Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments

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

ABSTRACT

Investigating and enhancing the performance of genetic algorithms in dynamic environments have attracted a growing interest from the community of genetic algorithms in recent years. This trend reflects the fact that many real world problems are actually dynamic, which poses serious challenge to traditional genetic algorithms. Several approaches have been developed into genetic algorithms for dynamic optimization problems. Among these approches, random immigrants and memory schemes have shown to be beneficial in many dynamic problems. This paper proposes a hybrid memory and random immigrants scheme for genetic algorithms in dynamic environments. In the hybrid scheme, the best solution in memory is retrieved and acts as the base to create random immigrants to replace the worst individuals in the population. In this way, not only can diversity be maintained but it is done more efficiently to adapt the genetic algorithm to the changing environment. The experimental results based on a series of systematically constructed dynamic problems show that the proposed memory-based immigrants scheme efficiently improves the performance of genetic algorithms in dynamic environments.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms (including Monte Carlo)*

General Terms

Algorithms, Experimentation, Performance

Keywords

Genetic Algorithms, Dynamic Optimization Problems, Memory Scheme, Random Immigrants Scheme

1. INTRODUCTION

Genetic algorithms (GAs) have been widely used for and mainly focused on stationary optimization problems. However, in recent years investigating the performance of GAs in dynamic environments has attracted a growing interest from GA's community because many real world problems are actually dynamic. In these dynamic optimization problems (DOPs), the problem-specific fitness evaluation function and constraints of the problem, such as design variables and environmental conditions, may change over time. This poses serious challenges to traditional GAs due to the convergence problem because once converged GAs cannot adapt well to the changing environment. In recent years, several approaches have been developed into GAs to address dynamic optimization problems [3], such as maintaining diversity during the run via random immigrants [8, 15], increasing diversity after a change [5], using memory schemes to reuse old good solutions [2], and multi-population approaches [4].

Among the approaches developed for GAs for DOPs, random immigrants and memory schemes have proved to be beneficial for many DOPs. Random immigrants schemes aim to maintain the diversity of the population by replacing worst or randomly selected individuals from the population with randomly created individuals. Memory schemes work by implicitly using redundant representation or explicitly storing good (usually best of population) solutions in certain time pattern during the run in an extra memory and reusing them when the environment changes. In this paper, a hybrid random immigrants and memory approach, called *memory-based immigrants*, is proposed and investigate for GAs in dynamic environments. In the hybrid approach, the best solution in the memory is retrieved and acts as the base to create random immigrants to replace the worst individuals in the current population. In this way, not only can diversity be maintained but it is done more efficiently to adapt the GA to the changing environment.

Based on the dynamic problem generator proposed in [16, 17], a series of dynamic test problems are constructed from several stationary functions and experimental study is carried out to compare the performance of several GA variants with random immigrants and memory schemes. Based on the experimental results, we carry out algorithm performance analysis regarding the weakness and strength of random immigrants and memory schemes for GAs in dynamic environments. The experiment results indicate that the proposed memory-based immigrants scheme efficiently improves the performance of GAs in dynamic environments.

The rest of this paper is outlined as follows. The next section briefly reviews random immigrants and memory approaches for GAs in dynamic environments and presents the GAs studied in this paper as peer GAs. Section 3 presents

GECCO'05, June 25–29, 2005, Washington, DC, USA.

```
begin
  t := 0
  initialize population P(0) with n random solutions
  evaluate population P(0)
  repeat
    P'(t) := selectForReproduction(P'(t))
    crossover(P'(t), p_c)
    mutate(P'(t), p_m)
    replace elite from P(t-1) into P'(t) randomly
    evaluate the interim population P'(t)

    if random immigrants used then    // for RIGA
      replace r_i × n worst individuals in P'(t) with
        random immigrants
      evaluate the random immigrants

    P(t+1) := P'(t)
  until terminated = true          // e.g., t > t_max
end
```

Figure 1: Pseudo-code for the standard GA (SGA) and the GA with random immigrants (RIGA).

the proposed memory-based immigrants scheme for GAs in dynamic environments. Section 4 describes the dynamic test environments for this study. The experimental results and relevant analysis are presented in Section 5. Section 6 concludes this paper with discussions on relevant future work.

2. RANDOM IMMIGRANTS AND MEMORY SCHEMES

2.1 Random Immigrants Based GA

The standard GA (SGA) maintains and evolves a population of candidate solutions through selection and variation, as shown in Figure 1. New populations are generated by first probabilistically selecting relatively fitter individuals from the current population and then performing crossover and mutation on them to create new off-springs. This process continues until some stop condition becomes true, e.g., the maximum allowable number of generations t_{max} is reached. Usually, with the iteration of the GA, individuals in the population will eventually converge to optimum solutions 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 usually becomes a big problem for GAs in dynamic environments because changing environments usually require GAs to keep certain population diversity level to maintain their adaptability. The random immigrants approach is a quite natural and simple way around the convergence problem [5]. It maintains the diversity level of the population through substituting some individuals of the current population with random individuals every generation. As to which individuals in the population should be substituted, usually there are two strategies: replacing random individuals or replacing the worst ones [15]. In order to avoid that random immigrants disrupt the ongoing search progress too much, especially during the period when the environment does not change, the ratio r_i of the number of random immigrants to the population size n is usually set to a small value, e.g., 0.1.

The pseudo-code for the standard GA with random immigrants investigated in this paper, denoted *RIGA*, is also shown in Figure 1, where random immigrants replace worst individuals in the population, p_c is the crossover probability, and p_m is the mutation probability.

2.2 Memory-Enhanced GA

While the random immigrants approach uses random individuals to maintain the population diversity level to adapt GAs to changing environments, the memory approach aims to enhance GA's performance in dynamic environments in a different way. The memory approach works by storing useful information from the current environment, either implicitly through redundant representations [6, 7, 9, 12] or explicitly by storing good (usually best) solutions of the current population in an extra memory space [1, 10, 11, 14]. And the stored information can be reused later in new environments. For example, for the explicit memory scheme, when the environment changes, old solutions in the memory that well fit the new environment will be reactivated and hence may adapt the GA to the new environment more directly than random immigrants would do. Especially, when the environment changes periodically, the memory approach can work very well because 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 the GA to the reappeared environment [2].

For explicit memory schemes, which are the concern of this paper, there are several technical considerations, regarding the content, the management and the 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 [13], to associate the new environment with stored good solutions in the memory to be re-activated.

For the memory management, since the memory space is usually limited, it is necessary to update memory solutions, when it is full, to make room for new ones. A general strategy is to select one memory point to be removed for the best individual from the population or to be moved toward it [1]. As to which memory point should be selected for updating, there are several memory replacement strategies, e.g., replacing the least important one with respect to the age, contribution to diversity and fitness, replacing the one with least contribution to memory variance, replacing the most similar one if the new individual is better, or replacing the less fit one of a pair of memory points that have the minimum distance among all pairs [2]. For the memory retrieval, a natural strategy is to use the best individual(s) in the memory to replace the worst individual(s) in the population. This can be done periodically (e.g., every generation), or only when the environment changes.

The GA with the memory scheme investigated in this paper, called *memory-enhanced GA* (MEGA), is shown in Figure 2, where $f(X)$ is the fitness of individual X. MEGA (and other memory based GAs studied in this paper) 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 initial-

```
begin
  t := 0 and t_M := rand(5, 10)
  initialize population P(0) and memory M(0) randomly
  evaluate population P(0)
  repeat
    evaluate memory M(t)
    if environmental change detected then
      P'(t) := retrieveBestMembersFrom(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 B_P(t)
      if still any random point in memory then
        replace a random point in memory with B_P(t)
      else
        find the memory point C_M(t) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t)) then
          replace C_M(t) with B_P(t)

    // normal genetic operation
    P'(t) := selectForReproduction(P'(t))
    crossover(P'(t), p_c)
    mutate(P'(t), p_m)
    replace elite from P(t − 1) into P'(t) randomly
    evaluate the interim population P'(t)

    if random immigrants used then    // for MRIGA
      replace r_i × n worst individuals in P'(t) with
        random immigrants
      evaluate the random immigrants

    P(t + 1) := P'(t)
  until terminated = true          // e.g., t > t_max
end
```

Figure 2: Pseudo-code for the memory-enhanced GA (MEGA) and the GA with memory and random immigrants schemes (MRIGA).

ized 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 (i.e., the most similar memory updating strategy [2]). And instead of updating the memory in a fixed time interval as in other memory-based GAs in the literature, the memory is updated in a dynamic time pattern as follows. After each memory updating, a random integer $R \in [5, 10]$ is generated to decide 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 (e.g., the memory is updated whenever the environment changes) is smoothed away. The memory is re-evaluated every generation to detect environmental changes[1]. If an environment change is detected, the memory is merged with the old population and the best $n - m$ individuals are selected as an interim population to undergo normal genetic operations for a new population while the memory remains unchanged.

[1]The environment is detected as changed if at least one individual in the memory has been detected changed its fitness.

```
begin
  t := 0 and t_M := rand(5, 10)
  initialize population P(0) and memory M(0) randomly
  evaluate population P(0)
  repeat
    evaluate memory M(t)
    if t = t_M then        // time to update memory
      t_M := t + rand(5, 10)
      denote the best individual in P(t) by B_P(t)
      if still any random point in memory then
        replace a random point in memory with B_P(t)
      else
        find the memory point C_M(t) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t)) then
          replace C_M(t) with B_P(t)

    // normal genetic operation
    P'(t) := selectForReproduction(P'(t))
    crossover(P'(t), p_c)
    mutate(P'(t), p_m)
    replace elite from P(t − 1) into P'(t) randomly
    evaluate the interim population P'(t)

    // perform memory-based immigration
    denote the best point in M(t) by B_M(t)
    P_I(t) := mutateBestMemoryPoint(B_M(t), r_i × n, p_m^i)
    replace r_i × n worst individuals in P'(t) with
      the memory-based immigrants in P_I(t)
    evaluate the memory-based immigrants

    P(t + 1) := P'(t)
  until terminated = true          // e.g., t > t_max
end
```

Figure 3: Pseudo-code for the GA with memory-based immigrants (MIGA).

2.3 GA with Memory + Random Immigrants

It is straightforward that the above discussed random immigrants and memory approaches can be combined into GAs to deal with DOPs. The GA investigated in this paper, which combines memory and random immigrants schemes, is also shown in Figure 2, denoted *MRIGA*. MRIGA differs from MEGA only in that before entering next generation, $r_i \times n$ random immigrants are swapped into the population.

3. MEMORY-BASED IMMIGRANTS GA

As discussed in the above section, the random immigrants approach aims to improve GA's performance in dynamic environments through maintaining the population diversity level with random immigrants and the memory approach aims to move the GA directly to an old environment that is similar to the new one through reusing old good solutions. These two approaches can be simply combined into GAs as in the MRIGA. In this paper we propose a more efficient memory and random immigrants hybrid approach, called *memory-based immigrants*, for GAs to deal with dynamic optimization problems. The pseudo-code for the GA with the proposed memory-based immigrants scheme, denoted *MIGA*, is shown in Figure 3.

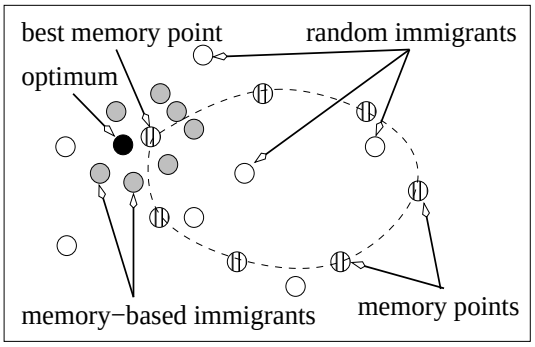

Figure 4: Illustration of immigrants schemes in GAs.

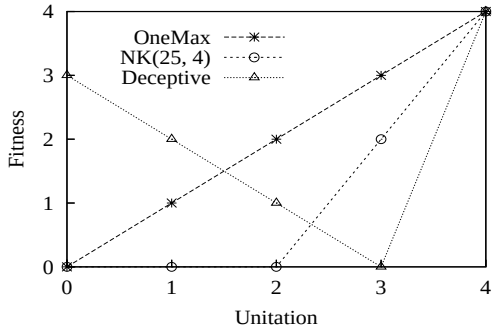

Figure 5: Building block of the stationary functions.

Comparing Figure 2 with Figure 3, it can be seen that MIGA uses the same memory updating mechanism as MEGA and MRIGA. However, the memory retrieval does not depend on the detection of environmental changes and is hybridized into the random immigrants scheme together with the mutation mechanism. For every generation, the memory is reevaluated and the best memory point $B_M(t)$ is retrieved as the base for creating immigrants. From $B_M(t)$, a set $P_I(t)$ of $r_i \times n$ individuals are iteratively generated by performing general bit flip mutation with a probability p_m^i on $B_M(t)$. The generated individuals then act as immigrants and replace the worst $r_i \times n$ individuals in the population.

The key idea behind MIGA is that the memory is used to guide the immigrants to make them more biased to the current environment, be it a new one or not, than random immigrants. This is illustrated in Figure 4. For the random immigrants approach, immigrants are distributed over the whole search space while for the memory-based immigrants approach, immigrants are distributed around the best memory point and hence more precisely around the optimum of the current environment. This bias is expected to further improve GA's performance in dynamic environments.

4. DYNAMIC TEST ENVIRONMENTS

The dynamic problem generator used in [16] can construct 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 is changed every τ generations. For each environmental period k, an XORing mask $\vec{M}(k)$ is incrementally generated as follows:

$$\vec{M}(k) = \vec{M}(k-1) \oplus \vec{T}(k) \tag{1}$$

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:

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(k)) \tag{2}$$

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 value of ρ means severer environmental change and greater challenge to GAs.

In this paper, in order to compare GAs in dynamic environments, three 100-bit binary-encoded stationary functions, denoted $OneMax$, $NK(25,4)$ and $Deceptive$ respectively, are selected. They all consist of 25 contiguous 4-bit building blocks (BBs) and have an optimum fitness of 100. As shown in Figure 5, 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 $OneMax$ subfunction, 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 GAs in the order from $OneMax$ to $NK(25,4)$ to $Deceptive$.

Dynamic test environments are constructed from the three stationary functions using above dynamic problem generator. The landscape is periodically changed every τ generations during the run of GAs. In order to compare the performance of GAs in different dynamic environments, the generator parameters τ is set to 10, 50 and 100 and ρ is set to 0.1, 0.2, 0.5, and 0.9 respectively. Totally, a series of 12 DOPs, 3 values of τ combined with 4 values of ρ, are constructed from each stationary function.

5. EXPERIMENTAL STUDY

5.1 Experimental Design

Experiments were carried out to compare different GAs on above constructed dynamic environments. For all GAs, generators and parameters are set as follows: generational, uniform crossover with $p_c = 0.6$, flip mutation with $p_m = 0.01$, and fitness proportionate selection with the stochastic universal sampling scheme and elitism of size 1. The population size n, including memory size if memory is used, is set to 100 and the memory size is $m = 0.1 * n = 10$ if used. For RIGA, MRIGA and MIGA, the ratio of immigrants r_i is set to 0.1 and for MIGA the probability p_m^i of bitwisely mutating the best memory point for immigrants is 0.01.

For each experiment of an algorithm on a dynamic test problem, 20 independent runs were executed with the same set of random seeds. And for each run 100 environmental changes were allowed and the best-of-generation fitness was recorded every generation. The overall performance of an algorithm on a problem is formulated as below:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N} F_{BOG_{ij}}) \tag{3}$$

where $G = 100 * \tau$ is the total number of generations for

Table 1: Experimental results with respect to overall performance of algorithms.

Performance	$OneMax$				$NK(25,4)$				$Deceptive$			
$\tau = 10$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
SGA	74.7	70.4	65.3	63.3	60.4	51.0	40.3	37.4	54.9	52.4	51.0	54.3
$RIGA$	75.5	71.4	66.5	63.9	62.2	52.5	42.3	38.6	53.6	52.1	51.3	52.5
$MEGA$	75.9	70.8	65.4	63.3	60.8	50.7	40.2	37.6	55.6	52.6	50.9	54.8
$MRIGA$	76.5	71.7	66.5	64.2	62.3	52.3	42.2	39.2	54.0	52.2	51.2	52.8
$MIGA$	78.9	70.6	64.8	64.0	60.0	47.9	39.6	38.7	59.3	54.2	51.7	57.7
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
SGA	83.5	80.1	73.3	67.5	77.7	70.8	57.6	47.5	65.4	61.7	58.5	66.0
$RIGA$	83.7	80.5	75.4	73.7	81.4	73.8	61.3	61.0	62.6	59.1	57.4	61.5
$MEGA$	84.0	80.4	73.5	71.7	77.7	70.8	57.9	63.5	65.9	61.9	58.5	66.2
$MRIGA$	84.2	80.7	75.5	75.0	82.0	74.0	61.5	69.4	63.2	59.1	57.4	62.0
$MIGA$	94.4	88.1	78.0	87.1	89.0	76.9	59.8	75.7	74.3	68.5	64.7	77.2
$\tau = 100$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
SGA	86.5	83.8	78.5	73.3	82.5	78.0	68.2	58.2	70.0	66.4	63.8	70.9
$RIGA$	86.7	84.1	80.4	79.4	86.7	81.7	73.2	73.8	67.9	64.1	61.6	68.2
$MEGA$	86.9	84.1	78.8	79.3	82.7	78.0	68.5	75.4	70.2	66.7	63.9	71.1
$MRIGA$	87.1	84.3	80.5	80.8	87.3	82.1	73.6	80.7	68.3	64.1	61.8	68.6
$MIGA$	97.5	94.0	86.5	94.0	94.8	87.8	73.0	88.1	77.1	74.1	72.0	83.1

Table 2: The t-test results of comparing algorithms on dynamic test problems.

t-test Result	$OneMax$				$NK(25,4)$				$Deceptive$			
$\tau = 10$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
$RIGA - SGA$	+	+	+	+	+	+	+	+	−	−	+	−
$MEGA - SGA$	+	+	∼	∼	∼	∼	−	+	+	+	−	+
$MRIGA - RIGA$	+	+	∼	+	∼	∼	∼	+	+	∼	∼	+
$MRIGA - MEGA$	+	+	+	+	+	+	+	+	−	−	+	−
$MIGA - RIGA$	+	−	−	∼	−	−	−	∼	+	+	+	+
$MIGA - MEGA$	+	−	−	+	−	−	−	+	+	+	+	+
$MIGA - MRIGA$	+	−	−	−	−	−	−	−	+	+	+	+
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
$RIGA - SGA$	+	+	+	+	+	+	+	+	−	−	−	−
$MEGA - SGA$	+	+	+	+	∼	∼	+	+	+	+	∼	+
$MRIGA - RIGA$	+	+	+	+	+	+	+	+	+	∼	∼	+
$MRIGA - MEGA$	+	+	+	+	+	+	+	+	−	−	−	−
$MIGA - RIGA$	+	+	+	+	+	+	−	+	+	+	+	+
$MIGA - MEGA$	+	+	+	+	+	+	+	+	+	+	+	+
$MIGA - MRIGA$	+	+	+	+	+	+	−	+	+	+	+	+
$\tau = 100$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
$RIGA - SGA$	+	+	+	+	+	+	+	+	−	−	−	−
$MEGA - SGA$	+	+	+	+	+	∼	+	+	∼	∼	+	+
$MRIGA - RIGA$	+	+	+	+	+	+	+	+	+	∼	∼	+
$MRIGA - MEGA$	+	+	+	+	+	+	+	+	−	−	−	−
$MIGA - RIGA$	+	+	+	+	+	+	−	+	+	+	+	+
$MIGA - MEGA$	+	+	+	+	+	+	+	+	+	+	+	+
$MIGA - MRIGA$	+	+	+	+	+	+	−	+	+	+	+	+

a run, $N = 20$ is the total runs, $F_{BOG_{ij}}$ is the best-of-generation fitness of generation i of run j, and $\overline{F}_{BOG}$ is the offline performance, i.e., the best-of-generation fitness averaged over the 20 runs and over the data gathering period.

5.2 Experimental Results and Analysis

The experimental results of GAs on the dynamic problems are presented in Table 1. The statistical results of comparing GAs by one-tailed t-test with 38 degrees of freedom at a 0.05 level of significance are given in Table 2. In Table 2, the t-test result regarding Alg. 1 − Alg. 2 is shown as "+", "−", or "∼" when Alg. 1 is significantly better than, significantly worse than, or statistically equivalent to Alg. 2 respectively. The results are also plotted in Figure 6 to Figure 8 for the three series of dynamic problems respectively. The dynamic behaviour of GAs for the first 10 environmental periods is plotted with respect to best-of-generation fitness against generation on the dynamic problems with $\tau = 50$ and $\rho = 0.1$, 0.5, and 0.9 in Figure 9 to Figure 11 respectively, where the data were averaged over 20 runs. From the tables and figures several results can be observed.

First, a prominent result is that MIGA significantly out-

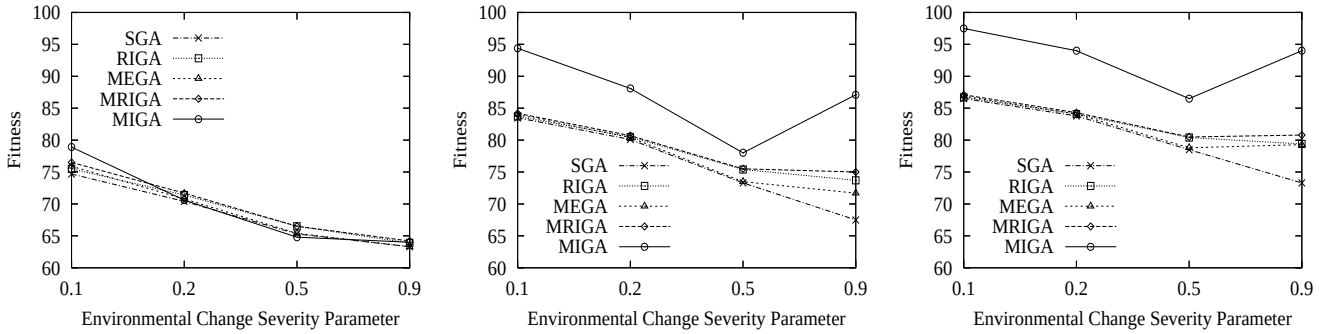

Figure 6: Results on dynamic $OneMax$ functions with different ρ and (a) $\tau = 10$, (b) $\tau = 50$, and (c) $\tau = 100$.

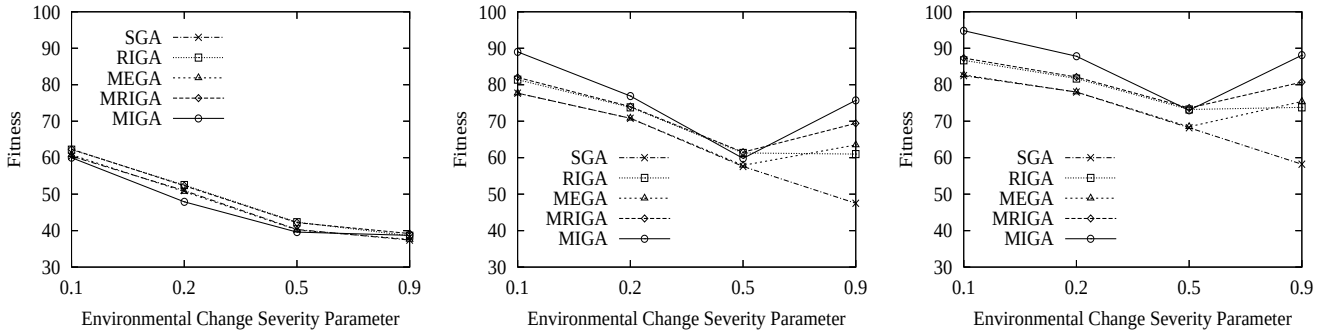

Figure 7: Results on dynamic $NK(25, 4)$ functions with different ρ and (a) $\tau = 10$, (b) $\tau = 50$, and (c) $\tau = 100$.

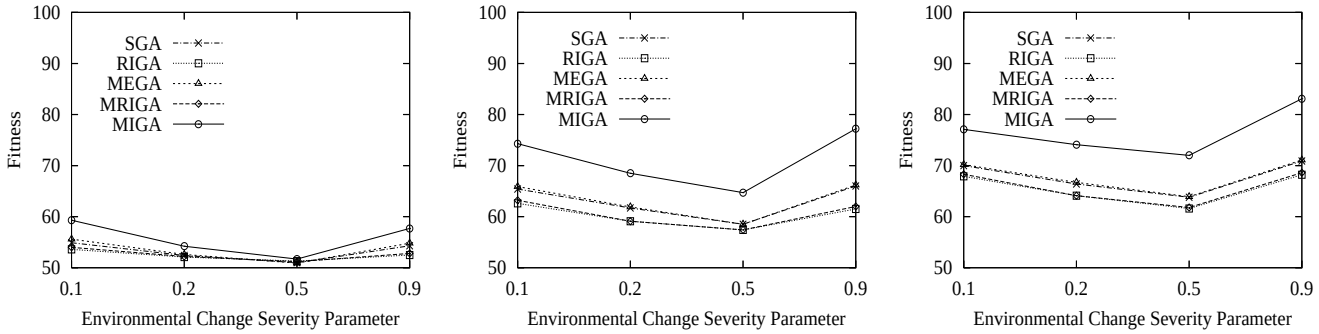

Figure 8: Results on dynamic $Deceptive$ functions with different ρ and (a) $\tau = 10$, (b) $\tau = 50$, and (c) $\tau = 100$.

performs other GAs on most dynamic test problems, see the t-test results regarding $MIGA - RIGA$, $MIGA - MEGA$, and $MIGA - MRIGA$ in Table 2. This result validates our expectation of the memory-based immigrants scheme for GAs in dynamic environments. When $\tau = 10$, MIGA is beaten by other GAs on some dynamic $OneMax$ and $NK(25, 4)$ functions. The reason is that when the environment changes quickly, the best memory point may not be able to track the optimum of the current environment and hence biases the immigrants to not well fit area.

Second, traditional memory scheme improves GA's performance in most cases and consistently over the three series of dynamic problems, see the t-test results regarding $MEGA - SGA$ and $MRIGA - RIGA$ in Table 2. In contrast, the random immigrants scheme has different effect on different problems. On all non-deceptive dynamic test environments, i.e., dynamic $OneMax$ and $NK(25, 4)$ functions, RIGA and MRIGA outperform their peers, SGA and MEGA, respectively. In these environments, the diversity introduced by the random immigrants really improves GA's adaptability. However, on dynamic *Deceptive* functions RIGA and MRIGA are outperformed by SGA and MEGA respectively, see the t-test results in Table 2. This is because random immigrants can easily break global optimal building blocks existing in the deceptive functions via crossover.

From Figure 6 to 8, it can be seen another difference between the memory and random immigrants schemes lies in that the random immigrants scheme has much stronger effect on GA's performance than the memory scheme. For example, $|\overline{F}_{BOG}(RIGA) - \overline{F}_{BOG}(SGA)|$ is much larger than

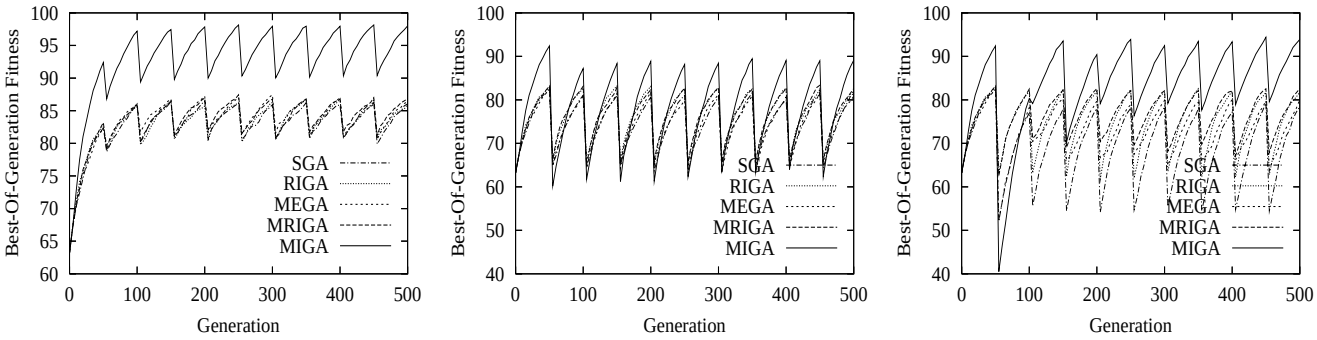

Figure 9: Dynamic behaviour of GAs on dynamic $OneMax$ functions with $\tau = 50$ and (a) $\rho = 0.1$, (b) $\rho = 0.5$, and (c) $\rho = 0.9$.

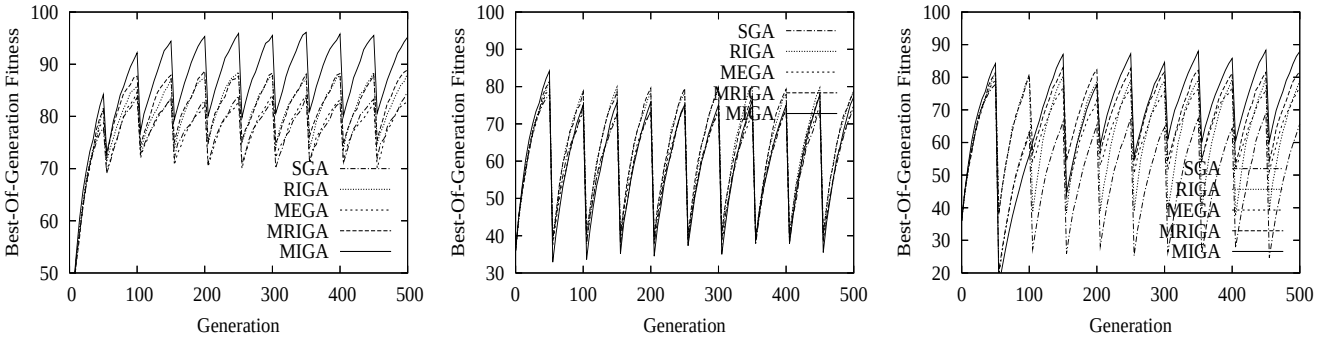

Figure 10: Dynamic behaviour of GAs on dynamic $NK(25, 4)$ functions with $\tau = 50$ and (a) $\rho = 0.1$, (b) $\rho = 0.5$, and (c) $\rho = 0.9$.

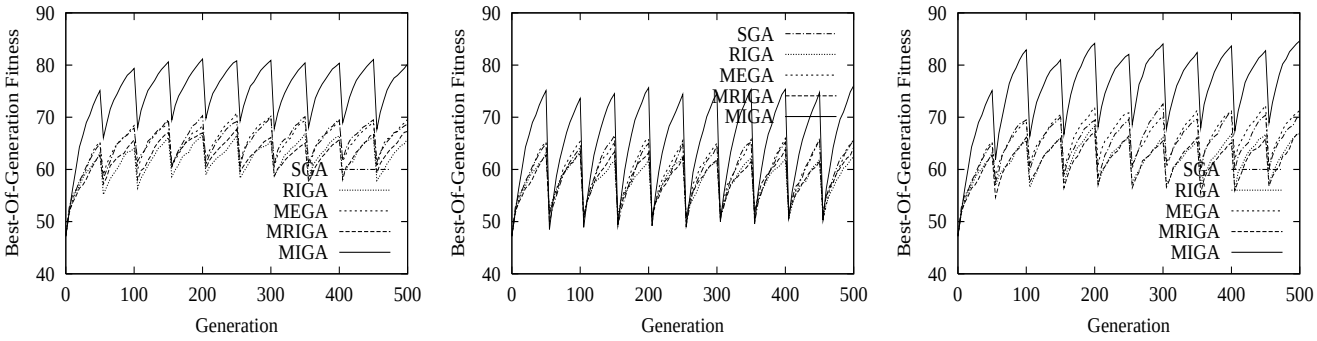

Figure 11: Dynamic behaviour of GAs on dynamic $Deceptive$ functions with $\tau = 50$ and (a) $\rho = 0.1$, (b) $\rho = 0.5$, and (c) $\rho = 0.9$.

$|\overline{F}_{BOG}(MEGA) - \overline{F}_{BOG}(SGA)|$. The memory-based immigrants approach combines the two aspects: the strong effect of random immigrants and the consistent performance of memory. This leads to MIGA's good performance.

Third, for each fixed value of ρ, when τ increases, the performance of all GAs increases. This is natural. When the environment changes slower, GAs get more time to reach higher fitness level before next change. And, also naturally, for a fixed τ, when the severity of environmental change increases, i.e., when ρ increases, the performance of GAs decreases. This is because bigger ρ leads to sharper fitness dropping when change occurs, e.g., compare GAs' dynamic behaviour in Figure 9(a), 10(a) and 11(a) with Figure 9(b), 10(b) and 11(b) respectively. When $\rho = 0.5$, each time the environment changes all GAs are put into a totally random environment and hence their behaviour during dynamic environmental periods is just like that for the first stationary period, see Figure 9(b), 10(b) and 11(b). When $\rho = 0.9$, for several situations GAs with memory perform better than when $\rho = 0.5$. This is because when $\rho = 0.9$ memory points stored at environmental period k may be quite efficiently reused at period $k + 2$, i.e., every other period. On dynamic

deceptive functions, GAs without memory also perform better when $\rho = 0.9$ than $\rho = 0.5$. This is because the sort of switching between global optimum and deceptive optimum within the building blocks makes the fitness drop less sharp when change occurs, compare Figure 11(b) and 11(c).

Finally, viewing from Figure 6 down to Figure 8 and Table 1 it can be seen that the increasing difficulty of stationary $OneMax$ and $NK(25, 4)$ functions is also reflected in corresponding dynamic functions. That is, with the same τ and ρ, GAs perform worse on dynamic $NK(25, 4)$ functions than on dynamic $OneMax$ functions. However, the existence of deceptivity in deceptive functions makes things different. In several cases GAs perform better on dynamic $Deceptive$ functions than on dynamic $NK(25, 4)$ functions, especially when the environment changes severely.

6. CONCLUSIONS AND FUTURE WORK

Random immigrants and memory schemes are two major approaches developed into GAs to address dynamic optimization problems. This paper proposes a memory-based immigrants scheme for GAs in dynamic environments, where the memory (in fact, best memory point) is used to bias the immigrants toward the current environment. This way, the diversity introduced by the immigrants becomes more adapted and hence more efficient for the current environment, be it a new one or not. From the experiment results on a series of systematically constructed dynamic problems, we can draw following conclusions.

First, random immigrants are beneficial for non-deceptive dynamic functions but may deteriorate GA's performance in dynamic deceptive functions. Second, traditional memory scheme has more consistent positive but maybe less strong effect than the random immigrants scheme on GA's performance for DOPs. Third, the proposed memory-based immigrants scheme combines the working principles of the memory and random immigrants approaches and hence improves GA's performance in dynamic environments. Fourth, the difficulty level of non-deceptive stationary functions is reflected in corresponding dynamic problems for GAs.

As relevant future work, it would be interesting to investigate the effect of the memory-based immigrants and other memory schemes for GAs in periodically returning dynamic environments, where an old environment will return exactly in the search space after certain changes. Usually, the memory scheme is expected to be more beneficial in these dynamic environments. Another interesting future work would be developing other memory organization and retrieval mechanisms in order to make the memory direct the immigrants more efficiently, e.g., in rapidly changing environments.

7. REFERENCES

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

[2] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proc. of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1875–1882, 1999.

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

[4] J. Branke, T. Kaußler, C. Schmidth, and H. Schmeck. A multi-population approach to dynamic optimization problems. In *Proc. of the Adaptive Computing in Design and Manufacturing*, pages 299–308, 2000.

[5] H. G. Cobb and J. J. Grefenstette. Genetic algorithms for tracking changing environments. In S. Forrest, editor, *Proc. of the 5th Int. Conf. on Genetic Algorithms*, pages 523–530, 1993.

[6] D. Dasgupta and D. McGregor. Nonstationary function optimization using the structured genetic algorithm. In R. Männer and B. Manderick, editors, *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, pages 145–154, 1992.

[7] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pages 59–68, 1987.

[8] J. J. Grefenstette. Genetic algorithms for changing environments. In R. Männer and B. Manderick, editors, *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, pages 137–144, 1992.

[9] E. H. J. Lewis and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature*, pages 139–148, 1998.

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

[11] H. K. N. Mori and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In *Proc. of the 7th Int. Conf. on Genetic Algorithms*, pages 299–306. Morgan Kaufmann Publishers, 1997.

[12] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1997.

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

[14] K. Trojanowski and Z. Michalewicz. Searching for optima in non-stationary environments. In *Proc. of the 1999 Congress on Evolutionary Computation*, pages 1843–1850, 1999.

[15] F. Vavak and T. C. Fogarty. A comparative study of steady state and generational genetic algorithms for use in nonstationary environments. In T. C. Fogarty, editor, *AISB Workshop on Evolutionary Computing, Lecture Notes in Computer Science*, volume 1143, pages 297–304. Springer, 1996.

[16] S. Yang. Non-stationary problem optimization using the primal-dual genetic algorithm, In *Proc. of the 2003 Congress on Evol. Comput.*, Vol. 3, pages 2246–2253, 2003.

[17] S. Yang and X. Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 2005, to be published.