Hyper-Learning for Population-Based Incremental Learning in Dynamic Environments

Shengxiang Yang and Hendrik Richter

Abstract— **The population-based incremental learning (PBIL) algorithm is a combination of evolutionary optimization and competitive learning. Recently, the PBIL algorithm has been applied for dynamic optimization problems. This paper investigates the effect of the learning rate, which is a key parameter of PBIL, on the performance of PBIL in dynamic environments. A hyper-learning scheme is proposed for PBIL, where the learning rate is temporarily raised whenever the environment changes. The hyper-learning scheme can be combined with other approaches, e.g., the restart and hypermutation schemes, for PBIL in dynamic environments. Based on a series of dynamic test problems, experiments are carried out to investigate the effect of different learning rates and the proposed hyper-learning scheme in combination with restart and hypermutation schemes on the performance of PBIL. The experimental results show that the learning rate has a significant impact on the performance of the PBIL algorithm in dynamic environments and that the effect of the proposed hyper-learning scheme depends on the environmental dynamics and other schemes combined in the PBIL algorithm.**

I. Introduction

Dynamic optimization problems (DOPs) are a class of challenging optimization problems that involve changes over time regarding the optmization goal, problem instances, and/or constraints. DOPs are pervasive in real world optimization problems. They challenge traditional optimization algorithms as well as conventional evolutionary algorithms (EAs) due to the requirement of adapting to the changing environment with time. For DOPs, the aim is to develop algorithms that can track the changing optimum instead of locating a fixed optimum in the search space.

When applied for DOPs, conventional EAs face the convergence problem: once converged, EAs can not adapt well to the following new environments. In order to enhance the performance of EAs in dynamic environments, several approaches have been developed in the literature [10]. Generally speaking, these approaches can be classified into four types. The first type of approaches belongs to the *diversity scheme*, which maintains the diversity level of the population by inserting random immigrants [9] or guided immigrants [19] into the population during the run of EAs. The second type uses *memory* [14], [6], [16], [22], to store and reuse useful information to efficiently adapt EAs in dynamic environments, especially in cyclic dynamic environments. The third type uses *multi-population* schemes [7], [15] to distribute the search forces into the search space. The fourth type uses *adaptive* or *self-adaptive* schemes to adjusts genetic operators and/or relevant parameters to adapt EAs to the new environment whenever a change occurs, e.g, the hypermutation scheme [8] and self-adaptive evolution strategies [1], [2]. Of these four types of approaches devised for EAs for DOPs, the fourth type of adaptive schemes has received relatively less research so far. This paper investigates a method that belongs to the fourth type of approaches for EAs in dynamic environments.

The population-based incremental learning (PBIL) algorithm was first proposed by Baluja [3], which combines the idea of evolutionary optimization and competitive learning. PBIL explicitly maintains the statistics contained in the population of EAs [4]. PBILs have been successfully applied for numerous stationary benchmark and real-world problems [12]. Recently, the PBIL algorithm has been investigated for DOPs in the literature [21], [22].

This paper investigates the effect of the learning rate on the performance of PBILs in dynamic environments. A *hyper-learning* scheme is proposed for PBILs to address DOPs, where the learning rate is temporarily raised whenever the environment changes. The hyper-learning scheme can be combined with other diversity schemes, e.g., restart and hypermutation schemes, for PBILs in dynamic environments. Using the dynamic problem generator proposed in [17], [21], a series of DOPs are constructed as the dynamic test environments and experiments are carried out to investigate the performance of PBILs with different learning rates and the performance of PBILs with the hyper-learning scheme in combination with restart and hypermutation schemes for DOPs. Based on the experimental results, the effect of the learning rate and the hyper-learning scheme on the performance of PBILs in dynamic environments is analysed.

The rest of this paper is organized as follows. The next section describes the PBIL algorithm and some work on PBIL for DOPs. Section III describes the proposed hyper-learning scheme for PBILs and some PBILs studied in this paper, which integrate the hyper-learning scheme in combination with restart and hypermutation schemes. Section IV presents the experimental design, including the dynamic test environments, parameter settings, and performance measure. Section V presents the experimental results and analysis. Finally, Section VI concludes this paper with discussions on relevant future work.

Shengxiang Yang is with the Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom (email: s.yang@mcs.le.ac.uk).

Hendrik Richter is with HTWK Leipzig, Fachbereich Elektrotechnik und Informationstechnik, Institut Mess–, Steuerungs– und Regelungstechnik, D–04125 Leipzig, Germany (email: richter@fbeit.htwk-leipzig.de)

The work by Shengxiang Yang was supported by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom under Grant EP/E060722/1.

```
$t := 0$
initialize the probability vector $\vec{P}(0) := \vec{0.5}$
generate a set $S(0)$ of $n$ samples by $\vec{P}(0)$
**repeat**
  evaluate samples in $S(t)$
  learn $\vec{P}(t)$ toward the best sample $\vec{B}(t)$ in $S(t)$
    according to Eq. (1)
  mutate $\vec{P}(t)$ according to Eq. (2)
  generate a set $S(t)$ of $n$ samples by $\vec{P}(t)$
  retrieve the best sample from $S(t-1)$ to replace a
    random sample in $S(t)$
  $t := t + 1$
**until** a termination condition holds    // e.g., $t > t_{max}$
```

Fig. 1. Pseudocode of the standard PBIL algorithm (SPBIL) with the elitism scheme.

II. Population-Based Incremental Learning

The PBIL algorithm 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. Each element P_i ($i = 1, \ldots, l$) in the probability vector is the probability of creating an allele "1" in the locus i. Hence, a solution is sampled from the 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. The pseudocode for the standard PBIL algorithm (SPBIL) investigated in this paper is shown in Fig. 1.

The standard PBIL starts from a probability vector that has a value of 0.5 for each element. This probability vector can be 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 $\vec{P}(t)$[1]. The samples are then evaluated using the problem-specific fitness function. Then, the probability vector is learnt towards the best solution $\vec{B}(t)$ of the set $S(t)$ according to the following learning rule.

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

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, it may undergo a bitwise mutation process [5]. Mutation is applied to PBILs studied in this paper. In order to keep the diversity of sampling, 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 ($i = 1, \ldots, l$), if a random number $r = rand(0.0, 1.0) < p_m$ (p_m is the mutation probability), then P_i is mutated as follows:

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

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

As the search progresses, the elements in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0, which will produce high evaluation solutions when the probability vector is sampled. 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.

PBIL has been applied for many optimization problems with promising results [12]. Most of these applications are for stationary problems. Recently, there have been some works on studying PBIL algorithms for DOPs. Yang and Yao [21] have investigated PBIL for DOPs by introducing dualism and a scheme similar to the random immigrants method [9] to improve their performance in dynamic environments. In [18], [22], an associative memory scheme has been introduced into PBIL for DOPs with some promising results. In this paper, a hyper-learning scheme is proposed for PBIL in dynamic environments, which is described in the following section.

III. Hyper-Learning for PBIL

As aforementioned, the PBIL algorithm maintains a probability vector and evolves it through creating samples from it and learning toward the best sample created. The driving force for PBIL to solve an optimization problem lies in the learning of the probability vector toward the best sample created from it iteratively. Usually, with the running of PBIL, the probability vector will eventually converge to the one with either 0.0 or 1.0 in each element, which will produce the optimal solution(s) when sampled in stationary environments due to the learning process. Here, the learning rate parameter α controls how fast the probability vector moves toward the best sample in each iteration. The bigger the value of α, the faster the evolving process (though the probability vector may converge into local optima).

Usually, PBIL with a proper learning rate can converge well to the optimal solution(s) in stationary environments, which gives PBIL an advantage in comparison with other EAs [5]. However, in dynamic environments, convergence becomes a big problem for PBIL algorithms since it deprives the diversity of the samples created and hence make it hard for PBIL algortihms to adapt to the new environment when a change occurs. To address the convergence problem, several approaches can be developed to re-introduce diversity after a change occurs, e.g., the restart scheme.

However, using only these diversity schemes may not adapt PBILs to a new environment to its best. We may need

[1]The elisitsm of size 1 is used in all PBIL algorithms studied in this paper. That is, the best sample created by $\vec{P}(t-1)$ is inserted into $S(t)$, replacing a random sample in $S(t)$ created by $\vec{P}(t)$.

```
$t := 0$
initialize the probability vector $\vec{P}(0) := \vec{0.5}$
generate a set $S(0)$ of $n$ samples by $\vec{P}(0)$
**repeat**
  evaluate smaples in $S(t)$
  learn $\vec{P}(t)$ toward the best sample $\vec{B}(t)$ in $S(t)$
    according to Eq. (1)
  mutate $\vec{P}(t)$ according to Eq. (2)
  generate a set $S(t)$ of $n$ samples by $\vec{P}(t)$
  retrieve the best sample from $S(t-1)$ to replace a
    random sample in $S(t)$

  **if** the environment changes **then**
    re-initialize $\vec{P}(t) := \vec{0.5}$
    **if** hyper-learning is used **then**     // for PBILrl
      raise $\alpha$ from $\alpha^l$ to $\alpha^u$ for $n_{hl}$ generations

  $t := t + 1$
**until** a termination condition holds     // e.g., $t > t_{max}$
```

Fig. 2. Pseudo-code for the PBIL with re-start (PBILr) and the PBIL with restart and hyper-learning (PBILrl).

```
$t := 0$
initialize the probability vector $\vec{P}(0) := \vec{0.5}$
generate a set $S(0)$ of $n$ samples by $\vec{P}(0)$
**repeat**
  evaluate smaples in $S(t)$
  learn $\vec{P}(t)$ toward the best sample $\vec{B}(t)$ in $S(t)$
    according to Eq. (1)
  mutate $\vec{P}(t)$ according to Eq. (2)
  generate a set $S(t)$ of $n$ samples by $\vec{P}(t)$
  retrieve the best sample from $S(t-1)$ to replace a
    random sample in $S(t)$

  **if** the environment changes **then**
    raise $p_m$ from $p_m^l$ to $p_m^u$ for $n_{hm}$ generations
    **if** hyper-learning is used **then**     // for PBILml
      raise $\alpha$ from $\alpha^l$ to $\alpha^u$ for $n_{hl}$ generations

  $t := t + 1$
**until** a termination condition holds     // e.g., $t > t_{max}$
```

Fig. 3. Pseudo-code for the PBIL with hypermutation (PBILm) and the PBIL with hypermutation and hyper-learning (PBILml).

to apply a high learning rate to learn the probability vector faster toward those really useful samples produced in a new environment and hence adapt PBIL algorithms more quickly toward the new environment. This thinking naturally leads to the introduction of the *hyper-learning* scheme into PBIL: whenever an environmental change occurs, the learning rate is temporarily raised for several generations from the normal learning rate.

Obviously, to realize its best advantage, the hyper-learning scheme should be combined with other diversity approaches for PBIL algorithms to address DOPs. In this paper, the hyper-learning scheme is combined with restart and hypermutation [8] schemes for PBIL, which are described in the following sub-sections respectively.

A. Hyper-Learning with Restart

Restart is a simple and natural way for EAs to address DOPs. For PBIL with the restart scheme, whenever the environment changes, the probability vector is re-initialized to the central probability vector. The pseudo-code of the PBIL with restart, denoted *PBILr* in this paper, is shown in Fig. 2. The corresponding PBIL with restart and hyper-learning schemes, denoted *PBILrl* in this paper, is also shown in Fig. 2. Within PBILrl, whenever the environment changes, the learning rate α is raised from the basic low value α^l to a high value α^u for the following n_{hl} generations.

B. Hyper-Learning with Hypermutation

Hypermutation is another scheme to re-introduce the population diversity for EAs to address DOPs and has been studied in several works [8], [13]. Hypermutation can also be integrated into PBIL to deal with DOPs. The pseudo-code of the PBIL with hypermutation, denoted *PBILm* in this paper, is shown in Fig. 3. Within PBILm, whenever the environment changes, the mutation probability p_m is raised from the basic low value p_m^l to a high value p_m^u for the following n_{hm} generations. The corresponding PBIL with the hypermutation and hyper-learning schemes, denoted *PBILml* in this paper, is also shown in Fig. 3.

IV. Experimental Design

A. Dynamic Test Environments

For this paper, we use the DOP generator proposed in [17], [21] to construct dynamic test problems. This generator can construct DOPs from any binary-encoded stationary function $f(\vec{x})$ as follows. Suppose the environment changes every τ generations. For each environment k, an XORing mask $\vec{M}(k)$ is incrementally generated as follows:

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

where "$\oplus$" is a bitwise exclusive-or (XOR) operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, and $0 \oplus 0 = 0$) and $\vec{T}(k)$ is an intermediate binary template generated for environment k. $\vec{T}(k)$ is generated with $\rho \times l$ ($\rho \in (0.0, 1.0]$) random loci set to 1 while the remaining loci set to 0. For the initial environment $k = 1$, $\vec{M}(1)$ is set to a zero vector, i.e., $\vec{M}(1) = \vec{0}$.

Given the above descriptions, an individual at generation t is then evaluated as follows:

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

where $k = \lceil t/\tau \rceil$ is the environmental index at time t. With this XOR DOP generator, τ and ρ control the speed and severity of environmental changes respectively. Smaller τ means faster changes while bigger ρ means severer changes.

In this paper, three 100-bit binary functions are selected as the base stationary functions to construct dynamic test environments. The first one is the well-known *OneMax* function that aims to maximize the number of ones in a binary string. The second one is a *Plateau* function, which consists of 25 contiguous 4-bit building blocks (BBs). Each BB contributes 4 to the total fitness if all bits inside it have the allele of one; otherwise, it contributes 0. The third problem is a 100-item 0-1 knapsack problem with the weight and profit of each item randomly created in the range of $[1, 30]$ and the capacity of the knapsack set to be half of the total weight of all items. The fitness of a feasible solution is the sum of the profits of the selected items. If a solution overfills the knapsack, its fitness is set to the difference between the total weight of all items and the weight of selected items, multiplied by a small factor 10^{-5} in order to make it in-competitive with those solutions that do not overfill the knapsack.

Dynamic environments are constructed from each of the three base functions using the aforementioned XOR DOP generator. For each dynamic environment, the landscape is periodically changed every τ generations during the run of a PBIL algorithm. In order to compare the performance of PBIL algorithms in different dynamic environments, the speed of change parameter τ is set to 20 and 50 respectively. The severity of change parameter ρ is set to 0.1, 0.2, 0.5, and 0.9 respectively.

B. Parameter Settings and Performance Measure

Two sets of experiments were carried out in this paper on the above constructed dynamic test environments. The first set of experiments investigates the effect of the learning rate on the performance of the standard PBIL, i.e., SPBIL, for DOPs. The second set investigates the effect of the hyper-learning scheme on the performance of several PBIL algorithms with restart or hypermutation enhancements, as described in Section III.

For all PBIL algorithms, some common parameters are set as follows: the population size $n = 100$, the learning rate $\alpha = 0.10$, the mutation probability $p_m = 0.05$ with the mutation shift $\delta = 0.05$, and elitism of size 1. For the first set of experiments, the learning rate α in SPBIL is set to 0.05, 0.10, 0.25, 0.50, and 0.75 respectively (and the SPBIL is denoted as α-*SPBIL* accordingly). For the second set of experiments, the parameters are set as follows. The learning rate α is fixed to 0.10 for SPBIL, PBILr, and PBILm and is set to the base value $\alpha^l = 0.10$ for normal generations or the hyper value $\alpha^u = 0.25$ for the interim generations when the hyper-learning scheme is triggered for PBILrl and PBILml. For PBILm and PBILml, the mutation probability p_m is set to the base value $p_m^l = 0.05$ for normal generations or the hyper value $p_m^u = 0.3$ for the interim generations when the hypermutation scheme is triggered. Whenever the environment changes, the hyper-mutation or hyper-learning schemes for PBILs are triggered for 5 generations, i.e., $n_{hm} = 5$ and $n_{hl} = 5$.

For each experiment of a PBIL algorithm on a DOP, 50 independent runs were executed with the same set of random seeds. For each run, 50 environmental changes were allowed. For each run the best-of-generation fitness was recorded every generation. The overall performance of an algorithm on a DOP is defined as:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{50}\sum_{j=1}^{50} F_{BOG_{ij}}\right), \tag{5}$$

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

V. Experimental Results and Analysis

A. Experimental Results on Selection Pressure

The experimental results of the first set of experiments are plotted in Fig. 4. The corresponding statistical results of comparing PBILs by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance are given in Table I. The t-test result with respect to $Alg.\ 1 - Alg.\ 2$ is shown as "$+$", "$-$", "$s+$", or "$s-$" when $Alg.$ 1 is better than, worse than, significantly better than, or significantly worse than $Alg.$ 2 respectively. From Fig. 4 and Table I, the following two results can be observed.

First, it can be seen that the learning rate does have a significant effect on the performance of SPBIL on most dynamic test problems. This result can be clearly seen from Table I, where most t-test results are shown as either "$s+$" or "$s-$".

Second, the exact effect of increasing the learning rate on the performance of SPBIL depends on the base function used for DOPs and the environmental dynamics. The effect is quite different across dynamic OneMax, Plateau and Knapsack problems. For dynamic OneMax problems, it seems that a higher learning rate increases the performance of SPBIL except for severely changing environments (e.g., $\rho = 0.5$ for $\tau = 20$ and $\rho = 0.9$ for both $\tau = 20$ and $\tau = 50$). For dynamic Plateau and Knapsack problems, when the learning rate is raised from 0.05 to 0.10, the performance of SPBIL improves, as indicated by the t-test results regarding 0.10-SPBIL $-$ 0.05-SPBIL. When the learning rate is increased to 0.25, the performance of SPBIL degrades on dynamic Plateau problems while improving on dynamic Knapsack problems. When the learning rate is further increased to 0.50 and 0.75, the performance of SPBIL degrades on both dynamic Plateau and Knapsack problems, as indicated in the corresponding t-test results in Table I. When the environment involves severe changes (i.e., $\rho = 0.9$), the performance of SPBIL degrades when the learning rate increases on most DOPs.

Generally speaking, it seems that setting the learning rate parameter α to 0.10 gives the best or second best performance of SPBIL on the test DOPs in comparison with other settings. For the following experiments, we set $\alpha = 0.10$ for relevant PBIL algorithms.

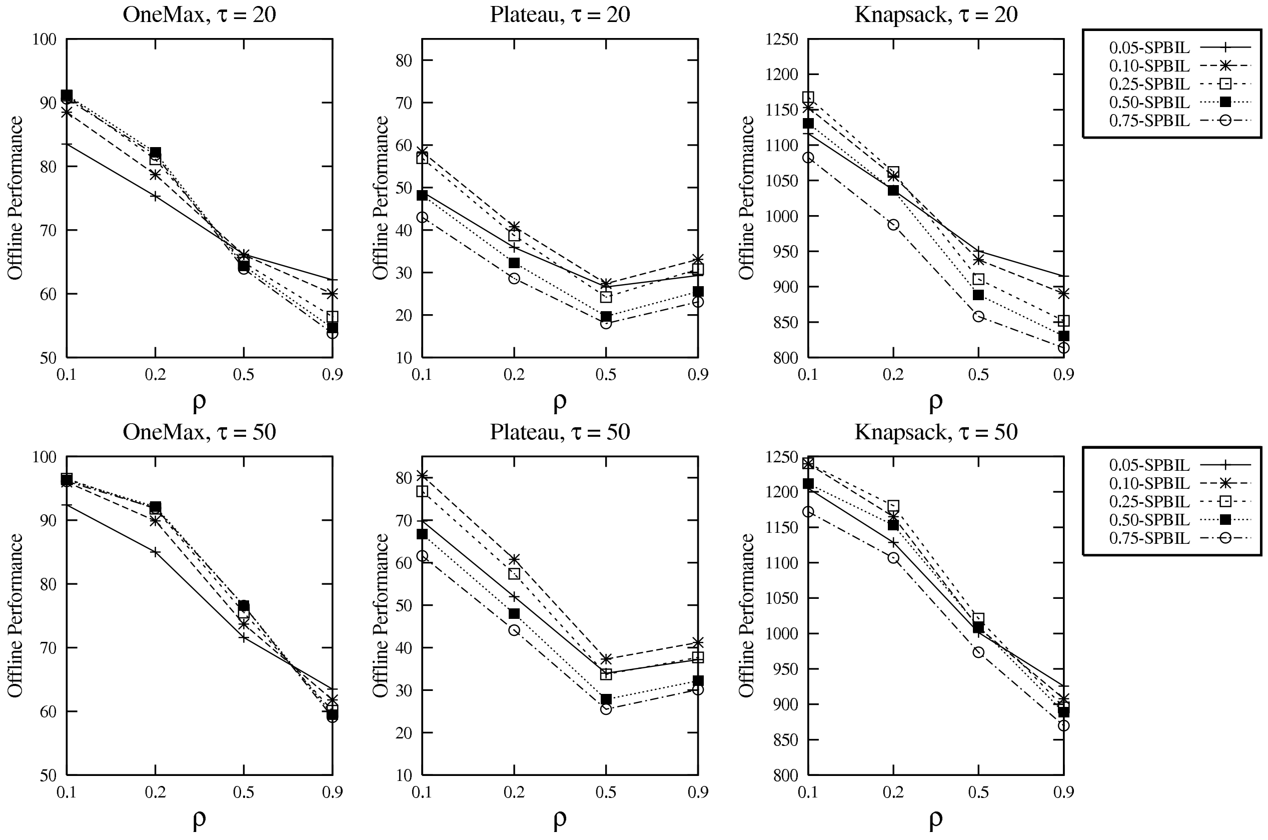

Fig. 4. Experimental results of comparing SPBILs with different learning rates α on DOPs.

TABLE I
THE t-TEST RESULTS OF COMPARING SPBILS WITH DIFFERENT LEARNING RATES ON DOPS.

t-test Result	*OneMax*				*Plateau*				*Knapsack*			
$\tau = 20$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
0.10-SPBIL − 0.05-SPBIL	$s+$	$s+$	$-$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s-$
0.25-SPBIL − 0.10-SPBIL	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$
0.50-SPBIL − 0.25-SPBIL	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.75-SPBIL − 0.50-SPBIL	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
0.10-SPBIL − 0.05-SPBIL	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
0.25-SPBIL − 0.10-SPBIL	$s+$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$+$	$s+$	$s+$	$s-$
0.50-SPBIL − 0.25-SPBIL	$s-$	$s+$	$s+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$
0.75-SPBIL − 0.50-SPBIL	$s-$	$s-$	$+$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$	$s-$

B. Experimental Results on Hyper-Learning

The experimental results of the second set of experiments regarding the effect of the hyper-learning scheme are plotted in Fig. 5. The corresponding t-test results of comparing PBILs are given in Table II. In order to better understand the performance of PBILs, the dynamic behaviour of PBILs with respect to the best-of-generation fitness against generations on DOPs with $\tau = 50$ and $\rho = 0.1$ and $\rho = 0.9$ is plotted in Fig. 6. In Fig. 6, the first 10 environmental changes, i.e., 500 generations, are shown and the data were averaged over 50 runs. From Figs. 5 and 6 and Table II, several results can be observed.

First, regarding the restart scheme, it can be seen that PBILr outperforms SPBIL on DOPs with ρ set to 0.5 or 0.9 while is beaten by SPBIL on DOPs with ρ set to 0.1 or 0.2, as indicated by the t-test results regarding PBILr − SPBIL in Table II. This happens because when the environment changes slightly, a high diversity introduced may divert the searching force too much and hence degrades the performance of PBIL algorithms. However, when the environment changes significantly, restart can bring in sufficient diversity

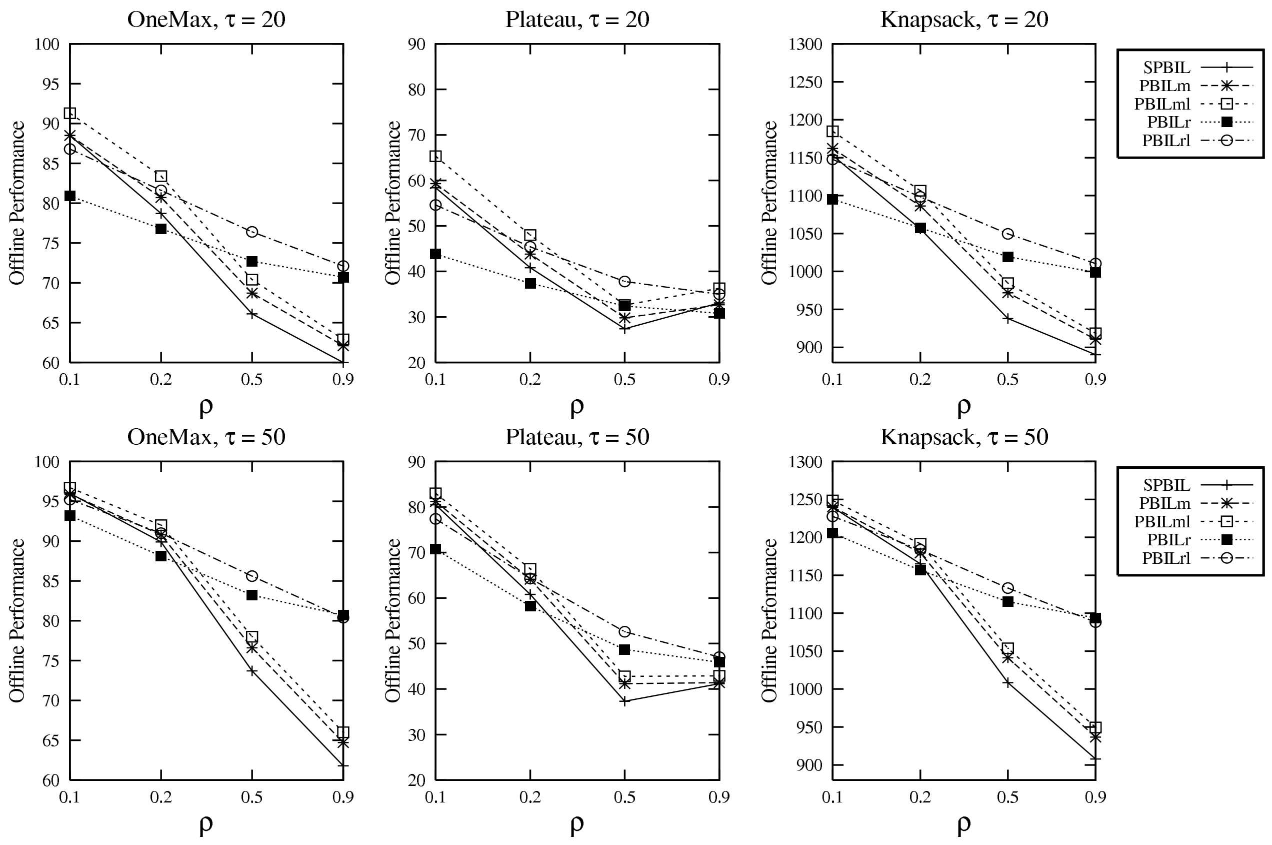

Fig. 5. Experimental results of comparing PBIL algorithms with and without hyper-learning on DOPs.

TABLE II
THE t-TEST RESULTS OF COMPARING DIFFERENT PBIL ALGORITHMS ON DOPS.

t-test Result	$OneMax$				$Plateau$				$Knapsack$			
$\tau = 20$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
PBILm − SPBIL	$+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$
PBILr − SPBIL	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$+$	$s+$	$s+$
PBILr − PBILm	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s+$
PBILml − PBILm	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
PBILrl − PBILr	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
PBILrl − PBILml	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s-$	$s-$	$s-$	$s+$	$s+$
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
PBILm − SPBIL	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$+$	$s+$	$s+$	$s+$
PBILr − SPBIL	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
PBILr − PBILm	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$
PBILml − PBILm	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$
PBILrl − PBILr	$s+$	$s+$	$s+$	$s-$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s+$	$s-$
PBILrl − PBILml	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$	$s-$	$s-$	$s+$	$s+$

for PBIL to search for the new optima, which may be far away from the optima of the previous environment.

Second, regarding the hypermutation scheme, it can be seen that hypermutation is beneficial for the performance of PBILs on almost all DOPs. This result indicates that it is important to introduce a proper level of diversity when the environment changes. Another observation lies in that the performance of PBILm and PBILml is more sensitive to the value of ρ. Their performance drops sharply when the value of ρ increases from 0.1 to 0.2, 0.5 to 0.9.

It can also be seen that the hypermutation scheme outperforms the restart scheme when the environment changes slightly, i.e., when $\rho = 0.1$ and 0.2. On the contrast, when the environment changes significantly, i.e., when $\rho = 0.5$

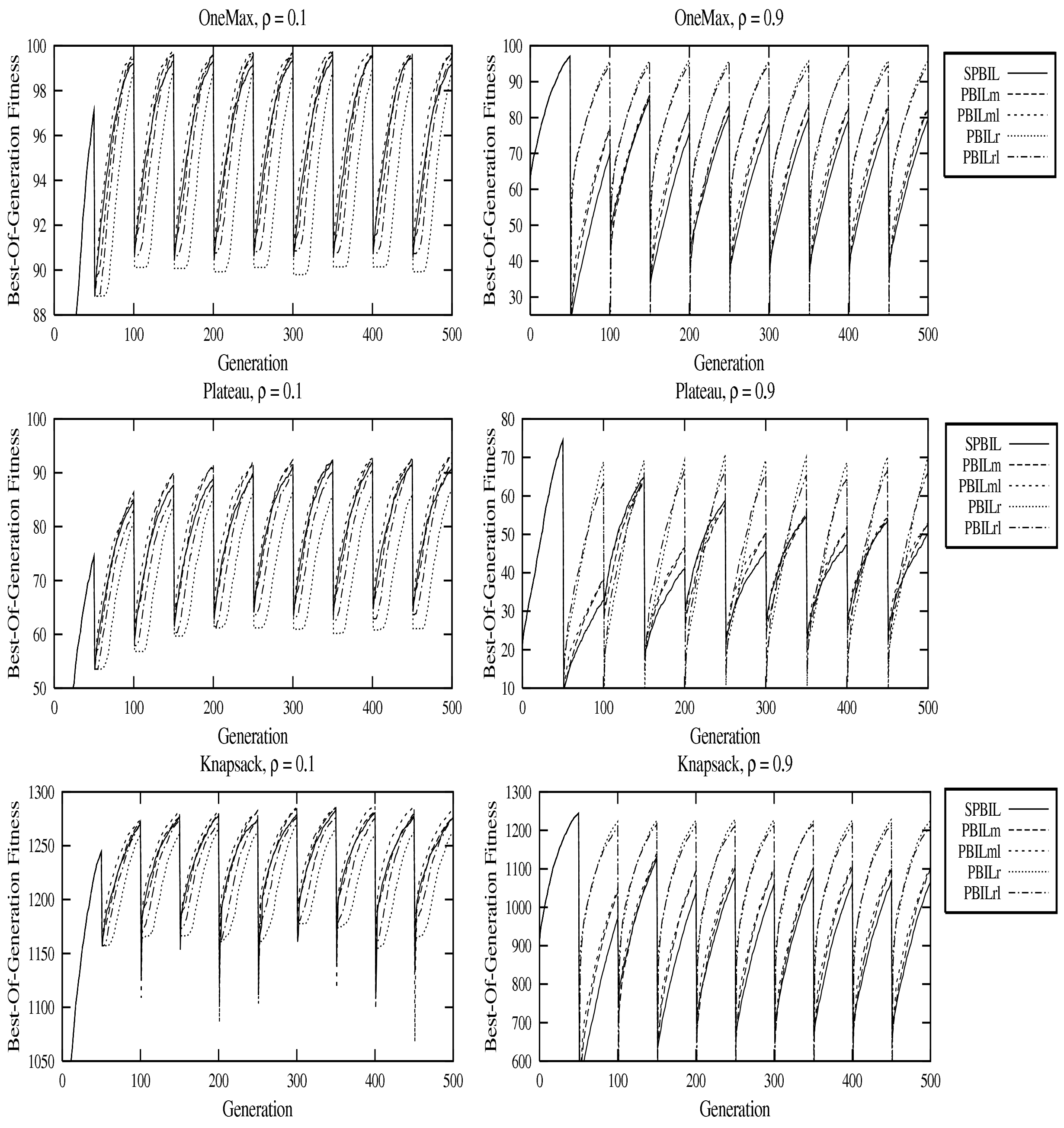

Fig. 6. Dynamic behaviour of PBIL algorithms on DOPs with $\tau = 50$ and $\rho = 0.1$ (Left) and $\rho = 0.9$ (Right).

and 0.9, hypermutation is beaten by restart, as indicated by the t-test results regarding PBILr − PBILm and PBILrl − PBILml in Table II respectively.

Finally, regarding the effect of the hyper-learning scheme, it can be seen that hyper-learning significantly improves the performance of both PBILr and PBILm on almost all dynamic functions, as indicated by the t-test results regarding PBILrl − PBILr and PBILml − PBILm in Table II respectively. This result confirms our expectation of combining the hyper-learning scheme with diversity schemes for PBIL algorithms in dynamic environments. The effect of the hyper-learning scheme can also be clearly seen from the dynamic behaviour of PBILrl on Dops with $\rho = 0.10$ in Fig. 6. Each time when the environment changes, it will take PBILr some time to make real searching progress while PBILrl can make real searching progress quite quickly after a change.

VI. Conclusion and Future Work

Developing adaptive genetic operators is one type of approaches for EAs to address dynamic environments. This paper investigates the effect of the learning rate on the performance of PBIL algorithms in dynamic environments and proposes a hyper-learning scheme for PBIL algorithms to address DOPs. When an environmental change occurs, the learning rate is temporarily raised. This hyper-learning scheme can be combined with other schemes in the literature, e.g. restart and hypermutation, for PBIL algorithms in dynamic environments.

The effect of the learning rate and the hyper-learning scheme for PBIL algorithms in dynamic environments were experimentally studied based on a series of dynamic test problems. From the experimental results and relevant analysis, three major conclusions can be drawn on the dynamic test environments. First, the learning rate does have a significant effect on the performance of PBIL algorithms in dynamic environments. Second, the effect of increasing the learning rate on the performance of PBIL in dynamic environments depends on the problem and environmental dynamics. Third, the proposed hyper-learning scheme significantly improves the performance of PBIL algorithms with diversity schemes in dynamic environments.

Generally speaking, this paper investigates the effect of the learning rate and the hyper-learning scheme for PBIL algorithms in dynamic environments with some preliminary experiments. The results observed may be used to guide the design of new PBIL algorithms for DOPs. For example, developing more efficient learning schemes that can adjust the learning rate adaptively during the running of PBIL algorithms may be an interesting future work. Combining the hyper-learning scheme with other mechanisms for PBIL in dynamic environments is another interesting future work.

REFERENCES

[1] D. V. Arnold and H.-G. Beyer. Random dynamics optimum tracking with evolution strategies. *Parallel Problem Solving from Nature VII*, pp. 3–12, 2002.

[2] D. V. Arnold and H.-G. Beyer. Optimum tracking with evolution strategies. *Evolutionary Computation*, 14(3): 291–308, 2006.

[3] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. *Technical Report CMU-CS-94-163*, Carnegie Mellon University, USA, 1994.

[4] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. *Proc. of the 12th Int. Conf. on Machine Learning*, pp. 38-46, 1995.

[5] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. *Technical Report CMU-CS-95-193*, Carnegie Mellon University, USA, 1995.

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

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

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

[9] J. J. Grefenstette. Genetic algorithms for changing environments. *Parallel Problem Solving from Nature II*, pp. 137–144, 1992.

[10] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments: a survey. *IEEE Trans. on Evol. Comput.*, 9(3): 303–317, 2005.

[11] J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. *Proc. of the 4th Int. Conf. on Parallel Problem Solving from Nature*, pp. 139–148, 1998.

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

[13] R. W. Morrison and K. A. De Jong. Triggered hypermutation revisited. *Proc. of the 2000 IEEE Congress on Evol. Comput.*, pp. 1025-1032, 2000.

[14] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1995.

[15] D. Parrott and X. Li. Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. on Evol. Comput.*, 10(4): 444-458, 2006.

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

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

[18] S. Yang. Population-based incremental learning with memory scheme for changing environments. *Proc. of the 2005 Genetic and Evolutionary Computation Conference*, vol. 1, pp. 711-718, 2005.

[19] S. Yang. Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 16(3): 385-416, 2008.

[20] S. Yang and R. Tinós. Hyper-selection in dynamic environments. *Proc. of the 2008 IEEE Congress on Evol. Comput.*, pp. 3185-3192, 2008.

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

[22] S. Yang and X. Yao. Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evol. Comput.*, 12(5): 542-561, 2008.