Open access • Journal Article • DOI:10.1162/EVCO.2008.16.3.385

Genetic algorithms with memory-and elitism-based immigrants in dynamic environments — Source link

Shengxiang Yang

Institutions: University of Leicester

Published on: 01 Sep 2008 - Evolutionary Computation (MIT Press.)

Topics: Genetic algorithm and Population

Genetic Algorithms with Memory- and Elitism-Based Immigrants in Dynamic Environments

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

Abstract

In recent years the genetic algorithm community has shown a growing interest in studying dynamic optimization problems. Several approaches have been devised. The random immigrants and memory schemes are two major ones. The random immigrants scheme addresses dynamic environments by maintaining the population diversity while the memory scheme aims to adapt genetic algorithms quickly to new environments by reusing historical information. This paper investigates a hybrid memory and random immigrants scheme, called memory-based immigrants, and a hybrid elitism and random immigrants scheme, called elitism-based immigrants, for genetic algorithms in dynamic environments. In these schemes, the best individual from memory or the elite from the previous generation is retrieved as the base to create immigrants into the population by mutation. This way, not only can diversity be maintained but it is done more efficiently to adapt genetic algorithms to the current environment. Based on a series of systematically constructed dynamic problems, experiments are carried out to compare genetic algorithms with the memory-based and elitism-based immigrants schemes against genetic algorithms with traditional memory and random immigrants schemes and a hybrid memory and multi-population scheme. The sensitivity analysis regarding some key parameters is also carried out. Experimental results show that the memory-based and elitism-based immigrants schemes efficiently improve the performance of genetic algorithms in dynamic environments.

Keywords

Genetic algorithms, dynamic optimization problems, memory, random immigrants, memory-based immigrants, elitism-based immigrants.

1 Introduction

Traditionally, the research and application of genetic algorithms (GAs) have been mainly focused on stationary optimization problems (Holland, 1975; Goldberg, 1989). However, a significant part of real world problems are in fact dynamic optimization problems (DOPs) (Branke, 2002). For DOPs, the evaluation function and/or problem-specific constraints, such as design variables and environmental conditions, may change over time. Changes may occur due to many factors, such as the arrival of stochastic tasks, machine faults, climatic modification, or economic and financial changes. DOPs pose serious challenges to traditional GAs due to the convergence problem. Once converged, GAs cannot adapt well to the changing environment.

In recent years investigating the performance of GAs in dynamic environments has attracted a growing interest from the GA community. One simple way to deal with DOPs is to regard the problem as a new one when a change occurs and restart GAs

from scratch. However, it could be useful in terms of computational cost to utilize the information obtained in the current solutions to find new good solutions in the newly changed environment. This is usually true if the new problem is closely related to the old one. Over the years, several approaches have been developed for GAs to address dynamic environments (Branke, 2002; Morrison, 2004; Weicker, 2003), such as maintaining diversity during the run via random immigrants (Grefenstette, 1992; Vavak and Fogarty, 1996), increasing diversity after a change (Cobb and Grefenstette, 1993; Morrison and De Jong, 2000), using memory schemes to reuse stored useful information (Branke, 1999; Trojanowski and Michalewicz, 1999, 2000; Yang, 2005a, b), and multi-population approaches (Branke et al., 2000). Among these approaches, 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 solutions, usually the best ones of the population, regularly during the run in an extra memory and reusing them when the environment changes.

Recently, a hybrid random immigrant and memory approach, called *memory-based immigrants*, and a hybrid random immigrants and elitism approach, called *elitism-based immigrants*, have been proposed for GAs in dynamic environments with some preliminary experiments and promising results in Yang (2005a, 2007). In the memory-based immigrants scheme, the best solution in the memory is retrieved as the base to create random immigrants to replace the worst individuals in the current population. In the elitism-based immigrants scheme, the elite from the previous generation is retrieved as the base to create random immigrants to replace the worst individuals in the current population. This way, for both schemes, not only can diversity be maintained, but it is done more efficiently to adapt GAs to the changing environment.

This paper further investigates the performance of the memory-based immigrants and elitism-based immigrants schemes for GAs in dynamic environments. In Yang (2005a) and Yang (2007), the memory-based immigrants and elitism-based immigrants schemes were tested based on a set of random dynamic test problems constructed by the DOP generator proposed (Yang, 2003; Yang and Yao, 2005) from some stationary functions. This paper extends the experiments to a series of cyclic, cyclic with noise, and random dynamic test environments, which are systematically constructed from several stationary functions using the generalized DOP generator proposed (Yang and Yao, 2008). Experiments are carried out based on these constructed DOPs to compare the performance of GAs with the memory-based immigrants and elitism-based immigrants schemes against GAs with traditional random immigrants and memory schemes and a hybrid memory and multi-population scheme (Branke, 1999). Based on the experimental results, an algorithm performance analysis with respect to the weakness and strength of random immigrants, memory, memory-based immigrants, and elitism-based immigrants schemes for GAs in dynamic environments was carried out. This paper also carries out experiments on sensitivity analysis with respect to several important parameters, such as the ratio of immigrants and the degree of environmental noise, on the performance of the investigated GAs. The effect of traditional memory and random immigrants schemes for GAs in dynamic environments is also investigated in this paper.

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

```
t := 0 and initialize population P(0) randomly
repeat
  evaluate population P(t)
  replace the worst individual in P(t) by the elite E(t − 1) from P(t − 1)
  if random immigrants used then    // for RIGA
    replace the worst r_i * n individuals in P(t) by random individuals
    evaluate the random immigrants
  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
  P(t + 1) := P'(t)
until the termination condition is met    // e.g., t > t_max
```

Figure 1: Pseudocode for the standard GA (SGA) and the GA with random immigrants (RIGA). Here, elitism of size one is used in both SGA and RIGA.

immigrants schemes for GAs in dynamic environments, previously proposed in Yang (2005b) and Yang (2007), respectively. Section 4 describes the generalized DOP generator proposed in Yang and Yao (2008) and the dynamic test environments for this study. The experimental results and analysis, including the sensitivity analysis of relevant parameters, are presented in Section 5. Section 6 concludes this paper with a discussion on future work.

2 Random Immigrants and Memory Schemes

2.1 GAs with Random Immigrants

The standard GA maintains and evolves a population of candidate solutions via 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 offspring. This process continues until some termination condition becomes true, for example, the maximum allowable number of generations t_{max} is reached. The pseudocode for the standard GA, henceforth denoted *SGA*, is shown in Figure 1, where p_c and p_m are the crossover and mutation probabilities, respectively. The elite in SGA (and other GAs studied in this paper) is updated every generation. That is, the best individual generated in generation t is taken as the elite at generation t, no matter whether it is better than the elite at generation $t-1$ or not.

Usually, with the iteration of SGA, individuals in the population will eventually converge to the optimal solution(s) in stationary environments due to the selection pressure. Convergence at a proper pace, instead of premature, may be beneficial and in fact is expected for GAs 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 change occurs, it is hard for GAs to adapt to the new environment. Hence, changing environments require GAs to

keep a certain level of population diversity or reintroduce diversity after a change is detected to maintain their adaptability.

The random immigrants scheme is a quite simple and natural method to address the convergence problem (Cobb and Grefenstette, 1993). It maintains the diversity level of the population through substituting a portion of individuals in the current population with random individuals (immigrants) every generation. As to which individuals in the population should be replaced, there are two strategies: replacing random individuals or replacing the worst ones (Vavak and Fogarty, 1996). In order to avoid the problem 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, for example, 0.2.

The pseudocode of the GA with random immigrants, denoted *RIGA* in this paper, is also shown in Figure 1. RIGA differs from SGA only in that after the evaluation of the population, $r_i * n$ worst individuals in the population are replaced by random immigrants.

2.2 Memory-Enhanced GAs

While the random immigrants scheme uses random individuals to maintain the population diversity to adapt GAs to changing environments, the memory scheme aims to enhance GA's performance for DOPs in a different way. It works by storing useful information from the current environment, either implicitly through redundant representations (Dasgupta and McGregor, 1992; Goldberg and Smith, 1987; Lewis and Ritchie, 1998; Ng and Wong, 1995; Uyar and Harmanci, 2005) or explicitly by storing good (usually best) solutions of the current population in an extra memory (Branke, 1999; Louis and Xu, 1996; Mori and Nishikawa, 1997). 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 fit the new environment well will be reactivated and hence may adapt GAs to the new environment more directly than random immigrants would do. Especially, when the environment changes cyclically, memory can work very well. This is because in cyclic dynamic environments, with time moving forward, the environment will return to some old environment precisely and the solution in the memory, which has been optimized with respect to the old environment, will instantaneously move the GA to the reappeared optimum of that environment.

For explicit memory schemes, which are the concern of this paper, there are several technical considerations: what to be stored in the memory, how to update the memory, and how to retrieve the memory (i.e., how to reuse stored information). For the first aspect, usually good solutions are stored and reused directly when the environment changes (Louis and Xu, 1996; Yang, 2005c). This is called *direct memory scheme*. It is also interesting to store environmental information together with good solutions, which is called *associative memory scheme*. For example, Ramsey and Greffenstette (1993) devised a GA for the robot control problem, where good solutions are stored in the memory together with the current environmental information. When the robot comes to a new environment similar to a stored environmental instance, the associated solution in the memory is reactivated. In Yang (2005b) and Yang and Yao (2008), an associative memory scheme was developed for the population-based incremental learning algorithms (Baluja, 1994) for DOPs, where the probability vector (model) is also stored and associated with the best solution sampled from it in the memory. When the environment

changes, the stored model associated with the best reevaluated solution in the memory is retrieved as the future working model to sample solutions for the new environment.

For the memory updating strategy, since the memory space is usually limited and fixed for the efficiency of space usage and computational cost, it is necessary to remove memory solutions, when it is full, to make room for new ones. A general strategy is to select one memory point to be replaced by the best individual from the population or to be moved toward it (Bendtsen and Krink, 2002). This can be done periodically, for example every certain number of generations. As to which memory point should be selected for updating, there are several memory replacement strategies. For example, we can replace the least important one with respect to the age, contribution to diversity and fitness, replace the one with least contribution to memory variance, replace the most similar one if the new individual is better, or replace the less fit one of a pair of memory points that have the minimum distance among all pairs (Branke, 1999).

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.

The GA with the memory scheme studied in this paper, called *memory-enhanced GA* (MEGA), is shown in Figure 2, where $f(\cdot)$ is the fitness function. MEGA (and other memory based GAs studied in this paper) uses a memory of size $m = 0.1 * n$. The memory in MEGA is reevaluated every generation to detect environmental changes. The environment is detected as changed if the fitness of at least one individual in the memory has been detected to have changed its fitness. 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 standard genetic operations for a new population while the memory remains unchanged.

The memory in MEGA is randomly initialized. Instead of updating the memory regularly as in other memory-based GAs in the literature, the memory in MEGA is updated in a stochastic time pattern as follows. After each memory updating, a random integer 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 $t_M = t + rand(5, 10)$. In order to store the most relevant information to an environment in the memory, each time an environmental change is detected, the memory is also updated according to the population just before the environmental change. When the memory is due to update, if any of the randomly initialized points still exists in the memory, the best individual of the current population (if the memory update is due to $t = t_M$) or the elite from the previous population (if the memory update is because an environmental change is detected) will replace one of them randomly; otherwise, the best individual or the elite will replace the closest memory point if it is fitter according to the current environment or the previous environment, respectively. It can be seen that the most similar memory updating strategy is applied in MEGA.

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 (Trojanowski and Michalewicz, 1999). The pseudocode for 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 in MRIGA before entering the next generation, $r_i * n$ random immigrants are swapped into the population to replace the worst ones.

```
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)
  replace the worst in P(t) by the elite E(t − 1) from P(t − 1)

  if change detected then P'(t) :=retrieveBestMembersFrom(P(t), M(t))
  else P'(t) := P(t)

  if t = t_M || change detected then // time to update memory
    if t = t_M then B_P(t) := retrieveBestMemberFrom(P'(t))
    if change detected then B_P(t) := E(t − 1)
    if still any random point in memory then
      replace a random point in memory with B_P(t)
    else    // replace the most similar memory point
      if t = t_M then find the memory point C_M(t) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t)) then C_M(t) := B_P(t)
      if change detected then find the memory point C_M(t−1) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t − 1)) then C_M(t − 1) := B_P(t)
    t_M := t + rand(5, 10)

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

  // standard genetic operations
  P''(t) := selectForReproduction(P'(t))
  crossover(P''(t), p_c) // p_c is the crossover probability
  mutate(P''(t), p_m)    // p_m is the mutation probability
  P(t + 1) := P''(t)
until the termination condition is met    // e.g., t > t_max
```

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

2.4 The Memory/Search Genetic Algorithm

Branke (1999, 2002) proposed a *memory/search GA* that combines the multi-population and memory schemes. In this paper, a similar memory/search GA, denoted *MSGA*, is also studied as a peer GA. Figure 3 shows the pseudocode of MSGA. In MSGA, in addition to the memory, MSGA maintains two populations P_1 and P_2 that evolve independently. The population sizes n_1 and n_2 for P_1 and P_2, respectively, are equally initialized to $0.45 * n$, where n is the total population size, including the memory. In order to give the better performing population more chance to search, n_1 and n_2 are slightly adjusted every generation within the range of $[0.3 * n, 0.6 * n]$ according to their relative performance. The winner population wins $\delta = 0.05 * n$ for its population size from the loser; if the two populations tie, their sizes do not change. This adaptive population size scheme was first used in Yang and Yao (2005).

As in MEGA, the memory in MSGA has a size $m = 0.1 * n$, is randomly initialized, and is updated in the stochastic time pattern with the most-similar updating strategy.

$t := 0$ and $t_M := rand(5, 10)$
initialize memory $M(0)$ and populations $P_1(0)$ and $P_2(0)$ randomly
repeat
  evaluate $P_1(t)$, $P_2(t)$ and $M(t)$
  replace the worst in $P_1(t)$ by the elite $E(t-1)$ from $P_1(t-1)$ and $P_2(t-1)$
  adjust next population sizes for $P_1(t)$ and $P_2(t)$ respectively

  if *change detected* **then**
    $P_1'(t)$:=retrieveBestMembers($P_1(t), M(t)$) and $P_2'(t)$:=re-initialize($P_2(t)$)
  else $P_1'(t) := P_1(t)$ and $P_2'(t) := P_2(t)$

  if $t = t_M$ || *change detected* **then** // *time to update memory*
    if $t = t_M$ **then** $B_P(t) :=$ retrieveBestMemberFrom($P_1'(t), P_2'(t)$)
    if *change detected* **then** $B_P(t) := E(t-1)$
    if *still any random point in memory* **then**
      replace a random point in memory with $B_P(t)$
    else // replace the most similar memory point
      if $t = t_M$ **then** find the memory point $C_M(t)$ closest to $B_P(t)$
        if $f(B_P(t)) > f(C_M(t))$ **then** $C_M(t) := B_P(t)$
      if *change detected* **then** find the memory point $C_M(t-1)$ closest to $B_P(t)$
        if $f(B_P(t)) > f(C_M(t-1))$ **then** $C_M(t-1) := B_P(t)$
    $t_M := t + rand(5, 10)$

  // *normal genetic operations for* $P_1'(t)$ *and* $P_2'(t)$ *respectively*
  $(P_1''(t), P_2''(t))$:= selectForReproduction($P_1'(t), P_2'(t)$)
  crossover($P_1''(t), P_2''(t), p_c$) // p_c *is the crossover prob.*
  mutate($P_1''(t), P_2''(t), p_m$) // p_m *is the mutation prob.*
  $P_1(t+1) := P_1''(t)$ and $P_2(t+1) := P_2''(t)$
until the termination condition is met // *e.g.,* $t > t_{max}$

Figure 3: Pseudocode for the memory/search GA (MSGA).

When the memory is due to update, the best individual over P_1 and P_2 will replace the closest memory solution if it is fitter than the memory solution. The memory is reevaluated every generation. When an environmental change is detected, the memory is merged with the old population P_1 and the best individuals are selected as a new interim population P_1 with the memory unchanged. That is, only P_1 retrieves the memory and hence is called the *memory population*. The second population P_2 is restarted (reinitialized) when an environmental change is detected, in order to search new areas in the search space and is hence called the *search population*.

3 Memory- and Elitism-Based Immigrants

3.1 The Memory-Based Immigrants Scheme

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

```
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)
  replace the worst individual in P(t) by the elite E(t − 1) from P(t − 1)

  if t = t_M || change detected then // time to update memory
    if t = t_M then B_P(t) := retrieveBestMemberFrom(P'(t))
    if change detected then B_P(t) := E(t − 1)
    if still any random point in memory then
      replace a random point in memory with B_P(t)
    else    // replace the most similar memory point
      if t = t_M then find the memory point C_M(t) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t)) then C_M(t) := B_P(t)
      if change detected then find the memory point C_M(t−1) closest to B_P(t)
        if f(B_P(t)) > f(C_M(t − 1)) then C_M(t − 1) := B_P(t)
    t_M := t + rand(5, 10)

  // 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)
  evaluate the memory-based immigrants in P_I(t)
  replace the worst r_i*n individuals in P(t) by immigrants in P_I(t)

  // standard genetic operations
  P'(t) := selectForReproduction(P(t))
  crossover(P'(t), p_c) // p_c is the crossover probability
  mutate(P'(t), p_m)    // p_m is the mutation probability
  P(t + 1) := P'(t)
until the termination condition is met    // e.g., t > t_max
```

Figure 4: Pseudocode for the GA with memory-based immigrants (MIGA).

solutions. These two approaches can be simply combined into GAs as in the MRIGA. However, a more efficient approach of hybridizing memory and random immigrants for GAs to deal with dynamic environments is the memory-based immigrants scheme proposed in Yang (2005a). The pseudocode for the GA with the memory-based immigrants scheme, denoted *MIGA* in this paper, is shown in Figure 4.

From Figures 4 and 2, it can be seen that MIGA uses the same memory updating scheme as MEGA and MRIGA. However, the memory retrieval does not depend on the detection of environmental changes and is hybridized with the random immigrants scheme via the mutation mechanism. For every generation, the memory is reevaluated and the best memory point $B_M(t)$ is retrieved as the base to create immigrants. From $B_M(t)$, a set $P_I(t)$ of $r_i * n$ individuals are iteratively generated by performing general bitwise flip mutation with a probability p_m^i on $B_M(t)$. The generated individuals then act as immigrants and replace the worst $r_i * n$ 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

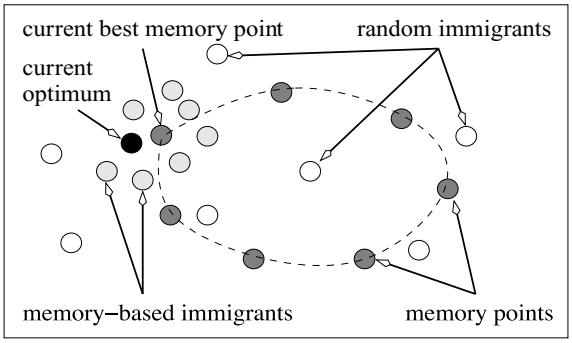

Figure 5: Illustration of immigrants schemes for GAs in dynamic environments. The memory-based immigrants are distributed more closely around the current optimum than random immigrants.

random immigrants. This is illustrated in Figure 5. 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 base memory point. Since the base memory point is evaluated as the best one of the memory in the current environment (and hence may be close to the current optimum of the dynamic problem), the immigrants created from it are distributed more precisely around the optimum of the current environment. This bias enables the memory-based immigrants to track the moving optima more efficiently than random immigrants can do and hence is expected to better improve the GA's performance in dynamic environments.

Note that the mechanisms of diversity and memory in biology have been applied to GAs for DOPs. For example, Simões and Costa (2003a, b) proposed an immune system based GA (ISGA) for dynamic environments. Their ISGA maintains two populations. The first one is the main population and evolves as follows: the individuals with the best matches to the optimum are selected and cloned into the next generation. At times, the best individual is stored in the second memory population and is attached a value of the average fitness of the first population. When an environmental change is detected, the memory individual most proximal to the new environment is activated, cloned, and replaced into the first population. The proximity is measured by the average fitness of the first population and the value attached to the memory individuals. A set of gene libraries are used in ISGA, each containing a set of gene segments. During the cloning process, an individual is subject to a *transformation* process with a probability (Simões and Costa, 2001) as follows. First, one gene segment is randomly selected from one randomly chosen gene library. Then, a transformation locus is randomly selected in the individual and the chosen gene segment is incorporated into the individual, replacing the genes after the transformation locus.

MIGA differs from Simões and Costa's ISGA in two aspects. First, MIGA uses traditional crossover operators while ISGA uses the cloning scheme from biology to generate offspring. Second, ISGA maintains an extra gene pool to provide gene materials for the cloning process while MIGA is based on traditional memory and mutation schemes and is more straightforward. According to our preliminary experiments, Simões and

$t := 0$ and initialize population $P(0)$ randomly
repeat
  evaluate population $P(t)$
  replace the worst individual in $P(t)$ by the elite $E(t-1)$ from $P(t-1)$
  // perform elitism-based immigration
  $P_I(t) :=$ mutateElite$(E(t-1), r_i * n, p_m^i)$
  evaluate the elitism-based immigrants in $P_I(t)$
  replace the worst $r_i * n$ individuals in $P(t)$ by immigrants in $P_I(t)$
  // standard genetic operations
  $P'(t) :=$ selectForReproduction$(P(t))$
  crossover$(P'(t), p_c)$ *// p_c is the crossover probability*
  mutate$(P'(t), p_m)$ *// p_m is the mutation probability*
  $P(t+1) := P'(t)$
until the termination condition is met *// e.g., $t > t_{max}$*

Figure 6: Pseudocode for the elitism-based immigrants GA (EIGA).

Costa's ISGA underperforms MIGA on most dynamic test problems studied in this paper and underperforms other GAs investigated in this paper on many dynamic test problems and hence will not be further discussed in this paper.

3.2 The Elitism-Based Immigrants Scheme

The traditional random immigrants approach works by inserting random individuals into the population. This may increase the population diversity and improve the performance of GAs in dynamic environments. However, in a slowly changing environment, random immigrants introduced may divert the searching force of GAs during each environment before a change occurs and hence may degrade the performance. On the other hand, if the environment only changes slightly in terms of severity of changes, random immigrants may not have any actual effect even when a change occurs because individuals in the previous environment may still be quite fit in the new environment.

Based on the above consideration, an elitism-based immigrants approach was proposed for GAs to address DOPs in Yang (2007). Figure 6 shows the pseudocode for the GA with the elitism-based immigrants scheme, denoted *EIGA* in this paper. Within EIGA, for each generation t, before the normal genetic operations (i.e., selection and recombination), the elite $E(t-1)$ from the previous generation is used as the base to generate a set of $r_i \times n$ individuals iteratively by a bitwise mutation with a probability p_m^i, where r_i is the ratio of the elitism-based immigrants to the population size. The generated immigrants replace the worst individuals in the current population.

The elitism-based immigrants scheme combines the idea of elitism with the random immigrants scheme. It differs from the aforementioned memory-based immigrants scheme in that the elite from the previous population instead of the best memory point is used to guide the immigrants toward the current environment.

Another thing to notice is the difference between EIGA and a standard GA with a high selection pressure. In EIGA, a set of immigrants (or offspring) are created only

from the elite of the previous generation with the standard mutation. For a standard GA, when the selection pressure is very high (e.g., when a tournament selection is used with a tournament size much greater than two), it may also lead to several offspring created by crossing over the elite with the elite (i.e., no effect) and then mutating it. However, a very high selection pressure may deprive the diversity of the population too much and hence may degrade the performance of GAs in dynamic environments. For EIGA, the effect of the elite is limited by the ratio of immigrants (i.e., r_i).

The effect of the selection pressure for a standard GA in dynamic environments has been studied recently by Yang and Tinós (2008). Their experimental results show that increasing the selection pressure does have an important effect on the performance of the standard GA in dynamic environments, but whether the effect is positive or negative depends on the dynamic problem. Their experimental results also show that EIGA outperforms the standard GA with different levels of selection pressure and an adaptive selection pressure scheme, called *hyper-selection*, on most tested DOPs. Because of the experimental results in Yang and Tinós (2008) and also because the selection pressure is not the main concern of this study, it will not be further studied in this paper.

4 Dynamic Test Environments

4.1 General Dynamic Environment Generators

In order to compare the performance of the developed GA approaches in dynamic environments, researchers have developed a number of dynamic problem generators. Generally speaking, dynamic problems are constructed via changing (the parameters of) stationary base problem(s). And ideally through proper control, different dynamic environments can be constructed from the stationary base problem(s) regarding the characteristics of the environmental dynamics, such as the frequency, severity, predictability, and cyclicity of environmental changes.

In the early days, the dynamic environment generators were quite simple and just switch between two or more stationary problems (or states of a problem). For example, the dynamic knapsack problem where the knapsack capacity oscillates between two or more fixed values has been frequently used in the literature (Dasgupta and McGregor, 1992; Lewis and Ritchie, 1998; Mori and Nishikawa, 1997; Ng and Wong, 1995). Cobb and Grefenstette (1993) used a dynamic environment that oscillates between two different fitness landscapes. Later in 1999, several researchers independently developed several dynamic environment generators by changing a base fitness landscape predefined in n-dimensional real space (Branke, 1999; Grefenstette, 1999; Morrison and De Jong, 1999; Trojanowski and Michalewicz, 1999). This base landscape consists of a number of peaks. Each peak can change its own morphology independently, such as the height, slope and location of the peak. The center of the peak with the highest height is taken as the optimal solution of the landscape. Dynamic problems can be created through changing the parameters of each peak.

Recently, a dynamic problem generator based on the bitwise exclusive-or (XOR) operator, called the *XOR generator* in short henceforth, was proposed in Yang (2003), and Yang and Yao (2005). This XOR generator can construct dynamic environments from any binary-encoded stationary function $f(\vec{x})$ ($\vec{x} \in \{0, 1\}^l$ where l is the length of binary representation) as follows. Suppose the environment is periodically changed every τ generations. For each environmental period k, an XOR mask $\vec{M}(k)$ is first 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 the XOR generator, the parameter τ controls the change speed while $\rho \in (0.0, 1.0)$ controls the severity of environmental changes. A bigger value of ρ implies a more severe environmental change and hence greater challenge to GAs.

4.2 Dynamic Environment Generator for Testing Memory Schemes

In order to better test the memory schemes for GAs in dynamic environments, we need a dynamic problem generator that can control the cyclicity of the environments constructed since memory schemes are expected to work well in cyclic environments. Only under dynamic environments of varying cyclicity can the memory schemes for GAs be fully tested and justified. The aforementioned XOR generator in fact can construct *non-cyclic dynamic environments*, also called *random dynamic environments* in this paper, because there is no guarantee that the environment will return to a previous one after certain changes. Recently, the XOR generator has been extended to construct *cyclic dynamic environments* in Yang (2005c) and *cyclic dynamic environments with noise* further in Yang and Yao (2008).

With the XOR generator, cyclic dynamic environments can be constructed as follows. First, we can generate $2K$ XOR masks $\vec{M}(0), \ldots, \vec{M}(2K-1)$ 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 are evaluated as follows:

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(I_t)) = f(\vec{x} \oplus \vec{M}(k\%(2K))) \tag{3}$$

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 that the environment is in at generation t.

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

$$\vec{M}(i+1) = \vec{M}(i) \oplus \vec{T}(i\%K), \quad i = 0, \ldots, 2K-1 \tag{4}$$

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

The templates $\vec{T}(0), \ldots, \vec{T}(K-1)$ are first used to create K masks till $\vec{M}(K) = \vec{1}$ and then orderly reused to construct another K XOR masks till $\vec{M}(2K) = \vec{M}(0) = \vec{0}$. The Hamming distance between two neighbor XOR 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 above cyclic environment generator, we can further construct cyclic dynamic environments with noise as below. Each time the environment is about to move to a next base state $\vec{M}(i)$, $\vec{M}(i)$ is bitwise flipped with a small probability, denoted p_n in this paper.

4.3 Dynamic Test Environments for This Study

In this paper, three 100-bit binary-encoded problems are selected as the stationary functions. The first one is the *OneMax* function, which aims to maximize the number of ones in a chromosome. The second one, denoted *Plateau*, consists of 25 contiguous four-bit building blocks. Each building block for *Plateau* contributes four (or two) to the total fitness if its unitation (i.e., the number of ones inside the building block) is four (or three); otherwise, it contributes zero. The third problem is a 100-item 0-1 knapsack problem with the weight and profit of each item randomly created in the range of $[1, 30]$ and the capacity of the knapsack set to half of the total weight of all items. The fitness of a feasible solution is the sum of the profits of the selected items. If a solution overfills the knapsack, its fitness is set to the difference between the total weight of all items and the weight of selected items, multiplied by a small factor 10^{-5} to make it in-competitive with those solutions that do not overfill the knapsack.

Three kinds of dynamic environments, cyclic, cyclic with noise, and random, are constructed from each of the three base functions using the aforementioned extended XOR generator. For all the dynamic environments, the landscape is periodically changed every τ generations during the run of an algorithm. In order to test the effect of environmental change speed on the performance of algorithms, τ is set to 10 and 50. The environmental change severity parameter ρ is set to 0.1, 0.2, 0.5, and 1.0 for all dynamic problems. With this setting of ρ, for cyclic dynamic problems, with and without noise, the environment cycles among 20, 10, 4, and 2 base states respectively. For cyclic dynamic problems with noise, the probability p_n is set to 0.05 in the basic experiments and to different values in the sensitivity analysis experiments.

In total, a series of 24 DOPs, that is, two values of τ combined with four values of ρ under three kinds of dynamic environments, are constructed from each stationary function.

5 Experimental Study

5.1 Experimental Design

Experiments were carried out to compare different GAs on the above described dynamic environments. For the basic experiments, typical generators and parameters are used for all GAs as follows: generational GAs with uniform crossover with $p_c = 0.6$, flip mutation with $p_m = 0.01$, and fitness proportionate selection and elitism of size one. In order to have fair comparisons among GAs, the population size and immigrants ratios are set such that each GA has 120 fitness evaluations per generation as follows.

$$(1 + r_i) * n = 120 \tag{5}$$

where r_i is the immigrants ratio for GAs with immigrants and n is the whole population size including the memory size if memory is used. If memory is used, its size m is set to $m = 0.1 * n$. Hence, we have $n = 120$ for SGA, MEGA, and MSGA, and $n = 100$ for RIGA, MRIGA, EIGA, and MIGA, and $m = 12$ for MEGA and MSGA and $m = 10$ for MRIGA and MIGA. The immigrants ratio r_i for RIGA, MRIGA, EIGA, and MIGA is set to 0.2. For EIGA and MIGA p_m^i of bitwise mutating the elite of previous generation or best individual from the memory for immigrants is set to 0.01.

For each experiment of an algorithm on a dynamic test problem, 50 independent runs were executed with the same set of random seeds. For each run, 200 environmental changes were allowed, which are equivalent to 2000 and 10,000 generations for $\tau = 10$ and 50, respectively. For each run, the best-of-generation fitness was recorded every generation. The overall offline performance of a GA on a DOP is defined as:

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

where $G = 200 \times \tau$ is the total number of generations for a run, $N = 50$ is the total number of runs, and $F_{\mathrm{BOG}_{ij}}$ is the best-of-generation fitness of generation i of run j. $\overline{F}_{\mathrm{BOG}}$ is the off-line performance, that is, the best-of-generation fitness averaged over the 50 runs and then over the data gathering period.

5.2 Basic Experimental Results and Analysis

The basic experimental results of GAs on the three kinds of dynamic environments: cyclic, cyclic with noise, and random, are presented in Figure 7 to Figure 9, respectively. The corresponding statistical results of comparing GAs by a one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table 1 to Table 3, respectively. In these tables, 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 the investigated GAs in dynamic environments, the dynamic behavior of GAs with respect to best-of-generation fitness against generations on the dynamic test functions with $\tau = 50$ and $\rho = 0.2$ under cyclic and random environments is plotted in Figure 10 and Figure 11, respectively, where the last cycle of 10 environmental changes (i.e., 500 generations) is shown. From Figures 7 to 11 and Tables 1 to 3, several results can be observed and are analyzed as follows.

First, a prominent result is that MIGA significantly outperforms SGA, RIGA, MEGA, MRIGA, and MSGA (the performance of EIGA will be analyzed later) on most dynamic test problems; see the relevant t-test results in Table 1 to Table 3. This result validates our expectation of the memory-based immigrants scheme for GAs in dynamic environments. When $\tau = 10$, MIGA underperforms some of these GAs on a few cyclic with noise and random dynamic 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 may misguide the immigrants to a less fit area.

The good performance of MIGA over SGA, RIGA, MEGA, MRIGA, and MSGA can be further observed in the dynamic behavior of GAs plotted in Figures 10 and 11, where MIGA is able to maintain a much higher fitness level in the dynamic environments. For example, the performance of MIGA is $\overline{F}_{\mathrm{BOG}}$(MIGA) = 95.7, 91.3, and 1218.7 on random

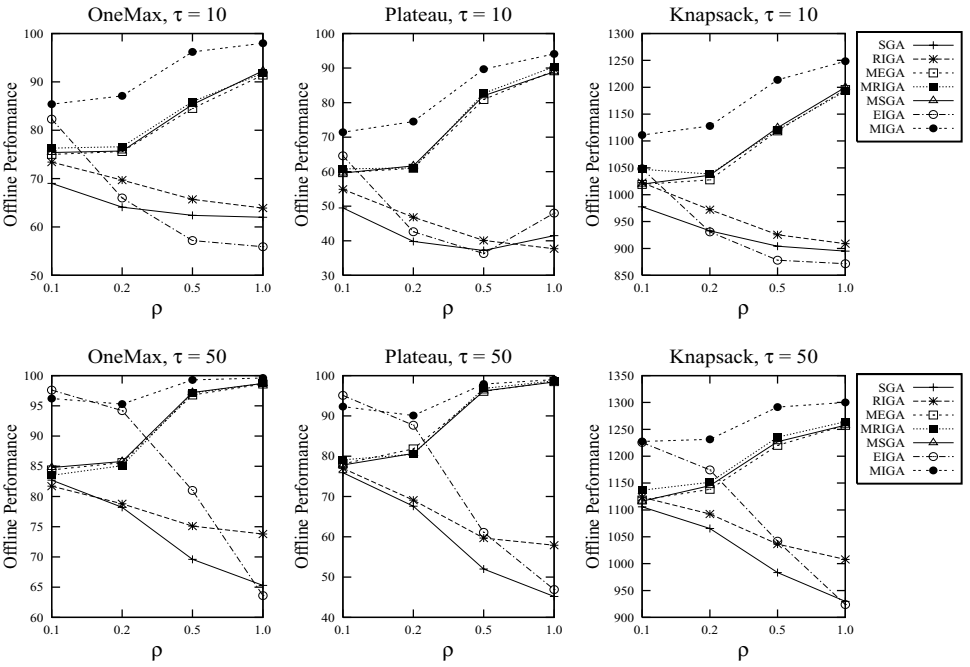

Figure 7: Experimental results of GAs in cyclic dynamic environments.

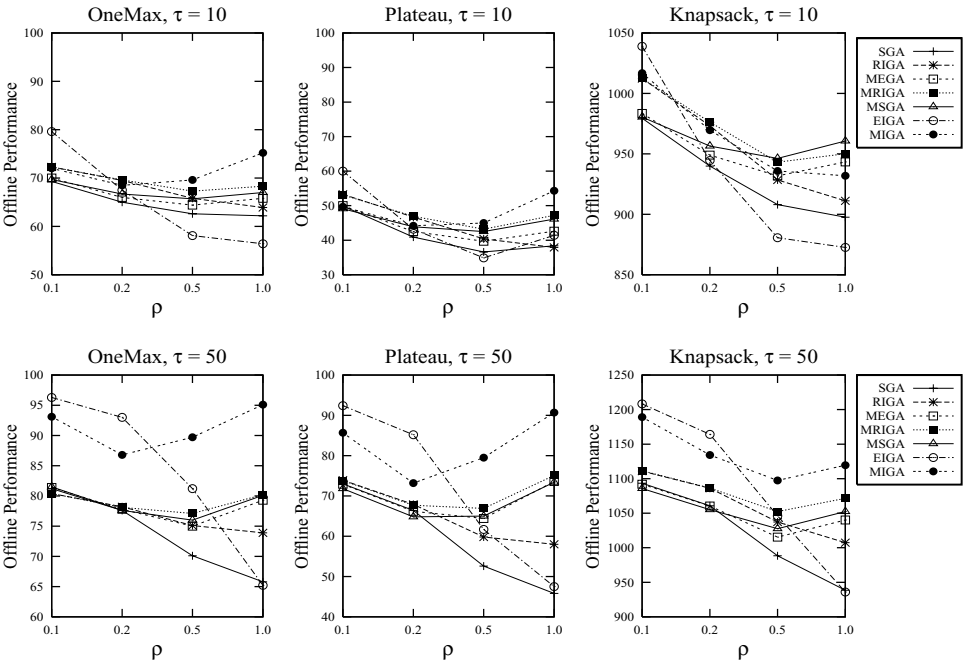

Figure 8: Experimental results of GAs in cyclic dynamic environments with noise.

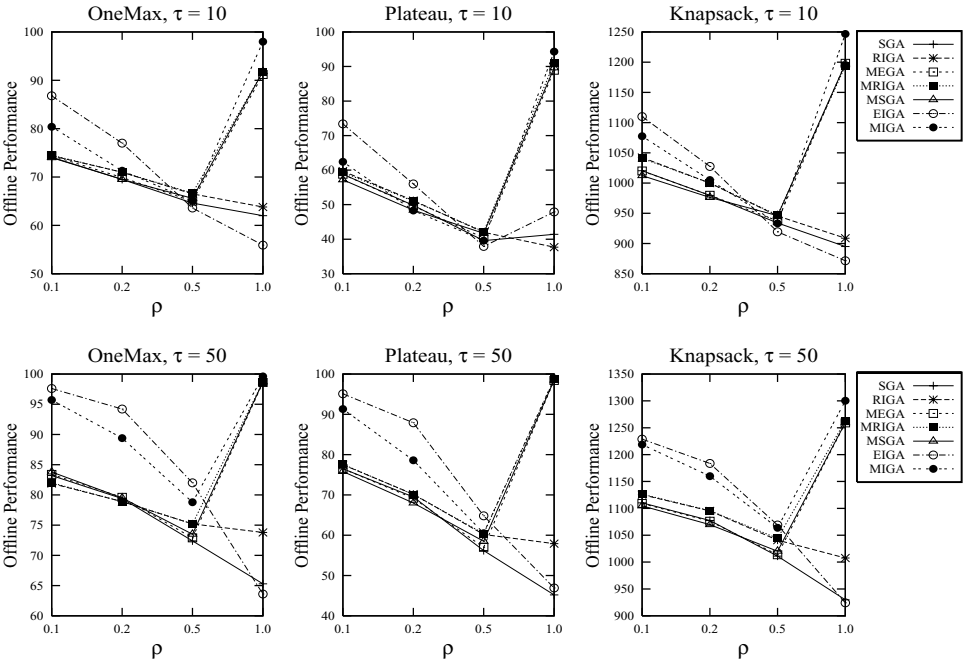

Figure 9: Experimental results of GAs in random dynamic environments.

OneMax, *Plateau*, and *Knapsack* with $\tau = 50$ and $\rho = 0.1$, respectively. These results are significantly better than the best performance of SGA, RIGA, MEGA, MRIGA, and MSGA on the three DOPs, respectively, that is, $\overline{F}_{\mathrm{BOG}}(\mathrm{MSGA}) = 83.8$ on random *OneMax*, $\overline{F}_{\mathrm{BOG}}(\mathrm{MRIGA}) = 77.6$ on random *Plateau*, and $\overline{F}_{\mathrm{BOG}}(\mathrm{MRIGA}) = 1126.4$ on random *Knapsack*, respectively.

Second, the traditional memory scheme improves the performance of GAs for most DOPs; see the t-test results regarding MEGA – SGA and MRIGA – RIGA in Table 1 to Table 3. And when viewing across Figure 7 to Figure 9, it can be seen that the impact of memory in MEGA and MRIGA changes with the cyclicity of dynamic environments. For the same τ and ρ, the performance improvement of MEGA over SGA and MRIGA over RIGA, that is, $\overline{F}_{\mathrm{BOG}}(\mathrm{MEGA}) - \overline{F}_{\mathrm{BOG}}(\mathrm{SGA})$ and $\overline{F}_{\mathrm{BOG}}(\mathrm{MRIGA}) - \overline{F}_{\mathrm{BOG}}(\mathrm{RIGA})$, reaches the highest value in cyclic environments while it is significantly reduced in cyclic with noise and random environments. And under cyclic environments for each DOP with the same value of τ, the impact of memory increases with the value of ρ. From each individual picture in Figure 7, it can be seen that the performance of GAs with memory, that is, MEGA, MRIGA, MSGA, and MIGA, basically increase from $\rho = 0.1$ to $\rho = 1.0$. This is because a bigger ρ means fewer base states and hence memory points are more accurate to relevant base states when they were stored. However, the performance of SGA, RIGA, and EIGA on cyclic DOPs decreases when ρ increases. This is natural because, without memory, a bigger ρ means more severe environmental changes to SGA, RIGA, and EIGA.

When $\rho = 1.0$, the environment switches between two fitness landscapes that are complementary to each other. The dynamic behavior of GAs with respect to the

Table 1: The t-test Results of Comparing GAs on Cyclic DOPs

t-test Result	*OneMax*				*Plateau*				*Knapsack*			
$\tau = 10, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$	$s+$	$+$	$-$
MSGA − MRIGA	$s-$	$s-$	$s-$	$s+$	$s-$	$+$	$s-$	$s-$	$s-$	$-$	$+$	$s+$
EIGA − RIGA	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$
EIGA − MRIGA	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$+$	$s-$	$s-$	$s-$
EIGA − MSGA	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$
MIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MSGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − EIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
$\tau = 50, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − MEGA	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MSGA − MRIGA	$s+$	$s+$	$+$	$-$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIGA − RIGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
EIGA − MRIGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
EIGA − MSGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
MIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MSGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − EIGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$

best-of-generation fitness against generations on the cyclic and noisy *Plateau* functions with $\tau = 50$ and $\rho = 1.0$ is plotted for the first 500 generations in Figure 12. The dynamic performance of GAs is the same on the random *Plateau* function with $\tau = 50$ and $\rho = 1.0$ as on the cyclic *Plateau* function with $\tau = 50$ and $\rho = 1.0$ and is not plotted in Figure 12. This is because when $\rho = 1.0$ a cyclic environment is equivalent to a random one. From Figure 12, it can be seen that after several environmental changes the memory scheme clearly drives GAs with memory toward a high fitness level while SGA, RIGA, and EIGA struggle to climb from a low fitness level during each environmental period.

Third, MSGA outperforms MEGA on many DOPs. This result justifies the introduction of an extra population that restarts when the environment changes. However, MSGA is beaten by MRIGA on most DOPs; see the t-test results regarding MSGA – MRIGA in Tables 1 to 3. This result shows that the simple restart scheme in MSGA may not be as efficient as the random immigrants scheme in MRIGA.

Fourth, we now examine the performance of EIGA on the DOPs. From Figures 7 to 9, it can be seen that, generally speaking, the performance of EIGA drops with the rising value of ρ on each DOP with fixed τ under three types of dynamic environments and the degree of performance dropping of EIGA is much more significant than that of SGA and RIGA. For example, on the random Knapsack problem with $\tau = 50$, the

Table 2: The t-test Results of Comparing GAs on Cyclic DOPs with Noise

t-test Result	*OneMax*				*Plateau*				*Knapsack*			
$\tau = 10, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − RIGA	$-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
MRIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MSGA − MRIGA	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$
EIGA − RIGA	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$
EIGA − MRIGA	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$
EIGA − MSGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$
MIGA − RIGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
MIGA − MRIGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$
MIGA − MSGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$
MIGA − EIGA	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$
$\tau = 50, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$-$	$s+$	$s+$
MRIGA − RIGA	$-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$+$	$+$	$s+$	$s+$
MRIGA − MEGA	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MSGA − MRIGA	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIGA − RIGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
EIGA − MRIGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
EIGA − MSGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$
MIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MSGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − EIGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$

performance of EIGA $\overline{F}_{\text{BOG}}$(EIGA) drops from 1228.8 at $\rho = 0.1$ to 1183.6 at $\rho = 0.2$, 1069.1 at $\rho = 0.5$, and 924.2 at $\rho = 1.0$ while the performance of SGA $\overline{F}_{\text{BOG}}$(SGA) drops from 1110.3 at $\rho = 0.1$ to 1077.4 at $\rho = 0.2$, 1011.1 at $\rho = 0.5$, and 929.7 at $\rho = 1.0$.

In comparison with the performance of other GAs, EIGA performs quite inconsistently over the DOPs. When the environment changes slightly, for example, $\rho = 0.1$, EIGA outperforms other GAs on most cases; see the t-test results regarding EIGA – RIGA, EIGA – MRIGA, EIGA – MSGA, and MIGA – EIGA in Tables 1 to 3 (other t-test results are not shown). This happens because under slightly changing environments, when a change occurs, the elitism mechanism will drive EIGA to a high fitness level directly since the elite from the previous generation is quite likely to still fit the new environment. This can be more clearly observed in the dynamic behavior of GAs in Figures 10 and 11, where EIGA can climb to a high fitness level quite fast after a change occurs.

It is noticeable that on random DOPs with small ρ, MIGA also outperforms other GAs except EIGA. This is because on such DOPs the memory-based immigrants scheme in MIGA is in fact similar to the elitism-based immigrants scheme in EIGA since the best memory solution extracted as the base for immigrants is usually the elite from the previous environment. This result can be further observed in the dynamic behavior of

Table 3: The t-test Results of Comparing GAs on Random DOPs

t-test Result	*OneMax*				*Plateau*				*Knapsack*			
$\tau = 10, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$-$	$s-$	$s+$	$s+$	$-$	$+$	$s+$	$s+$
MRIGA − RIGA	$+$	$+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$-$	$s+$	$s+$
MRIGA − MEGA	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$-$
MSGA − MRIGA	$s-$	$s-$	$s-$	$-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$+$
EIGA − RIGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s-$
EIGA − MRIGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
EIGA − MSGA	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
MIGA − RIGA	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$
MIGA − MSGA	$s+$	$s+$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$
MIGA − EIGA	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
$\tau = 50, \rho \Rightarrow$	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
RIGA − SGA	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$	$s+$	$-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
MRIGA − RIGA	$s+$	$+$	$s+$	$s+$	$+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MRIGA − MEGA	$s-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MSGA − MRIGA	$s+$	$s+$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
EIGA − RIGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
EIGA − MRIGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
EIGA − MSGA	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s-$
MIGA − RIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − MSGA	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
MIGA − EIGA	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$

MIGA in Figure 11: similar to EIGA, MIGA can also climb to a high fitness level quite quickly after a change occurs. When the degree of environmental change increases, for example, $\rho = 0.5$ and 1.0, EIGA underperforms other GAs in most cases, see Figures 7 to 9 and relevant t-test results in Tables 1 to 3. This happens because when a change occurs with big ρ the elite from the previous generation will be far away from the optima of the new environment and hence will be less efficient or even misleading in guiding immigrants toward the new environment. This effect can be observed in Figure 12 regarding the dynamic behavior of EIGA over other GAs on the random *Plateau* function with $\tau = 50$ and $\rho = 1.0$. Here, when a change occurs, EIGA climbs toward the high fitness level even slower than RIGA because the elitism scheme is misleading immigrants in EIGA and hence has a negative effect.

Another observation regarding the performance of EIGA for DOPs is that on DOPs with $\tau = 50$ its relative performance to other GAs is better than on DOPs with $\tau = 10$. This happens because the elitism mechanism in EIGA works well when the environment does not change. The longer the time between two environmental changes, the better the chance for the elitism mechanism in EIGA to express its effect.

Fifth, as mentioned before, the key idea behind MIGA and EIGA is to use the memory or elite from a previous generation to bias immigrants toward the current

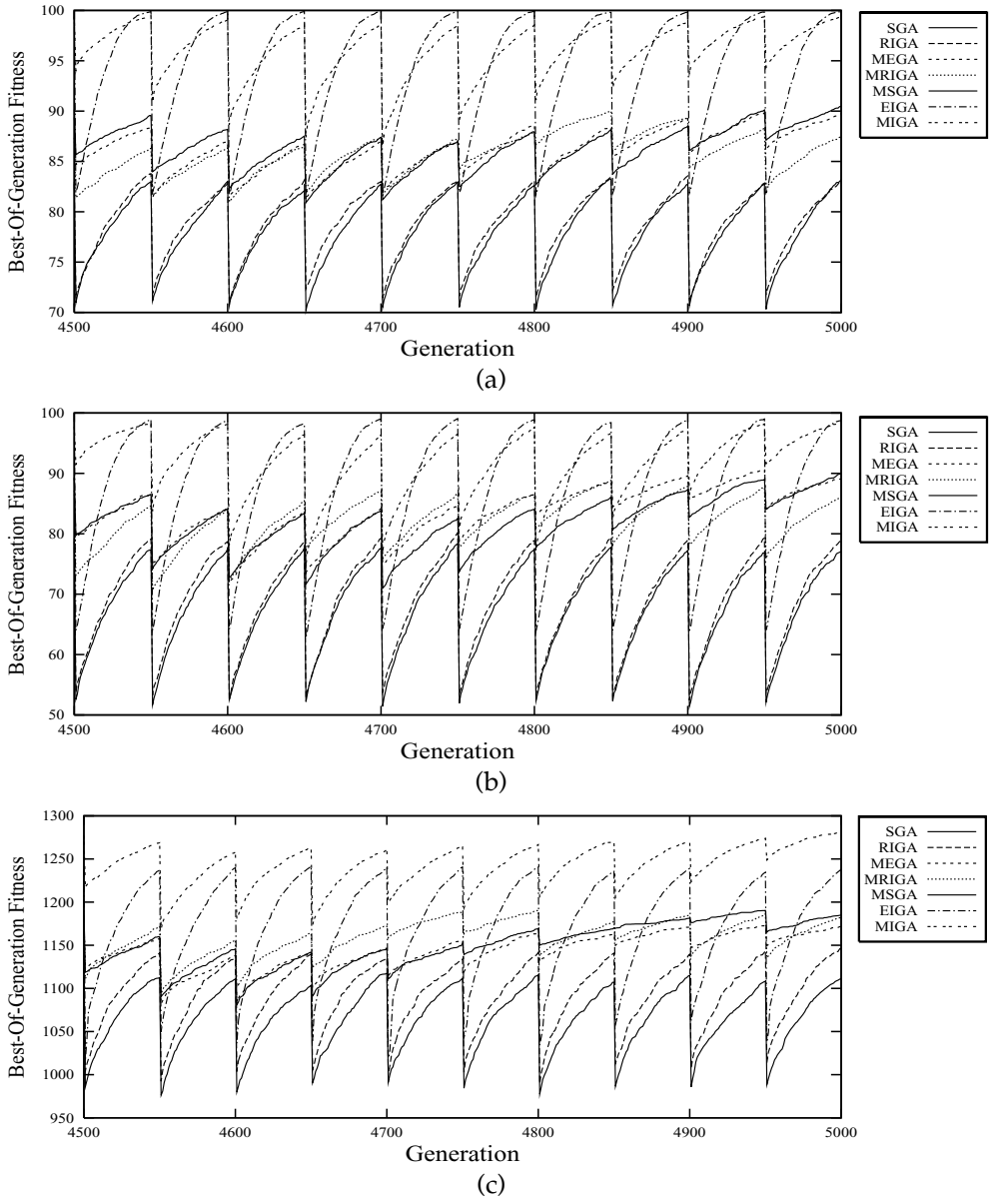

Figure 10: Dynamic behavior of GAs on cyclic DOPs with $\tau = 50$ and $\rho = 0.2$: (a) *OneMax*, (b) *Plateau*, and (c) *Knapsack*.

environment. In order to show this bias, the diversity of the population was also recorded every generation in the experiments. The diversity of the population at time t in the k-th run of a GA on a DOP is defined as:

$$Div(k,t) = \frac{1}{ln(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} HD(I_i, I_j) \tag{7}$$

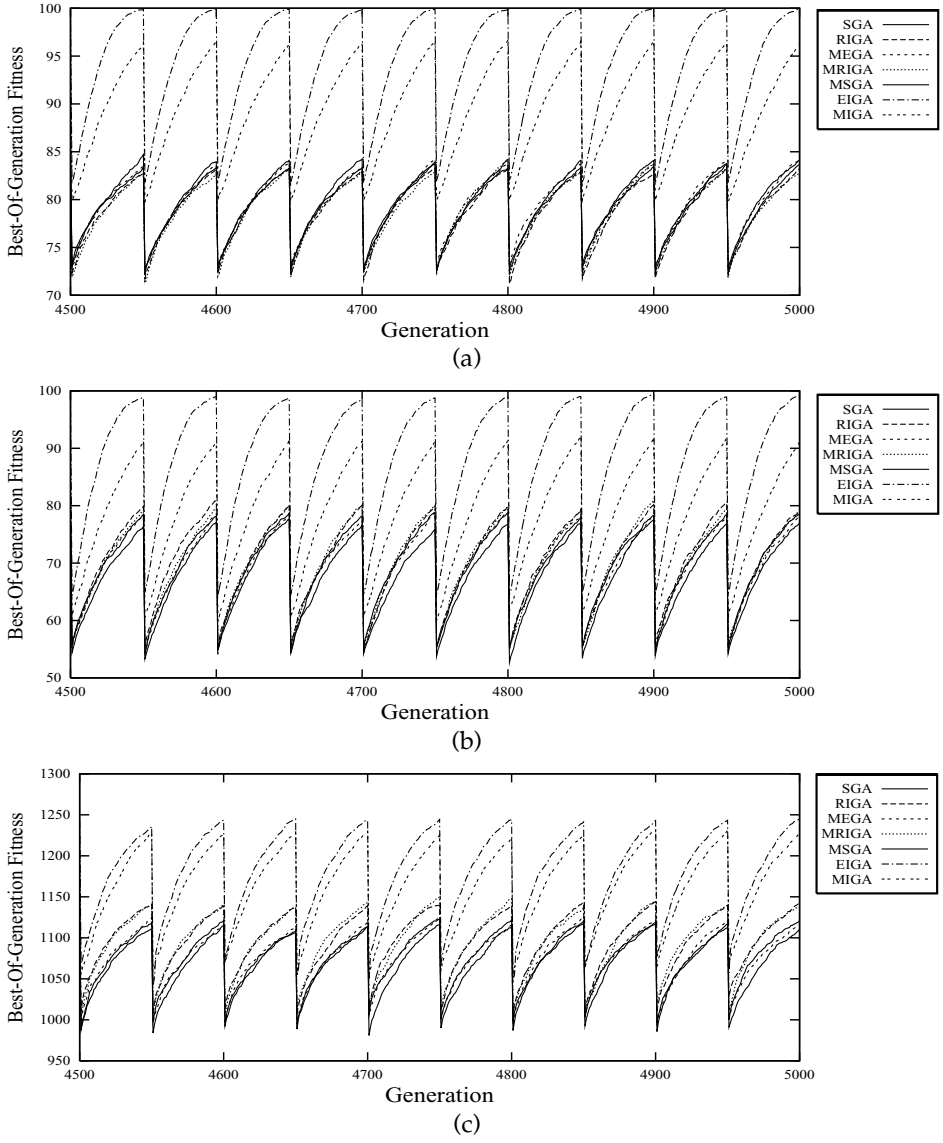

Figure 11: Dynamic behavior of GAs on random DOPs with $\tau = 50$ and $\rho = 0.2$: (a) *OneMax*, (b) *Plateau*, and (c) *Knapsack*.

where $l = 100$ is the encoding length, n is the population size, and $HD(I_i, I_j)$ is the Hamming distance between the i-th individual I_i and the j-th individual I_j in the population. The overall diversity of a GA on a DOP over 50 runs is calculated as follows.

$$\overline{Div} = \frac{1}{G} \sum_{t=1}^{G} \left(\frac{1}{50} \sum_{k=1}^{50} Div(k, t) \right) \tag{8}$$

where $G = 200 \times \tau = 5000$ is the total number of generations for a run.

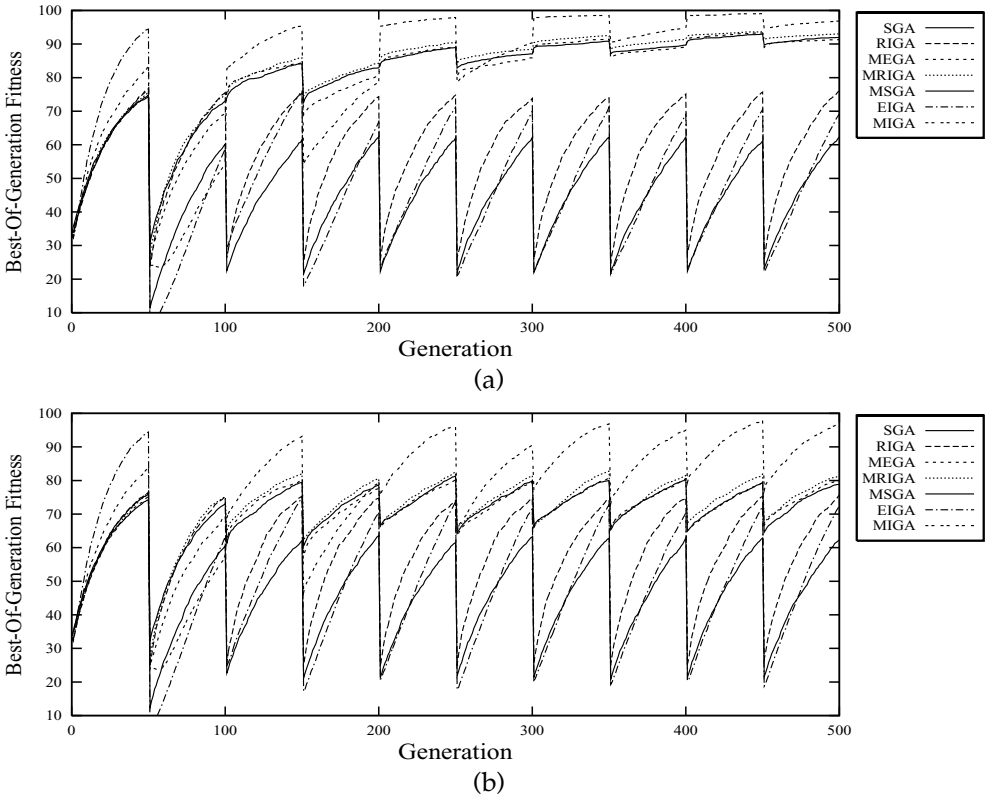

Figure 12: Dynamic behavior of GAs on dynamic *Plateau* function with $\tau = 50$ and $\rho = 1.0$ under (a) cyclic environment and (b) cyclic environment with noise.

The overall diversity of GAs on cyclic and random DOPs with $\tau = 50$ and different values of ρ is plotted in Figure 13. From Figure 13, it can be seen that both MIGA and EIGA maintain a much lower diversity level than other GAs while random immigrants based RIGA and MRIGA maintain the highest diversity level. This result shows that immigrants in MIGA and EIGA are really guided. Whether this guidance is helpful or not depends on the GAs and the DOPs. For MIGA, the memory mechanism efficiently directs the immigrants toward the current optimum, which results in low diversity but high fitness at the same time. Hence, MIGA significantly outperforms other GAs on these DOPs; see the corresponding t-test results in Tables 1 and 3. However, for EIGA the situation is different. As analyzed before, the guidance of the elitism toward immigrants is helpful when ρ is small while it may be harmful when ρ is large.

Finally, for each DOP with fixed ρ, the performance of GAs rises when the value of τ increases from 10 to 50. This is easy to understand. When the environment changes slower, that is, when τ is large, GAs have more time to reach a higher fitness level between two environmental changes. And GAs perform worse on dynamic *Plateau* problems than on corresponding *OneMax* problems with the same environmental dynamics, that is, the same values of τ and ρ. This is natural because the dynamic problem

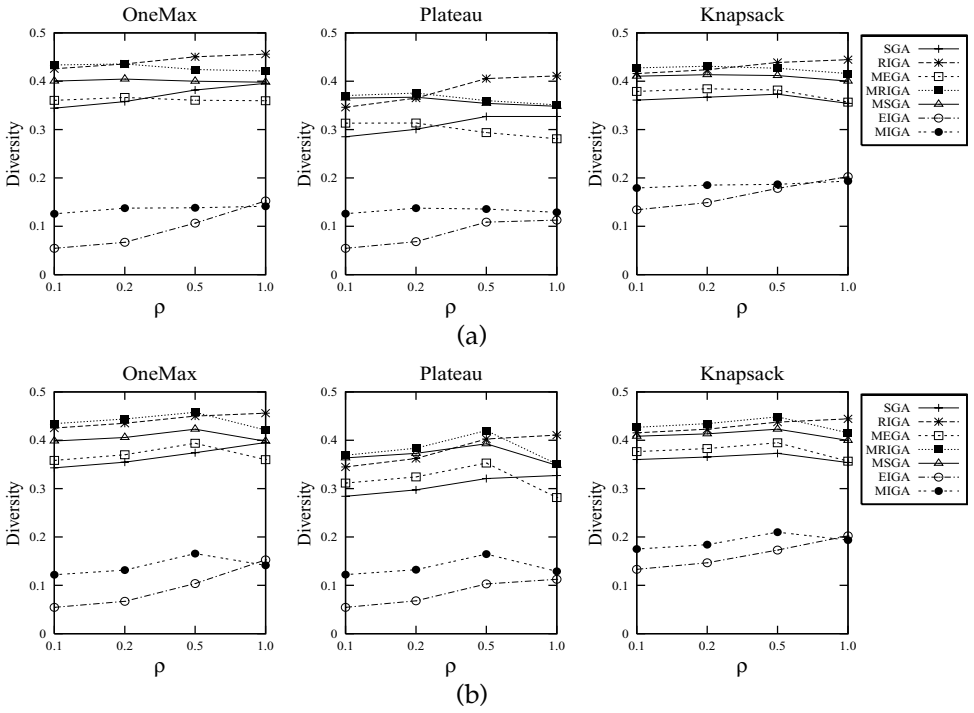

Figure 13: Experimental results regarding the diversity of the population of GAs on (a) cyclic and (b) random DOPs with $\tau = 50$.

during each environmental period can be taken as a stationary problem and the stationary *Plateau* problem is harder than the stationary *OneMax* problem for GAs.

5.3 Sensitivity Analysis on the Effect of the Noise Probability p_n

In the basic experimental results, shown in Figures 7 to 9, it can be seen that given the same ρ and τ, when the cyclicity of dynamic environments decreases from cyclic to cyclic with noise, the performance of GAs degrades. That is, cyclic environments with noise are harder than cyclic environments. The existence of noise reduces the effect of memory schemes because the environment will return to previous states less accurately. However, GAs perform better on some DOPs in random environments than in cyclic environments with noise with the same ρ and τ. This means that noise may outweigh randomness in terms of the difficulty of dynamic environments.

In order to investigate the effect of the degree of noise on the performance of GAs, we further carry out experiments in cyclic environments with noise with $\tau = 50$ and $\rho = 0.1$ and 0.5. The noise probability p_n, which determines the degree of noise, is now set to 0.01, 0.02, 0.5, and 0.1, respectively. The other experimental settings are the same as in the basic experiments. The experimental results are plotted in Figure 14.

From Figure 14, the following results can be observed. First, the noise probability p_n does affect the performance of GAs and the effect differs with GAs. Generally speaking, when the value of p_n increases from 0.01 to 0.1, the performance of GAs decreases.

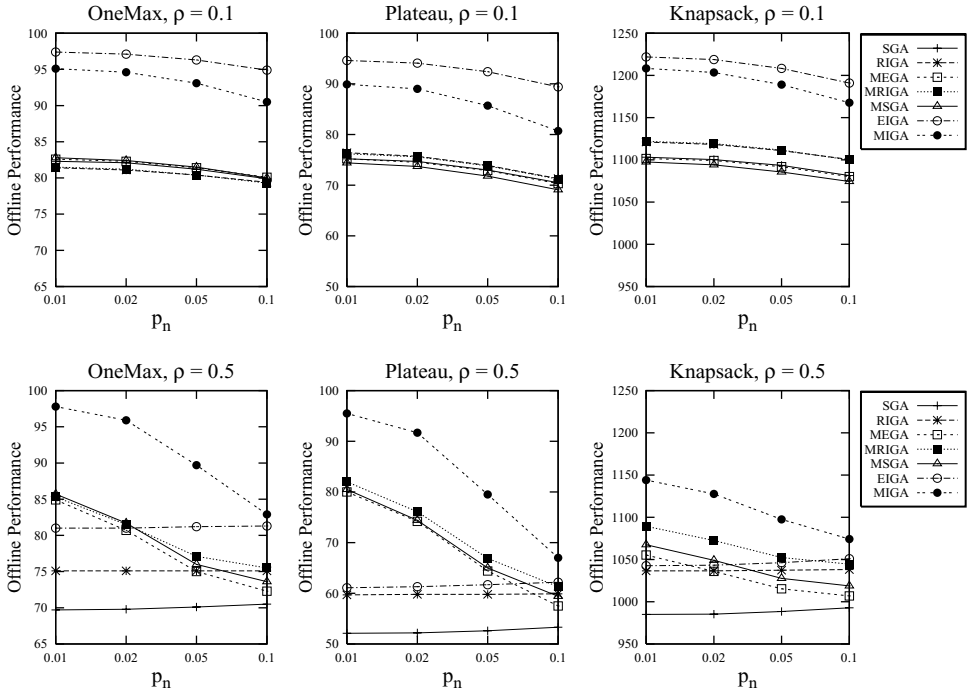

Figure 14: Experimental results of GAs in cyclic environments with noise with different noise probability p_n and $\tau = 50$ and $\rho = 0.1$ and 0.5.

Especially, the performance of memory based GAs decreases quite significantly with the rising of p_n. For example, on the *OneMax* function with $\rho = 0.5$, the performance of MIGA drops from 97.8 at $p_n = 0.01$ to 95.9 at $p_n = 0.02$, to 89.7 at $p_n = 0.05$, and to 82.9 at $p_n = 0.1$. This effect is natural since the higher degree of noise means the environment will return to a base state less accurately, which makes the memory scheme less efficient. However, for GAs without memory, that is, SGA, RIGA, and EIGA, the value of p_n is much less sensitive to their performance since they have no way to sense the accuracy of the environment returning to an old state. When p_n is big, the random immigrants scheme in RIGA may beat the memory scheme in some GAs. For example, when $p_n = 0.1$, RIGA outperforms MEGA on the *OneMax* function with $\rho = 0.5$.

Second, the value of p_n does not affect much the relative ranking of the performance of GAs on DOPs. However, p_n does affect the performance difference between GAs. In general, with the increase of p_n, the performance difference between GAs decreases. For example, on the *OneMax* function with $\rho = 0.5$, the performance improvement of MIGA over EIGA, that is, $\overline{F}_{\text{BOG}}(\text{MIGA}) - \overline{F}_{\text{BOG}}(\text{EIGA})$, decreases from $97.8 - 81.0 = 16.8$ at $p_n = 0.01$ to $82.9 - 81.3 = 1.6$ at $p_n = 0.1$.

5.4 Sensitivity Analysis on the Effect of Parameters r_i and p_m^i

There are several parameters within the investigated GAs, of which one key parameter is the immigrants ratio r_i that determines the degree of diversity introduced to the current

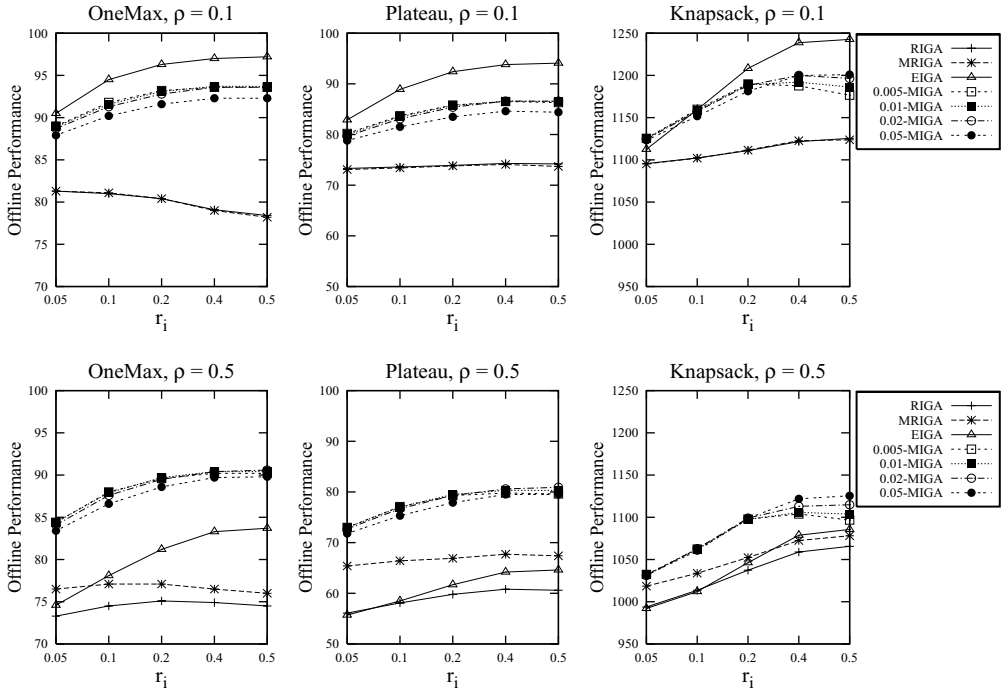

Figure 15: Experimental results of RIGA, MRIGA, EIGA, and MIGAs with different immigrants ratio on cyclic DOPs with noise with $\tau = 50$ and $\rho = 0.1$ and 0.5.

population by the immigrants and for MIGA the probability p_m^i of bitwise mutating the best memory point for immigrants determines the degree of diversity among the immigrants. In the basic experiments, we have set r_i and p_m^i to fixed values. In order to investigate the effect of immigrants ratio r_i on the performance of GAs and the effect of p_m^i on MIGA (and EIGA), we further carried out experiments on RIGA, MRIGA, EIGA, and MIGA on DOPs with $\tau = 50$ and $\rho = 0.1$ and 0.5. We now set r_i for RIGA, MRIGA, EIGA, and MIGA to 0.005, 0.1, 0.2, 0.4, and 0.5, respectively, and the population size n and memory size m for GAs were set according to Eq. (5) correspondingly. For example, for MIGA with $r_i = 0.5$, we set $n = 80$ and $m = 8$. For MIGA, we set p_m^i to 0.005, 0.01, 0.02, and 0.05, and for EIGA we still fix p_m^i to 0.01 since our preliminary experiments show that p_m^i has the similar effect on EIGA as on MIGA. The other experimental settings were the same as in the basic experiments.

The experimental results in cyclic environments with noise[2] are plotted in Figure 15, where MIGA with p_m^i is marked as p_m^i-MIGA. Several results can be observed from Figure 15 and are described as follows.

First, the immigrants ratio r_i does affect the performance of relevant GAs on the DOPs. Generally speaking, the performance of GAs increases when the value of r_i increases from 0.05 to 0.4 (with an exceptional case on the *OneMax* function with

[2]The experimental results in cyclic and random environments show similar observations and are not presented here.

$\rho = 0.1$ where the performance of RIGA and MRIGA decreases with the rising of r_i). For example, on the *OneMax* with $\rho = 0.5$, the performance of EIGA improves from 74.6 at $r_i = 0.05$ to 83.3 at $r_i = 0.4$. When r_i is further increased from 0.4 to 0.5, the performance of GAs does not increase much. For example, the performance of EIGA on the *OneMax* with $\rho = 0.5$ is 83.7 at $r_i = 0.5$, which is only a little better than the performance 83.3 at $r_i = 0.4$.

The degree of sensitivity of r_i on the performance of GAs depends on the DOP and the value of ρ. For example, MRIGA is much less sensitive to the value of r_i on the *OneMax* and *Plateau* functions than on the *Knapsack* problems. And EIGA is much less sensitive to r_i on DOPs with $\rho = 0.5$ than on DOPs with $\rho = 0.1$.

Second, the effect of p_m^i is relatively smaller than the effect of r_i on the performance of MIGAs. For example, on the *Knapsack* problem with $\rho = 0.5$, the maximum performance improvement of MIGAs with fixed r_i happens at $r_i = 0.5$ where $\overline{F}_{\text{BOG}}(0.05\text{-MIGA}, r_i = 0.5) - \overline{F}_{\text{BOG}}(0.005\text{-MIGA}, r_i = 0.5) = 1125.4 - 1096.7 = 28.7$. On the contrary, the maximum performance improvement of MIGAs with fixed p_m^i occurs at $p_m^i = 0.05$ where $\overline{F}_{\text{BOG}}(0.05\text{-MIGA}, r_i = 0.5) - \overline{F}_{\text{BOG}}(0.05\text{-MIGA}, r_i = 0.05) = 1125.4 - 1031.3 = 94.1$.

The effect of p_m^i on the performance of MIGAs depends on the dynamic test problems. With the increasing of the value of p_m^i, the performance of MIGAs decreases on the dynamic *OneMax* and *Plateau* functions, while increases on the dynamic *Knapsack* problems.

Third, comparing Figure 15 with Figures 7, 8, 9, and 14, it can be seen that the effect of r_i and p_m^i on the performance of relevant GAs is much less significant than the effect of the environmental dynamics parameters τ, ρ, and p_n.

5.5 Experiments on Random DOPs with Random Severities of Changes

In the basic experiments, we have investigated the performance of GAs under three kinds of dynamic environments with different but fixed values of ρ. However, for real world problems, the environment may be subject to different severities of changes over time. That is, each environment change may involve different values of ρ. In order to study the performance of GAs in dynamic environments with random degrees of changes, experiments are further carried out on random DOPs with $\tau = 50$ and the value of ρ randomly generated with a uniform distribution in $[0.0, 1.0]$ (i.e., $\rho = rand(0.0, 1.0)$) for each environmental change. Here, the experimental settings, including genetic operators and relevant parameters for GAs and the performance measure, are the same as those for the basic experiments and 50 runs of each GA on a DOP are also executed.

The experimental results regarding the offline performance and the standard deviation are presented in Table 4. The statistical results of comparing GAs by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are also given in Table 4. The dynamic performance of a typical run of GAs on the dynamic *Knapsack* problem with $\tau = 50$ and $\rho = rand(0.0, 1.0)$ for the first 15 environments is plotted in Figure 16(a) and the value of ρ for each environmental change for this typical run is shown in Figure 16(b).

The results in Table 4 basically match our previous analysis regarding the comparison of GAs. It is interesting to see that MIGA outperforms EIGA. This seems contrary to the result in our basic experiments under random environments; see the t-test results regarding MIGA – EIGA in Table 3 and Table 4. This result can be explained via observing the dynamic behavior of GAs in a typical run shown in Figure 16. Whenever the environment involves a significant change, for example, $\rho = 0.91$ at generation 150,

Table 4: The Experimental Results on Random DOPs with $\tau = 50$ and $\rho = rand(0.0, 1.0)$

Function	*OneMax*	*Plateau*	*Knapsack*
Offline Performance (Standard Deviation)			
SGA	72.7 (0.45)	57.2 (0.75)	1010.8 (4.85)
RIGA	76.5 (0.25)	63.8 (0.53)	1052.7 (3.38)
MEGA	74.6 (0.32)	61.9 (0.52)	1029.6 (3.14)
MRIGA	76.7 (0.30)	64.7 (0.62)	1060.8 (3.83)
MSGA	75.6 (0.32)	62.9 (0.42)	1038.1 (2.88)
EIGA	81.6 (0.64)	65.6 (1.04)	1069.7 (5.77)
MIGA	82.6 (0.53)	66.8 (0.89)	1092.7 (5.09)
t-test Result			
RIGA − SGA	$s+$	$s+$	$s+$
MEGA − SGA	$s+$	$s+$	$s+$
MRIGA − RIGA	$s+$	$s+$	$s+$
MRIGA − MEGA	$s+$	$s+$	$s+$
MSGA − MRIGA	$s-$	$s-$	$s-$
EIGA − RIGA	$s+$	$s+$	$s+$
EIGA − MRIGA	$s+$	$s+$	$s+$
EIGA − MSGA	$s+$	$s+$	$s+$
MIGA − RIGA	$s+$	$s+$	$s+$
MIGA − MEGA	$s+$	$s+$	$s+$
MIGA − MRIGA	$s+$	$s+$	$s+$
MIGA − MSGA	$s+$	$s+$	$s+$
MIGA − EIGA	$s+$	$s+$	$s+$

the performance of EIGA drops significantly because the elite from the previous generation does not fit at all in the new environment. On the contrary, the best point in the memory in MIGA may be much fitter in the new environment than the elite from the previous generation, which leads to a much less significant drop of the performance of MIGA. This gives MIGA an overall better performance over EIGA. When the environment involves a slight change, for example, $\rho = 0.08$ at generation 550, the performance of MIGA and EIGA only drops slightly and then quickly rises, due to the embedded elitism-based immigrants, to a higher level than achieved at the end of the previous environment. This leads to better performance of both MIGA and EIGA over other GAs.

6 Conclusions and Future Work

Random immigrants and memory are two major approaches developed for GAs to address dynamic environments. This paper investigates a memory-based immigrants scheme and an elitism-based immigrants scheme for GAs in dynamic environments, where the best memory point or elite from the previous generation is used as the base to generate immigrants via mutation. This way, the immigrants may be more adapted and hence more efficient for the current environment. Using the extended XOR generator (Yang and Yao, 2008), dynamic test problems under three kinds of environments, that is, cyclic, cyclic with noise, and random, were systematically constructed. Experiments were carried out to compare the performance of the memory- and elitism-based immigrants GAs against several peer GAs. From the experimental results and analysis, several conclusions can be drawn on the dynamic test problems.

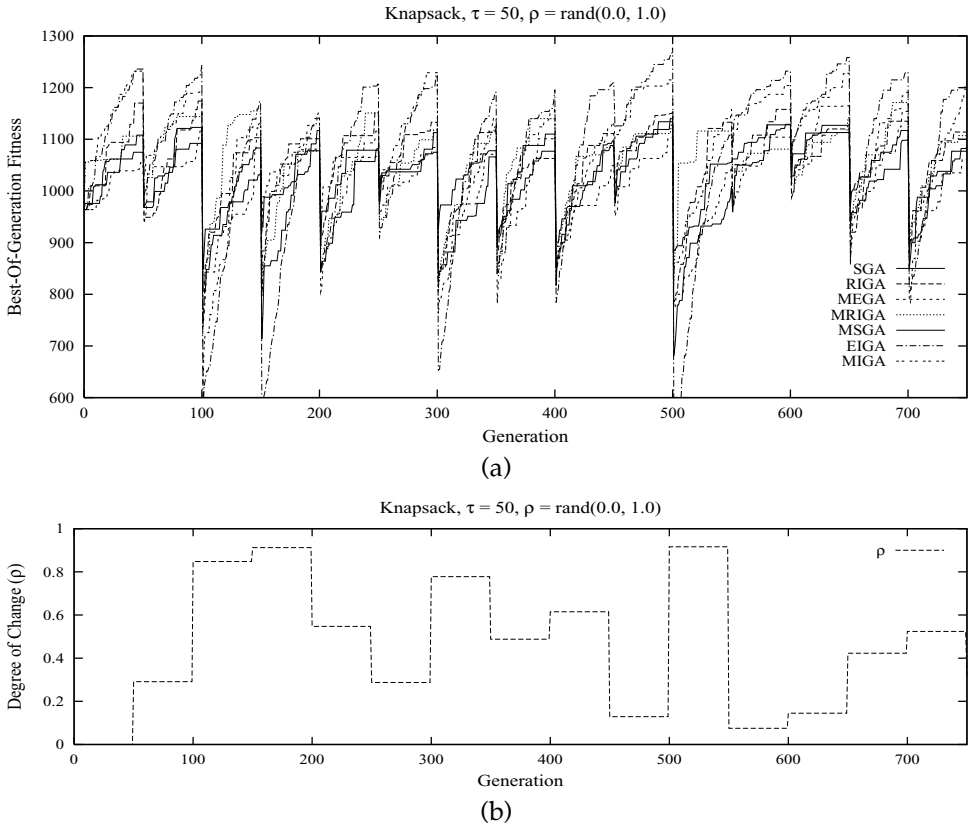

Figure 16: Dynamic behavior of a typical run of GAs on the dynamic *Knapsack* problem with $\tau = 50$ and $\rho = rand(0.0, 1.0)$ for the first 15 environments: (a) dynamic performance and (b) the value of ρ for each environmental change.

First, the memory-based immigrants scheme combines the principles of memory and random immigrants and consistently improves the performance of GAs in dynamic environments. On the other hand, the elitism-based immigrants scheme has inconsistent effect on the performance of GAs in dynamic environments. When the environment involves slight changes (i.e., small ρ) consistently, EIGA outperforms MIGA.

Second, random immigrants are usually beneficial to improve the performance of GAs in dynamic environments. The immigrants ratio generally has a significant effect on the performance of GAs and can be set to the range of [0.2, 0.4]. However, maintaining a high diversity level in the population, though usually useful, may not lead to better performance for GAs in dynamic environments. It is problem dependent. On the contrary, the traditional memory scheme has a more consistently positive effect than the random immigrants scheme on the performance of GAs in dynamic environments.

Third, introducing an extra population with restart scheme into memory based GA improves the performance in dynamic environments. However, the effect may not be as strong as the effect of combining the random immigrants scheme with memory on the performance of GAs in dynamic environments.

Fourth, the difficulty of DOPs for GAs depends on the difficulty of the base function and depends significantly on the environmental dynamics, for example, the speed,

severity, and cyclicity of environmental changes. Under cyclic with noise environments, the degree of noise significantly affects the performance of memory-based GAs and when the degree of noise reaches a certain level, DOPs may become harder than corresponding DOPs under random environments for GAs. This is due to the fact that for a fixed ρ with a random environment, the number of bits to be flipped by the DOP generator for a new environment is constant, while for a cyclic with noise environment, it is not, and in random environments, some previous changes may be reversed.

Fifth, the internal parameters, for example, r_i and p_m^i, of GAs do affect the performance of GAs in dynamic environments. However, their effect is generally less significant than the effect of external environmental dynamics parameters, for example, τ, ρ, p_n, and cyclicity of environments. And the difference between different parameter settings of an algorithm is much less significant than the difference between different algorithms. This means that the type of algorithm is more important.

Generally speaking, the experimental results indicate that MIGA can be a good optimizer under dynamic environments and EIGA can be another good choice under dynamic environments where the environment is subject to slight and slow changes.

This paper, though experimentally validating the efficiency of MIGA and EIGA for DOPs, also inspires several topics for future research. First, the experimental results show that higher diversity schemes lead to a good performance on some DOPs and/or within some GAs while a bad performance on other DOPs and/or within other GAs. This contrary phenomenon surely deserves further research in order to achieve efficient diversity for GAs in dynamic environments.

Second, another interesting work is to investigate the effect of combining the memory- and elitism-based immigrants schemes with other approaches, for example, multi-population (Branke et al., 2000) and adaptive selection (Yang and Tínos, 2008) schemes, for GAs for problem optimization in dynamic environments.

Third, the memory schemes studied in this paper are mainly explicit direct memory schemes. It is worth carrying out a higher level of comprehensive comparison of memory-based evolutionary algorithms (EAs), including GAs studied in this paper, GAs with implicit memory schemes, such as the diploidy GAs (Ng and Wong, 1995; Uyar and Harmanci, 2005), and EAs with associative memory schemes, such as the associative memory-based PBIL algorithms (Yang, 2005b; Yang and Yao, 2008), for dynamic optimization problems.

Finally, although the environmental dynamics parameters have a larger impact on the performance of GAs in dynamic environments than the algorithm parameters, it is still important to tune the algorithm parameters. This is because, usually, a dynamic problem with its environmental dynamics parameters has to be accepted as given. All one can do is try to find the best algorithm for the particular problem, and parameter settings are part of the algorithm. However, instead of tuning algorithm parameters with a prescribed policy for DOPs, it may be more important to devise algorithms that can capture the environmental dynamics and adapt to the dynamic environments accordingly. This will be an important issue to pursue in the future.

Acknowledgments

The author would like to thank the anonymous action editor and reviewers for their thoughtful suggestions and constructive comments. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom under Grant EP/E060722/01.

References

Baluja, S. (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, Pittsburgh, PA.

Bendtsen, C. N. and Krink, T. (2002). Dynamic memory model for non-stationary optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 145–150.

Branke, J. (1999). Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1875–1882.

Branke, J. (2002). *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Branke, J., Kaußler, T., Schmidth, C., and Schmeck, H. (2000). A multi-population approach to dynamic optimization problems. In *Proceedings of the 4th International Conference on Adaptive Computing in Design and Manufacturing*, pages 299–308.

Cobb, H. G. and Grefenstette, J. J. (1993). Genetic algorithms for tracking changing environments. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 523–530.

Dasgupta, D. and McGregor, D. (1992). Nonstationary function optimization using the structured genetic algorithm. In R. Männer and B. Manderick, editors, *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature*, pages 145–154.

Goldberg, D. E. and Smith, R. E. (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pages 59–68.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In R. Männer and B. Manderick, editors, *Proceedings of the 2nd International Conference on Parallel Problem Solving from Nature*, pages 137–144.

Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 3, pages 2031–2038.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Lewis, E. H. J. and Ritchie, G. (1998). A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 139–148.

Louis, S. J. and Xu, Z. (1996). Genetic algorithms for open shop scheduling and re-scheduling. In *Proceedings of the 11th ISCA International Conference on Computers and their Applications*, pages 99–102.

Mori, H. K. N. and Nishikawa, Y. (1997). Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 299–306.

Morrison, R. W. and De Jong, K. A. (1999). A test problem generator for non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation*, Vol. 3, pages 2047–2053.

Morrison, R. W. and De Jong, K. A. (2000). Triggered hypermutation revisited. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1025–1032.

Morrison, R. W. (2004). *Designing Evolutionary Algorithms for Dynamic Environments*. Springer-Verlag, Berlin.

Ng, K. P. and Wong, K. C. (1995). A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 159–166.

Ramsey, C. L. and Greffenstette, J. J. (1993). Case-based initialization of genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 84–91.

Simões, A. and Costa, E. (2001). On biologically inspired genetic operators: Using transformation in the standard genetic algorithm. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 584–591.

Simões, A. and Costa, E. (2003a). An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory. In *Proceedings of the 6th International Conference on Neural Networks and Genetic Algorithms*, pages 168–174.

Simões, A. and Costa, E. (2003b). Improving the genetic algorithm's performance when using transformation. In *Proceedings of the 6th International Conference on Neural Networks and Genetic Algorithms*, pages 175–181.

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

Trojanowski, K. and Michalewicz, Z. (2000). Evolutionary optimization in non-stationary environments. *Journal of Computer Science and Technology*, 1(2): 93–124.

Uyar, A. Ş. and Harmanci, A. E. (2005). A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. *Soft Computing*, 9(11): 803–815.

Vavak, F. and Fogarty, T. C. (1996). 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 1143*, pages 297–304, Springer, Berlin.

Weicker, K. (2003). *Evolutionary Algorithms and Dynamic Optimization Problems*. Der andere Verlag, Osnabrück, Germany.

Yang, S. (2003). Non-stationary problem optimization using the primal-dual genetic algorithm. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Vol. 3, pages 2246–2253.

Yang, S. (2005a). Memory-based immigrants for genetic algorithms in dynamic environments. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, Vol. 2, pages 1115–1122.

Yang, S. (2005b). Population-based incremental learning with memory scheme for changing environments. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, Vol. 1, pages 711–718.

Yang, S. (2005c). Memory-enhanced univariate marginal distribution algorithms for dynamic optimization problems. In *Proceedings of the 2005 Congress on Evolutionary Computation*, Vol. 3, pages 2560–2567.

Yang, S. (2007). Genetic algorithms with elitism-based immigrants for changing optimization problems. In *Applications of Evolutionary Computing, Lecture Notes in Computer Science 4448*, pages 627–636.

Yang, S. and Tinós, R. (2008). Hyper-selection in dynamic environments. *Proceedings of the 2008 Congress on Evolutionary Computation*, pages 3184–3191.

Yang, S. and Yao, X. (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11): 815–834.

Yang, S. and Yao, X. (2008). Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation*. IEEE Press, Piscataway, NJ, to appear.