this paper. Section 3 presents the dynamic test environment for this study. The experimental results and relevant analysis are presented in Section 4. Section 5 concludes this paper with discussions on relevant future work.

2 Description of Investigated Algorithms

2.1 The Standard UMDA

Mühlenbein [14] introduced the UMDA as the simplest version of Estimation of Distribution Algorithms (EDAs) [13]. Thereafter, there have been several modifications and mathematical analysis of UMDAs [11, 21] and UMDAs have been applied for many optimization problems [8]. In the binary search space, UMDA evolves a probability vector $p(\vec{x},t) = (p(x_1,t),\ldots,p(x_l,t))$ ($\vec{x} = (x_1,\ldots,x_l) \in \{0,1\}^l$) where all the variables are assumed to be independent of each other. The pseudo-code for the UMDA with mutation investigated in this paper, denoted *UMDAm*, is shown in Figure 1.

UMDAm 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

$t := 0$ and $p(\vec{x}, 0) := \vec{0.5}$
repeat
  sample a population S_t of n individuals from $p(\vec{x}, t)$
  if $random\ immigrants\ used$ **then** *// for UMDAi*
    replace $r_i * n$ random immigrants into S_t randomly
  select best $\mu < n$ individuals from S_t to form an interim population D_t
  build $p'(\vec{x}, t)$ according to D_t by Eq. (1)
  mutate $p'(\vec{x}, t)$ by Eq. (2)
  $p(\vec{x}, t+1) := p'(\vec{x}, t)$
until $terminated =$ **true** *// e.g., $t > t_{max}$*

Figure 1: Pseudo-code of the UMDA with mutation (UMDAm) and the UMDA with mutation and random immigrants (UMDAi).

0 on each locus is equal[1]. At iteration t, a population S_t of n individuals are sampled from the probability vector $p(\vec{x}, t)$. The samples are evaluated and an interim population D_t is formed by selecting μ ($\mu < n$) best individuals, denoted $\vec{x}_t^1, \cdots, \vec{x}_t^\mu$, from S_t. Then, the probability vector is updated by extracting statistics information from D_t as follows:

$$p'(\vec{x}, t) := \frac{1}{\mu} \sum_{k=1}^{k=\mu} \vec{x}_t^k \tag{1}$$

After the probability vector is updated according to D_t, in order to keep the diversity of sampling in dynamic environments, a bitwise mutation is applied in this paper. 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 $p(x_i)$ using the following formula:

$$p(x_i) = \begin{cases} p(x_i) * (1.0 - \delta_m), & p(x_i) > 0.5 \\ p(x_i), & p(x_i) = 0.5 \\ p(x_i) * (1.0 - \delta_m) + \delta_m, & p(x_i) < 0.5, \end{cases} \tag{2}$$

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 high evaluation solutions. The search stops when some termination condition becomes true, e.g., the maximum allowable number of iterations t_{max} is reached.

In this paper, we also investigate the effect of random immigrants on the performance of UMDAs in dynamic environments. The pseudo-code for the investigated UMDA with mutation and random immigrants, denoted *UMDAi*, is also shown in Figure 1, where r_i is the ratio of random immigrants to the total population size. Within UMDAi, for each iteration after the probability vector is sampled, a set of individuals are randomly selected and replaced with randomly created individuals (immigrants).

[1]When sampling the probability vector for a solution, for each locus i if a randomly created number $r = rand(0.0, 1.0) < p(x_i, t)$, it is set to 1; otherwise, it is set to 0.

2.2 Memory-Enhanced UMDAs

The memory scheme has proved to be able to enhance EA's performance in dynamic environments. As mentioned in Section 1, the memory scheme works by storing and reusing useful information either implicitly or explicitly. In this paper we focus on explicit memory schemes.

For explicit memory schemes, there are several technical considerations, regarding the content, management and retrieval strategies of the memory. For the first aspect, usually good solutions are stored and reused directly when the environment changes [10]. It is also an interesting policy to store environmental information together with good solutions. When the environment changes, the stored environmental information is used, for example, as the similarity measure [16], to associate the new environment with stored solutions in the memory to be re-activated. For the memory management, usually it has fixed size and when it is full, one memory point is selected to be removed to make room for new ones, i.e., the best individuals from the population. As to selecting which memory point to be updated, there are several memory replacement strategies, e.g., replacing the most similar one if the new individual is better [4]. For 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 (e.g., every generation), or only when the environment changes.

In this paper we investigate an explicit memory scheme for UMDAs in dynamic environments. The pseudo-code for the investigated memory-enhanced UMDAs without and with random immigrants, denoted *MUMDA* and *MUMDAi* respectively, is shown in Figure 2. In Figure 2, n is the number of evaluations per iteration including the memory samples and $f(\vec{x})$ denotes the fitness of individual $\vec{x}$.

In MUMDA and MUMDAi, a memory of size $m = 0.1 * n$ is used to store best samples from the population. The most similar measure, as discussed in [4], is used as the memory replacement strategy. That is, 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 MUMDA and MUMDAi is updated using a dynamic time pattern as follows. After each memory updating, a random integer $R \in [5, 10]$ is generated to determine the next memory updating time t_M. For example, suppose a memory updating happens at generation t, then the next memory updating time is $t_M = t + R = t + rand(5, 10)$. This way, the potential effect that the environmental change period coincides with the memory updating period can be smoothed away.

The memory 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

$t := 0$ and $t_M := rand(5, 10)$
initialize $p(\vec{x}, 0) := \vec{0.5}$ and memory $M(0)$ randomly
repeat
  sample a population S_t of $n-m$ individuals by $p(\vec{x},t)$
  if $random\ immigrants\ used$ **then** // *for MUMDAi*
    replace $r_i * n$ random samples into S_t randomly
  evaluate the population S_t and memory $M(t)$
  if *environmental change detected* **then**
    $S'(t)$:=retrieveBestMembers$(S_t, M(t))$
  else $S'(t) := S_t$
  select best $\mu < n - m$ individuals from S'_t to form an interim population D_t
  build $p'(\vec{x}, t)$ according to D_t by Eq. (1)
  mutate $p'(\vec{x}, t)$ by Eq. (2)
  if $t = t_M$ **then** // *time to update memory*
    $t_M := t + rand(5, 10)$
    denote the best individual in $S'(t)$ by $\vec{x}_t^P$
    if *still any random points in memory* **then**
      replace a random point in memory with $\vec{x}_t^P$
    else find the memory point $\vec{x}_t^M$ closest to $\vec{x}_t^P$
      if $f(\vec{x}_t^P) > f(\vec{x}_t^M)$ **then** $\vec{x}_t^M := \vec{x}_t^P$
  $p(\vec{x}, t + 1) := p'(\vec{x}, t)$
until $terminated = \mathbf{true}$ // *e.g.,* $t > t_{max}$

Figure 2: Pseudo-code of the memory-enhanced UMDAs: without random immigrants (MUMDA) and with random immigrants (MUMDAi).

the current population to form an intermit population. If no environmental change is detected, MUMDA and MUMDAi progress just as the standard UMDA does.

2.3 Memory-Enhanced Genetic Algorithm

As for stationary problems, GAs are the most studied EAs for dynamic environments. In this study a memory-enhanced GA with random immigrants, denoted *MEGAi*, is taken as a peer EA to compare the performance of UMDAs for dynamic problems. Figure 3 shows the pseudo-code of MEGAi. MEGAi has typical configuration of standard GAs as follows: generational, uniform crossover, bit flip mutation, fitness proportionate selection with the elitist scheme.

MEGAi uses a memory of size $m = 0.1 * n$, where n is the total number of individuals evaluated in each generation. The memory is randomly initialized and, as in MUMDA and MUMDAi, is updated in a dynamic time pattern with the most similar memory updating strategy [4]. When the memory is due to update, if any 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 memory is re-evaluated every generation. If an environmental 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 normal genetic operations for a new population while the memory remains unchanged.

$t := 0$ and $t_M := rand(5, 10)$
initialize population $P(0)$ and memory $M(0)$ randomly
repeat
  evaluate population $P(t)$ and memory $M(t)$
  if *environmental change detected* **then**
    $P'(t)$:=retrieveBestMembers$(P(t), M(t))$
  else $P'(t) := P(t)$
  if $t = t_M$ **then** // *time to update memory*
    $t_M := t + rand(5, 10)$
    denote the best individual in $P'(t)$ by $\vec{x}_t^P$
    if *still any random points in memory* **then**
      replace a random point in memory with $\vec{x}_t^P$
    else find the memory point $\vec{x}_t^M$ closest to $\vec{x}_t^P$
      if $f(\vec{x}_t^P) > f(\vec{x}_t^M)$ **then** $\vec{x}_t^M := \vec{x}_t^P$
  // normal genetic operation
  $P'(t) :=$ 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 $r_i * n$ random immigrants into $P'(t)$ randomly
  replace elite from $P(t-1)$ into $P'(t)$ randomly
  $P(t+1) := P'(t)$
until $terminated = \mathbf{true}$ // *e.g.,* $t > t_{max}$

Figure 3: Pseudo-code for the memory-enhanced GA with random immigrants (MEGAi).

3 Dynamic Test Environments

The DOP generator proposed in [18, 19] can construct *random* dynamic environments from any binary-encoded stationary function $f(\vec{x})$ ($\vec{x} \in \{0,1\}^l$) by a bitwise exclusive-or (XOR) operator. Suppose the environment changes every τ generations. For each environmental period k, an XORing mask $\vec{M}(k)$ is incrementally generated as follows:

$$\vec{M}(k) = \vec{M}(k-1) \oplus \vec{T}(k), \quad (3)$$

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

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(k)), \quad (4)$$

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 *cyclical* dynamic environments as follows (see [20] for a formal description). First, we can generate $2K$ XORing masks $\vec{M}(0), \vec{M}(1), \cdots, \vec{M}(2K-1)$ 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:

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(I_t)) = f(\vec{x} \oplus \vec{M}(k\%(2K))), \quad (5)$$

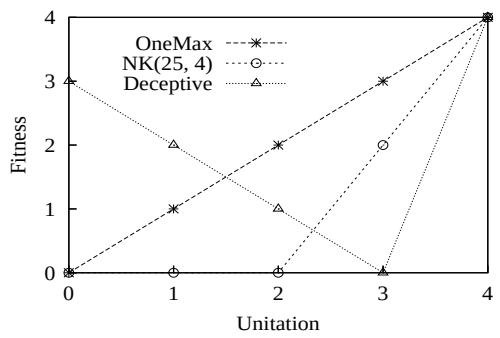

Figure 4: Building block of the three stationary functions.

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

$$\vec{M}(i+1) = \vec{M}(i) \oplus \vec{T}(i\%K), i = 0, \cdots, 2K-1 \quad (6)$$

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

In this paper, three 100-bit binary functions, denoted $OneMax$, $NK(25, 4)$ and $Deceptive$ respectively, are selected as base stationary functions to construct dynamic test environments. They all consist of 25 contiguous 4-bit building blocks (BBs) and have an optimum fitness of 100. As shown in Figure 4, the BB for each function is defined based on the unitation function, i.e., the number of ones inside the BB. The BB for $OneMax$ is just a $OneMax$ sub-function, which aims to maximize the number of ones in a chromosome. The BB for $NK(25, 4)$ contributes 4 (or 2) to the total fitness if its unitation is 4 (or 3), otherwise, it contributes 0. The BB for $Deceptive$ is fully deceptive. These three stationary functions have increasing difficulty for EAs in the order from $OneMax$ to $NK(25, 4)$ to $Deceptive$.

Two kinds of dynamic environments, random and cyclical, are constructed from each of the three base functions using the aforementioned dynamic problem generator. For kind of dynamic environments, the landscape is periodically changed every τ generations during the 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 24 DOPs, 3 values of τ combined with 4 values of ρ under two kinds of dynamic environments, are constructed from each stationary function.

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

4 Experimental Study

4.1 Experimental Design

Experiments were carried out to compare the performance of algorithms on the dynamic test environments. For all UMDAs, the parameters are set as follows: total sample size $n = 100$ (including memory samples $m = 10$ if used), $\mu = 0.5 * n$ or $0.5 * (n - m)$ (if memory is used), and the mutation probability $p_m = 0.02$ with the mutation shift $\delta_m = 0.05$. For MEGAi, parameters are set as: crossover probability $p_c = 0.6$, mutation probability $p_m = 0.01$, elitist size 1, and population size $n = 100$ including the memory size $m = 10$. If random immigrants are used, r_i is set to 0.2.

For each experiment of an algorithm on a dynamic problem, 20 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 an algorithm on a problem is defined as:

$$\overline{F}_{BOG} = \frac{1}{G} \sum_{i=1}^{G} (\frac{1}{N} \sum_{j=1}^{N} F_{BOG_{ij}}), \quad (7)$$

where $G = 5000$ is the total number of generations for a run, $N = 20$ 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 20 runs and then averaged over the data gathering period.

4.2 Experimental Results and Analysis

The experimental results of algorithms on the test problems under cyclical and random dynamic environments are plotted in Figure 5 and Figure 6 respectively. The corresponding statistical results of comparing algorithms by one-tailed t-test with 38 degrees of freedom at a 0.05 level of significance are given in Table 1 and Table 2 respectively. In Table 1 and 2, 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.

In order to better understand the performance of investigated algorithms in dynamic environments, the dynamic behaviour of algorithms with respect to best-of-generation fitness against generations on the dynamic test problems with $\tau = 25$ and $\rho = 0.2$ is plotted in Figure 7. In Figure 7, the last 10 environmental changes (i.e., 250 generations) are shown and the data were averaged over 20 runs. From the tables and figures several results can be observed.

First, both the memory-enhanced UMDAs, MUMDA and MUMDAi, perform significantly better than UMDAm on most dynamic test problems. This result validates the

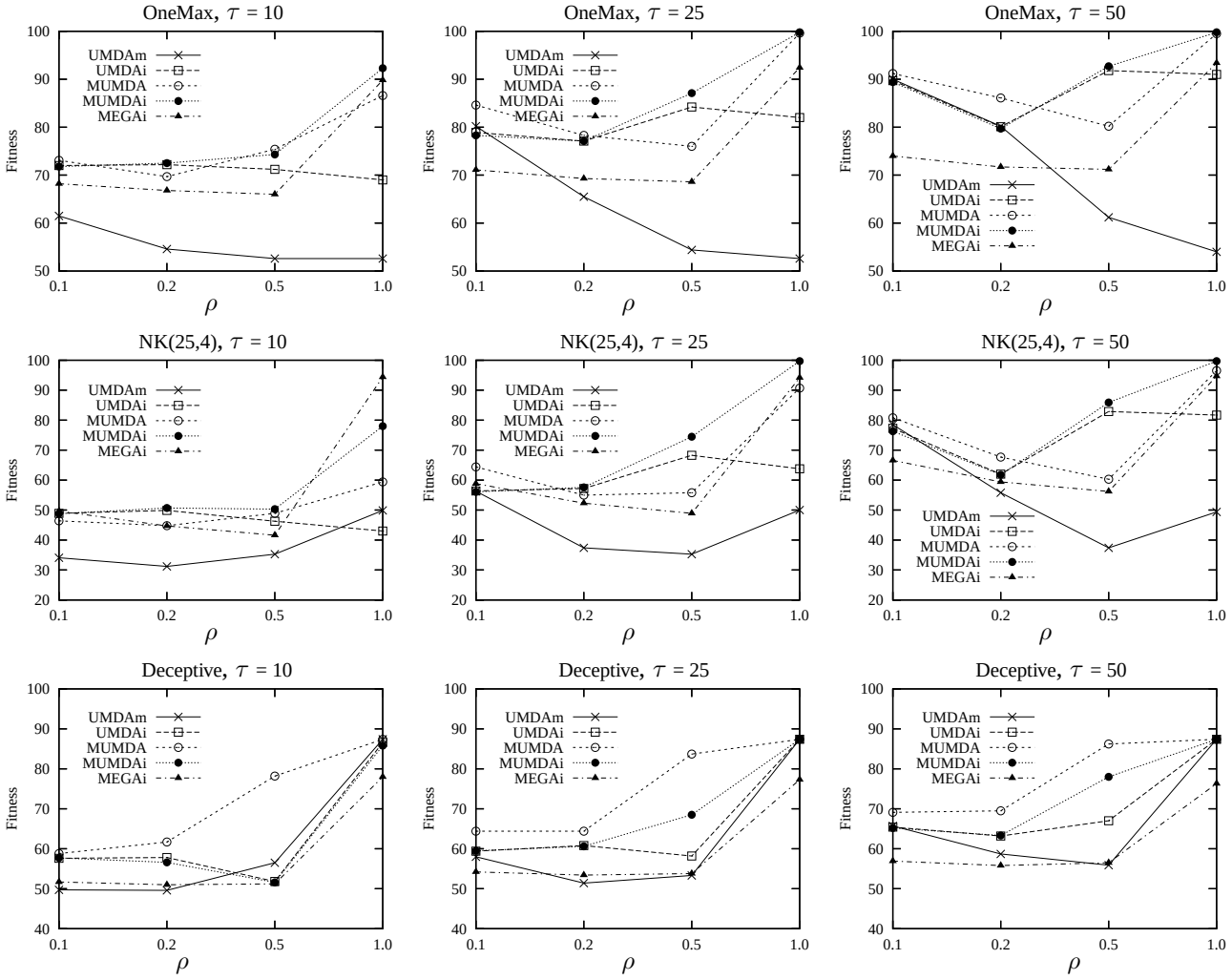

Figure 5: Experimental results of UMDAm, UMDAi, MUMDA, MUMDAi, and MEGAi on cyclical dynamic problems.

Table 1: The t-test results of comparing investigated algorithms on cyclical dynamic environments.

t-test Result	$OneMax$				$NK(25,4)$				$Deceptive$			
$\tau = 10, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$
$MUMDA - UMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
$MUMDA - UMDAi$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$MUMDAi - MUMDA$	$s-$	$s+$	$s-$	$s+$	$s+$	$s+$	$+$	$s+$	$s-$	$s-$	$s-$	$s-$
$UMDA - MEGAi$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
$\tau = 25, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s-$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
$MUMDA - UMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
$MUMDA - UMDAi$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$
$MUMDAi - MUMDA$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
$UMDA - MEGAi$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$-$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$\tau = 50, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s-$	$-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
$MUMDA - UMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
$MUMDA - UMDAi$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$
$MUMDAi - MUMDA$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$
$UMDA - MEGAi$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s+$	$-$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

efficiency of introducing the memory scheme into UMDA. From Figure 7, it can be seen that the performance of MUMDA and MUMDAi stays at a much higher fitness level than UMDAm. And both MUMDA and MUMDAi achieve

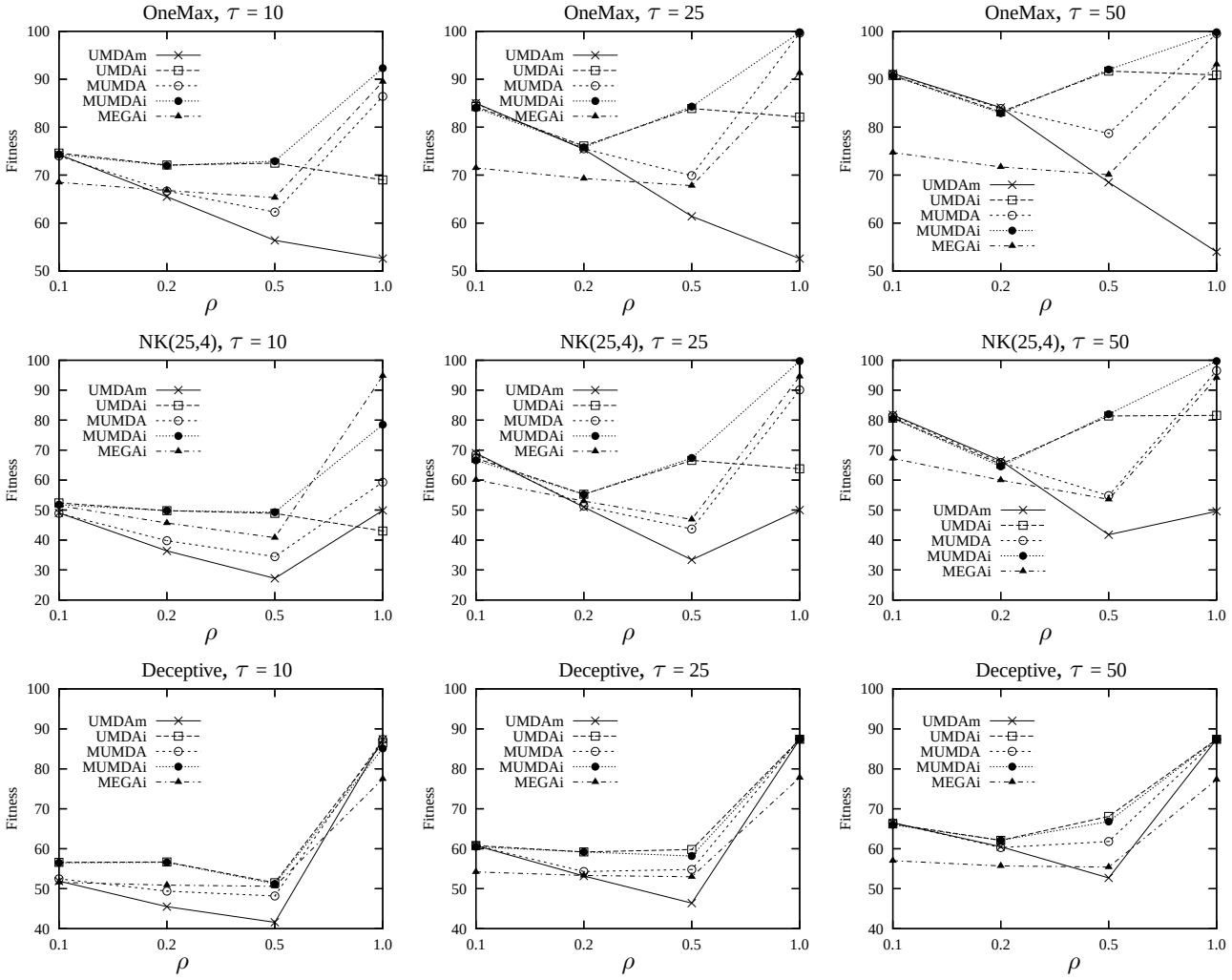

Figure 6: Experimental results of UMDAm, UMDAi, MUMDA, MUMDAi, and MEGAi on random dynamic problems.

Table 2: The t-test results of comparing investigated algorithms on random dynamic environments.

t-test Result	$OneMax$				$NK(25,4)$				$Deceptive$			
$\tau = 10, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
$MUMDA - UMDA$	$s-$	$s+$	$s+$	$s+$	$-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$
$MUMDA - UMDAi$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$
$MUMDAi - MUMDA$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
$UMDA - MEGAi$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
$\tau = 25, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s-$
$MUMDA - UMDA$	$-$	$+$	$s+$	$s+$	$-$	$+$	$s+$	$s+$	$+$	$s+$	$s+$	$+$
$MUMDA - UMDAi$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$+$	$s-$	$s-$	$s+$
$MUMDAi - MUMDA$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
$UMDA - MEGAi$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$-$	$s-$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$\tau = 50, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
$UMDAi - UMDA$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
$MUMDA - UMDA$	$-$	$s-$	$s+$	$s+$	$-$	$s-$	$s+$	$s+$	$-$	$s-$	$s+$	$+$
$MUMDA - UMDAi$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$
$MUMDAi - MUMDA$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$
$UMDA - MEGAi$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$
$MUMDAi - MEGAi$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

better performance improvement over UMDAm on cyclical environments than on random dynamic environments. This result means that the effect of the memory scheme depends on whether the environment changes cyclically or not.

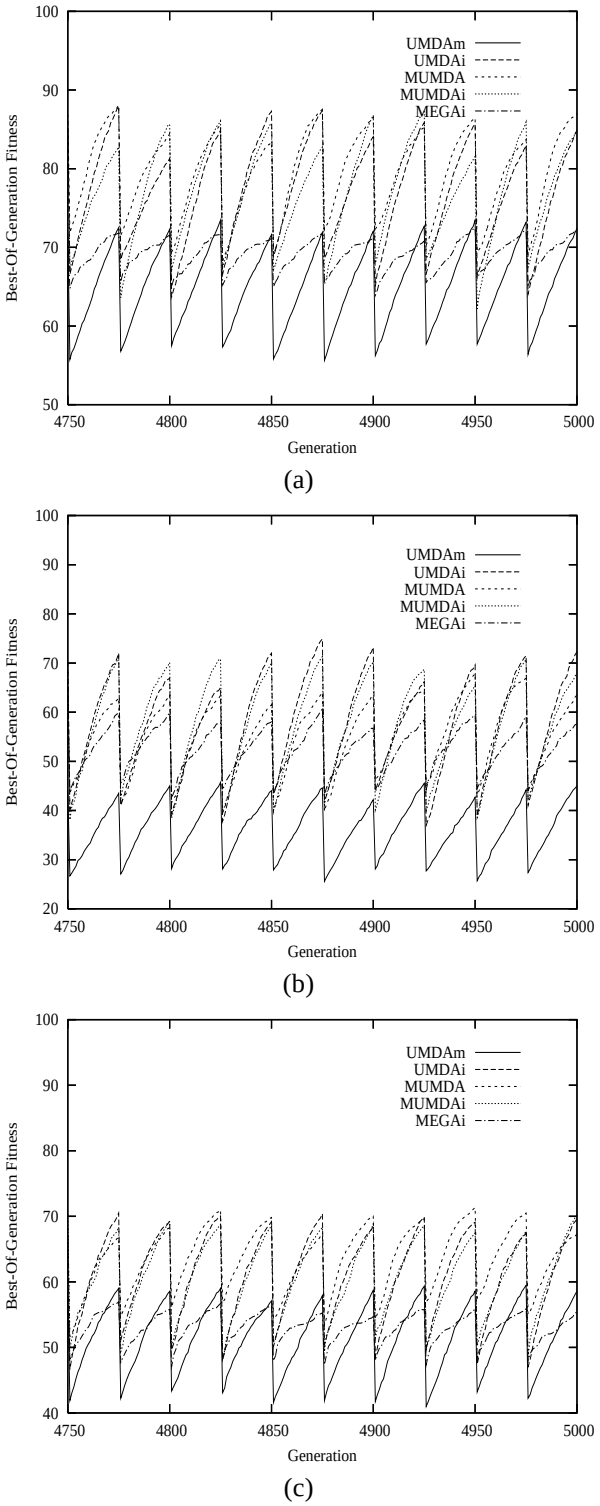

Figure 7: Dynamic behaviour of algorithms on cyclical dynamic (a) $OneMax$, (b) $NK(25,4)$ and (c) $Deceptive$ functions with $\tau = 25$ and $\rho = 0.2$.

Second, the addition of the random immigrants scheme improves the performance of UMDA on most dynamic problems, see the t-test results regarding UMDAi – UMDAm. This also validates the benefit of maintaining the diversity for UMDA in dynamic environments. However, when ρ is small, e.g., 0.1, for several cases the random immigrants scheme has negative effect on the performance of UMDAm. This happens because when the severity of environmental changes is low, the random immigrants added slow down the continuous search progress of original UMDAs when change occurs.

Third, when comparing the memory scheme and the random immigrants scheme, it can be seen that the effect of the memory scheme is significantly greater (better) than the random immigrants scheme on most cyclical dynamic problems, see the t-test results regarding MUMDA – UMDAi in Table 1. However, for random dynamic problems, the random immigrants scheme outperforms the memory scheme on most cases. This happens because for random DOPs, random immigrants may track the new environment more efficient than memory samples.

Fourth, when examining the interactive effect between the memory scheme and the random immigrants scheme, it can be seen that MUMDAi outperforms UMDAi on most cyclical and random dynamic problems. This means when the random immigrants scheme is used, the addition of the memory scheme always has positive effect on UMDA's performance. However, MUMDAi is beaten by MUMDA for many cyclical problems. That is, when the memory scheme is used, the addition of the random immigrants may have negative effect in cyclical dynamic environments.

Finally, comparing the performance of MEGAi with UMDAs, it can be seen that MEGAi outperforms UMDAm under many dynamic environments, see the t-test results regarding UMDAm – MEGAi. However, MEGAi is significantly beaten by both MUMDA and MUMDAi on most dynamic test problems, see the relevant t-test results. This happens due to two factors. The first factor lies in that UMDAs have better search capacity than MEGAi and this factor contributes to the fact that even UMDA outperforms MEGAi on several dynamic test problems. This point can be seen from Figure 7. On almost all dynamic problems UMDAs achieve a much higher fitness improvement than MEGAi does during each environmental period.

The second factor is because that the memory scheme in MUMDA and MUMDAi has a stronger effect than the memory scheme in MEGAi. This can be clearly seen in the dynamic behaviour of algorithms in Figure 7. On cyclical dynamic problems MUMDA and MUMDAi are able to maintain a higher fitness level than MEGAi does.

5 Conclusions and Future Work

This paper investigates the application of the memory scheme for UMDAs in dynamic environments. Within this memory scheme, the best samples are stored in the memory by replacing the most similar memory point. When the environmental change is detected, the memory is merged with the population to build new probability vectors. This paper also investigates the relationship between memory and random immigrants for UMDAs in dynamic environments. An experimental study was carried out based on a series of systematically constructed dynamic test environments. From the experimental results, several conclusions can be drawn on the dynamic test environments.

First, the memory scheme is efficient to improve the performance of UMDAs in dynamic environments. Second, the experimental results indicate that different interaction exists between the random immigrants scheme and the memory scheme. In general, when random immigrants is used for UMDA, the addition of the memory scheme has a positive effect on UMDA's performance in dynamic environments. Random immigrants improves the performance of UMDA when no memory is used. However, when memory is used for UMDA, random immigrants have a negative effect on UMDA's performance in the cyclical dynamic test environments. Third, the memory-enhanced UMDAs seem a better choice than the memory-enhanced GA for the dynamic test problems.

The work studied in this paper 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 [2, 13] in dynamic environments. Comparing the memory scheme investigated in this paper with the associative memory scheme studied in [20] is another future work. And it is also an interesting work to further investigate the interactions between the memory scheme and other approaches, such as multi-population and adaptive operators, for UMDAs in dynamic environments.

Bibliography

[1] T. Bäck. On the behaviour of evolutionary algorithms in dynamic fitness landscape. *Proc. of the 1998 IEEE Int. Conf. on Evol. Comput.*, pp. 446-451, 1998.

[2] S. Baluja. 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, 1994.

[3] C. N. Bendtsen and T. Krink. Dynamic memory model for non-stationary optimization. *Proc. of the 2002 Congress on Evol. Comput.*, pp. 145-150, 2002.

[4] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. *Proc. of the 1999 Congress on Evol. Comput.*, vol. 3, pp. 1875-1882, 1999.

[5] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2002.

[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] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[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] S. J. Louis and Z. Xu. Genetic algorithms for open shop scheduling and re-scheduling. *Proc. of the 11th ISCA Int. Conf. on Computers and their Applications*, pp. 99-102, 1996.

[11] T. Mahnig and H. Mühlenbein. Mathematical analysis of optimization methods using search distributions. *Proc. of the 2000 Genetic and Evol. Comput. Conference Workshop Program*, pp. 205-208, 2000.

[12] 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.

[13] H. Mühlenbein and G. Paaß. 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, 1996.

[14] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, vol. 5, pp. 303-346, 1998.

[15] 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.

[16] C. L. Ramsey and J. J. Greffenstette. Case-based initializtion of genetic algorithms. *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 1993.

[17] K. Trojanowski and Z. Michalewicz. Searching for optima in non-stationary environments. *Proc. of the 1999 Congress on Evol. Comput.*, pp. 1843-1850, 1999.

[18] S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm. *Proc. of the 2003 Congress on Evol. Comput.*, vol. 3, pp. 2246-2253, 2003.

[19] S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, published online first on 22 April 2005.

[20] S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. Submitted to *IEEE Trans. on Evol. Comput.*, 2005.

[21] Q. Zhang. On stability of fixed points of limit models of univariate marginal distribution algorithm and factorized distribution algorithm. *IEEE Trans. on Evol. Comput.*, vol. 8, no. 1, pp. 80-93, 2004.