Population-Based Incremental Learning With Associative Memory for Dynamic Environments

Shengxiang Yang, *Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—In recent years, interest in studying evolutionary algorithms (EAs) for dynamic optimization problems (DOPs) has grown due to its importance in real-world applications. Several approaches, such as the memory and multiple population schemes, have been developed for EAs to address dynamic problems. This paper investigates the application of the memory scheme for population-based incremental learning (PBIL) algorithms, a class of EAs, for DOPs. A PBIL-specific associative memory scheme, which stores best solutions as well as corresponding environmental information in the memory, is investigated to improve its adaptability in dynamic environments. In this paper, the interactions between the memory scheme and random immigrants, multipopulation, and restart schemes for PBILs in dynamic environments are investigated. In order to better test the performance of memory schemes for PBILs and other EAs in dynamic environments, this paper also proposes a dynamic environment generator that can systematically generate dynamic environments of different difficulty with respect to memory schemes. Using this generator, a series of dynamic environments are generated and experiments are carried out to compare the performance of investigated algorithms. The experimental results show that the proposed memory scheme is efficient for PBILs in dynamic environments and also indicate that different interactions exist between the memory scheme and random immigrants, multipopulation schemes for PBILs in different dynamic environments.**

***Index Terms*—Associative memory scheme, dynamic optimization problems (DOPs), immune system-based genetic algorithm (ISGA), memory-enhanced genetic algorithm, multipopulation scheme, population-based incremental learning (PBIL), random immigrants.**

I. Introduction

EVOLUTIONARY algorithms (EAs) are a class of metaheuristic algorithms inspired by principles of natural evolution, such as selection and population genetics. Traditionally, research on EAs has been focused on stationary optimization problems, where problems are precisely given in advance and remain fixed during the evolutionary process. Due to their ease of use and good performance, EAs have been widely applied for solving many stationary optimization problems [1], [23], [48]. However, the environments of many real-world problems are often dynamic [8], [18], [19], [28]. For these dynamic optimization problems (DOPs), the fitness function, design variables, and environmental conditions may change over time due to such factors as the stochastic arrival of new tasks, machine faults and degradation, climatic change, market fluctuation, and economic and financial factors.

Usually, for stationary optimization problems, the aim is to design EAs that can quickly and precisely locate the optimum solution(s) in the search space. However, for DOPs, the situation is quite different. For DOPs, fast and precise EAs are not sufficient and, in fact, in some cases perform worse than their slower and less precise peer EAs due to the convergence problem. For traditional (fast and precise) EAs when the environment changes, the population may have converged to some optimum solutions or areas and hence is trapped there and cannot perform well in the new environment. Hence, for DOPs what is more important is to develop algorithms that can track and adapt to the changing environment. Though this poses great challenges to traditional EAs, EAs with proper enhancement can also be good solvers for DOPs due to their intrinsic inspiration from natural evolution, which is itself always subject to an ever-changing environment.

In recent years, studying EAs for DOPs has attracted a growing interest due to its importance in EA's real-world applications. The simplest way of addressing DOPs is to restart EAs from scratch whenever an environmental change is detected. Though the restart scheme really works for some cases [41], for many DOPs, it is more efficient to develop other approaches that make use of knowledge gathered from old environments. Several approaches have been developed into EAs to address DOPs. These approaches include maintaining and reintroducing diversity during the run [9], [16], [27], [35], memory schemes [6], memory and diversity hybrid schemes [33], [43], [46], and multipopulation schemes [7].

Population-based incremental learning (PBIL) algorithms were first proposed by Baluja [2] as an abstraction of genetic algorithms (GAs), which explicitly maintain the statistics contained in a GA's population [3]. As a class of EAs, PBILs have proved to be very successful on numerous stationary benchmark and real-world problems [21]. Recently, Yang and Yao [41] have investigated PBILs for DOPs by introducing dualism and a scheme similar to the random immigrants method [16] to improve their performance in dynamic environments, and in [42], a memory scheme has been introduced into PBILs for DOPs with some preliminary results.

In this paper, the PBIL-specific explicit memory scheme introduced in [42] is further investigated to improve its adaptability in dynamic environments. With this memory scheme,

Manuscript received July 1, 2005; revised January 4, 2007. First published February 22, 2008; current version published October 1, 2008. This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) of the U.K. under Grant EP/E060722/1 and Grant EP/E058884/1 and in part by the National Natural Science Foundation (NNSF) of China under Grant 60428202.

S. Yang is with the Department of Computer Science, University of Leicester, Leicester LE1 7RH, U.K. (e-mail: s.yang@mcs.le.ac.uk).

X. Yao is with the Cercia, School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K., and also with the Nature Inspired Computation and Applications Laboratory (NICAL), University of Science and Technology of China (USTC), Hefei, Anhui 230027, China (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TEVC.2007.913070

the best sample created by the working probability vector together with the probability vector are stored in the memory in a certain time and space pattern. When an environmental change is detected, the probability vector associated with the memory sample that is reevaluated to be the best in the new environment is retrieved to compete with the current working probability vector for further iterations. This paper also investigates the interactions between the memory scheme and random immigrants, multipopulation, and restart schemes for PBILs in dynamic environments.

In order to test the investigated PBILs for DOPs, a DOP generator that aims to systematically construct dynamic environments for testing EAs, especially EAs with memory schemes, is also proposed in this paper. Based on this generator, a series of DOPs are constructed from three stationary functions as the dynamic test environments and experimental study is carried out to compare the performance of investigated PBILs and two state-of-the-art memory-enhanced GAs: the memory/search GA by Branke [6] and an immune system based GA recently developed in [44].

The rest of this paper is organized as follows. Section II reviews existing memory approaches for EAs in dynamic environments. Section III details several EAs investigated in this paper, including our memory-enhanced PBILs. Section IV first briefly reviews existing DOP generators, then presents our proposed dynamic environment generator for testing memory schemes for EAs, and finally describes the dynamic test environments constructed for the experimental study of this paper. The basic experimental results and analysis regarding the proposed memory scheme and random immigrants for PBILs are presented in Section V. Section VI studies the effect of multipopulation and restart schemes on the memory scheme for PBILs in dynamic environments. Finally, Section VII concludes this paper with discussions on future work.

II. Memory Schemes for Evolutionary Algorithms (EAs) in Dynamic Environments

The application of memory schemes has proved to be able to enhance EA's performance in dynamic environments, especially when the environment changes cyclicly in the search space.[1] The basic principle of memory schemes is to store information, such as good solutions, from the current environment and reuse it later in new environments. This useful information can be stored in two ways: by implicit memory mechanisms and by explicit memory mechanisms [6].

A. Implicit Memory Schemes

For implicit memory schemes, EAs use genotype representations that contain redundant information to store good (partial) solutions to be reused later. Here, the redundant representation acts as memory, which is implicit for the EA to use appropriately. Typical examples of implicit memory schemes are GAs based on diploidy or multiploidy representations. Goldberg and Smith [14] first extended the simple haploid GA to a diploid GA with a tri-allelic dominance scheme. Thereafter, Ng and Wong [30] proposed a dominance scheme with four alleles for a diploidy-based GA. Lewis *et al.* [22] further investigated an additive diploidy scheme where a gene becomes 1 if the addition of all alleles exceeds a certain threshold, or 0 otherwise. Recently, Uyar and Harmanci [38] proposed an adaptive dominance change mechanism for diploid GAs, where the dominance characteristics for each locus is dynamically adapted through feedback from the current population.

In addition to multiploidy GAs, Dasgupta and McGregor [11] proposed a quite different implicit memory scheme in the so called structured GA, which is haploid based but has a multileveled structure. In this representation, high-level genes can regulate the activation of a set of low-level genes. The set of low-level genes can memorize good (partial) solutions in old environments that can be reactivated by high-level genes in new environments. Similar to diploid GAs, recently Yang and Yao [41] proposed a dual PBIL for dynamic problems inspired by the principle of dualism in nature.

B. Explicit Memory Schemes

While implicit memory schemes for EAs in dynamic environments depend on redundant representations to store useful information for EAs to exploit during the run, explicit memory schemes make use of precise representations but split an extra storage space where useful information from the current generation can be explicitly stored and reused in later generations or environments. Explicit memory schemes mainly involve three concerns: what to store in the memory, how to organize and update the memory, and how to retrieve the memory.

For the first concern, a natural choice is to store good solutions and reuse them when the environment change is detected. This can be called *direct memory scheme*. For example, Louis and Xu [24] studied the open shop rescheduling problem. They used a memory to store best individuals during a run. Whenever a change (in a known pattern) occurs, the GA is restarted from a population with partial (5%–10%) individuals retrieved from the memory corresponding to the previous run, while the rest is initialized randomly. The authors reported significant improvements of their GA over the GA with totally random restart scheme. Instead of storing good solutions only, information that associates good solutions with their environments can also be stored with good solutions. This information can be used for similarity measure to associate a new environment with certain stored good solutions and then reuse these associated solutions more efficiently. This can be called *indirect memory scheme* or *associative memory scheme*. For example, Ramsey and Grefenstette [31] studied a GA for a robot control problem, where good candidate solutions are stored in a permanent memory together with information about the robot's current environment. When the robot incurs a new environment that is similar to a stored environment instance, the associated stored controller solution is reactivated. This scheme was reported to yield significant improvements. Recently, an associative memory scheme has been developed for PBILs for DOPs [42] with some promising preliminary results, which will be further investigated in this paper and will be described in details in Section III-B.

The memory space or size is usually limited (and fixed) for computational and searching efficiency. This leads to the second

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

concern of explicit memory schemes: memory organization and updating mechanisms. As to the memory organization, there exist two mechanisms: *local mechanism*, where the memory is individual-oriented and *global mechanism*, where the memory is population-oriented. Trojanowski and Michalewicz [36], [37] introduced a local memory approach, where for each individual the memory stores a number of its ancestors. When the environment changes, the current individual and its ancestors are reevaluated and compete together with the best becoming the active individual while the others are stored in the memory. The global memory mechanism is more natural and popular, see [6], [25]. In these global memory mechanisms, the best individual of the population is stored in the memory of certain generations, while deleting one individual from the memory according to some measure.

As to the memory updating mechanism, a general principle is to select one memory individual to be removed for or updated by the best individual from the population in order to make the stored individuals to be of above average fitness, not too old, and distributed across several promising areas of the search space. Branke [6] has discussed several memory replacement strategies: 1) replacing the least important one with the importance value of individuals being the linear combination of age, contribution to diversity, and fitness; 2) replacing the one with the least contribution to memory variance; 3) replacing the most similar one if the new individual is better; and 4) replacing the less fit of a pair of memory individuals that has the minimum distance among all pairs. The third strategy seems the most practical one and will be used in this paper. In addition to replacing memory point, Bendtsen and Krink [5] proposed a different memory updating scheme, where the memory individual closest to the best population individual is, instead of being removed from the memory, moved toward the best population individual.

For the third concern on explicit memory, i.e., how to retrieve the memory, a natural idea is to retrieve the best memory individual(s) to replace the worst individual(s) in the population. This can be done every generation or only when a change occurs. The memory retrieval is sort of coupled with the above two concerns. For example, for the direct memory scheme, the whole memory individuals may enter the new population as in [24] or compete with the population individuals for the new population as in [6], while for the associative memory scheme only associated memory individual(s) may enter the new population [31], and for the local memory organization scheme, the best ancestor of an active individual competes with it to become active in the population [36], while for the global memory scheme, the best memory individual(s) may compete with all individuals in the main population.

The memory retrieval has also been combined with diversity schemes to improve the performance of GAs for DOPs, which shows promising results. For example, Simões and Costa [33], [34] have proposed an immune system based GA for DOPs, where new individuals are cloned from selected memory solutions and replaced into the population. Recently, Yang has developed a memory-based immigrants scheme for GAs in dynamic environments [43], [46], where the best memory solution is retrieved every generation as the base to create new individuals via a normal bit flip mutation operation to replace worst individuals in the main population.

$t := 0$ and initialize probability vector $\vec{P}(0) := \vec{0.5}$
generate a set $S(0)$ of n samples by $\vec{P}(0)$
repeat
  evaluate samples in $S(t)$
  if *random immigrants used* **then** // *for SPBILi*
    replace $r_i * n$ worst samples in $S(t)$ by random ones
  learn $\vec{P}(t)$ toward best sample $\vec{B}(t)$ in $S(t)$ by Eq. (1)
  mutate $\vec{P}(t)$ by Eq. (2)
  generate a set $S(t)$ of n samples by $\vec{P}(t)$
until *termination condition holds* // *e.g.,* $t > t_{max}$

Fig. 1. Pseudocode of the standard PBIL without random immigrants (SPBIL) and the PBIL with random immigrants (SPBILi).

III. Description of Algorithms Investigated

A. Standard Population-Based Incremental Learning (PBIL)

The PBIL algorithm is a combination of evolutionary optimization and competitive learning [2]. PBIL aims to generate a real-valued probability vector $\vec{P} = \{P_1, \ldots, P_l\}$ (l is the binary-encoding length), which creates high-quality solutions with high probabilities when sampled.[2] The pseudocode for the standard PBIL, denoted *SPBIL*, is shown in Fig. 1.

The standard PBIL starts from a probability vector that has a value of 0.5 for each bit location. This probability vector is called the *central probability vector* since it falls in the central point of the search space. Sampling this initial probability vector creates random solutions because the probability of generating a 1 or 0 on each locus is equal. At iteration t, a set $S(t)$ of n solutions are sampled from the probability vector $\vec{P}(t)$. The samples are evaluated using the problem-specific fitness function. Then, the probability vector is learned towards the best solution $\vec{B}(t)$ of the set $S(t)$ as follows:

$$P_i(t+1) := (1-\alpha) * P_i(t) + \alpha * B_i(t), \quad i = \{1, \ldots, l\} \quad (1)$$

where α is the learning rate, which determines the distance the probability vector is pushed for each iteration.

After the probability vector is updated toward the best sample, in order to keep the diversity of sampling, it may undergo a bitwise mutation process [4]. Mutation is applied to PBILs studied in this paper and the mutation operation always changes the probability vector toward the central probability vector, i.e., the central point in the search space. The mutation operation is carried out as follows. For each locus $i = \{1, \ldots, l\}$, if a random number $r = rand(0.0, 1.0) < p_m$ (p_m is the mutation probability), then mutate P_i using the following formula:

$$P_i' = \begin{cases} P_i * (1.0 - \delta_m), & P_i > 0.5 \\ P_i, & P_i = 0.5 \\ P_i * (1.0 - \delta_m) + \delta_m, & P_i < 0.5 \end{cases} \quad (2)$$

[2]A solution is sampled from a probability vector $\vec{P}$ as follows: for each locus i, if a randomly created number $r = rand(0.0, 1.0) < P_i$, it is set to 1; otherwise, it is set to 0.

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

As the search progresses, the elements in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0, representing high evaluation solutions. The search progress stops when some termination condition is satisfied, e.g., the maximum allowable number of iterations $t_{\max}$ is reached or the probability vector is converged to either 0.0 or 1.0 for each bit position.

In this paper, we also investigate the effect of random immigrants on the performance of PBILs in dynamic environments. In [41], a technique similar to random immigrants is used for PBILs for DOPs by adding a subpopulation that is sampled by the central probability vector. In this paper, we use an equivalent but more direct random immigrants scheme for PBILs. With this random immigrants scheme, for each iteration after the probability vector is sampled, a set of worst samples are selected and replaced with randomly created samples. The pseudocode for the PBIL with random immigrants, denoted *SPBILi*, is also shown in Fig. 1, where r_i is the ratio of random immigrants to the total population size.

B. Memory-Enhanced PBILs

As reviewed in Section II, several memory schemes have been developed for EAs to deal with DOPs. In [41], Yang and Yao proposed a dualism-based PBIL for dynamic problems where a dual probability vector, which is symmetric to the main probability vector with respect to the central point in the search space, is associated and competes with the main probability vector to generate samples. The dual PBIL has proved successful in dynamic environments where significant changes exist in the genotypic space.

In this paper, we investigate a new explicit associative memory scheme for PBILs in dynamic environments. The key idea is to store good solutions, as well as associated environmental information in the memory for PBIL to reuse. Since PBILs aim to evolve a probability vector toward the intrinsic allele distribution of the current environment, the evolved probability vector can be taken as the natural representation of the current environmental information and can be stored together with the best sample generated from it in the memory.

The pseudocode for the memory-enhanced PBILs without and with random immigrants, denoted MPBIL and MPBILi, respectively, is shown in Fig. 2, where n is the total number of samples per iteration including the memory samples and $f(X)$ denotes the fitness of individual X. Within MPBIL and MPBILi, a memory of size $m = 0.1 * n$ is used to store samples and probability vectors. Each memory point consists of a pair: a sample and its associated probability vector. The most similar measure, as discussed in [6], is used as the memory replacement strategy. That is, when the memory is due to update, we first find the memory point with its sample $\vec{B}_M(t)$ closest to the best population sample $\vec{B}(t)$ in terms of the Hamming distance. If the best population sample has a higher fitness than this memory sample, it is replaced by the best population sample; otherwise, the memory remains unchanged. When a best population sample $\vec{B}(t)$ is stored in the memory, the current working probability vector $\vec{P}(t)$ that generates $\vec{B}(t)$ is also stored in the memory and is associated with $\vec{B}(t)$. Similarly, when replacing a memory point, both the sample and the associated probability vector within the memory point are replaced by the best population sample and the working probability vector, respectively.

```
t := 0 and t_M := rand(5, 10)
initialize prob. vector P(0) := 0.5 and memory M(0) := φ
generate a set S(0) of n − m samples by P(0)
repeat
  evaluate samples in S(t) and M(t)
  denote the best memory sample by B_M(t) and its
    associated prob. vector by P_M(t)
  if random immigrants used then    // for MPBILi
    replace r_i * n worst samples in S(t) by random ones
  denote the best sample in S(t) by B(t)
  if t = t_M then        // time to update memory
    t_M := t + rand(5, 10)
    if memory not full then store B(t), P(t) into M(t)
    else find the memory sample C_M(t) closest to B(t)
          and its associated prob. vector P_C(t)
      if f(B(t)) > f(C_M(t)) then
        C_M(t) := B(t) and P_C(t) := P(t)
  if environmental change detected then
    if f(B_M(t)) > f(B(t)) then P(t) := P_M(t)
  else learn P(t) toward B(t) by Eq. (1)
  mutate P(t) by Eq. (2)
  generate a set S(t) of n − m samples by P(t)
until termination condition holds    // e.g., t > t_max
```

Fig. 2. Pseudocode of the memory-enhanced PBILs: without random immigrants (MPBIL) and with random immigrants (MPBILi).

Instead of updating the memory in a fixed time interval as in other memory-enhanced EAs in the literature, the memory in MPBIL and MPBILi and other memory-enhanced EAs studied in this paper is updated using a stochastic time pattern as follows. After each memory updating, a random integer $R \in [5, 10]$ is generated to determine the next memory updating time t_M. For example, suppose a memory updating happens at generation t, then the next memory updating time is $t_M = t + R = t + rand(5, 10)$. This stochastic time pattern has two advantages over a fixed time pattern in terms of fairly comparing EAs with memory schemes. First, different memory-enhanced EAs may favor different fixed memory updating intervals. The stochastic time pattern can smooth away this potential effect. Second, the environmental change period is unknown before an EA is running or may be faulty to detect. Different fixed updating intervals will have a different effect even for the same memory-enhanced EA. It would be ideal that the environmental change period coincides with the memory updating period, e.g., the memory is updated just before the

environment changes. However, for a fair comparison among EAs with and without memory, this potential effect should be smoothed away, which can be achieved by the stochastic memory updating time pattern.

The memory in the memory-enhanced PBILs is reevaluated every iteration. If any memory sample has its fitness changed, the environment is detected to be changed. Then, the memory probability vector associated with the best reevaluated memory sample will replace the current working probability vector if the best memory sample is fitter than the best sample created by the current working probability vector. If no environmental change is detected, MPBIL and MPBILi progress just as the standard PBIL does.

From the above description, it can be seen that the proposed memory scheme for PBILs uses similar ideas as the memory scheme devised by Ramsey and Grefenstette [31] for GAs, i.e., storing environmental information in the memory. However, the stored environmental information is reused in a different way. For MPBIL and MPBILi, the stored environmental information, the probability vector, is used to directly reactivate an old environment it represents for MPBIL and MPBILi, which may be similar to the newly changed problem environment, and the stored solutions, besides their role as environmental change detectors and memory replacement locators, are used to indicate which associated environment should be reactivated.

C. Immune System-Based Genetic Algorithm (GA)

The human immune system protects our body against potentially harmful pathogens, called *antigens*. Our body maintains a large number of immune cells. Some belong to the adaptive immune system, called *lymphocytes*, which circulate through the body. There are two types of lymphocytes, namely, *T-cells* and *B-cells*, which cooperate in the immune response with different roles [20].

When a pathogen invades the body for the first time, a few B-cells can recognize its peptides and will be activated to respond as follows. When a B-cell is activated, it proliferates and produces many short-lived clones through cell division. B-cell cloning is subject to a form of mutation termed *somatic hypermutation*. The mutated B-cell clones will undergo a differentiation process. Those clones that have a low affinity to the antigen will die, while those with a high affinity will survive and differentiate into *plasma* or *memory B-cells*. Plasma B-cells secrete antibodies that can bind to the antigen and destroy or neutralize it. This process is called the *primary response*. Meanwhile, memory B-cells will retain in the circulation. If the same pathogen attacks the body again, the memory B-cells can respond immediately. This is called the *second response*, which is much faster and more efficient than the primary response. The immune system can recognize a large number of antigens because it has a gene library that aggregates modular chunks of genes or gene segments. These gene segments can be recombined to build up diverse antibodies.

The mechanisms of memory and diversity in the human immune system have been applied into GAs for DOPs, see [12] and [13]. Simões and Costa [33], [34] proposed an immune system-based GA for DOPs. The basic idea is to view the environment as the antigen and environmental changes as the appearance of different antigens. Their GA maintains two populations: the first one consists of plasma B-cell individuals, while the second consists of memory B-cell individuals. The first population is the main one and evolves as follows: the individuals with the best matches to the optimum (antigen) are selected and cloned into the next generation. At times, the best plasma B-cell individual is stored in the second population (and hence becomes a memory B-cell individual) and is attached a value of the average fitness of the first population, which is used as the affinity measurement to match memory B-cells to a new environment. The degradation of the population average fitness is taken as the environmental change detection mechanism. When a change is detected, the most proximal memory B-cell[3] is then activated, cloned, and reintroduced into the first population, replacing the worst individuals.

Simões and Costa used a set of gene libraries, each containing a set of fixed length gene segments. The libraries are randomly initialized and then kept constant during the running of the GA. They are used in the cloning process. During the cloning, every individual, be it a plasma or memory B-cell, is subject to a *transformation* modification with a probability p_t. Transformation, proposed by Simões and Costa in [32], is similar to the somatic hypermutation of B-cells. An individual is transformed as below. First, one gene segment is randomly selected from one randomly chosen gene library. Then, a random transformation locus is chosen in the individual. Finally, the chosen gene segment is incorporated into the individual, replacing the genes after the transformation locus.

In [44], a variant of Simões and Costa's immune system-based GA was proposed, which is studied as a peer GA in this paper and denoted ISGA. ISGA significantly outperforms Simões and Costa's GA according to the experiments [44]. The pseudocode of ISGA is shown in Fig. 3. ISGA differs from Simões and Costa's GA in four aspects.

First, ISGA uses a gene pool (instead of gene libraries) to hold a set of fixed length gene segments. The gene segments in the gene pool are divided into two groups: random and nonrandom. Both groups are randomly initialized and then updated every generation. The gene segments in the random group are just randomly reinitialized, while those in the nonrandom group are updated according to the current plasma B-cell population using a binary tournament selection as follows. For each gene segment in the nonrandom group, we first randomly select two individuals from the plasma B-cell population, and then from the fitter individual we select a contiguous segment of genes of fixed length from a random locus as the new gene segment and the starting locus is recorded and associated with the new gene segment.

Second, ISGA uses an *aligned transformation* scheme. When cloning an individual, we first randomly select a gene segment from the gene pool. If it is from the random group, it will be replaced into the individual from a random locus; otherwise, it will be replaced into the individual from the recorded starting locus.

Third, in the ISGA, the memory is updated similarly as in MPBIL and is reevaluated every generation to detect environmental changes. If an change is detected, the memory individual with the highest reevaluated fitness is retrieved to clone $r_i * n$ (r_i

[3]The proximity is measured by the average fitness of the first population and the value attached to each memory B-cell.

```
t := 0 and t_M := rand(5, 10)
initialize population P(0) and memory M(0) randomly
initialize gene pool G(0) randomly
repeat
  evaluate population P(t) and memory M(t)
  replace the worst member in P(t) by elite from P(t−1)
  update gene pool G(t)
  if environmental change detected then
    retrieve the best memory point B_M(t)
    clone r_i * n individuals from B_M(t)
    replace the worst individual in P(t) by the clones
  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 C_M(t) := B_P(t)
  P'(t) :=selectForReproduction(P(t))
  clone(P'(t), G(t), p_t)  // p_t is the transformation prob.
  mutate(P'(t), p_m)       // p_m is the mutation prob.
until termination condition holds      // e.g., t > t_max
```

Fig. 3. Pseudocode of the investigated ISGA with aligned transformation.

is the clone immigrants ratio) individuals and replace the worst ones in the plasma B-cell population.

Fourth, in Simões and Costa's GA, mutation was not used. In ISGA, mutation is switched on, which gives better performance according to our preliminary experiments.

IV. Constructing Dynamic Test Environments

A. General Dynamic Environment Generators

Over the years in parallel with developing approaches into EAs for dynamic problems, researchers have also developed a number of dynamic problem generator to create dynamic test environments to compare the performance of developed approaches. Generally speaking, existing generators can be roughly divided into three types.

The first type of dynamic environment generators is quite simple and just switches between two or more stationary problems (or states of a problem). For example, the dynamic knapsack problem where the weight capacity of the knapsack oscillates between two or more fixed values has been frequently used in the literature [11], [22], [25], [30]. Cobb and Grefenstette [10] used a dynamic environment that oscillates between two different fitness landscapes. For this type of generator, the environmental dynamics is mainly characterized by the speed of change measured in EA generations.

The second type of generators construct dynamic environments by reshaping a predefined fitness landscape. Usually, this base landscape is defined in n-dimensional real space and consists of a number of component landscapes (e.g., cones), see [6], [17], and [36]. Each of the components can change its own morphology independently with such parameters as peak height, peak slope, and peak location, and the center of the peak with the highest height is taken as the optimum solution of the landscape. For example, Morrison and De Jong [26] defined the base landscape in the n-dimensional real space as

$$f(\vec{x}) = \max_{i=1,\ldots,m} \left[H_i - R_i \times \sqrt{\sum_{j=1}^{n} (x_j - X_{ij})^2} \right] \tag{3}$$

where $\vec{x} = (x_1, \ldots, x_n)$ is a point in the landscape, m specifies the number of cones, and each cone i is independently specified by its height H_i, its slope R_i, and its center $X_i = (X_{i1}, \ldots, X_{\text{in}})$. These cones are blended together by the $\max$ function. Based on this stationary landscape, dynamic problems can be created through changing the parameters of each component.

Recently, Yang [40] proposed a dynamic environment generator based on the concept of problem difficulty justified by Goldberg [15], claiming that the problem difficulty can be decomposed along the lines of building block processing into three core elements: *deception*, *scaling*, and *exogenous noise*. A framework of binary decomposable trap function was proposed as the base to construct dynamic environments by changing the three core difficulty elements. From this framework, it is possible to systematically construct dynamic environments of changing but bounded difficulty.

For this type of generator, the environmental dynamics is characterized by the magnitude or step size of parameter change and the speed of changes in EA time.

The third type of generator was proposed in [39] and [41], which can generate dynamic environments from any binary-encoded stationary problem based on a bitwise exclusive-or (XOR) operator. Given a stationary problem $f(\vec{x})$ ($\vec{x} \in \{0,1\}^l$, where l is the length of binary representation), dynamic environments can be constructed from it as follows. Suppose the environment is periodically changed every τ generations.[4] For each environmental period k, an XORing mask $\vec{M}(k)$ is first incrementally generated as follows:

$$\vec{M}(k) = \vec{M}(k-1) \oplus \vec{T}(k) \tag{4}$$

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 $\rho \times l$ ($\rho \in [0.0, 1.0]$) ones inside it for environmental period k. Initially, $\vec{M}(0)$ is set to a zero vector. Then, the individuals at generation t are evaluated using the following formula:

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(k)) \tag{5}$$

where $k = \lfloor t/\tau \rfloor$ is the environmental period index.

With this generator, the environmental dynamics can be easily tuned by two parameters: τ controls the change speed, while ρ controls the severity each time the environment changes. A

[4]The generator can be easily modified to construct nonperiodical dynamic environments, where τ varies with time instead of being a fixed value.

bigger value of ρ means more severe environmental changes and, hence, a greater challenge to EAs.

This XOR operator-based generator can be combined with other ones to create complex dynamic environments, as seen in [40]. It can also be modified to construct dynamic environments for testing specific approaches developed for EAs in dynamic environments, e.g., the generator described below for testing memory schemes.

B. An Extended DOP Generator for Testing Memory Schemes

Generally speaking, the above reviewed dynamic environment generators are usually used to construct general dynamic environments for testing all approaches for EAs. However, in order to better understand a certain approach, it would be better to construct dynamic environments that pay special attention to the exact approach. For example, for memory schemes, it would be interesting to construct dynamic environments with tunable cyclicity[5] since the effect of memory schemes on EAs depends heavily on whether the environment changes cyclicly or not. In this paper, based on the XOR operator-based generator described above, an extended dynamic environment generator is proposed for testing memory schemes for EAs. The generator is described as follows.

Given a binary-encoded stationary problem $f(\vec{x})$ ($\vec{x} \in \{0,1\}^l$ where l is the length of binary representation), three types of dynamic environments, *noncyclic*, *cyclic*, and *partially cyclic*, can be constructed from it using the XOR operator. The first type of dynamic environment is exactly what the above described XOR operator generator constructs. This type of dynamic environment is also called *random* dynamic environment in this paper since with respect to cyclicity the environment moves randomly in the search space, even though each time it may move with a fixed Hamming distance away from the current environment. This is illustrated in Fig. 4(a), where a noncyclic dynamic environment is constructed from a 10-bit function with $\rho = 0.5$ and the XORing mask is used to represent the environmental state. Each time the environment changes, it moves $\rho \times l = 0.5 \times 10 = 5$ bits away randomly from the current state and will not guarantee to return to the initial state represented by $\vec{M}(0) = 0000000000$.

Next, we describe how to construct cyclic dynamic environments. The idea is quite simple: first construct a fixed number of states (environments), called *base states*, in the search space randomly or in certain pattern, and then move the environment among these base states in a fixed order cyclicly. Suppose there are $2K$ base states, then the environment will return to its initial state when it changes every $2K$ times. With the XOR operator, we can generate $2K$ XORing masks $\vec{M}(0), \vec{M}(1), \ldots, \vec{M}(2K-1)$ as the base states. These XORing masks form a logical ring representing the cyclicly changing environment. Suppose the environment is periodically changed every τ generations, then the individuals at generation t are evaluated using the following formula:

$$f(\vec{x}, t) = f(\vec{x} \oplus \vec{M}(I_t)) = f(\vec{x} \oplus \vec{M}(k\%(2K))) \quad (6)$$

[5]In the real-world, there are many DOPs that are subject to cyclic or approximately cyclic environments, which motivates the study of cyclic DOPs in this paper. For example, the climate may change cyclically over a year and the conditions in the traffic system may change cyclically over a day.

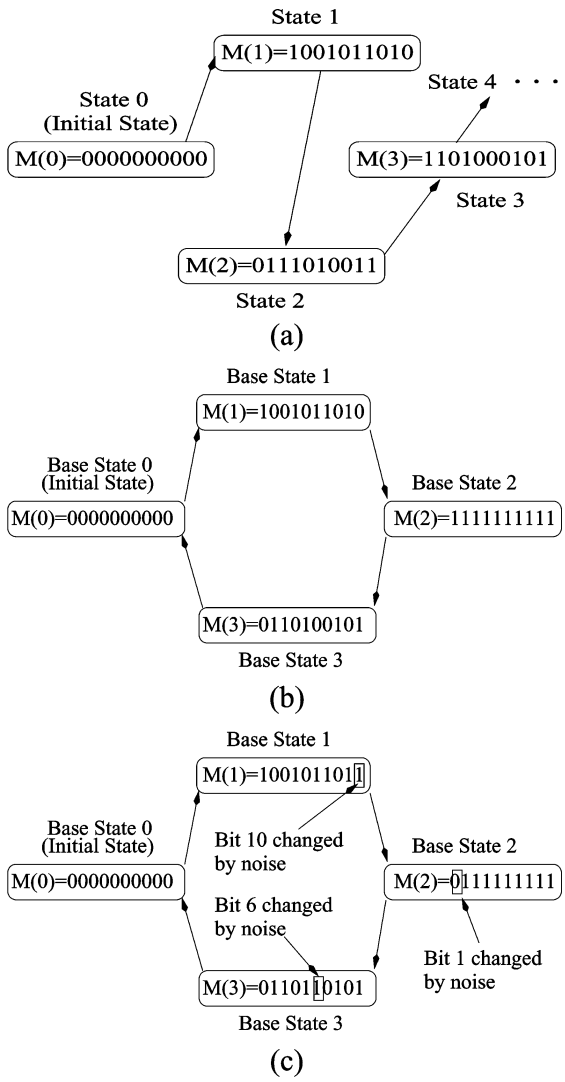

Fig. 4. Illustration of three kinds of dynamic environments constructed from a 10-bit encoded function with $\rho = 0.5$. (a) Noncyclic. (b) Cyclic. (c) Cyclic with noise.

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

The $2K$ XORing 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.[6] Let $\vec{M}(0) = \vec{0}$ denote the initial state. Then, the other XORing masks are generated iteratively as follows:

$$\vec{M}(i+1) = \vec{M}(i) \oplus \vec{T}(i\%K), \quad i = 0, \ldots, 2K-1. \quad (7)$$

With the above formula, 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 XORing masks till $\vec{M}(2K) = \vec{M}(0) = \vec{0}$. The constructed XORing masks have

[6]In the partition each template $\vec{T}(i)$ $(i = 0, \ldots, K-1)$ has randomly but exclusively selected $\rho \times l = l/K$ bits set to 1 while other bits to 0. For example, $\vec{T}(0) = 0101$ and $\vec{T}(1) = 1010$ form a partition of the 4-bit search space. Here, ρ (and $K = 1/\rho$) is determined such that l/K is an integer.

equal Hamming distance between two neighbors. That is, if we denote $d(\vec{M}(i), \vec{M}(j))$ the Hamming distance between $\vec{M}(i)$ and $\vec{M}(j)$, we have

$$\begin{aligned} d(\vec{M}(1), \vec{M}(0)) &= d(\vec{M}(2), \vec{M}(1)) = \cdots \\ &= d(\vec{M}(2K-1), \vec{M}(2K-2)) \\ &= d(\vec{M}(0), \vec{M}(2K-1)) \\ &= \rho \times l \end{aligned} \tag{8}$$

where $\rho \in [1/l, 1.0]$ is the distance factor, which determines the number of base states. For example, Fig. 4(b) shows a cyclic dynamic environment constructed from a 10-bit function with $\rho = 0.5$ and two templates $\vec{T}(0) = 1001011010$ and $\vec{T}(1) = 0110100101$ [not shown in Fig. 4(b)].

From the above cyclic dynamic environment generator, we can construct partially cyclic dynamic environments, also called *cyclic dynamic environments with noise*, by introducing noise to the base states.[7] There are two mechanisms, called *deterministic* and *probabilistic*, of adding noise in terms of the number of bits to be changed in the base states. For the deterministic mechanism, each time the environment is about to move to a next base state $\vec{M}(i)$, a noise template $\vec{T}_n$ with a small portion of ones is randomly created and integrated (XOR-ed) into $\vec{M}(i)$ as follows:

$$\vec{M}'(i) = \vec{M}(i) \oplus \vec{T}_n \tag{9}$$

where $\vec{M}'(i)$ is the new base state. The number of ones in $\vec{T}_n$ can be set to be linear with the Hamming distance between base states, i.e., $\gamma \times \rho \times l$ ($\gamma \in (0.0, 1.0)$). For example, Fig. 4(c) illustrates a noisily cyclic dynamic environment constructed from a 10-bit function with $\rho = 0.5$ and $\gamma = 0.2$, where each base state has one bit changed by noise. For the probabilistic noise mechanism, 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.

With the above generator for cyclic dynamic environments, noisy or not, there are two run mechanisms with respect to the base states for different runs of an algorithm on a test problem. For the first one, a set of $2K$ base states is first created and then used as the common base states for all runs of an algorithm. That is, all runs of an algorithm undergo the same cyclic dynamic environment. For the second mechanism, for each run of an algorithm a set of $2K$ base states is first created and then used for only this run. That is, different runs of an algorithm undergo different cyclic dynamic environments. The second mechanism will be used in the experimental study of this paper.

One thing to note is that for the proposed generator there exist certain relationships between the three kinds of dynamic environments. For example, when $\gamma = 0$ noisy cyclic dynamic environments become cyclic and when $\rho = 1.0$ noisy cyclic dynamic environments are comparable (though not equivalent) to noncyclic ones with $\rho = 1.0 - \gamma$, and cyclic environments with $\rho = 1.0$ are equivalent to noncyclic ones with $\rho = 1.0$. By tuning the values of ρ and γ (or p_n), we can easily tune the cyclicity of dynamic environments and hence the level of difficulty for memory-enhanced EAs.

[7]This is analogous to many dynamic environments in nature. For example, in the natural climate environment, spring (a base state) is spring but every spring is different.

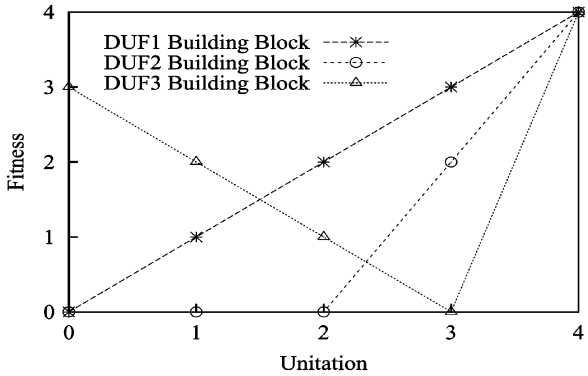

Fig. 5. The building blocks used for the three stationary DUFs.

C. Dynamic Test Environments for This Study

1) General Decomposable Unitation-Based Functions: Decomposable unitation-based functions (DUFs), such as trap and deceptive functions, have been widely studied in EA's community in the attempt to understand what constructs difficulty problems for EAs, especially for GAs [15]. A unitation function of a binary string returns the number of ones in the string. In this paper, in order to compare the performance of investigated algorithms in dynamic environments, three DUFs, denoted *DUF1*, *DUF2* and *DUF3*, are selected as stationary base functions to construct dynamic test environments.

All three DUFs consist of 25 copies of 4-bit building blocks. Each building block of the three DUFs is a unitation-based function and contributes a maximum value of 4 to the total fitness, as shown in Fig. 5. The fitness of a bit string is the sum of contributions from all building blocks, which gives an optimal fitness of 100 for all three DUFs.

DUF1 is, in fact, an $OneMax$ function, which aims to maximize the number of ones in a chromosome. OneMax functions are usually taken as easy functions for EAs since low-order building blocks inside the functions clearly lead to high-order building blocks. For DUF2, in the search space of the 4-bit building block, the unique optimal solution is surrounded by only four suboptimal solutions, while all the other 11 solutions form a wide plateau with zero fitness. The existence of this wide gap makes EA's searching on DUF2 much harder than on DUF1. For DUF3, it is a fully deceptive function [15]. Fully deceptive functions are usually considered hard problems for EAs because the low-order building blocks inside the functions do not combine to form the higher order optimal building block: instead they combine into deceptive suboptimal building block [47].

Generally speaking, the three DUFs form an increasing difficulty for EAs in the order from DUF1 to DUF2 to DUF3.

2) Constructing Dynamic DUFs (DDUFs): Dynamic test environments for this study are constructed from the three stationary DUFs, denoted *DDUF1*, *DDUF2* and *DDUF3*, respectively. From each DUF, three kinds of dynamic DUFs, cyclic, cyclic with noise, and random, are constructed by the aforementioned dynamic problem generator.

For each constructed dynamic problem, the fitness landscape is periodically changed every τ generations during the run of algorithms. In order to test the effect of environmental change speed on the performance of algorithms, τ is set to 10 and 25,

respectively. The environmental change severity parameter ρ is set to 0.1, 0.2, 0.5, and 1.0 for all DDUFs. With this setting of ρ, for cyclic dynamic problems, with and without noise, the environment cycles among 2, 4, 10, and 20 bases states, respectively. For cyclic dynamic problems with noise, the probabilistic mechanism of adding noise is used with the probability of flipping the base states $p_n = 0.05$.

Totally, a series of 24 dynamic problems, 2 values of τ combined with 4 values of ρ under 3 types of environments, are constructed from each stationary DUF.

V. Experimental Study on PBILs With Memory and Random Immigrants Schemes

A. Experimental Design and Results

Experiments were carried out to compare the performance of algorithms on the dynamic test environments constructed above. All algorithms have the following common parameters: total population size is set to $n = 100$, which includes the memory size $m = 0.1 * n = 10$ if memory is used, and $r_i = 0.2$ for algorithms with random immigrants, including ISGA. For all PBILs, the parameters are set to typical values without tuning as follows: the learning rate $\alpha = 0.25$ for all working probability vectors and the mutation probability $p_m = 0.02$ with the mutation shift $\delta_m = 0.05$. For ISGA, parameters are set as follows: the transformation probability $p_t = 0.9$ (according to our preliminary experiments), the bit flip mutation probability $p_m = 0.02$, and elitism of size 1 without reevaluating the elite. The gene pool contains 200 gene segments of fixed length 5, of which the random group contains 60 gene segments, while the nonrandom group contains 140.

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

$$\overline{F}_{\mathrm{BOG}} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N}F_{\mathrm{BOG}_{ij}}\right) \tag{10}$$

where $G = 5000$ 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 offline performance, i.e., the best-of-generation fitness averaged across the 50 runs, and then over the data gathering period.

In order to understand the effect of memory and random immigrants scheme on the population diversity during the running of an algorithm, we also recorded the diversity of the population every generation. The diversity of the population at time t in the kth run of an EA on a DOP is defined as

$$\mathrm{Div}(k,t) = \frac{1}{ln(n-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n}\mathrm{HD}(i,j) \tag{11}$$

where l is the encoding length, n is the population size, and $\mathrm{HD}(i,j)$ is the Hamming distance between the ith and jth individuals in the population. The mean population diversity of an EA on a DOP at time t over 50 runs is calculated as below

$$\overline{\mathrm{Div}}(t) = \frac{1}{50}\sum_{k=1}^{50}\mathrm{Div}(k,t)). \tag{12}$$

The experimental results of algorithms on DDUFs with $\tau = 10$ and $\tau = 25$ are given in Figs. 6 and 7, respectively. The statistical results of comparing algorithms by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table I. In Table I, 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 behavior of algorithms in dynamic environments, their dynamic performance regarding the best-of-generation fitness against generations on DDUFs with $\tau = 25$ and $\rho = 0.2$ is plotted in Fig. 8. In Fig. 8, the last ten environmental changes (i.e., 250 generations) are shown, which corresponds to one cycle of environmental changes for cyclic DDUFs, and the data were averaged over 50 runs. The dynamic population diversity of algorithms against generations on DDUF2 and DDUF3 with $\tau = 25$ and $\rho = 0.2$ is plotted in Fig. 9 for the last ten environ- mental changes, where the data were averaged over 50 runs.

From Figs. 6–9 and Table I, several results and phenomena can be observed and are analyzed below from two aspects: regarding the comparison between investigated algorithms and regarding the effect of environmental dynamics on the performance of algorithms in general.

B. Experimental Analysis Regarding Algorithm Comparisons

Comparing the performance of algorithms on the DDUFs, several results can be observed and are analyzed as follows.

First, a prominent result is that both the memory-enhanced PBILs (i.e., MPBIL and MPBILi) perform significantly better than SPBIL, on most dynamic test problems. This validates the efficiency of introducing the memory scheme into PBILs. The effect of the memory scheme can be clearly seen in the dynamic performance of MPBIL and MPBILi shown in Fig. 8. For cyclic DDUFs, when the environment changes, the performance of MPBIL and MPBILi drops, and then the memory scheme rapidly brings the performance back to a high fitness level. For example, on DDUF2 with $\tau = 25$ and $\rho = 0.2$, when a change occurs at generation 4800, the performance of MPBIL drops from 95.9 to 77.5 at generation 4801, and then jumps up back to 80.7 at generation 4802. This performance jumping is due to the newly reactivated memory probability vector.

Both MPBIL and MPBILi achieve a better performance improvement over SPBIL on cyclic environments than on cyclic environments with noise and noncyclic environments. For example, when $\tau = 10$ and $\rho = 0.2$, $\overline{F}_{\mathrm{BOG}}(\mathrm{MPBIL}) - \overline{F}_{\mathrm{BOG}}(\mathrm{SPBIL}) = 90.5 - 55.9 = 34.6$ for cyclic DDUF1, $\overline{F}_{\mathrm{BOG}}(\mathrm{MPBIL}) - \overline{F}_{\mathrm{BOG}}(\mathrm{SPBIL}) = 64.8 - 57.2 = 7.6$ for cyclic DDUF1 with noise, and $\overline{F}_{\mathrm{BOG}}(\mathrm{MPBIL}) -$

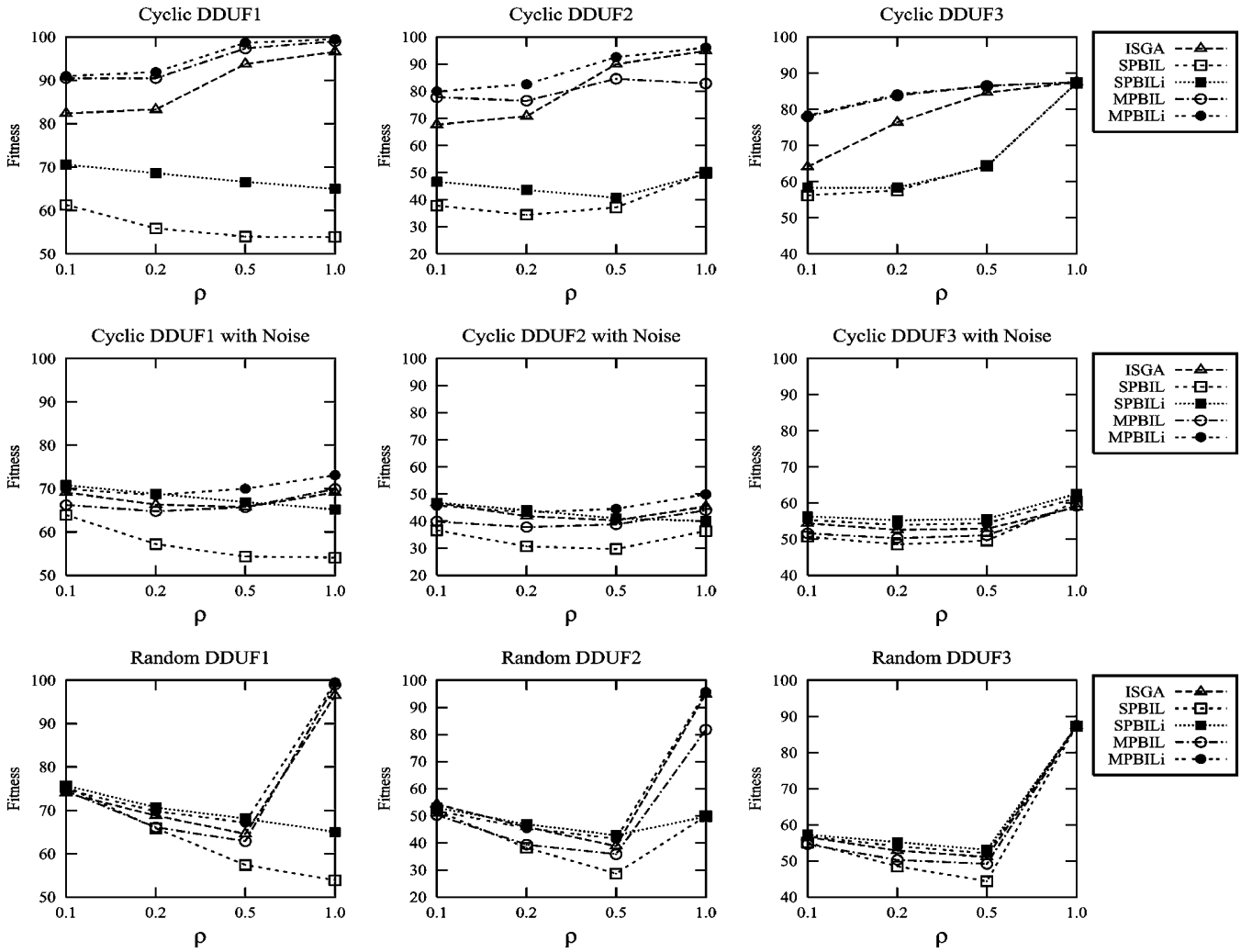

Fig. 6. Experimental results of ISGA, SPBIL, SPBILi, MPBIL, and MPBILi on DDUFs with $\tau = 10$.

$\overline{F}_{\mathrm{BOG}}(\mathrm{SPBIL}) = 66.1 - 65.9 = 0.2$ for random DDUF1. This result means that the effect of the memory scheme depends on the cyclicity of environments.

Second, the addition of the random immigrants scheme improves the performance of SPBIL and MPBIL on almost all DDUFs, see the t-test results regarding SPBILi – SPBIL and MPBILi – MPBIL. Random immigrants improve the population diversity, see Fig. 9 for the population diversity dynamics. Meanwhile, by replacing worst individuals in the population, random immigrants help improve the average fitness level of the population.

Comparing memory against random immigrants, it can be seen that the effect of the memory scheme is significantly greater (better) than the random immigrants scheme for all cyclic DDUFs, see the t-test results regarding MPBIL – SPBILi in Table I. However, for cyclic with noise and random DDUFs, the random immigrants scheme outperforms the memory scheme on most DDUFs. This happens because for these DDUFs, the environment is less likely to return precisely to those memorized environments, and hence random immigrants may track the new environment more efficient than memory samples.

When examining the effect of the memory scheme on PBIL with random immigrants, it can be seen that MPBILi outperforms SPBILi on most cyclic DDUFs, with or without noise. However, MPBILi is beaten by SPBILi for many random DDUFs. That is, when the random immigrants scheme is used, the addition of the memory scheme may have a negative effect in random dynamic environments.

Third, comparing the performance of ISGA with PBILs, it can be seen that ISGA outperforms SPBIL on most DDUFs and outperforms MPBIL on most random DDUFs and cyclic DDUFs with noise, see the t-test results regarding SPBIL – ISGA and MPBIL – ISGA, respectively. The memory and diversity hybrid scheme (i.e., memory-based cloning) inside ISGA gives it an advantage over SPBIL totally and over MPBIL on random and cyclic with noise environments. In fact, Fig. 9 shows that ISGA maintains the highest level of population diversity.

However, ISGA is significantly beaten by MPBIL on cyclic DDUFs and by MPBILi on almost all DDUFs, see the relevant t-test results. This happens due to two factors. First, PBILs have better search capacity than ISGA and this factor contributes to the fact that even SPBIL outperforms ISGA on several slightly

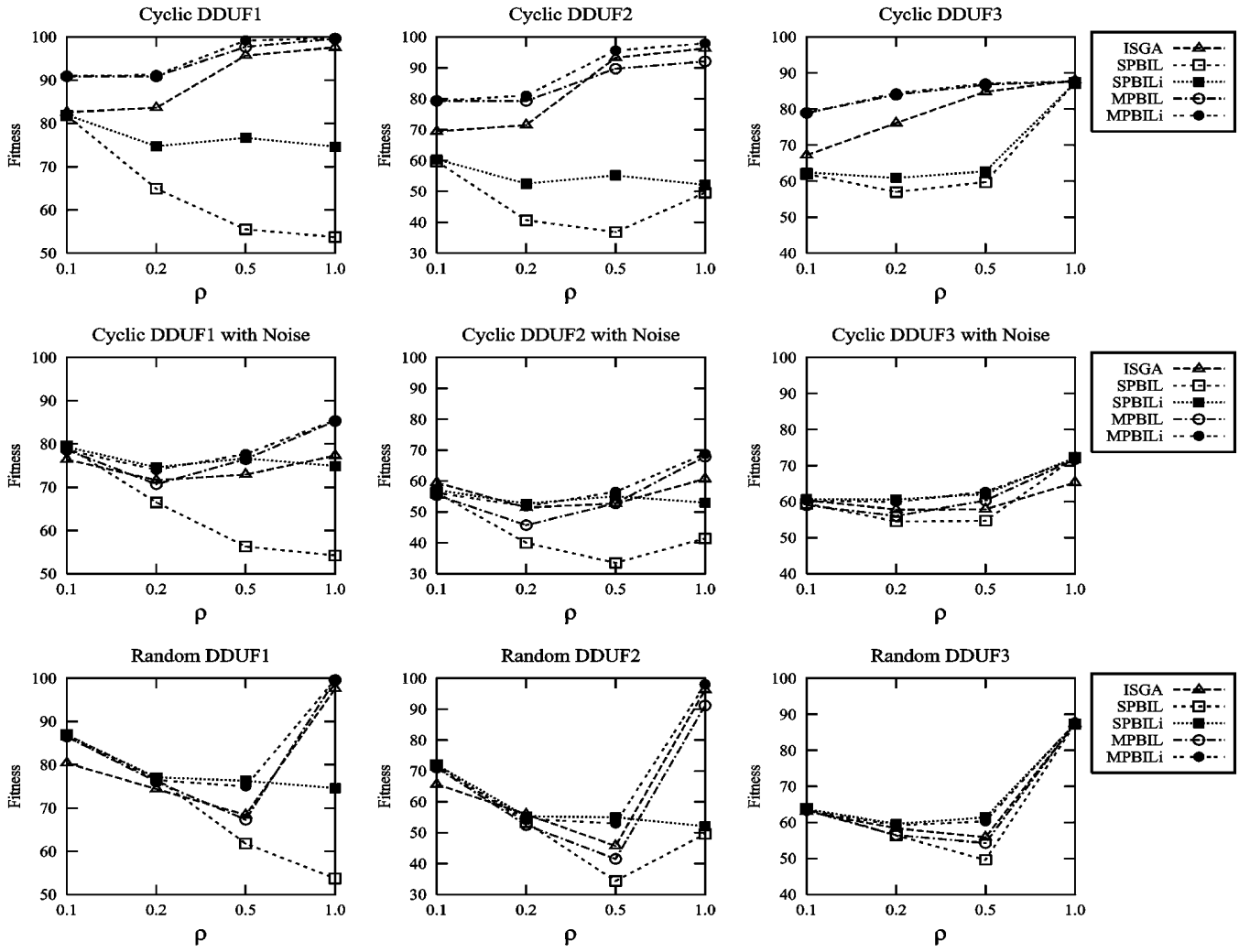

Fig. 7. Experimental results of ISGA, SPBIL, SPBILi, MPBIL, and MPBILi on DDUFs with $\tau = 25$.

changing DDUFs. This point can be seen from Fig. 8. On almost all DDUFs, PBILs achieve a higher fitness improvement than ISGA does during each environmental period. The second factor is because the memory scheme in MPBIL and MPBILi has a stronger effect than that in ISGA. This can be clearly seen in the dynamic performance of algorithms in Fig. 8. On cyclic DDUFs, MPBIL and MPBILi are able to maintain a higher fitness level than ISGA does. In order to better understand this point, the dynamic performance of algorithms on cyclic DDUF2 and DDUF3 with $\tau = 25$ and $\rho = 0.2$ over the first two cycles of environmental changes, i.e., 500 generations, is also shown in Fig. 10, where the data were averaged over 50 runs. From Fig. 10, it can be seen that after several early environmental changes the memory scheme in MPBIL and MPBILi clearly starts to take effect. For example, just after the first cycle of ten environmental changes, at generation 250 when the environment changes the memorized probability vector brings MPBIL and MPBILi directly to a high fitness level. On the contrast, the effect of the memory scheme in ISGA is much less visible from Fig. 10.

Stronger search capacity of PBIL, stronger memory scheme, and random immigrants together lead to MPBILi's better performance over the ISGA on almost all DDUFs.

C. Experimental Analysis Regarding Dynamic Environments

When examining the effect of dynamic environments on the performance of algorithms investigated, the following results can be observed.

First, comparing Fig. 6 with Fig. 7 shows that for each DDUF with a fixed ρ, the performance of algorithms rises when the value of τ increases from 10 to 25. This is easy to understand. When the environment changes slower, i.e., τ is larger, the algorithms have more time to reach higher fitness level before the environment changes.

Second, with each fixed τ, when the value of ρ increases from 0.1 to 0.2 to 0.5, the performance of algorithms generally decreases. This is natural since a bigger ρ means more severe environmental changes. However, on many DDUFs when $\rho = 1.0$ the algorithms perform better than when $\rho = 0.5$. This is because when $\rho = 1.0$ the environment switches between two

TABLE I
THE t-TEST RESULTS OF COMPARING ISGA, SPBIL, SPBILI, MPBIL, AND MPBILI ON DDUFS

t-test Result	$DDUF1$				$DDUF2$				$DDUF3$			
Environment Dynamics	$\tau = 10$		$\tau = 25$		$\tau = 10$		$\tau = 25$		$\tau = 10$		$\tau = 25$	
Cyclic, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$SPBILi - SPBIL$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s-$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $-$ $-$		$s+$ $s+$ $s+$ $-$	
$MPBIL - SPBIL$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$SPBIL - ISGA$	$s-$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$	
$MPBIL - ISGA$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s-$ $s-$		$s+$ $s+$ $-$ $s-$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL - SPBILi$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBILi - MPBIL$	$s+$ $s+$ $s+$ $s+$		$+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$+$ $s+$ $+$ $-$		$-$ $+$ $s+$ $+$	
$MPBILi - ISGA$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
Cyclic with Noise, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$SPBILi - SPBIL$	$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $+$	
$MPBIL - SPBIL$	$s+$ $s+$ $s+$ $s+$		$s-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s-$		$s-$ $s+$ $s+$ $s-$	
$SPBIL - ISGA$	$s-$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$	
$MPBIL - ISGA$	$s-$ $s-$ $s-$ $s+$		$s+$ $s-$ $s+$ $s+$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $+$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s+$ $s+$	
$MPBIL - SPBILi$	$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$	
$MPBILi - MPBIL$	$s+$ $s+$ $s+$ $s+$		$+$ $s+$ $s+$ $+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $+$	
$MPBILi - ISGA$	$s-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s-$ $+$ $s+$ $s+$		$s-$ $s-$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
Random, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$SPBILi - SPBIL$	$s+$ $s+$ $s+$ $s+$		$+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s-$		$+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $-$		$-$ $s+$ $s+$ $-$	
$MPBIL - SPBIL$	$s-$ $s+$ $s+$ $s+$		$s-$ $s-$ $s+$ $s+$		$s-$ $s+$ $s+$ $s+$		$s-$ $s-$ $s+$ $s+$		$s-$ $s+$ $s+$ $s+$		$s-$ $+$ $s+$ $s+$	
$SPBIL - ISGA$	$s+$ $s-$ $s-$ $s-$		$s+$ $s+$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s+$		$s+$ $s-$ $s-$ $s+$	
$MPBIL - ISGA$	$s+$ $s-$ $s-$ $s+$		$s+$ $s+$ $s-$ $s+$		$s-$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s+$		$s+$ $s-$ $s-$ $s+$	
$MPBIL - SPBILi$	$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$	
$MPBILi - MPBIL$	$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $+$		$+$ $s+$ $s+$ $+$	
$MPBILi - ISGA$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s-$ $s-$ $s+$ $s+$		$s+$ $s-$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	

landscapes and the algorithms may wait during one environment for the return of the other environment to which they converged well. For example, Fig. 11 shows the dynamic performance of algorithms on cyclic DDUF2, with and without noise, with $\tau = 25$ and $\rho = 1.0$. From Fig. 11(a), it can be seen that SPBIL clearly shows the waiting phenomenon during even environment periods.

An interesting result is that on cyclic DDUFs, the performance of MPBIL and MPBILi increases with the value of ρ, see the top row in Figs. 6 and 7. This happens because when the value of ρ increases, the number of base states decreases and hence the memory probability vectors in MPBIL and MPBILi represent the environments more precisely when they are stored and updated. This leads to the better effect of the memory scheme and hence the better overall performance of MPBIL and MPBILi when ρ increases. When the cyclicity of environments decreases, the effect of memory decreases due to the less precise matching between memorized environments and new environments, and bigger ρ brings in more severe changes and, hence, leads to the worse performance for MPBIL and MPBILi.

Third, viewing from top to down in Figs. 6 and 7, it can be seen that given the same values for ρ and τ, when the cyclicity of dynamic environments decreases from cyclic to cyclic with noise, the performance of algorithms degrades. That is, cyclic environments with noise are relatively harder than cyclic environments. The existence of noise reduces the effect of memory or the waiting behavior of SPBIL, see Fig. 11(b), and it seems that algorithms perform a little better on random environments than on cyclic environments with noise. This means noise may overweigh randomness with respect to the difficulty of dynamic environments.

Finally, viewing from left to right in Figs. 6 and 7, it can be seen the algorithms perform worse on DDUF2 problems than on corresponding DDUF1 problems with the same environmental dynamics. This shows the difficulty of dynamic problems for EAs not only depends on the environmental dynamics but also depends on the difficulty of relevant stationary problems, and the difficulty of stationary problems seems to be inherited to dynamic environments. This is natural since the problem during each environment period can be taken as a stationary problem. However, when deception exists inside the problem, the situation is quite different. For DDUF3 problems, when the environment changes the deceptive building blocks inside DUF3 will draw the population in the new environment toward them faster than the optimal building blocks in DUF2 can do. Though deceptive attractors are not globally optimal they are suboptimal with relatively high fitness. This leads to the result that algorithms perform better on most DDUF3 problems than on corresponding DDUF2 problems with the same environmental dynamics.

This result can be clearly seen from the dynamic behavior of algorithms in Fig. 8. The performance of algorithms stays at a

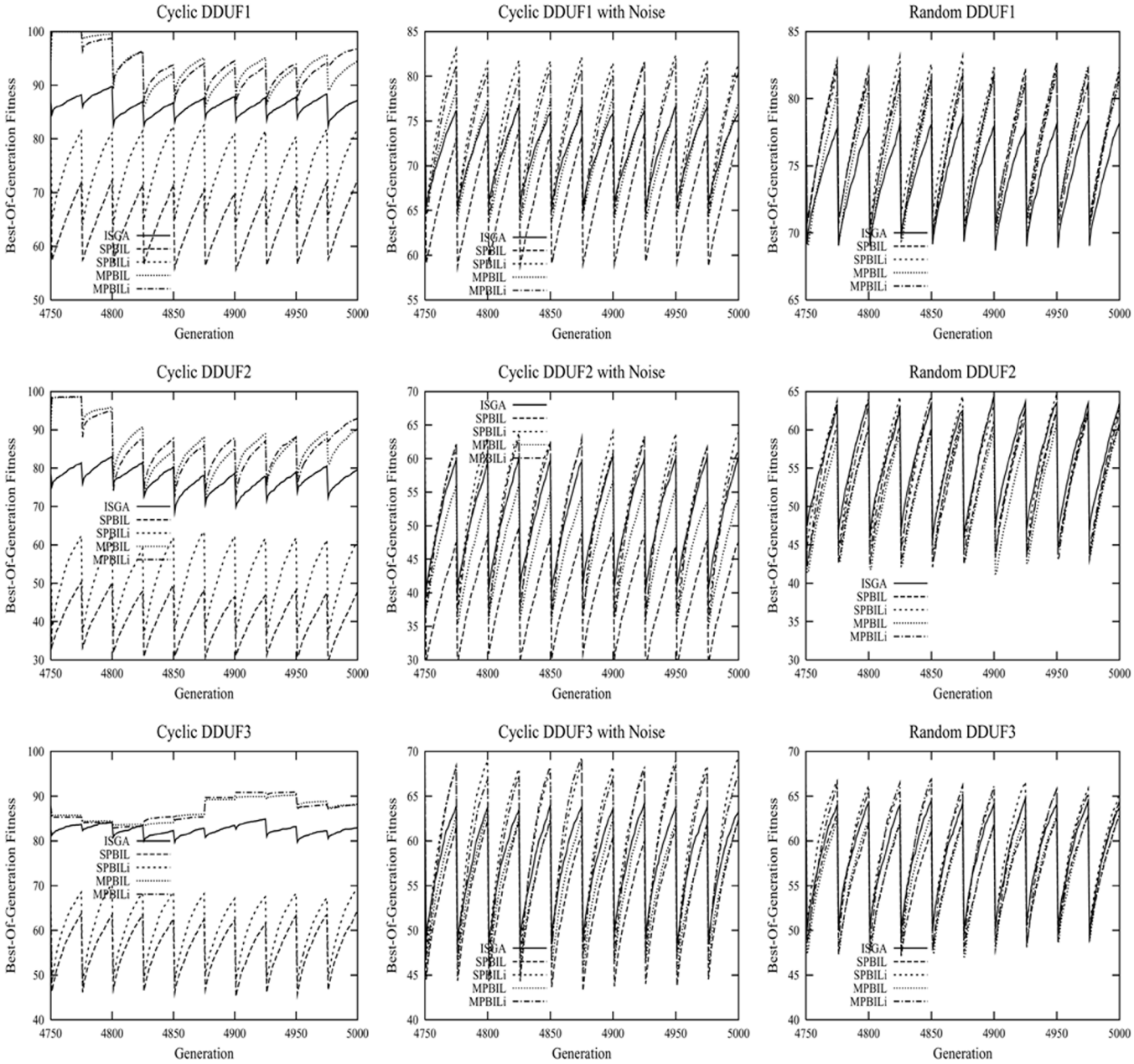

Fig. 8. Dynamic performance of algorithms for the last ten environmental changes on DDUFs with $\tau = 25$ and $\rho = 0.2$.

higher fitness level on DDUF1 and DDUF3 problems than on DDUF2 problems with the same environmental dynamics, and the existence of deception in DDUF3 problems makes the fitness fluctuation of algorithms less significantly over time on DDUF3 problems than on corresponding DDUF2 problems.

VI. Experimental Study on PBILs With Memory and Multipopulation Schemes

Other than memory schemes, multipopulation schemes are another kind of approaches that has been integrated into EAs to deal with dynamic environments. As discussed in [6] and [7], the multipopulation scheme has two advantages. On the one hand, using multiple but independently evolving populations can increase the diversity in the overall population. On the other hand, through assigning different responsibilities to different populations the available number of individuals in the overall population can be used more efficiently.

In this paper, in order to study the effect of multipopulation on the memory scheme for PBILs in dynamic environments, PBILs with two probability vectors are further investigated, and a memory-enhanced GA with two populations is also investigated as a peer EA for performance comparisons.

A. The Memory/Search GA

In [6] and [8], Branke proposed a *memory/search GA* that aims to combine the advantages of multipopulation and memory schemes together. In this study, a similar memory-enhanced GA with two populations, denoted *MEGA2r*, is also studied as a peer EA. MEGA2r differs from Branke's memory/search GA in two aspects: first, the memory in MEGA2r is updated in a stochastic

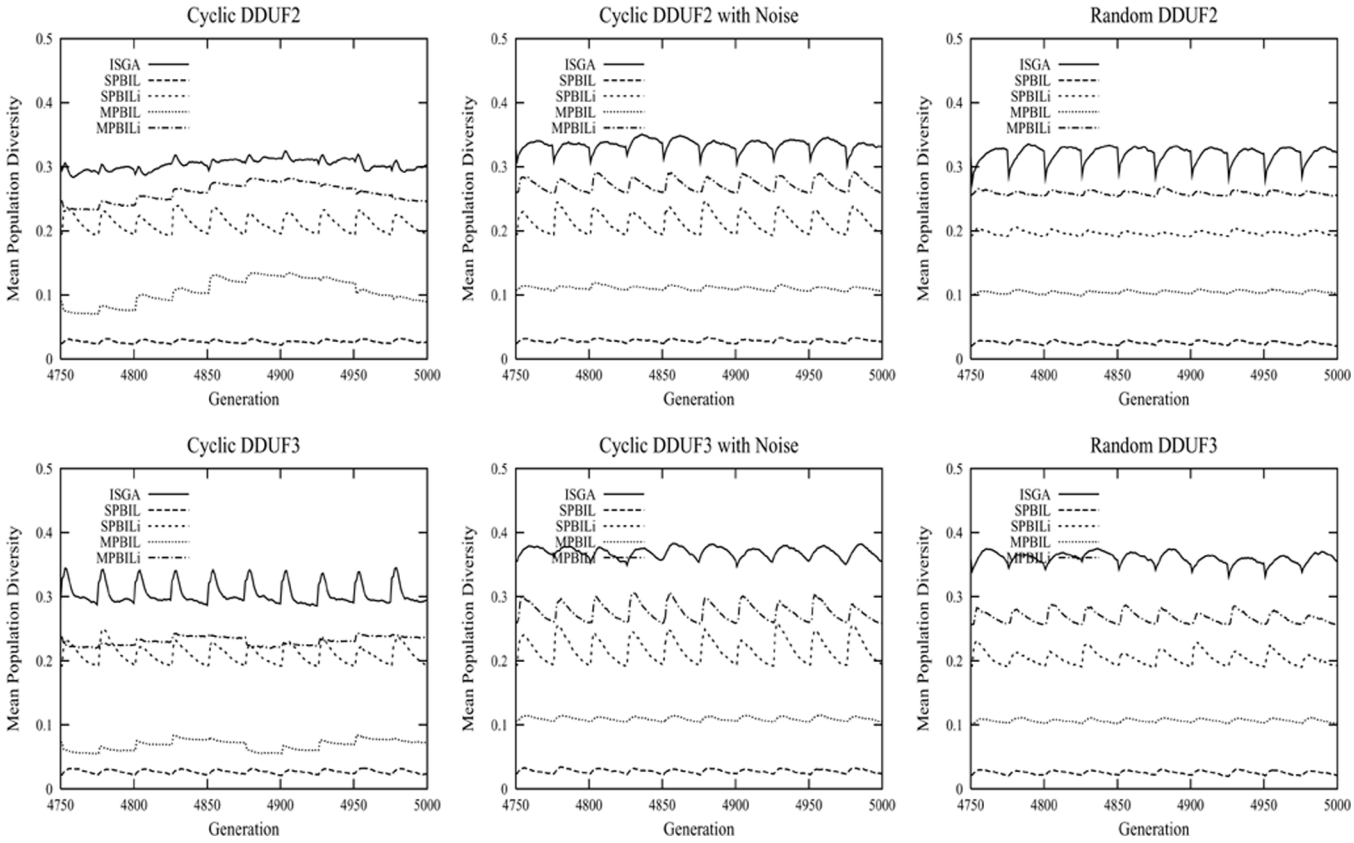

Fig. 9. Dynamic population diversity of algorithms for the last ten environmental changes on DDUF2 qnd DDUF3 with $\tau = 25$ and $\rho = 0.2$.

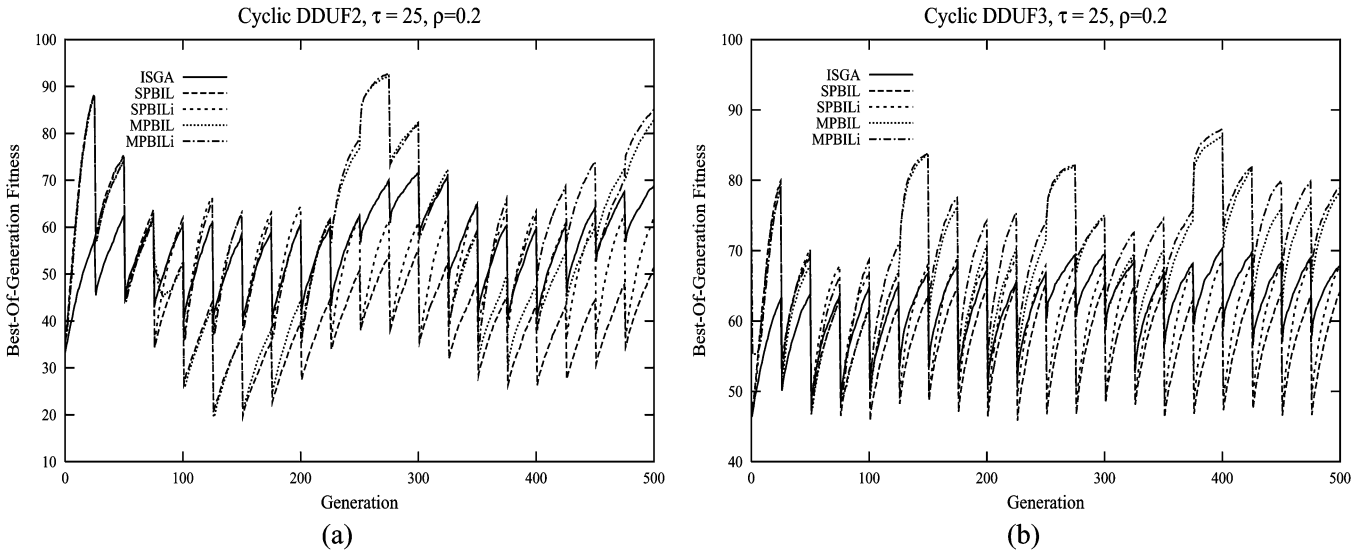

Fig. 10. Dynamic performance of algorithms for the first 20 environmental changes on (a) cyclic DDUF2 and (b) cyclic DDUF3 with $\tau = 25$ and $\rho = 0.2$.

time pattern; second, the population sizes in MEGA2r are adaptively adjusted, which was applied in [41]. Fig. 12 shows the pseudocode of MEGA2r.

In MEGA2r, the two populations P_1 and P_2 evolve independently and each has the following configuration: generational, uniform crossover, bit flip mutation, and fitness proportionate selection with the elitist scheme. 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 number of individuals, including the memory. In order to give the better performed population more chance to search, the population sizes n_1 and n_2 are slightly adjusted every generation within the range of $[0.3*n, 0.6*n]$ according to their performance. The winner population gets $\Delta = 0.05*n$ for its size from the loser; if the two populations tie, their sizes do not change.

As in ISGA, the memory in MEGA2r has a size $m = 0.1*n$, is randomly initialized, and updated in a stochastic time pattern with the most similar updating strategy. When the memory is

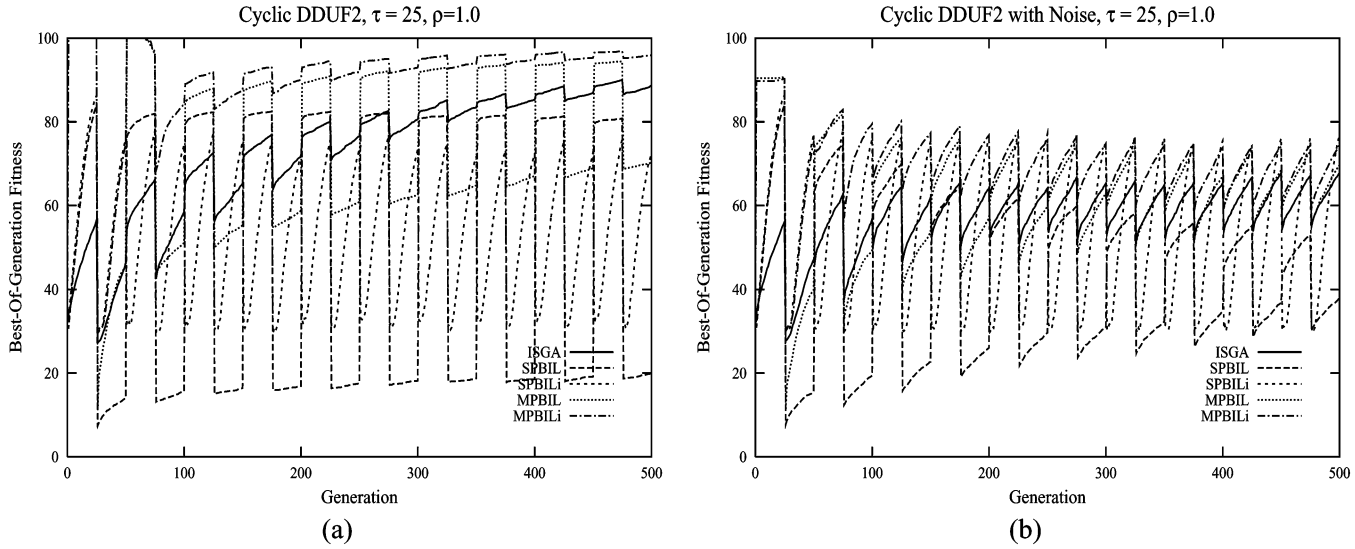

Fig. 11. Dynamic performance of algorithms for the first 20 environmental changes on (a) cyclic and (b) cyclic with noise DDUF2 with $\tau = 25$ and $\rho = 1.0$.

$t := 0$ and $t_M := rand(5, 10)$
initialize memory $M(0)$ and populations $P_1(0)$ and $P_2(0)$
repeat
  evaluate $P_1(t)$, $P_2(t)$ and $M(t)$
  adjust next population sizes for $P_1(t)$ and $P_2(t)$ resp.
  if *environmental change detected* **then**
    $P'_1(t)$:=retrieveBestMembers$(P_1(t), M(t))$
    $P'_2(t)$:=re-initialize$(P_2(t))$
  else $P'_1(t) := P_1(t)$ and $P'_2(t) := P_2(t)$
  if $t = t_M$ **then** // *time to update memory*
    $t_M := t + rand(5, 10)$
    denote best individual in $P'_1(t)$ and $P'_2(t)$ by $B_P(t)$
    if *still any random points 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** $C_M(t) := B_P(t)$
  // *normal genetic operations for* $P'_1(t)$ *and* $P'_2(t)$ *resp.*
  replace elite from $P_1(t-1)$ and $P_2(t-1)$ into $P'_1(t)$
  $(P'_1(t), P'_2(t)) :=$ 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.*
until *termination condition holds* // *e.g.,* $t > t_{max}$

Fig. 12. Pseudocode for the memory-enhanced GA with two populations and restart scheme (MEGA2r).

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 called the *memory population*. The second population P_2 is restarted (reinitialized) in order to search new areas in the search space and is hence called the *search population*.

B. PBILs With Multipopulation and Memory Schemes

For PBILs, the multipopulation scheme can be realized by maintaining and evolving multiple probability vectors in parallel. Fig. 13 shows the pseudocode of three variants of PBILs with two probability vectors that are investigated in this paper. The first variant, denoted *SPBIL2*, has no memory, while the other two are memory-enhanced, denoted *MPBIL2* and *MPBIL2r* respectively.

In SPBIL2, MPBIL2, and MPBIL2r, the two probability vectors work in parallel. Each one is sampled independently and is learnt toward the best sample generated by itself. The probability vector $\vec{P}_1$ is initialized to the central probability vector, while $\vec{P}_2$ is randomly initialized. The sample sizes for $\vec{P}_1$ and $\vec{P}_2$ are equally initialized to $0.5 * n$ for PBIL2 and $0.45 * n$ for MPBIL2 and MPBIL2r, where n is the total evaluations per iteration. For MPBIL2 and MPBIL2r, the memory size is fixed to $0.1 * n$. As in MEGA2r, the sample sizes for $\vec{P}_1$ and $\vec{P}_2$ are slightly adjusted within the range of $[0.3*n, 0.7*n]$ for SPBIL2 and $[0.3*n, 0.6*n]$ for MPBIL2 and MPBIL2r according to their performance. The winner probability vector gets $\Delta = 0.05 * n$ for its sample size from the loser for the next iteration.

For MPBIL2 and MPBIL2r, both populations can store data into the memory in a similar time and space pattern as in MPBIL1. When it is time to update the memory, the working probability vector that creates the best overall sample, i.e., the winner of $\vec{P}_1$ and $\vec{P}_2$, will be stored together with the best sample in the memory if it is fitter than the closest memory sample. The memory is reevaluated every iteration. When an environmental change is detected, in order to avoid that $\vec{P}_1$ and $\vec{P}_2$ converge into one, only $\vec{P}_1$ will be replaced by the best memory probability vector if the associated memory sample is fitter than the best sample generated by $\vec{P}_1$.

$t := 0$ and $t_M := rand(5, 10)$
initialize $\vec{P}_1(0) := \vec{0.5}$ and $\vec{P}_2(0) := \vec{rand}(0.0, 1.0)$
if *memory used* **then** // *for MPBIL2 and MPBIL2r*
  initialize memory $M(0) := \phi$ and $t_M := rand(5, 10)$
$S_1(0) :=$ sample$(\vec{P}_1(0))$ and $S_2(0) :=$ sample$(\vec{P}_2(0))$
repeat
  evaluate $S_1(t)$ and denote its best sample by $\vec{B}_1(t)$
  evaluate $S_2(t)$ and denote its best sample by $\vec{B}_2(t)$
  adjust next sample sizes for $\vec{P}_1(t)$ and $\vec{P}_2(t)$
  if *no memory used* **then** // *for PBIL2*
    learn $\vec{P}_1(t)$ toward $\vec{B}_1(t)$ and $\vec{P}_2(t)$ toward $\vec{B}_2(t)$
  else // *for MPBIL2 and MPBIL2r*
    denote the better of $\vec{B}_1(t)$ and $\vec{B}_2(t)$ by $\vec{B}_B(t)$ and its corresponding prob. vector by $\vec{P}_B(t)$
    evaluate $M(t)$ and denote the best memory sample by $\vec{B}_M(t)$ and its associated prob. vector by $\vec{P}_M(t)$
    if $t = t_M$ **then** // *time to update memory*
      $t_M := t + rand(5, 10)$
      if *memory not full* **then** save $\vec{B}_B(t), \vec{P}_B(t)$ in $M(t)$
      else find memory sample $\vec{C}_M(t)$ closest to $\vec{B}_B(t)$ and its associated prob. vector $\vec{P}_C(t)$
        if $f(\vec{B}_B(t)) > f(\vec{C}_M(t))$ **then**
          $\vec{C}_M(t) := \vec{B}_B(t)$ and $\vec{P}_C(t)) := \vec{P}_B(t))$
    if *environmental change detected* **then**
      if $f(\vec{B}_M(t)) > f(\vec{B}_B(t))$ **then** $\vec{P}_1(t) := \vec{P}_M(t)$
      if *restart used* **then** $\vec{P}_2(t) := \vec{0.5}$ // *for MPBIL2r*
    else learn $\vec{P}_1(t)$ toward $\vec{B}_1(t)$ & $\vec{P}_2(t)$ toward $\vec{B}_2(t)$
  mutate $\vec{P}_1(t)$ and $\vec{P}_2(t)$
  $S_1(t) :=$ sample$(\vec{P}_1(t))$ and $S_2(t) :=$ sample$(\vec{P}_2(t))$
until *termination condition holds* // *e.g.,* $t > t_{max}$

Fig. 13. Pseudocode for PBILs with two probability vectors: without memory (SPBIL2), with memory (MPBIL2), and with memory and restart (MPBIL2r).

MPBIL2 and MPBIL2r differ in that MPBIL2r uses the restart scheme. Whenever an environmental change is detected, $\vec{P}_2$ in MPBIL2r is reset to the central probability vector, while nothing happens for $\vec{P}_2$ in MPBIL2. It can be seen that MPBIL2r uses the idea similar to the above memory/search GA. The first probability vector $\vec{P}_1$ is devoted to make use of the memory, while the second probability vector $\vec{P}_2$ aims to search through the solution space for new promising areas in new environments.

C. Experimental Results and Analysis

Experiments are carried out to investigate the performance of MEGA2r, SPBIL2, MPBIL2, and MPBIL2r on the same DDUF problems as used in Section V. The experimental settings and the parameter settings for algorithms are also the same as used in Section V. MEGA2r uses uniform crossover with $p_c = 0.6$, the bit flip mutation with $p_m = 0.02$, and elitism of size 1 without reevaluating the elite. The experimental results of algorithms on the DDUFs with $\tau = 25$ are presented in Fig. 14. The statistical results of comparing algorithms by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table II. The dynamic performance of algorithms on the last ten environmental changes with respect to best-of-generation fitness against generations on DDUF2 with $\tau = 25$ and $\rho = 0.2$ is plotted in Fig. 15 and the corresponding dynamic population diversity of algorithms is also plotted in Fig. 16. From Figs. 14–16 and Table II, several results can be observed and are analyzed as follows.

First, SPBIL2 significantly outperforms SPBIL1 on almost all dynamic problems, see the t-test results regarding SPBIL2 – SPBIL1. This validates the efficiency of the multipopulation scheme on the performance of PBILs in dynamic environments. In SPBIL2, introducing an extra probability vector increases the diversity, and hence improves its adaptability in dynamic environments. This effect can be seen by comparing the dynamic population diversity of SPBIL2 and SPBIL1 in Figs. 9 and 16, respectively.

Second, SPBIL2 is still outperformed by all the memory-enhanced PBILs including MPBIL and MPBIL2 on most dynamic problems, especially under cyclic environments with or without noise (the t-test results with respect to MPBIL – SPBIL2 and MPBIL2 – SPBIL2 are not shown). This indicates that the memory scheme has a stronger effect than the multipopulation scheme on PBIL's performance in dynamic environments.

Third, an interesting result is that MPBIL2 is beaten by MPBIL on most DDUFs, see the t-test results with respect to MPBIL2 – MPBIL. This means when the memory scheme is used, introducing an extra probability vector $\vec{P}_2$ may be negative on PBIL's performance. This happens because in MPBIL the reactivated memory probability vector uses the sample size resource to its full (i.e., $0.9 * n$), which outweighs the diversity introduced by $\vec{P}_2$ in MPBIL2, comparing the dynamic population diversity of MPBIL2 and MPBIL1 in Figs. 9 and 16, respectively. In other words, the reactivated memory probability vector in MPBIL is better than $\vec{P}_2$ in MPBIL2 for most cases. However, when the restart scheme is used for $\vec{P}_2$ in MPBIL2r, the situation is totally different. MPBIL2r significantly outperforms both MPBIL and MPBIL2 on most DDUFs. The benefit of the restart scheme can be clearly seen from the dynamic performance of MPBIL2r shown in Fig. 15, especially on cyclic with noise DDUFs and random DDUFs.

Fourth, examining the performance of MEGA2r, it can be seen that ISGA significantly outperforms MEGA2r on all cyclic DDUFs. This happens because the memory scheme inside ISGA is stronger. However, on random and cyclic with noise DDUF1 and DDUF2 problems, MEGA2r beats ISGA due to the higher diversity brought in by the restart scheme, comparing the dynamic population diversity of ISGA and MEGA2r in Figs. 9 and 16, respectively. On all DDUF3 problems, higher diversity (not shown) may be negative due to its property of strong deception, which leads to ISGA's better performance over MEGA2r.

Comparing MEGA2r with memory-enhanced PBILs, it can be seen that MEGA2r outperforms MPBIL2 on approximately

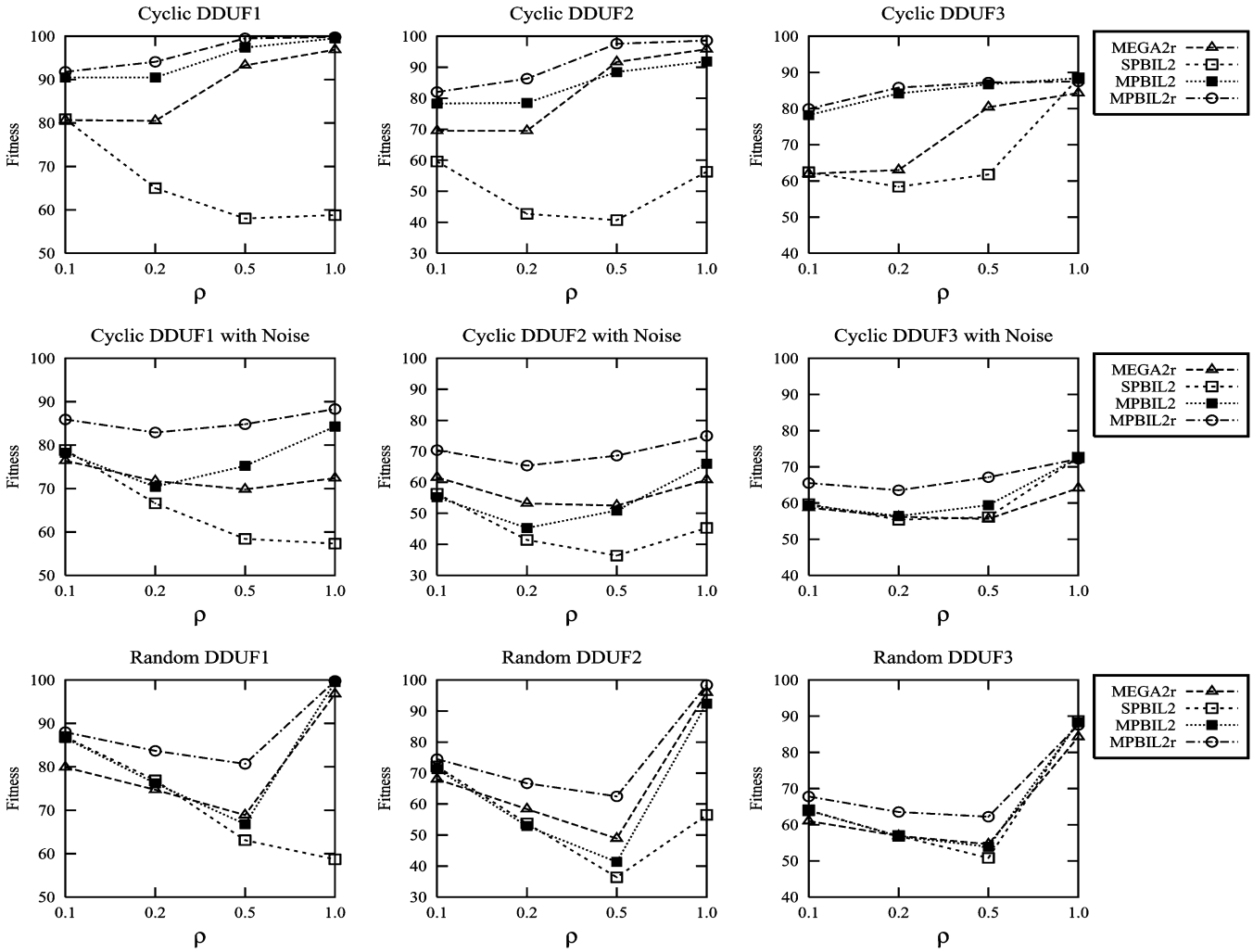

Fig. 14. Experimental results of MEGAr, SPBIL2, MPBIL2, and MPBIL2r on DDUFs with $\tau = 25$.

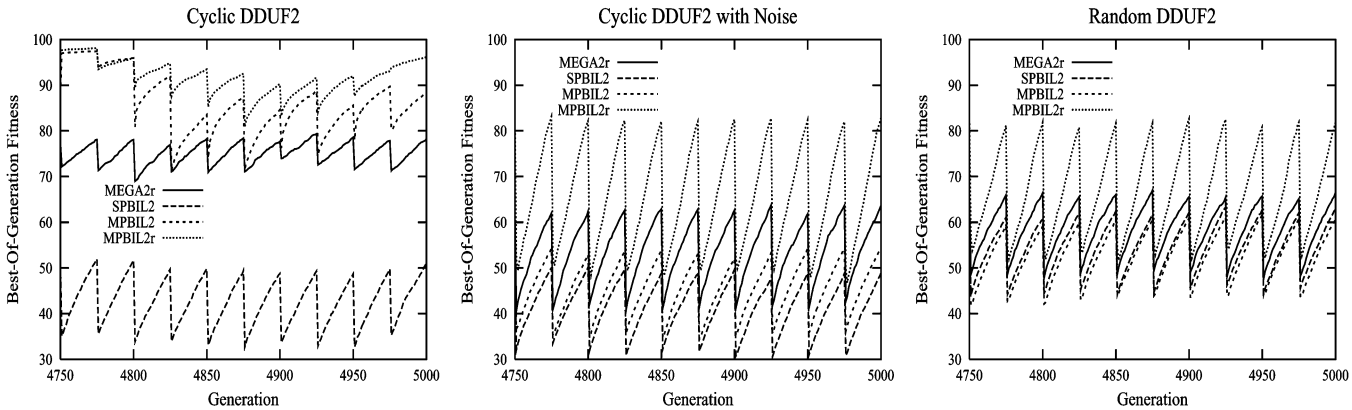

Fig. 15. Dynamic behavior of algorithms with two populations for the last ten environmental changes on DDUF2 with $\tau = 25$ and $\rho = 0.2$.

half of the DDUFs under cyclic with noise and random environments but is outperformed by MPBIL2 on almost all cyclic DDUFs. This happens because under cyclic with noise and random dynamic environments the restart scheme in MEGA2r contributes to its advantage over MPBIL2. But under cyclic environments the stronger memory scheme in MPBIL2 makes it win over MEGA2r significantly. When the restart scheme is combined with the memory scheme in MPBIL2r, MEGA2r is significantly outperformed by MPBIL2r on almost all DDUFs.

The great effect of combining memory and restart schemes in MPBIL2r can be clearly seen in the dynamic performance of MPBIL2r in Fig. 15. Under cyclic dynamic environments, the memory scheme enables MPBIL2r to maintain a quite high fitness level across changing environments; while in cyclic

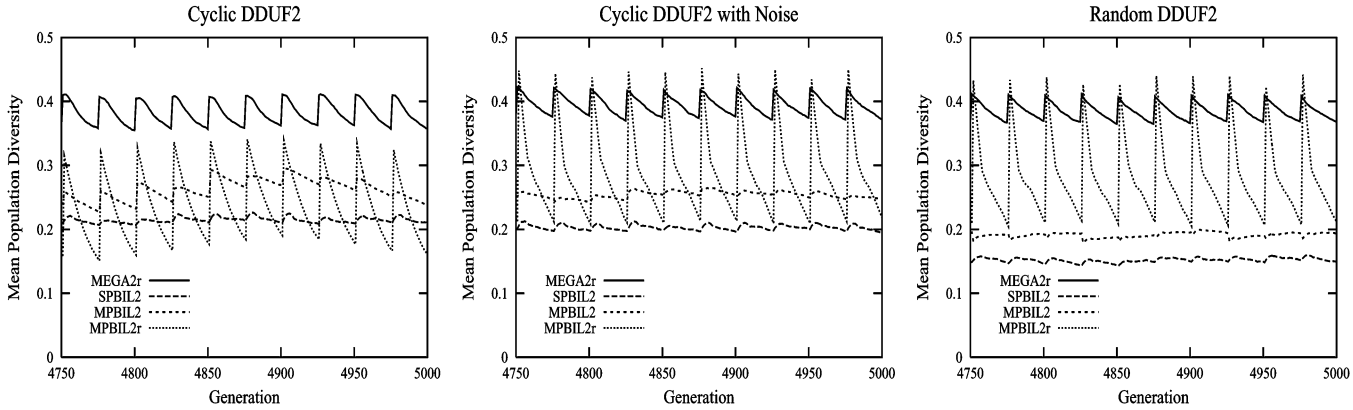

Fig. 16. Dynamic population diversity of algorithms with two populations for the last ten environmental changes on DDUF2 with $\tau = 25$ and $\rho = 0.2$.

TABLE II
THE t-TEST RESULTS OF COMPARING MEGA2R, SPBIL2, MPBIL2, AND MPBIL2R ON DDUFS

t-test Result	$DDUF1$				$DDUF2$				$DDUF3$			
Environment Dynamics	$\tau = 10$		$\tau = 25$		$\tau = 10$		$\tau = 25$		$\tau = 10$		$\tau = 25$	
Cyclic, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$MEGA2r - ISGA$	$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$	
$SPBIL2 - SPBIL$	$s+$ $s+$ $s+$ $s+$		$s-$ $+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2 - MPBIL$	$s-$ $s-$ $+$ $s-$		$s-$ $s-$ $-$ $s-$		$s-$ $s-$ $-$ $-$		$s-$ $s-$ $-$ $-$		$s-$ $s-$ $s-$ $s+$		$s-$ $+$ $s-$ $s+$	
$MPBIL2 - MEGA2r$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s-$ $s-$		$s+$ $s+$ $+$ $s-$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MEGA2r$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MPBIL$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s-$ $s-$ $s-$ $s-$		$s+$ $s+$ $s+$ $s-$	
$MPBIL2r - MPBIL2$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$-$ $-$ $s+$ $s-$		$s+$ $s+$ $s+$ $s-$	
Cyclic with Noise, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$MEGA2r - ISGA$	$s+$ $s+$ $s-$ $s-$		$s-$ $-$ $s-$ $s-$		$s+$ $s+$ $s+$ $s-$		$s+$ $s+$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$	
$SPBIL2 - SPBIL$	$s+$ $s+$ $s+$ $s+$		$s-$ $+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2 - MPBIL$	$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$		$-$ $s-$ $s-$ $s-$		$-$ $s-$ $s-$ $s-$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s-$ $s+$	
$MPBIL2 - MEGA2r$	$s-$ $s-$ $s-$ $s+$		$s+$ $s-$ $s+$ $s+$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s+$		$s-$ $s-$ $s-$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MEGA2r$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MPBIL$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MPBIL2$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $-$		$s+$ $s+$ $s+$ $s-$	
Random, $\rho \Rightarrow$	0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0		0.1 0.2 0.5 1.0	
$MEGA2r - ISGA$	$s-$ $s+$ $s+$ $s-$		$s-$ $s-$ $s+$ $s-$		$s+$ $s+$ $s+$ $-$		$-$ $s+$ $s+$ $s-$		$s-$ $s-$ $s-$ $s-$		$s-$ $s-$ $s-$ $s-$	
$SPBIL2 - SPBIL$	$-$ $s+$ $s+$ $s+$		$+$ $-$ $s+$ $s+$		$+$ $s+$ $s+$ $s+$		$+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2 - MPBIL$	$+$ $s+$ $s-$ $s-$		$+$ $+$ $s-$ $s-$		$+$ $s+$ $s-$ $-$		$+$ $s+$ $s-$ $+$		$s+$ $s+$ $s-$ $s+$		$s+$ $s+$ $s-$ $s+$	
$MPBIL2 - MEGA2r$	$s+$ $s-$ $s-$ $s+$		$s+$ $s+$ $s-$ $s+$		$s-$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s-$		$s+$ $s-$ $s-$ $s+$		$s+$ $s+$ $s-$ $s+$	
$MPBIL2r - MEGA2r$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$-$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$	
$MPBIL2r - MPBIL$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s-$		$s+$ $s+$ $s+$ $+$	
$MPBIL2r - MPBIL2$	$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s+$		$s+$ $s+$ $s+$ $s-$		$s+$ $s+$ $s+$ $s-$	

with noise and random dynamic environments, each time when change occurs, the restart scheme brings in a high population diversity and enables MPBIL2r to climb back to a relatively high fitness level during each environmental period.

VII. CONCLUSION AND FUTURE WORK

In this paper, an associative memory scheme is extensively investigated for PBIL algorithms in dynamic environments. Within this memory scheme, the working probability vector is taken as the environmental information and is stored together with the best sample it creates in the memory in a dynamic time pattern. When the environment changes, the stored probability vector associated with the memory sample with the best reevaluated fitness in the new environment is reactivated and directly brings PBILs into an associated old environment. This reactivated old environment may be very close to the newly changed environment. Hence, PBILs may reach a high fitness level quickly when the environment changes. In this paper, we also investigated the interactions between the memory scheme and several other approaches, e.g., random immigrants, multipopulation, and restart schemes, for PBILs in dynamic environments.

In order to test the performance of EAs, as another key contribution, this paper also proposes a DOP generator that can construct dynamic environments with tunable difficulty. With this generator, it is easy to construct cyclic, cyclic with noise, and random dynamic environments from any binary-encoded stationary problem. Hence, we can more thoroughly test and analyze memory schemes, in particular, and other approaches, in general, for EAs in dynamic environments.

Using the proposed DOP generator, a series of dynamic test problems are systematically constructed and experiments were carried out to compare the performance of investigated algorithms. From the experimental results, several conclusions can be drawn on the dynamic test environments.

First, the investigated memory scheme is efficient for improving PBIL's performance for DOPs, especially in cyclic dynamic environments.

Second, the interaction between memory and random immigrants depends on the dynamic environments. The addition of random immigrants improves the performance of memory-enhanced PBILs on most dynamic problems. However, when the random immigrants scheme is used, the effect of adding the memory scheme may be positive on PBIL's performance on cyclic DOPs and negative in random dynamic environments.

Third, there exist different interactions between memory and multipopulation schemes for PBILs. When memory is used, simply introducing an extra probability vector may be negative to PBIL's performance. However, when restart is combined with the multipopulation scheme, PBIL's performance can be significantly improved in different kinds of dynamic environments.

Fourth, the studied memory scheme for PBILs has a stronger effect than the memory scheme for GAs. This is because when a change occurs the reactivated probability vector in memory-enhanced PBILs can trigger an old environment more directly than the solutions in the memory in memory-enhanced GAs can do.

Fifth, the difficulty of DOPs depends on the environmental dynamics, including the cyclicity, severity and speed of changes, and the difficulty of the base stationary problems. As to the difficulty of environmental dynamics, the existence of noise on the cyclicity may overweigh randomness. The existence of deception in the base stationary problem may be beneficial to EA's performance in dynamic environments.

Generally speaking, the experimental results indicate that PBIL with the hybrid scheme of memory and multipopulation with restart can be a good EA optimizer for dynamic problems.

There are several future works relevant to this paper. A straightforward work is to extend the idea of associative memory scheme to other EAs. For example, extending the idea to the GA has shown some promising result [45]. We believe the proposed memory scheme should also improve the performance of those EAs based on probabilistic models in dynamic environments, such as the estimation of distribution algorithms [21], [29], of which PBILs are a subclass of EAs. Devising other memory management and retrieval mechanisms and hybrid memory schemes would be another interesting future work for PBILs and other EAs in dynamic environments. The third future work would be formally analyzing the behavior of PBILs and other EAs, with or without memory, under dynamic environments systematically constructed by the generator proposed in this paper. Finally, a comprehensive comparison of memory-enhanced EAs, including associative memory, direct memory, implicit memory [30], [38], and hybrid memory schemes [43], [46], is now under investigation.

ACKNOWLEDGMENT

The authors are grateful to Dr. G. Greenwood for being the Acting Editor-in-Chief of this paper and to the anonymous associate editor and reviewers for their thoughtful suggestions and constructive comments.

REFERENCES

[1] K. Atashkari, N. Nariman-Zadeh, A. Pilechi, A. Jamali, and X. Yao, "Thermodynamic Pareto optimization of turbojet engines using multi-objective genetic algorithms," *Int. J. Thermal Sci.*, vol. 44, no. 11, pp. 1061–1071, Nov. 2005.

[2] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.

[3] S. Baluja and R. Caruana, "Removing the genetics from the standard genetic algorithm," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 38–46.

[4] S. Baluja, "An empirical comparison of seven iterative and evolutionary function optimization heuristics," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-95-193, 1995.

[5] C. N. Bendtsen and T. Krink, "Dynamic memory model for non-stationary optimization," in *Proc. 2002 Congr. Evol. Comput.*, 2002, pp. 145–150.

[6] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," in *Proc. 1999 Congr. Evol. Comput.*, 1999, vol. 3, pp. 1875–1882.

[7] J. Branke, T. Kaußler, C. Schmidth, and H. Schmeck, "A multi-population approach to dynamic optimization problems," in *Proc. 4th Int. Conf. Adaptive Comput. Des. Manuf.*, 2000, pp. 299–308.

[8] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Norwell, MA: Kluwer, 2002.

[9] H. G. Cobb, "An investigation into the use of hypermutation as an Adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," Naval Research Lab., Washington, DC, Tech. Rep. AIC-90-001, 1990.

[10] H. G. Cobb and J. J. Grefenstette, "Genetic algorithms for tracking changing environments," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 523–530.

[11] D. Dasgupta and D. McGregor, "Nonstationary function optimization using the structured genetic algorithm," in *Proc. 2nd Int. Conf. Parallel Problem Solving From Nature*, 1992, pp. 145–154.

[12] A. Gaspar and P. Collard, "From gas to artificial immune systems: Improving adaptation in time dependent optimization," in *Proc. 1999 Congr. Evol. Comput.*, 1999, vol. 3, pp. 1859–1866.

[13] A. Gaspar and P. Collard, "Two models of immunization for time dependent optimization," in *Proc. 2000 IEEE Int. Conf. Syst., Man, Cybern.*, 2000, vol. 1, pp. 113–118.

[14] D. E. Goldberg and R. E. Smith, "Nonstationary function optimization using genetic algorithms with dominance and diploidy," in *Proc. 2nd Int. Conf. Genetic Algorithms*, J. J. Grefenstelle, Ed., 1987, pp. 59–68.

[15] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA: Kluwer, 2002.

[16] J. J. Grefenstette, "Genetic algorithms for changing environments," in *Proc. 2nd Int. Conf. Parallel Problem Solving From Nature*, R. Männer and B. Manderick, Eds., 1992, pp. 137–144.

[17] J. J. Grefenstette, "Evolvability in dynamic fitness landscapes: A genetic algorithm approach," in *Proc. 1999 Congr. Evol. Comput.*, 1999, vol. 3, pp. 2031–2038.

[18] H. Handa, L. Chapman, and X. Yao, "Robust route optimisation for gritting/salting trucks: A CERCIA experience," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 6–9, Feb. 2006.

[19] H. Handa, D. Lin, L. Chapman, and X. Yao, "Robust solution of salting route optimisation using evolutionary algorithms," in *Proc. 2006 IEEE Congr. Evol. Comput.*, 2006, pp. 10455–10462.

[20] D. L. Hartl and E. W. Jones, *Genetics: Principles and Analysis*, 4th ed. Sudbury, MA: Jones and Bartllet, 1998.

[21] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer, 2002.
[22] J. Lewis, E. Hart, and G. Ritchie, "A comparison of dominance mechanisms and simple mutation on non-stationary problems," in *Proc. 4th Int. Conf. Parallel Problem Solving From Nature*, A. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel, Eds., 1998, pp. 139–148.
[23] B. Li, J. Lin, and X. Yao, "A novel evolutionary algorithm for determining unified creep damage constitutive equations," *Int. J. Mech. Sci.*, vol. 44, no. 5, pp. 987–1002, 2002.
[24] S. J. Louis and Z. Xu, "Genetic algorithms for open shop scheduling and re-scheduling," in *Proc. 11th ISCA Int. Conf. Comput. Their Appl.*, M. E. Cohen and D. L. Hudson, Eds., 1996, pp. 99–102.
[25] N. Mori, H. Kita, and Y. Nishikawa, "Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed., San Mateo, CA, 1997, pp. 299–306.
[26] R. W. Morrison and K. A. De Jong, "A test problem generator for non-stationary environments," in *Proc. 1999 Congr. Evol. Comput.*, 1999, vol. 3, pp. 2047–2053.
[27] R. W. Morrison and K. A. De Jong, "Triggered hypermutation revisited," in *Proc. 2000 Congr. Evol. Comput.*, 2000, pp. 1025–1032.
[28] R. W. Morrison, *Designing Evolutionary Algorithms for Dynamic Environments*. Berlin, Germany: Springer-Verlag, 2004.
[29] H. Mühlenbein and G. Paaß, "From recombination of genes to the estimation of distributions I. binary parameters," in *Proc. 4th Int. Conf. Parallel Problem Solving From Nature*, H. M. Voigt, W. Ebeling, I. Rechenberg, and H. -P. Schwefel, Eds., 1996, pp. 178–187.
[30] K. P. Ng and K. C. Wong, "A new diploid scheme and dominance change mechanism for non-stationary function optimisation," in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., 1995, pp. 159–166.
[31] C. L. Ramsey and J. J. Grefenstette, "Case-based initialization of genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 84–91.
[32] A. Simões and E. Costa, "On biologically inspired genetic operators: Using transformation in the standard genetic algorithm," in *Proc. 2001 Genetic Evol. Comput. Conf.*, 2001, pp. 584–591.
[33] A. Simões and E. Costa, "An immune system-based genetic algorithm to deal with dynamic environments: Diversity and memory," in *Proc. 6th Int. Conf. Neural Networks and Genetic Algorithms*, 2003, pp. 168–174.
[34] A. Simões and E. Costa, "Improving the genetic algorithm's performance when using transformation," in *Proc. 6th Int. Conf. Neural Networks and Genetic Algorithms*, 2003, pp. 175–181.
[35] R. Tinos and S. Yang, "A self-organizing random immigrants genetic algorithm for dynamic optimization problems," *Genetic Programming and Evolvable Machines*, vol. 8, no. 3, pp. 255–286, Sep. 2007.
[36] T. Trojanowski and Z. Michalewicz, "Searching for optima in non-stationary environments," in *Proc. 1999 Congr. Evol. Comput.*, 1999, pp. 1843–1850.
[37] K. Trojanowski and Z. Michalewicz, "Evolutionary optimization in non-stationary environments," *J. Comput. Sci. Technol.*, vol. 1, no. 2, pp. 93–124, 2000.
[38] A. Ş Uyar and A. E. Harmanci, "A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments," *Soft Comput.*, vol. 9, no. 11, pp. 803–815, 2005.
[39] S. Yang, "Non-stationary problem optimization using the primal-dual genetic algorithm," in *Proc. 2003 Congr. Evol. Comput.*, 2003, vol. 3, pp. 2246–2253.
[40] S. Yang, "Constructing dynamic test environments for genetic algorithms based on problem difficulty," in *Proc. 2004 Congr. Evol. Comput.*, 2004, vol. 2, pp. 1262–1269.
[41] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comput.*, vol. 9, no. 11, pp. 815–834, 2005.
[42] S. Yang, "Population-based incremental learning with memory scheme for changing environments," in *Proc. 2005 Genetic Evol. Comput. Conf.*, 2005, vol. 1, pp. 711–718.
[43] S. Yang, "Memory-based immigrants for genetic algorithms in dynamic environments," in *Proc. 2005 Genetic Evol. Comput. Conf.*, 2005, vol. 2, pp. 1115–1122.
[44] S. Yang, "A comparative study of immune system based genetic algorithms in dynamic environments," in *Proc. 2006 Genetic Evol. Comput. Conf.*, 2006, pp. 1377–1384.
[45] S. Yang, "Associative memory scheme for genetic algorithms in dynamic environments," in *Applications of Evolutionary Computing.*, Berlin, 2006, vol. 3907, Lecture Notes in Computer Science, pp. 788–799.
[46] S. Yang, "Genetic algorithms with memory and elitism based immigrants in dynamic environments," *Evolutionary Computation*, 2008.
[47] L. D. Whitley, "Fundamental principles of deception in genetic search," in *Foundations of Genetic Algorithms 1*, G. J. E. Rawlins, Ed., 1991, pp. 221–241.
[48] J. X. Yu, X. Yao, C.-H. Choi, and G. Gou, "Materialized view selection as constrained evolutionary optimization," *IEEE Trans. Syst., Man, Cybern.*, vol. 33, pt. C: Applications and Reviews, pp. 458–467, Nov. 2003.

Shengxiang Yang (M'00) received the B.Sc. and M.Sc. degrees in automatic control and the Ph.D. degree in systems engineering from Northeastern University, Shenyang, China, in 1993, 1996, and 1999 respectively.

He was a Postdoctoral Research Associate in the Department of Computer Science, King's College London, from October 1999 to October 2000. He is currently a Lecturer in the Department of Computer Science, University of Leicester, Leicester, U.K. He serves as the area editor, associate editor, or editorial board member for three international journals. He has published over 60 papers in books, journals, and conferences. His current research interests include evolutionary and genetic algorithms, artificial neural networks for combinatorial optimization problems, scheduling problems, dynamic optimization problems, and network flow problems and algorithms.

Dr. Yang is a member the ACM Special Interest Group on Genetic and Evolutionary Computation (SIGEVO). He is a member of the Working Group on Evolutionary Computation in Dynamic and Uncertain Environments, Evolutionary Computation Technical Committee, IEEE Computational Intelligence Society.

Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, in 1985, and the Ph.D. degree from USTC in 1990.

He was an Associate Lecturer and Lecturer from 1985 to 1990 at USTC, while working towards the Ph.D. degree. He took up a Postdoctoral Fellowship in the Computer Sciences Laboratory, Australian National University (ANU), Canberra, in 1990, and continued his work on simulated annealing and evolutionary algorithms. He joined the Knowledge-Based Systems Group, CSIRO Division of Building, Construction and Engineering, Melbourne, in 1991, working primarily on an industrial project on automatic inspection of sewage pipes. He returned to Canberra in 1992 to take up a lectureship in the School of Computer Science, University College, University of New South Wales (UNSW), Australian Defence Force Academy (ADFA), where he was later promoted to a Senior Lecturer and Associate Professor. Attracted by the English weather, he moved to the University of Birmingham, Birmingham, U.K., as a Professor of Computer Science in 1999. Currently, he is the Director of the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), a Distinguished Visiting Professor of the University of Science and Technology of China, Hefei, and a visiting professor of three other universities. He has more than 200 publications. He is an associate editor or editorial board member of several journals. He is the Editor of the World Scientific Book Series on *Advances in Natural Computation*. He has given more than 45 invited keynote and plenary speeches at conferences and workshops worldwide. His major research interests include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint handling techniques, computational time complexity of evolutionary algorithms, coevolution, iterated prisoner's dilemma, data mining, and real-world applications.

Dr. Yao was awarded the President's Award for Outstanding Thesis by the Chinese Academy of Sciences for his Ph.D. work on simulated annealing and evolutionary algorithms. He won the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.