Soft Computing, 9(11): 815-834, 2005.
(DOI: 10.1007/s00500-004-0422-3)

Experimental Study on Population-Based Incremental Learning Algorithms for Dynamic Optimization Problems

Shengxiang Yang[1], **Xin Yao**[2]

[1] Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
`s.yang@mcs.le.ac.uk`

[2] School of Computer Science, University of Birmingham
Edgbaston, Birmingham B15 2TT, United Kingdom
`x.yao@cs.bham.ac.uk`

The date of receipt and acceptance will be inserted by the editor

Abstract Evolutionary algorithms have been widely used for stationary optimization problems. However, the environments of real world problems are often dynamic. This seriously challenges traditional evolutionary algorithms. In this paper, the application of Population-Based Incremental Learning (PBIL) algorithms, a class of evolutionary algorithms, for dynamic problems is investigated. Inspired by the complementarity mechanism in nature a Dual PBIL is proposed, which operates on two probability vectors that are dual to each other with respect to the central point in the genotype space. A diversity maintaining technique of combining the central probability vector into PBIL is also proposed to improve PBIL's adaptability in dynamic environments. In this paper, a new dynamic problem generator that can create required dynamics from any binary-encoded stationary problem is also formalized. Using this generator, a series of dynamic problems were systematically constructed from several benchmark stationary problems and an experimental study was carried out to compare the performance of several PBIL algorithms and two variants of standard genetic algorithm. Based on the experimental results, we carried out algorithm performance analysis regarding the weakness and strength of studied PBIL algorithms and identified several potential improvements to PBIL for dynamic optimization problems.

Key words Population-based incremental learning, dynamic optimization problem, dual population-based incremental learning, genetic algorithm, central probability vector, exclusive-or operator

1 Introduction

As a class of meta-heuristic algorithms, evolutionary algorithms (EAs) make use of principles of natural selection and population genetics. Due to the robust capability of finding solutions to difficult problems, EAs have become the optimization and search techniques of choice for many applications. Especially, they are widely applied for solving *stationary* optimization problems where the fitness landscape does not change during the course of computation [13]. However, the environments of real world optimization problems are often dynamic, where the problem fitness landscape changes over time. For example, in scheduling problems the scheduling demands and available resources may change over time. This forms a serious challenge to traditional EAs since they cannot adapt well to a changed environment once converged.

In recent years, there is a growing interest in the research of applying EAs for dynamic optimization problems since many of the problems that EAs are being used to solve are known to vary over time [1], [22]. Usually the dynamic environment requires EAs to maintain sufficient diversity for a continuous adaptation to the changing landscape. Researchers have developed many approaches into EAs to address this problem. Branke [7], [8] has grouped them into four categories: 1) increasing diversity after a change, such as the *hypermutation* scheme [9], [24]; 2) maintaining diversity throughout the run, such as the *random immigrants* scheme [15]; 3) memory-based methods, such as the *diploidy* and *multiploidy* approaches [12], [20], [26]; and 4) multi-population approaches [6].

In this paper, we investigate the application of a class of EAs, Population-Based Incremental Learning (PBIL) algorithms, for solving dynamic optimization problems. We study the effect of introducing several approaches into PBIL to address dynamic optimization problems, such as the multi-population and random immigrants methods. Inspired by the complementarity mechanism broadly existing in nature, we propose a Dual PBIL that operates on two probability vectors that are dual to each other with respect to the central point in the search space. To address the convergence problem, we also introduce a diversity maintaining technique, similar to the random immigrants method for GAs, into PBIL to improve its adaptability under dynamic environments.

In this paper, we also formalize a new dynamic problem generator, first applied in [30], which can generate required dynamics from a given stationary problem. Using this generator, we systematically construct a series of dynamic problems from two benchmark and one real-word stationary problems and carry out an experimental study comparing the investigated PBILs and two variants of standard genetic algorithm. Based on the analysis of the experimental results, we identify the weakness and strength of the studied PBILs and discuss some improvements to PBIL for dynamic optimization problems.

The rest of this paper is organized as follows. The next section briefly reviews some existing dynamic problem generators and presents the new dynamic problem generator. Sect. 3 details several algorithms investigated in this paper including our proposed Dual PBIL. Sect. 4 describes the test environment for this study, including the stationary test suite and related dynamic problems. The basic experimental results and relevant analysis are presented in Sect. 5. In Sect. 6, we investigate the introduction of a diversity maintaining probability vector into PBIL for dynamic optimization problems. Finally we conclude this paper in Sect. 7 and give out discussions on future work in Sect. 8.

2 Dynamic Problem Generators

2.1 Review of Existing Generators

Over the past few years, in order to study the performance of EAs for dynamic optimization problems researchers have developed a number of dynamic problem generators to create dynamic test environments. Generally speaking, these generators can be roughly divided into two types. The first type of constructing dynamic environments is quite simple. The environment is just switched between two or more stationary problems (or states of a problem). For example, many researchers have tested their algorithms on a time varying knapsack problem where the total weight capacity of the knapsack changes over time, usually oscillating between two or more fixed values [11], [20], [22], [26]. Cobb and Grefenstette [10] constructed a significantly changing environment that oscillates between two different fitness landscapes. For this type of generators, the dynamics of environmental change is mainly characterized by the speed of change. It can be fast or slow relative to EA time and is usually measured in EA generations.

The second type of dynamic problem generators starts from a predefined fitness landscape, usually constructed in n-dimensional real space [5], [16], [23], [28]. This stationary landscape is composed of a number of component landscapes (e.g., cones), each of which can change independently. Each component has its own morphology 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's generator [23], called $DF1$, defines the basic landscape in n-dimensional real space as follows:

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

where $\mathbf{x} = (x_1, \cdots, x_n)$ is a point in the landscape, m specifies the number of cones in the environment, and each cone i is independently specified by its height H_i, its slope R_i, and its center $X_i = (X_{i1}, \cdots, X_{in})$. These independently specified 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. With respect to how to change a parameter, there may be a variety of properties. For example, one property of the dynamics of environmental change is related to the magnitude or step size of change for each parameter. It may be large or small. Another dynamics property is related to the speed of change, which can be slow or fast.

2.2 A New Dynamic Problem Generator

In this paper, we formalize a new dynamic problem generator that can generate dynamic test problems from any binary encoded stationary problem. Given a stationary problem $f(\mathbf{x})$ ($\mathbf{x} \in \{0,1\}^l$ where l is the chromosome length), we can construct dynamic landscape from it as follows: we first create a binary mask $\mathbf{M} \in \{0,1\}^l$, randomly or in a controlled way, periodically or not. When evaluating an individual $\mathbf{x}$ in the population, we first perform the operation $\mathbf{x} \oplus \mathbf{M}$ on it, where "$\oplus$" is the bitwise exclusive-or (XOR) operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$). The resulting individual is then evaluated to obtain a fitness value for the individual $\mathbf{x}$. Suppose that the change happens at generation t, then we have $f(\mathbf{x}, t+1) = f(\mathbf{x} \oplus \mathbf{M})$.

In this way, we can revolve the fitness landscape but still keep certain properties of the original fitness landscape, e.g., the total number of optima and fitness values of optima though their locations are shifted. For example, if we apply a template $\mathbf{M} = 1111$ to Whitley's 4-bit deceptive function (to be described in Sect. 4.1), the original optimal point $\mathbf{x}^* = 1111$ becomes sub-optimal while the original deceptive solution $\mathbf{x} = 0000$ becomes the new optimal point in the changed landscape, but the optimal fitness value (i.e., 30) and the uniqueness of optimum keep invariant.

With the new dynamic problem generator, the dynamics of environmental change can be characterized by two parameters: the speed of change and the magnitude or degree of change in the sense of Hamming distance. As for other generators, the first parameter can be measured in EA generations. In this paper it will be referred to as the environmental change period, denoted by τ, and is defined as the number of EA generations between two changes. With respect to the degree of change, it can be measured by the ratio of ones in the mask $\mathbf{M}$, denoted by ρ. The more ones in the mask, the severer the change and the bigger the challenge to EAs. When $\rho = 0.0$, the problem stays stationary. When $\rho = 1.0$, it brings in the extreme or heaviest fitness landscape change in the sense of Hamming distance, analogous to natural environmental change between sunny daytime and dark night.

Putting things together, we can generate dynamic problems from a stationary problem as follows. Suppose that the environment is periodically changed every τ generations the dynamics can be formulated as follows:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(k)) \qquad (2)$$

where $k = \lceil t/\tau \rceil$ is the period index, t is the generation counter, and $\mathbf{M}(k)$ is the XORing mask for period k. And given a value for parameter ρ, $\mathbf{M}(k)$ can be incrementally generated as follows:

$$\mathbf{M}(k) = \mathbf{M}(k-1) \oplus \mathbf{T}(k) \qquad (3)$$

where $\mathbf{T}(k)$ is an intermediate binary template randomly created for period k containing $\rho \times l$ ones. For the first period $k = 1$, $\mathbf{M}(1)$ is initialized to be a zero vector.

Comparing with other generators, the new dynamic problem generator has the following properties.

– It is genotype-based. That is, it operates on the problem genotype instead of phenotype. Hence, we can carry out theoretical analysis more thoroughly in the genotype space.
– It is easy to realize required dynamics. We can not only test the speed of environmental change by tuning the parameter τ, but also test the degree of environmental change by tuning the parameter ρ easily.
– With this generator we can study the performance of algorithms on the dynamic version of many well studied benchmark problems in EA's community. For example, the royal road [21] and deceptive [29] functions are selected as test problems in this paper.
– It can be easily combined with other dynamic problem generators to generate required dynamic environments.

3 Description of Algorithms Investigated

3.1 Population-Based Incremental Learning

The Population-Based Incremental Learning (PBIL) algorithm, first proposed by Baluja [3], is a combination of evolutionary optimization and competitive learning. PBIL has proved to be very successful on numerous benchmark and real-world problems [4], [19]. Theoretical work on PBILs has also been carried out [14], [17].

The aim of PBIL is to generate a real valued probability vector which, when sampled, creates high quality solutions with high probability. PBIL starts from an initial probability vector with values of each entry set to 0.5[1]. This means when sampling by this initial probability vector random solutions are created because the probability of generating a 1 or 0 on each locus is equal. However, as the search progresses, the values in the probability vector are gradually learnt towards values representing high evaluation solutions. The evolution process is described as follows.

At each iteration, a set of samples (solutions) are created according to the current probability vector[2]. The set of samples are evaluated according to the problem-specific fitness function. Then the probability vector is learnt (pushed) towards the solution(s) with the highest fitness. The distance the probability vector is pushed depends on the learning rate parameter. After the probability vector is updated, a new set of solutions is generated by sampling from the new probability vector and this cycle is repeated. As the search progresses, the entries in the probability vector move away from their initial settings of 0.5 towards either 0.0 or 1.0. 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.

The pseudocode for the PBIL investigated in this paper is shown in Fig. 1. Within this PBIL, at iteration t a set S^t of $n = 120$ solutions are sampled from the probability vector P^t and only the best solution B^t from the set S^t is used to learn the probability vector P^t. The learning rate α is set to a commonly used value 0.05.

3.2 Parallel PBIL

Using multi-population instead of one population has proved to be a good approach for improving the performance of EAs for dynamic optimization problems. Similarly, multi-population can be introduced into PBIL by using multiple probability vectors [4], [27]. Each probability vector is sampled to generate solutions independently, and is learnt according to the best solution(s) generated by itself. For the sake of simplicity, in this paper we investigate a PBIL with two parallel probability

[1] For the convenience of description in this paper we will call the probability vector that has 0.5 for all of its entries *central probability vector* or just *central vector* because it represents the central point in the genotype space.

[2] For each bit position of a solution, assuming binary encoded, if a random created real number in the range of [0.0, 1.0] is less than the probability value of corresponding element in the probability vector, the bit is set to 1 (or 0), otherwise it is set to 0 (or 1 respectively).

```
begin
  $t := 0$;
  // initialize probability vector
  for $i := 1$ to $l$ do $P^0[i] := 0.5$; endfor;
  repeat
    $S^t :=$ generateSamplesFromProbVector($P^t, n$);
    evaluateSamples($S^t$);
    $B^t :=$ selectBestSolution($S^t$);
    // update probability vector toward best solution
    for $i := 1$ to $l$ do
      $P^t[i] := (1-\alpha) * P^t[i] + \alpha * B^t[i]$;
    endfor;
    $t := t + 1$;
  until $terminated =$ true;        // e.g., $t > t_{max}$
end;

PBIL's parameter settings:
  $l$: chromosome length (problem specific).
  $\alpha$: the learning rate (0.05).
  $n$: sample size generated by the prob. vector (120).
```

Fig. 1 Pseudocode for the PBIL with one probability vector.

vectors, called Parallel PBIL (PPBIL2). The pseudocode for PPBIL2 is shown in Fig. 2.

Within PPBIL2, one of the two probability vectors P_1 is initialized to the central probability vector (for the sake of performance comparison with the PBIL) and the other P_2 is randomly initialized. P_1 and P_2 are sampled and updated independently. Initially P_1 and P_2 have an equal sample size. However, in order to give the probability vector that performs better more chance to generate samples, the sample sizes of the probability vectors are slightly adapted within the range of $[n_{min}, n_{max}] = [0.2 * n, 0.8 * n] = [24, 96]$ according to their relative performance. If one probability vector outperforms the other, its sample size is increased by a constant value $\Delta = 0.05 * n = 6$ while the other's sample size is decreased by Δ; otherwise, if the two probability vectors tie, there is no change to the sample sizes. The learning rate for both P_1 and P_2 is the same as that for the PBIL.

3.3 Dual PBIL

Dualism and complementarity are quite common in nature. For example, in biology the DNA molecule consists of two complementary strands that are twisted together into a duplex chain. Inspired by the complementarity mechanism in nature, a primal-dual genetic algorithm has been proposed and applied for dynamic optimization problems [30]. In this paper we investigate the application of dualism into PBIL and propose a Dual PBIL, denoted DPBIL2. For the convenience of description, we first introduce the definition of dual probability vector here. Given a probability vector $P = (P[1], \cdots, P[l]) \in$

```
begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ to $l$ do
    $P_1^0[i] := 0.5$; $P_2^0[i] := rand[0.0, 1.0]$;
  endfor;
  // initialize sample sizes for probability vectors
  $n_1^0 := n_2^0 := n/2$;
  repeat
    $S_1^t :=$ generateSamplesFromProbVector($P_1^t, n_1^t$);
    $S_2^t :=$ generateSamplesFromProbVector($P_2^t, n_2^t$);
    evaluateSamples($S_1^t$, $S_2^t$);
    $B_1^t :=$ selectBestSolution($S_1^t$);
    $B_2^t :=$ selectBestSolution($S_2^t$);
    // update probability vectors toward best solutions
    for $i := 1$ to $l$ do
      $P_1^t[i] := (1-\alpha) * P_1^t[i] + \alpha * B_1^t[i]$;
      $P_2^t[i] := (1-\alpha) * P_2^t[i] + \alpha * B_2^t[i]$;
    endfor;
    // update sample sizes for probability vectors
    if $f(B_1^t) > f(B_2^t)$ then $n_1^t := min\{n_1^t + \Delta, n_{max}\}$;
    if $f(B_1^t) < f(B_2^t)$ then $n_1^t := max\{n_1^t - \Delta, n_{min}\}$;
    $n_2^t := n - n_1^t$;
    $t := t + 1$;
  until $terminated =$ true;        // e.g., $t > t_{max}$
end;

PPBIL2's parameter settings:
  $l$: chromosome length (problem specific).
  $\alpha$: the learning rate (0.05).
  $n$: total sample size by two prob. vectors (120).
  $n_1^t$, $n_2^t$: sample size by prob. vector 1 and 2 at time $t$.
  $\Delta$: constant step size of adjusting $n_1$ and $n_2$ (6).
  $n_{min}$: min sample size by each prob. vector (24).
  $n_{max}$: max sample size by each prob. vector (96).
```

Fig. 2 Pseudocode for the Parallel PBIL (PPBIL2).

$I = [0.0, 1.0]^l$ of fixed length l, its dual probability vector is defined as $P' = dual(P) = (P'[1], \cdots, P'[l]) \in I$ where $P'[i] = 1.0 - P[i]$ $(i = 1, \cdots, l)$. That is, a probability vector's dual probability vector is the one that is symmetric to it with respect to the central probability vector. With this definition, DPBIL2 consists of a pair of probability vectors that are dual to each other. The pseudocode of DPBIL2 is given in Fig. 3.

From Fig. 2 and Fig. 3 it can be seen that DPBIL2 differs from PPBIL2 only in the definition of the probability vector P_2 and the learning mechanism. The other aspects of DPBIL2, such as the sampling mechanism, the sample size updating mechanism, and relevant parameters, are the same as those of PPBIL2. Within DPBIL2 P_2 is now defined to be the dual probability vector of P_1. As the search progresses only P_1 is learnt from the best generated solution since P_2 changes with P_1 automatically. If the best overall solution is sampled by P_1^t (i.e.,

```
begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ to $l$ do
    $P_1^0[i] := 0.5$; $P_2^0[i] := 1.0 - P_1^0[i]$;
  endfor;
  // initialize sample sizes for probability vectors
  $n_1^0 := n_2^0 := n/2$;
  repeat
    $S_1^t :=$ generateSamplesFromProbVector($P_1^t, n_1^t$);
    $S_2^t :=$ generateSamplesFromProbVector($P_2^t, n_2^t$);
    evaluateSamples($S_1^t$, $S_2^t$);
    $B_1^t :=$ selectBestSolution($S_1^t$);
    $B_2^t :=$ selectBestSolution($S_2^t$);
    // update probability vectors
    for $i := 1$ to $l$ do
      if $f(B_1^t) \geq f(B_2^t)$ then    // learn $P_1^t$ toward $B_1^t$
        $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * B_1^t[i]$;
      else                              // learn $P_1^t$ away from $B_2^t$
        $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * (1.0 - B_2^t[i])$;
      $P_2^t[i] := 1.0 - P_1^t[i]$;
    endfor;
    // update sample sizes for probability vectors
    if $f(B_1^t) > f(B_2^t)$ then $n_1^t := min\{n_1^t + \Delta, n_{max}\}$;
    if $f(B_1^t) < f(B_2^t)$ then $n_1^t := max\{n_1^t - \Delta, n_{min}\}$;
    $n_2^t := n - n_1^t$;
    $t := t + 1$;
  until $terminated =$ true;      // e.g., $t > t_{max}$
end;

DPBIL2's parameter settings are the same as PPBIL2's.
```

Fig. 3 Pseudocode for the Dual PBIL (DPBIL2).

$f(B_1^t) \geq f(B_2^t)$) then P_1^t is updated towards B_1^t; otherwise, P_1^t is updated away from B_2^t, the best solution created by P_2^t. The reason to P_1^t learning away from B_2^t lies in that it is equivalent to P_2^t learning towards B_2^t.

The motivation of introducing a dual probability vector into PBIL lies in two aspects: increasing diversity of generated samples and fighting significant environmental changes. On the first aspect, usually with the progress of parallel PBILs the probability vectors will converge towards each other and the diversity of generated samples is reduced. This situation doesn't occur with dual probability vectors. On the second aspect, when the environment is subject to significant changes the dual probability vector is expected to generate high evaluation solutions and hence improve PBIL's adaptability.

3.4 Standard Genetic Algorithm

Genetic algorithms (GAs) are one kind of well studied evolutionary algorithms. The standard genetic algorithm maintains a population of individuals, usually encoded as fixed length binary strings. The initial population is randomly created. New populations are created through a process of selection, recombination (crossover) and mutation. At each generation, the fitness of each individual in the population is calculated according to the problem-specific evaluation function. Then the individuals are probabilistically selected from the current population based on their fitness to generate a mating pool, which is called *selection for reproduction*. Afterwards, the recombination and mutation operators are applied to some or all individuals in the mating pool. The recombination operator randomly combines parts of two "parents" that are randomly selected from the mating pool to produce two "offsprings". And the mutation operator randomly flips each bit of a string with a small probability p_m to create a new string. This process continues until some termination condition is satisfied, e.g., the maximum allowable number of generations t_{max} is reached [18]. Usually with the iteration of the GA, the average fitness of the population will progressively improve due to the selective pressure applied through the process. The best individual in the final population should be a highly evolved solution to the given problem.

```
begin
  $t := 0$;
  initializePopulation($P(0), n$);
  evaluatePopulation($P(0)$);
  repeat
    $P'(t) :=$ selectForReproduction($P(t)$);
    crossover($P'(t), p_c$);
    mutate($P'(t), p_m$);
    evaluatePopulation($P'(t)$);
    $t := t + 1$;
  until $terminated =$ true;      // e.g., $t > t_{max}$
end;

SGA's parameter settings:
  $n$: population size (120).
  $p_c$: uniform crossover probability (0.6).
  $p_m$: bit mutation probability (0.01).
```

Fig. 4 Pseudocode for the Standard GA (SGA).

GAs are closely related to PBILs. In fact PBIL is an abstraction of the GA that explicitly maintains the statistics contained in GA's population [4]. In this study, one variant of the standard GA (SGA), as shown in Fig. 4, is taken as a peer EA to compare the performance of PBILs for dynamic optimization problems. The peer SGA has the following typical genetic operator and parameter settings: generational, uniform crossover with a crossover probability $p_c = 0.6$, traditional bit mutation with a mutation probability $p_m = 0.01$, and fitness proportionate selection with the Stochastic Universal Sampling (SUS) [2] scheme. There is no elitist scheme used in the SGA and the population size n is set to 120.

4 Algorithm Test Environments

In order to compare different PBILs and SGA, a set of well studied stationary problems, including one GA-easy royal road function, one GA-hard deceptive function, and one real world knapsack problem, is selected as the test suite. A series of dynamic optimization problems are constructed from these stationary problems using the dynamic problem generator described in Sect. 2.2.

4.1 Stationary Test Problems

1. **Knapsack Problem**:

The knapsack problem is a well known NP-complete combinatorial optimization problem. The problem is to select from a set of items with varying weights and profits those items that will yield the maximal summed profit to fill in the knapsack without exceeding its limited weight capacity. Given a set of m items and a knapsack, the 0-1 knapsack problem can be described as follows:

$$max \;\; p(\mathbf{x}) = \sum_{i=1}^{i=m} p_i x_i \tag{4}$$

subject to the weight constraint

$$\sum_{i=1}^{i=m} w_i x_i \leq C \tag{5}$$

where $\mathbf{x} = (x_1 \cdots x_m)$, x_i is 0 or 1, w_i and p_i are the weight and profit of item i respectively, and C is the capacity of the knapsack. If $x_i = 1$, the ith item is selected.

In this paper, a knapsack problem with 100 items using strongly correlated sets of randomly generated data is constructed as follows:

$$w_i = \text{uniformly random integer}[1, 50] \tag{6}$$

$$p_i = w_i + \text{uniformly random integer}[1, 5] \tag{7}$$

$$C = 0.6 \times \sum_{i=1}^{i=100} w_i \tag{8}$$

And given a solution $\mathbf{x}$, its fitness $f(\mathbf{x})$ is evaluated as follows. If the sum of the item weights is within the capacity of the knapsack, the sum of the profits of the selected items is used as the fitness. If the solution selects too many items such that the summed weight exceeds the capacity of the knapsack, the solution is judged by how much it exceeds the knapsack capacity (the less, the better) and its fitness is evaluated to be the difference between the total weight of all items and the weight of selected items, multiplied by a small constant 10^{-10} to ensure that the solutions that overfill the knapsack are not competitive with those which do not. Together, the fitness of a solution $\mathbf{x}$ is evaluated as follows:

$$f(\mathbf{x}) = \begin{cases} \sum_{i=1}^{i=100} p_i x_i, & \text{if } \sum_{i=1}^{i=100} w_i x_i \leq C \\ 10^{-10} \times (\sum_{i=1}^{i=100} w_i - \sum_{i=1}^{i=100} w_i x_i), & \text{else} \end{cases} \tag{9}$$

2. **Royal Road Function**:

This function is the same as Mitchell, Forrest and Holland's royal road function $R1$ [21]. It is defined on a sixty-four bit string consisting of eight contiguous building blocks of eight bits, each of which contributes $c_i = 8$ $(i = 1, ..., 8)$ to the total fitness if all of the eight bits are set to one. The fitness of a bit string $\mathbf{x}$ is computed by summing the coefficients c_i corresponding to each of the given building blocks s_i of which $\mathbf{x}$ is an instance (denoted by $x \in s_i$). That is, the royal road function is defined as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{i=8} c_i \delta_i(x) \tag{10}$$

where $\delta_i(x) = \{1, \text{if } x \in s_i; 0, \text{otherwise}\}$. This function has an optimum fitness of 64.

3. **Deceptive Function**:

Deceptive functions are devised as difficult test functions for GAs. They are a family of functions where there exist low-order building blocks that do not combine to form higher-order building blocks: instead they form building blocks resulting in a solution, called *deceptive attractor* [29], which is sub-optimal itself or near a sub-optimal solution. It is even claimed that the only challenging problems for GAs are problems that involve some degree of deception. Based on an algorithm of constructing fully deceptive functions, Whitley [29] developed a 4-bit fully deceptive problem as follows:

```
f(0000)=28 f(0001)=26 f(0010)=24 f(0011)=18
f(0100)=22 f(0101)=6  f(0110)=14 f(0111)=0
f(1000)=20 f(1001)=12 f(1010)=10 f(1011)=2
f(1100)=8  f(1101)=4  f(1110)=6  f(1111)=30
```

In this study, we construct a deceptive function consisting of 30 copies of Whitley's 4-bit fully deceptive function (order-4 subproblem). This function has an optimum fitness of 900 and a representation of 120 bits.

4.2 Constructing Dynamic Test Environments

In this paper, we construct dynamic test environments from above stationary problems in the following way. The fitness landscape of each stationary problem is periodically changed every τ generations during the run of algorithms. Based on our preliminary experiments on the stationary problems (see Sect. 5.2), τ is set to 10, 100 and 200 respectively to create 3 dynamic problems with respect to this parameter only. The environmental change speed parameter τ is set to these values because on the stationary problems all algorithms are sort of consistently on different search stages at generations of these values. For example, on the stationary problems almost all algorithms are at quite early searching stage at generation 10, at medium searching stage at generation 100, and at late stage or converged at generation 200. By

Table 1 The index table for environmental dynamics parameter setting.

τ	Environmental Dynamics Index						
10	1	2	3	4	5	6	7
100	8	9	10	11	12	13	14
200	15	16	17	18	19	20	21
$\rho \Rightarrow$	0.05	0.2	0.4	0.6	0.8	0.95	$rand(0.01, 0.99)$

setting τ to these values we can test each algorithm's capability of adapting to dynamic environment under different degree of convergence (or searching stage).

In order to test the effect of another dynamics parameter, the degree of environmental change, on the performance of algorithms, the value of ρ is set to 0.05, 0.2, 0.4, 0.6, 0.8, and 0.95 respectively for each run of an algorithm on a problem. These values represent different environmental change levels, from very light shifting ($\rho = 0.05$) to medium variation ($\rho = 0.2, 0.4, 0.6, 0.8$) to significant change ($\rho = 0.95$). In order to study the behavior of algorithms in randomly changing environment we also set ρ to be a random number uniformly distributed in $[0.01, 0.99]$, i.e., $\rho = rand(0.01, 0.99)$.

Totally, we systematically construct a series of 21 dynamic problems, 3 values of τ combined with 7 values of ρ, from each stationary test problem. The environmental dynamics parameter settings are summarized in Table 1.

5 Experimental Study

5.1 Experimental Design

Experiments were carried out to compare the performance of PBILs as well as the SGA on the test environments constructed above. In addition to the above described PBILs, we also test the effect of the re-start scheme on the performance of PBIL in dynamic environments. A complete re-start of EAs after a change in the environment has occurred is the simplest option to maintain diversity in the population and react to changes in the environment. However, it is not always possible to detect a change and do a re-start deliberately. In this study, for the sake of algorithm performance comparison we also investigate the PBIL with an ideal re-start scheme, called PBILr, where whenever the environment changes the PBIL is re-started from scratch. That is, with PBILc all elements in the probability vector is reset to 0.5 whenever the environment changes.

For each experiment of combining different algorithm and test problem (no matter stationary or dynamic), 50 independent runs were executed with the same set of 50 random seeds. For each run of different algorithm on each problem, the best-of-generation fitness was recorded every generation. And for each run of an algorithm on a dynamic problem, 10 periods of environmental changes are allowed[3].

The overall performance of an algorithm on a problem is measured by the mean best-of-generation fitness. It is defined as the best-of-generation fitness averaged across the number of total runs and then averaged over the data gathering period. More formally this is:

$$\overline{F}_{BG} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N} F_{BG_{ij}}) \tag{11}$$

where $\overline{F}_{BG}$ is the mean best-of-generation fitness, G is the number of generations which is equivalent to 10 periods of environmental changes (i.e., $G = 10 * \tau$), $N = 50$ is the total number of runs, and $F_{BG_{ij}}$ is the best-of-generation fitness of generation i of run j.

5.2 Experimental Results on Stationary Problems

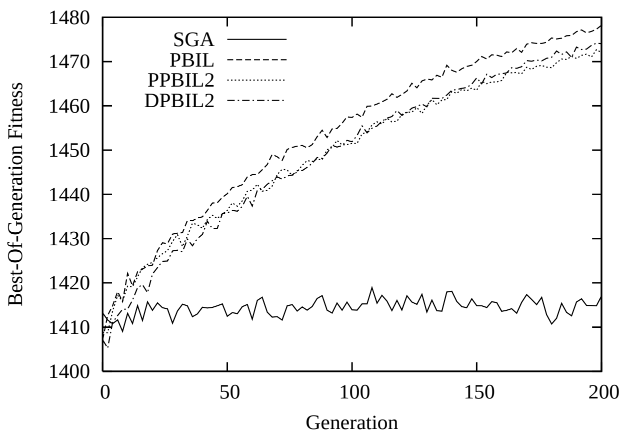
In order to help analyze the experimental results on dynamic problems later on in this paper, preliminary experiments were carried out on the stationary test problems. For each run of different algorithm on each problem the maximum allowable number of generations was set to 200. The preliminary experimental results are shown in Fig. 5 where the data were averaged over 50 runs.

From Fig. 5, it can be seen that in general all PBILs outperform SGA. This result is consistent with other researchers' study [4]. On the knapsack and royal road problems PBIL outperforms PPBIL2 and DPBIL2 while PPBIL2 performs as well as DPBIL2. This result shows that on stationary problems introducing extra probability vector may not be beneficial because the existence of an extra probability vector that performs worse may slow down the learning speed of the other probability vector that performs better. However, on the deceptive function the situation seems quite different. PBIL and PPBIL2 performs equally well while both are beaten by DPBIL2 during late searching stage. This happens because the deceptive attractor $\mathbf{x} = 00...0$ in this function strongly draws the probability vectors of PBIL and PPBIL2 towards its trap. The existence of the dual probability vector in DPBIL2 slows down the process of trapping, and after about 140 generations when the fitness level 840 of the deceptive attractor is reached, the dual probability vector helps escaping the local optimum and pushes the searching towards the global optimum.

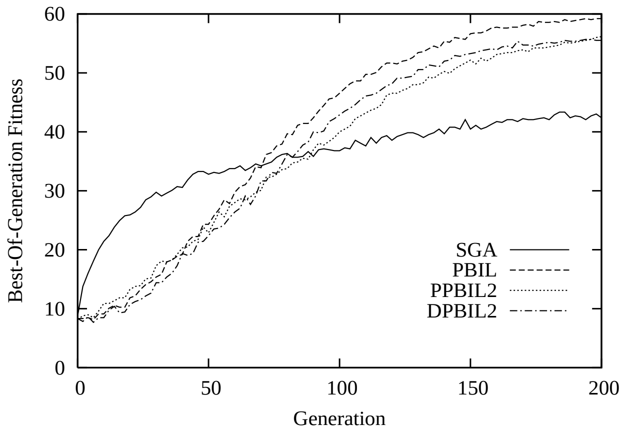
5.3 Experimental Results on Dynamic Problems

The experimental results on dynamic problems and some key statistical test results are summarized in Table 2

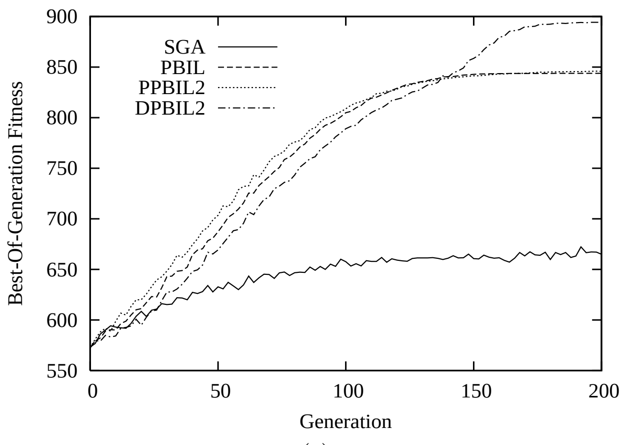
[3] For the convenience of analyzing experimental results on dynamic problems, we herein call the first period *stationary* since the behavior of an algorithm on a dynamic problem during this period is the same as that on the relevant stationary problem. And the other 9 periods are called *dynamic*.



(a)

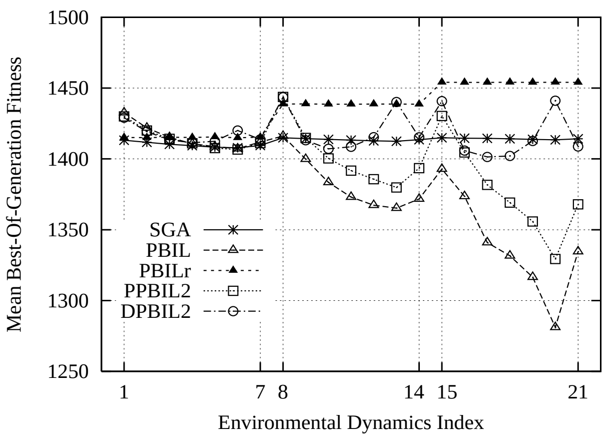


(b)

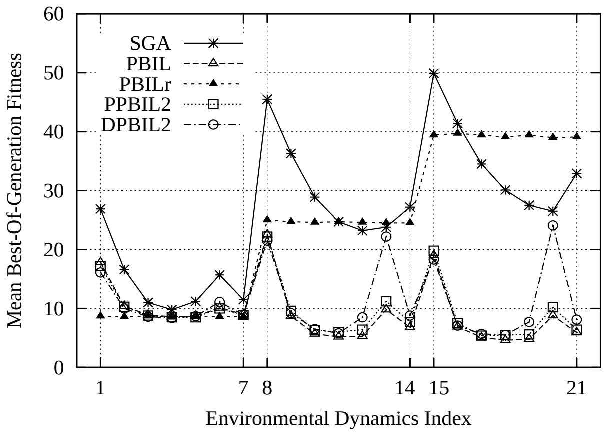


(c)

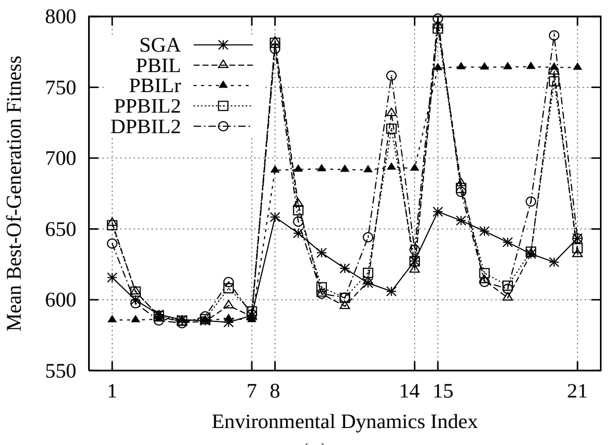
Fig. 5 Experimental results with respect to best-of-generation fitness against generations of algorithms on stationary problems: (a) Knapsack, (b) Royal Road, and (c) Deceptive. The data were averaged over 50 runs.



(a)



(b)



(c)

Fig. 6 Experimental results of SGA, PBIL, PBILr, PPBIL2, and DPBIL2 with respect to mean best-of-generation fitness against different environmental dynamics parameter settings on dynamic problems: (a) Knapsack, (b) Royal Road, and (c) Deceptive.

and Table 3 respectively. The experimental results are also plotted in Fig. 6, where the environmental dynamics setting can be indexed according to Table 1. From Table 2, Table 3 and Fig. 6 several results can be observed.

First, the performance of PBILr increases with the value of τ but doesnot change much with the value of ρ. This is easy to understand. Given the perfect re-start scheme, each time the environment changes PBILr is in fact starting from the same initial state to search the equivalent problem regardless of the changing degree, i.e., the value of ρ. And with the increasing of τ PBILr

Table 2 Experimental results of SGA, PBIL, PBILr, PPBIL2, and DPBIL2 on dynamic problems with respect to overall mean best-of-generation fitness.

Dynamics		Knapsack Problem					Royal Road Function					Deceptive Function				
τ	ρ	*SGA*	*PBIL*	*PBILr*	*PPBIL2*	*DPBIL2*	*SGA*	*PBIL*	*PBILr*	*PPBIL2*	*DPBIL2*	*SGA*	*PBIL*	*PBILr*	*PPBIL2*	*DPBIL2*
10	0.05	1413.2	1432.7	1415.1	1430.0	1429.1	26.9	17.8	8.7	17.2	16.1	615.7	654.2	585.7	652.6	639.7
10	0.2	1411.7	1421.9	1415.0	1420.1	1419.2	16.6	10.4	8.6	10.3	10.0	599.8	605.7	585.7	605.7	597.5
10	0.4	1410.3	1415.2	1415.3	1414.2	1413.4	11.0	8.9	8.7	8.8	8.6	589.6	588.6	586.2	588.6	585.4
10	0.6	1409.4	1410.7	1415.1	1411.0	1411.4	9.8	8.6	8.6	8.5	8.4	585.8	583.9	585.7	585.2	583.5
10	0.8	1408.2	1408.7	1415.5	1407.6	1412.5	11.2	8.7	8.6	8.5	8.7	585.2	584.7	585.5	586.3	588.3
10	0.95	1407.8	1407.2	1414.9	1406.6	1420.1	15.7	10.1	8.6	10.0	11.1	584.1	595.9	586.5	608.1	612.5
10	*rand*	1409.5	1411.9	1415.3	1411.1	1413.8	11.5	9.0	8.6	8.9	8.9	588.9	588.2	585.8	591.8	589.6
100	0.05	1414.9	1416.2	1438.7	1443.8	1443.6	45.5	22.5	25.0	22.2	21.5	658.4	782.1	691.4	781.4	777.1
100	0.2	1414.3	1399.8	1438.8	1414.8	1413.2	36.3	8.7	24.7	9.6	9.2	647.0	667.6	692.0	663.2	655.2
100	0.4	1413.8	1383.4	1438.6	1400.3	1407.1	28.9	5.7	24.6	6.3	6.5	633.1	604.2	692.3	608.8	604.3
100	0.6	1413.3	1373.1	1438.6	1391.7	1408.6	24.7	5.2	24.7	6.0	5.8	622.1	595.5	691.9	601.3	601.5
100	0.8	1412.8	1367.3	1438.7	1385.7	1415.3	23.2	5.3	24.6	6.4	8.5	611.8	612.8	691.6	619.0	644.1
100	0.95	1412.5	1365.2	1438.7	1379.8	1440.2	23.8	9.8	24.5	11.2	22.2	605.9	731.6	693.5	720.8	758.2
100	*rand*	1413.6	1371.7	1438.6	1393.5	1415.2	27.2	6.8	24.5	7.7	8.8	627.4	621.1	692.8	627.2	635.4
200	0.05	1414.9	1392.9	1454.1	1430.3	1440.8	49.9	19.0	39.4	19.8	18.3	662.2	793.5	763.7	791.5	798.6
200	0.2	1414.6	1373.6	1454.0	1404.6	1405.9	41.4	6.9	39.7	7.5	7.1	655.9	682.1	764.4	678.9	676.1
200	0.4	1414.5	1340.9	1454.0	1381.7	1401.4	34.5	5.1	39.4	5.4	5.7	648.4	613.6	764.2	618.8	612.6
200	0.6	1414.2	1331.6	1454.1	1369.1	1402.1	30.1	4.6	39.1	5.5	5.5	640.6	601.4	764.4	609.9	607.7
200	0.8	1413.8	1316.5	1454.0	1355.8	1412.7	27.5	4.8	39.4	5.6	7.7	632.4	632.3	764.6	634.0	669.4
200	0.95	1413.5	1281.1	1454.1	1329.3	1441.1	26.5	8.8	39.0	10.2	24.1	626.6	761.0	764.1	754.2	786.7
200	*rand*	1414.2	1334.6	1454.0	1367.9	1408.8	32.9	5.9	39.1	6.4	8.1	643.4	632.1	764.0	636.4	643.3

Table 3 Statistical comparison of algorithms on dynamic problems by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance. The t-test result regarding Alg. 1 − Alg. 2 is shown as "+", "−", or "∼" when Alg. 1 is significantly better than, significantly worse than, or statistically equivalent to Alg. 2 respectively.

t-test Result	Knapsack Problem							Royal Road Function							Deceptive Function						
$\tau = 10,\ \rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$DPBIL2 - SGA$	+	+	+	+	+	+	+	−	−	−	−	−	−	−	+	−	−	−	+	+	∼
$DPBIL2 - PBIL$	−	−	−	+	+	+	+	−	−	−	−	∼	+	∼	−	−	−	∼	+	+	∼
$DPBIL2 - PPBIL2$	∼	−	−	∼	+	+	+	−	−	∼	∼	+	+	∼	−	−	−	−	+	+	−
$PPBIL2 - PBIL$	−	−	−	∼	−	∼	∼	∼	∼	∼	∼	−	∼	∼	∼	∼	∼	+	+	+	+
$PBILr - PBIL$	−	−	∼	+	+	+	+	−	−	−	∼	∼	−	−	−	−	−	+	∼	−	−
$PBILr - PPBIL2$	−	−	+	+	+	+	+	−	−	∼	∼	∼	−	−	−	−	−	∼	∼	−	−
$PBILr - DPPBIL2$	−	−	+	+	+	−	+	−	−	∼	+	∼	−	−	−	−	∼	+	−	−	−
$\tau = 100,\ \rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$DPBIL2 - SGA$	+	∼	−	−	+	+	∼	−	−	−	−	−	−	−	+	+	−	−	+	+	+
$DPBIL2 - PBIL$	+	+	+	+	+	+	+	∼	+	+	+	+	+	+	−	−	∼	+	+	+	+
$DPBIL2 - PPBIL2$	∼	∼	+	+	+	+	+	∼	∼	∼	∼	+	+	+	−	−	−	∼	+	+	+
$PPBIL2 - PBIL$	+	+	+	+	+	+	+	∼	+	+	+	+	+	+	∼	−	+	+	+	−	∼
$PBILr - PBIL$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+
$PBILr - PPBIL2$	−	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+
$PBILr - DPPBIL2$	−	+	+	+	+	∼	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+
$\tau = 200,\ \rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$DPBIL2 - SGA$	+	−	−	−	∼	+	−	−	−	−	−	−	−	−	+	+	−	−	+	+	∼
$DPBIL2 - PBIL$	+	+	+	+	+	+	+	∼	∼	+	+	+	+	+	+	−	∼	+	+	+	+
$DPBIL2 - PPBIL2$	+	∼	+	+	+	+	+	∼	∼	∼	∼	+	+	+	+	∼	−	∼	+	+	∼
$PPBIL2 - PBIL$	+	+	+	+	+	+	+	∼	+	+	+	+	+	∼	∼	∼	+	+	∼	−	∼
$PBILr - PBIL$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	∼	+
$PBILr - PPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	+	+
$PBILr - DPPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+

has more time to search solutions with higher fitness before the next change.

PBILr outperforms other algorithms in many dynamic problems, especially when the environment changes slowly (and hence convergence becomes a problem). This is due to the maximum diversity the re-start scheme introduces into the population. However, in slightly changing environments ($\rho = 0.05$) PBILr is beaten by other PBILs in many cases due to the lack of information transfer from the last generation of the last dynamic period. Since it is usually not possible to detect environmental change timely and perform the re-start scheme immediately when the environment changes, we will exclude PBILr in following algorithm performance comparison and analysis.

Second, from Fig. 6 it is easy to see that for each fixed τ DPBIL2 outperforms other algorithms (even including PBILr) on most of the dynamic problems when the environment is subject to significant changes, e.g., when ρ is set to 0.95. In fact, from Table 3 it can be seen that when $\rho = 0.95$, DPBIL2 statistically significantly outperforms PBIL and PPBIL2 on all dynamic problems and SGA on all dynamic knapsack and deceptive functions. This result confirms our expectation of introducing the dual probability vector into DPBIL2. When the environment suffers significant changes, the dual probability vector takes effect quickly to adapt DPBIL2 to the changed environment. This effect on DPBIL2 also takes place when ρ is set to 0.6 and 0.8. DPBIL2 still statistically significantly outperforms PBIL and PPBIL2 on most dynamic problems when ρ equals 0.6 and 0.8 and SGA on most dynamic knapsack problems and deceptive functions when $\rho = 0.8$.

Third, PBIL is now beaten by both PPBIL2 and DPBIL2 on most dynamic problems except when the value of ρ is small. When ρ is small, the dynamic problems are close to their corresponding stationary problems. For stationary (and nearly stationary) problems introducing an extra probability vector may not be beneficial, which has been verified in our preliminary experiments in Sect. 5.2 (see Fig. 5). However, when the value of ρ increases the introduction of an extra probability vector helps improving PBIL's performance.

Fourth, as opposed to stationary problems, SGA now outperforms PBILs on many dynamic problems, especially when the value of τ is large. This happens because when τ is large the algorithms are given more time to search before the next environmental change and hence they are more likely to converge. Convergence deprives PBILs of the adaptability to changing environments. However, the mutation mechanism embedded in SGA gives it more diversity than PBILs and hence better adaptability to environmental changes. Hence, SGA outperforms PBILs in many dynamic problems.

It seems that SGA performs much better on dynamic royal road functions than on dynamic knapsack problems and dynamic deceptive functions when the value of τ is large. The reason lies in the intrinsic characteristics of the royal road function where there exists a big gap with respect to the fitness levels of its component building block. Only when all ones appear in a building block it will contribute 8 to the whole fitness, otherwise for all other cases it will contribute 0. This makes the effect of the mutation scheme in SGA more significant on dynamic royal road functions.

Fifth, given a value of τ when the environment changes randomly with respect to the changing severity, i.e., $\rho = rand(0.01, 0.99)$, the performance of algorithms is similar to the situation of setting ρ to medium values, e.g., 0.4 or 0.6. This fits well with the fact that the expected value of $rand(0.01, 0.99)$ is about 0.5. It is also notable that when $\rho = rand(0.01, 0.99)$, DPBIL2 still significantly outperforms PBIL and PPBIL2 on most dynamic problems and SGA on several cases. This happens because the dual probability vector inside DPBIL2 improves its adaptability if by chance the environment is subject to significant changes, i.e., ρ is randomly set to a big value.

Finally, from Fig. 6 an interesting result that can be seen is that for each fixed τ with the increasing of the value of ρ (excluding the random situation) DPBIL2 performs consistently across the three series of dynamic problems (knapsack, royal road and deceptive). When ρ increases from 0.05 to 0.2, 0.4, 0.6, 0.8 to 0.95 the performance curve of DPBIL2 looks like a big "U". PBIL and PPBIL2 have this performance curve on dynamic royal road and deceptive functions, while on the dynamic knapsack problems they have the performance curve of "falling stone". SGA has a "falling stone" performance curve on almost all dynamic problems. The reason to this observation lies in the intrinsic characteristics of the problems and will be further explained below.

In order to better understand the experimental results, we need to have a deeper look into the dynamic behavior of different algorithms. The dynamic behavior of different algorithms with respect to best-of-generation fitness against generations on the three series of dynamic problems is shown in Fig. 7 to Fig. 9 respectively, where the data were averaged over 50 runs. In these figures τ is set to 10 (left column) and 200 (right column) respectively, and ρ is set to 0.05, 0.4, and 0.95 from top row to bottom row respectively. From these figures it can be easily observed that for PBILr on all dynamic problems its dynamic behavior for each dynamic period is almost the same as that for the stationary period.

For DPBIL2 on all dynamic problems the dynamic performance drops heavier and heavier when the value of ρ increases from 0.05 to 0.4. However, when $\rho = 0.8$ (not shown in Fig. 7) and 0.95 the situation is different. Now, when $\tau = 10$ its performance rises instead of drops with the increment of dynamic periods due to less convergence and high adaptability brought in by the dual probability vector, while when $\tau = 200$ with the increment of dynamic periods DPBIL2's performance maintains almost the same on dynamic knapsack problems or drops much less severe on dynamic royal road and deceptive problems. For both values of τ whenever the environment changes the dual probability vector adapts

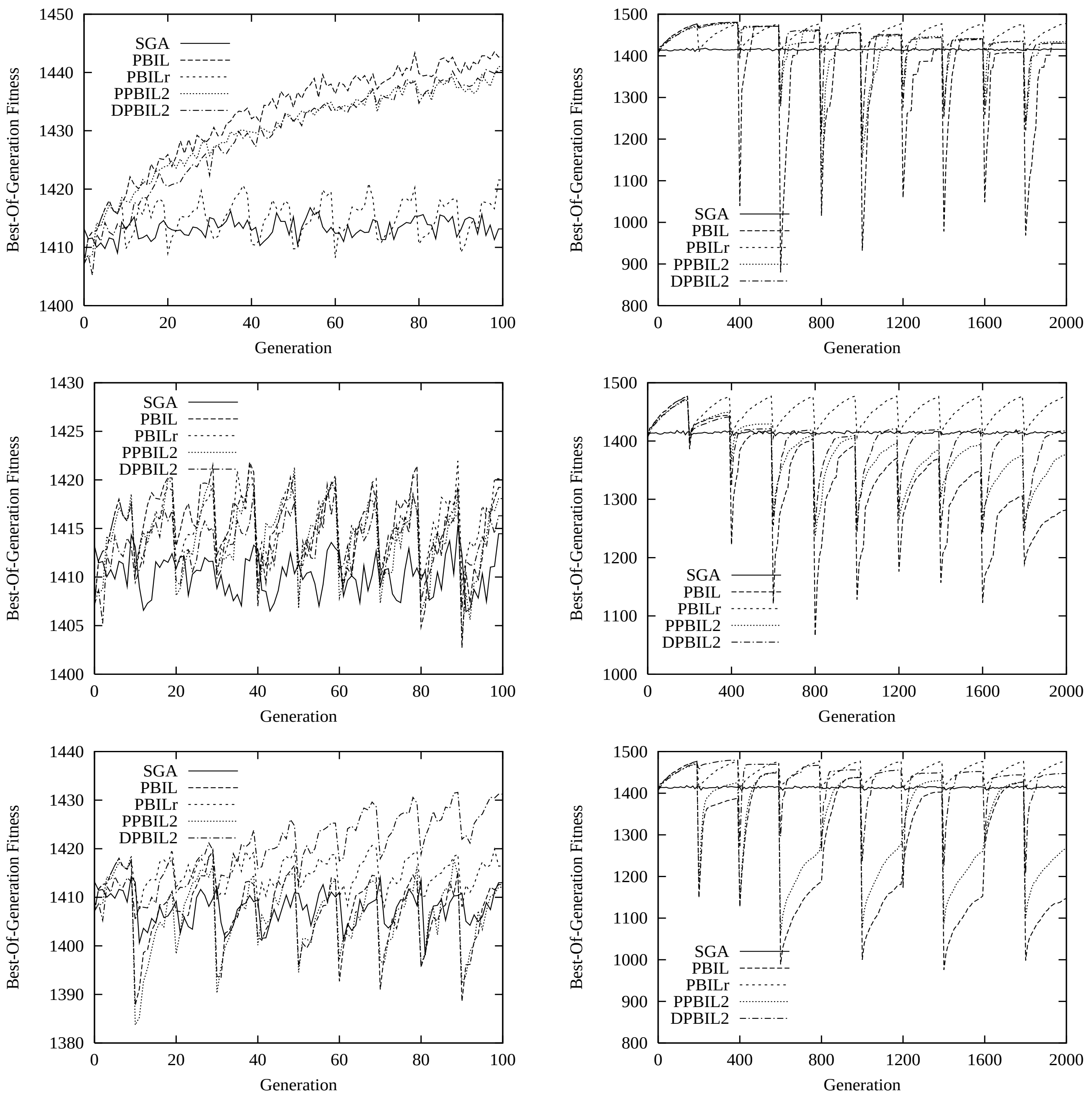


Fig. 7 Dynamic behavior of algorithms on dynamic knapsack problems. The environmental dynamics parameter τ is set to 10 (*Left Column*) and 200 (*Right Column*) respectively and ρ is set to 0.05, 0.4, and 0.95 from top to bottom row respectively.

DPBIL2 quickly to the new environment. This stops its performance from significant drop for dynamic periods. All in all, this results in DPBIL2's big "U" performance curve on all the dynamic problems.

For PBIL and PPBIL2, generally speaking, when the value of ρ increases from 0.05 to 0.4 their dynamic performance drops heavier and heavier on all dynamic problems, which is similar to DPBIL2's dynamic performance. However, when $\rho = 0.95$ their dynamic performance is different from DPBIL2's. When $\rho = 0.95$, the dynamic behavior of PBIL and PPBIL2 is sort of switching between odd and even environmental periods. They start from a harsher state for even environmental periods than for odd environmental periods. The reason to this lies in that after the stationary period for the following odd period the environment is in fact greatly returned or repeated from previous odd period given $\rho = 0.95$. Hence at the start of odd environmental periods the performance of PBIL and PPBIL2 doesnot drop as heavy as it does at the start of even periods. This benefits the whole performance of PBIL and PPBIL2 and also results in the big "U" overall performance curve for

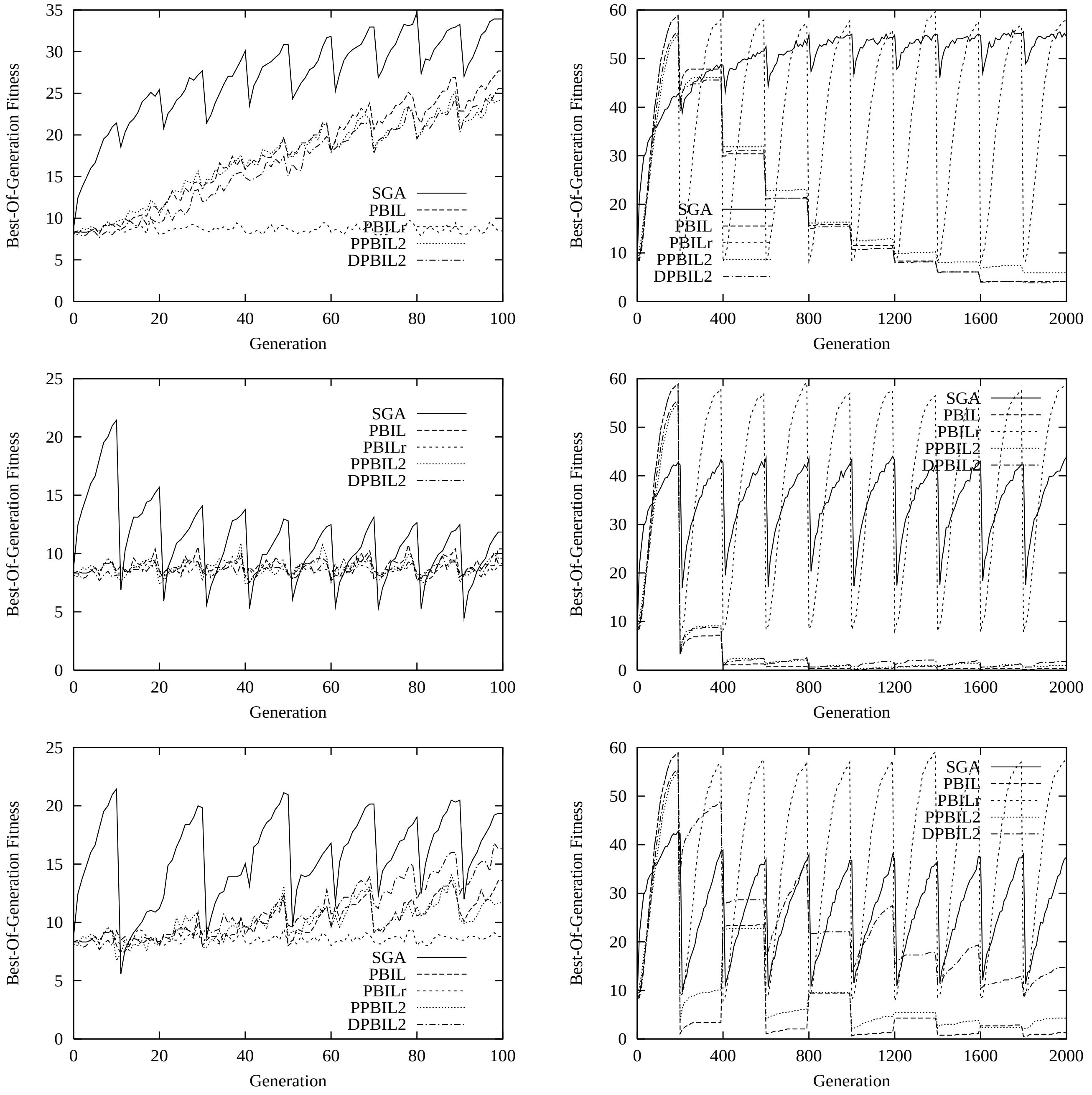


Fig. 8 Dynamic behavior of algorithms on dynamic royal road functions. The environmental dynamics parameter τ is set to 10 (*Left Column*) and 200 (*Right Column*) respectively and ρ is set to 0.05, 0.4, and 0.95 from top to bottom row respectively.

them on dynamic royal road and deceptive functions. However, the inside mechanism is that PBIL and PPBIL2 are sort of waiting for the return of previously well sought environment, which is totally different from DPBIL2 where the high performance is achieved by rapid adaptation to the newly changed environment. On dynamic knapsack problems, PBIL and PPBIL2 do not have the big "U" overall performance curve because their performance drops too much during the start of even environmental periods to be compensated by the benefit gained during odd periods.

In order to better understand the above discussion, in Fig. 10 we present extra experimental results with the extreme environmental dynamics of $\rho = 1.0$ and $\tau = 200$ on the dynamic problems. From Fig. 10 it can be clearly seen that when the environment changes DPBIL2 immediately adapts to the new environment while PBIL and PPBIL2 switch between two states: one low fitness state of even environmental periods where PBIL and PPBIL2 are poorly searching or in fact waiting for the return of their previously adapted environment, and the other high fitness state of odd periods.

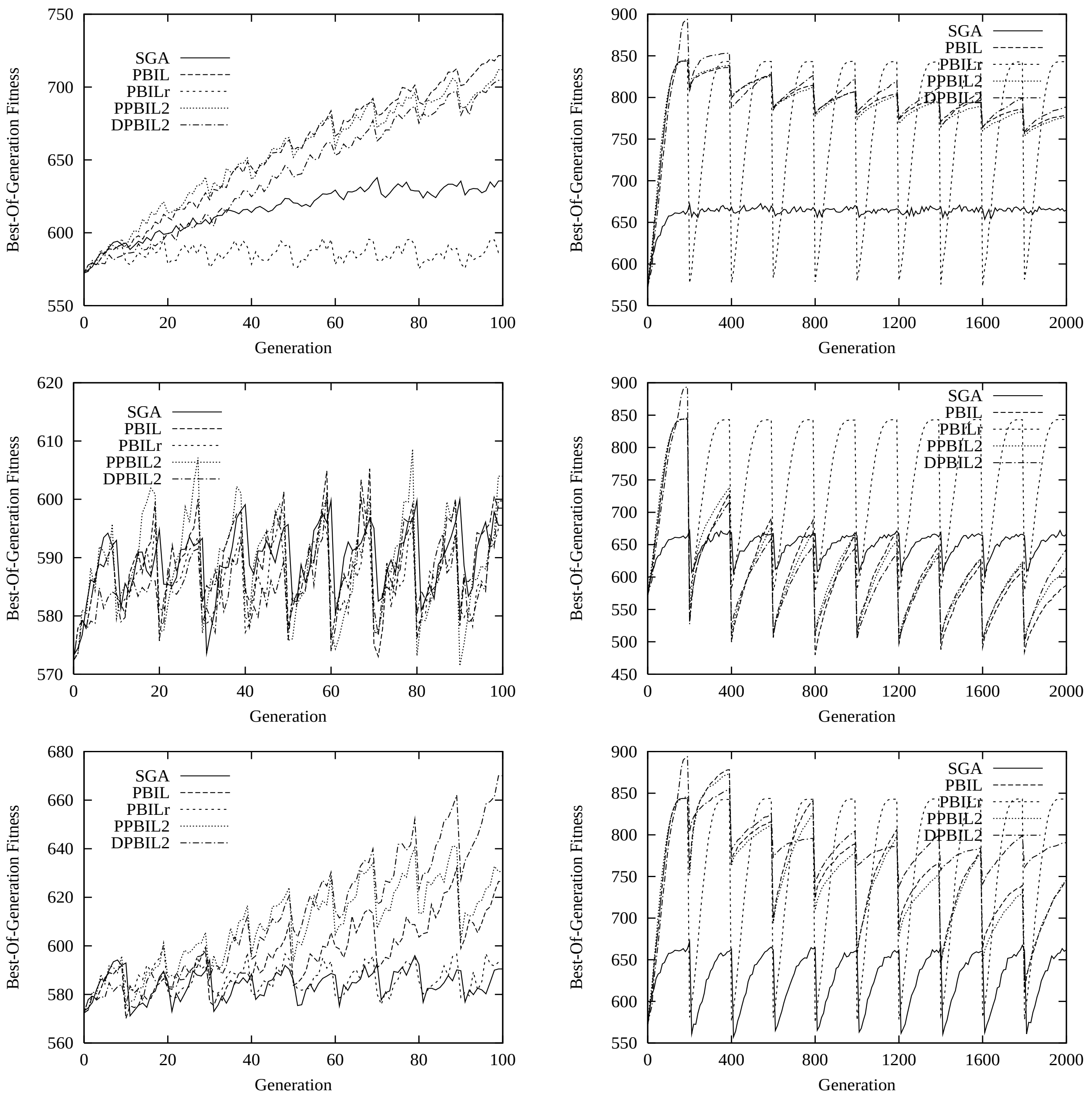


Fig. 9 Dynamic behavior of algorithms on dynamic deceptive functions. The environmental dynamics parameter τ is set to 10 (*Left Column*) and 200 (*Right Column*) respectively and ρ is set to 0.05, 0.4, and 0.95 from top to bottom row respectively.

For SGA the mutation scheme gives it certain learning capacity in dynamic environments. Hence, its average performance for each dynamic period does not drop too heavily with the growing of dynamic periods. However, when the environment undergoes severer and severer changes, i.e., when the value of ρ changes from 0.05 to 0.95, SGA faces harsher and harsher starting points when the environment changes. Hence, the performance of SGA degrades consistently with the increasing of ρ and SGA does not have a big "U" performance curve on most dynamic optimization problems.

6 Introducing the Central Probability Vector

6.1 Modified Algorithms

One major problem for EAs to solve dynamic optimization problems is due to the convergence of population or probability vector. Once converged, the EA loses the required diversity to adapt to the changing environment. To address this problem, Grefenstette [15] introduced the *random immigrants* approach into GAs where in every generation the population is partly replaced by ran-

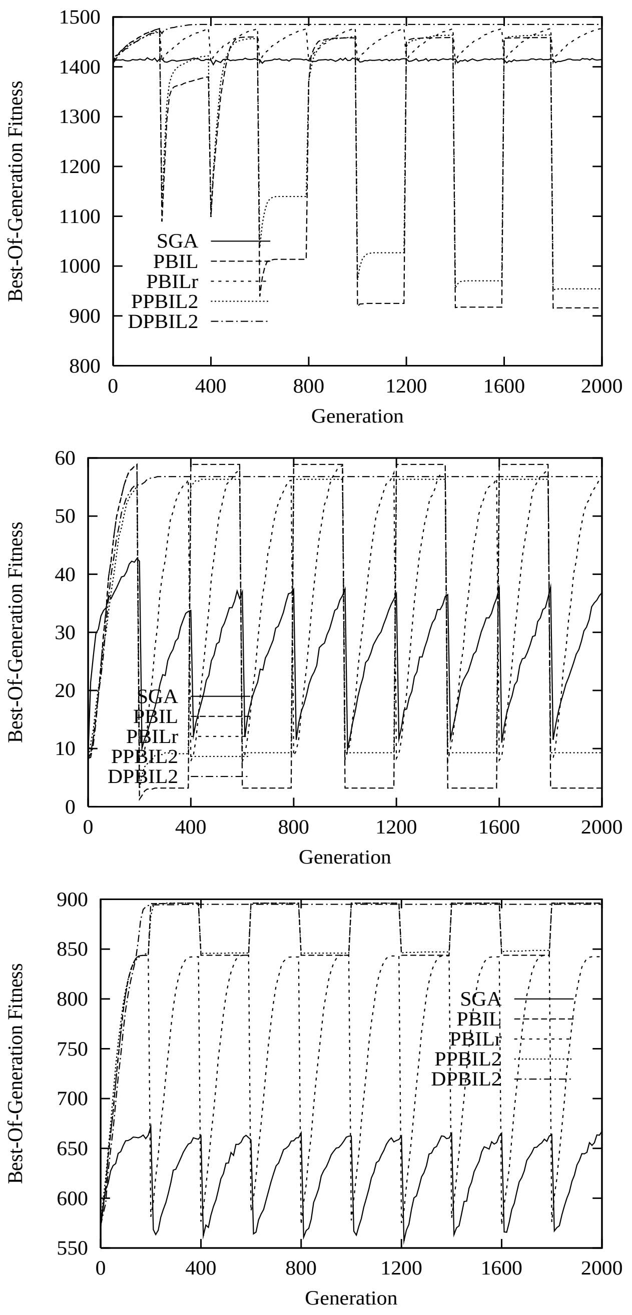


Fig. 10 Dynamic behavior of algorithms on dynamic problems: (*Top*) Knapsack, (*Middle*) Royal Road, and (*Bottom*) Deceptive. The environmental dynamics parameter τ is set to 200 and ρ is set to 1.0.

domly created individuals (random immigrants). Since the approach only replaces a small ratio, e.g. 10%, of the population, it introduces diversity without disrupting the ongoing search progress greatly.

In this paper, incorporating a similar technique into PBILs is also investigated. The idea is to introduce the central probability vector into PBIL since from it random solutions can be sampled. We add a central probability vector into PBIL, PPBIL2, and DPBIL2 and call

```
begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ to $l$ do $P_1^0[i] := 0.5$; $P_2[i] := 0.5$; endfor;
  // initialize sample sizes for probability vectors
  $n_1 := 0.9 * n$; $n_2 := 0.1 * n$;
  repeat
    $S_1^t :=$ generateSamplesFromProbVector($P_1^t, n_1$);
    $S_2^t :=$ generateSamplesFromProbVector($P_2, n_2$);
    evaluateSamples($S_1^t$, $S_2^t$);
    $B_1^t :=$ selectBestSolution($S_1^t$);
    $B_2^t :=$ selectBestSolution($S_2^t$);
    // update probability vectors
    for $i := 1$ to $l$ do
      if $f(B_1^t) \geq f(B_2^t)$ then // learn $P_1^t$ toward $B_1^t$
        $P_1^t[i] := (1-\alpha) * P_1^t[i] + \alpha * B_1^t[i]$;
      else $P_1^t[i] := (1-\alpha) * P_1^t[i] + \alpha * B_2^t[i]$;
    endfor;
    $t := t + 1$;
  until $terminated$ = true; // e.g., $t > t_{max}$
end;

PBILc's parameters $l$, $\alpha$ and $n$ are the same as PBIL's.
```

Fig. 11 Pseudocode for PBILc.

the obtained algorithms PBILc, PPBIL3, and DPBIL3 respectively. The pseudocodes for PBILc, PPBIL3, and DPBIL3 are shown in Fig. 11 to Fig. 13 respectively.

Within PBILc, PPBIL3, and DPBIL3, we set the sample size of the central probability vector to a small constant value, $0.1 * n$, in order to limit its effect on the algorithm as a whole. The sampling mechanism is the same as that in PPBIL2 and DPBIL2. The probability vectors are independently sampled to generate their own set of solutions. However, the central probability vector doesn't change or learn over time. Within PPBIL3 the other two probability vectors learn from the best solution generated by themselves independently. However, if the best solution generated by the central probability vector is better than their best solution, they learn from that solution. Within PBILc and DPBIL3, the learning mechanism is a little different. As in DPBIL2 only the first probability vector P_1 learns. In PBILc, if P_1's best sample B_1 has higher fitness than P_2's best sample B_2, P_1 will learn towards B_1; otherwise, P_1 will learn towards B_2. In DPBIL3 if P_1's best sample B_1 has the overall highest fitness, P_1 will learn towards B_1; if P_3's best sample B_3 has higher fitness than both P_1's and P_2's, P_1 will learn towards B_3; otherwise, if $f(B_2) > f(B_1)$ and $f(B_2) \geq f(B_3)$, P_1 will learn away from B_2.

As in PPBIL2 and DPBIL2, within PPBIL3 and DPBIL3 the sample size of the two varying probability vectors is initialized to an equal value and is slightly adapted by a constant value $\Delta = 6$ within the range of $[n_{min}, n_{max}]$

```
begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ to $l$ do
    $P_1^0[i] := 0.5$; $P_2^0[i] := rand[0.0, 1.0]$; $P_3[i] := 0.5$;
  endfor;
  // initialize sample sizes for probability vectors
  $n_1^0 := n_2^0 := 0.45 * n$; $n_3 := 0.1 * n$;
  repeat
    $S_1^t :=$ generateSamplesFromProbVector($P_1^t, n_1^t$);
    $S_2^t :=$ generateSamplesFromProbVector($P_2^t, n_2^t$);
    $S_3^t :=$ generateSamplesFromProbVector($P_3, n_3$);
    evaluateSamples($S_1^t$, $S_2^t$, $S_3^t$);
    $B_1^t :=$ selectBestSolution($S_1^t$);
    $B_2^t :=$ selectBestSolution($S_2^t$);
    $B_3^t :=$ selectBestSolution($S_3^t$);
    // update probability vectors
    for $i := 1$ to $l$ do
      if $f(B_1^t) \geq f(B_3^t)$ then // learn $P_1^t$ toward $B_1^t$
        $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * B_1^t[i]$;
      else $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * B_3^t[i]$;
      if $f(B_2^t) \geq f(B_3^t)$ then // learn $P_2^t$ toward $B_2^t$
        $P_2^t[i] := (1 - \alpha) * P_2^t[i] + \alpha * B_2^t[i]$;
      else $P_2^t[i] := (1 - \alpha) * P_2^t[i] + \alpha * B_3^t[i]$;
    endfor;
    // update sample sizes for probability vectors
    if $f(B_1^t) > f(B_2^t)$ then $n_1^t := min\{n_1^t + \Delta, n_{max}\}$;
    if $f(B_1^t) < f(B_2^t)$ then $n_1^t := max\{n_1^t - \Delta, n_{min}\}$;
    $n_2^t := n - n_3 - n_1^t$;
    $t := t + 1$;
  until $terminated$ = true;    // e.g., $t > t_{max}$
end;

PPBIL3's parameter settings:
  $l$, $\alpha$, $\Delta$, $n$, $n_1^t$, $n_2^t$: the same as PPBIL2's
  $n_3$: constant sample size by the 3rd prob. vector (12).
  $n_{min}$: min sample size by prob. vector 1 and 2 (24).
  $n_{max}$: max sample size by prob. vector 1 and 2 (84).
```

Fig. 12 Pseudocode for PPBIL3.

```
begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ to $l$ do $P_1^0[i] := P_2^0[i] := P_3[i] := 0.5$; endfor;
  // initialize sample sizes for probability vectors
  $n_1^0 := n_2^0 := 0.45 * n$; $n_3 := 0.1 * n$;
  repeat
    $S_1^t :=$ generateSamplesFromProbVector($P_1^t, n_1^t$);
    $S_2^t :=$ generateSamplesFromProbVector($P_2^t, n_2^t$);
    $S_3^t :=$ generateSamplesFromProbVector($P_3, n_3$);
    evaluateSamples($S_1^t$, $S_2^t$, $S_3^t$);
    $B_1^t :=$ selectBestSolution($S_1^t$);
    $B_2^t :=$ selectBestSolution($S_2^t$);
    $B_3^t :=$ selectBestSolution($S_3^t$);
    // update probability vectors
    for $i := 1$ to $l$ do
      if $f(B_1^t) \geq max\{f(B_2^t), f(B_3^t)\}$ then
        $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * B_1^t[i]$;
      else if $f(B_3^t) > max\{f(B_1^t), f(B_2^t)\}$ then
        $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * B_3^t[i]$;
      else $P_1^t[i] := (1 - \alpha) * P_1^t[i] + \alpha * (1.0 - B_2^t[i])$;
      $P_2^t[i] := 1.0 - P_1^t[i]$;
    endfor;
    // update sample sizes for probability vectors
    if $f(B_1^t) > f(B_2^t)$ then $n_1^t := min\{n_1^t + \Delta, n_{max}\}$;
    if $f(B_1^t) < f(B_2^t)$ then $n_1^t := max\{n_1^t - \Delta, n_{min}\}$;
    $n_2^t := n - n_3 - n_1^t$;
    $t := t + 1$;
  until $terminated$ = true;    // e.g., $t > t_{max}$
end;

DPBIL3's parameter settings are the same as PPBIL3's.
```

Fig. 13 Pseudocode for DPBIL3.

$= [0.2 * n, 0.7 * n] = [24, 84]$ according to their relative performance. If one probability vector outperforms the other, its sample size is increased by Δ while the other's decreased by Δ; otherwise, there is no change. The learning rate α is set to 0.05 for all PBILs.

In order to compare the effect of introducing the central probability vector into PBILs with the random immigrants technique in GAs one variant of SGA, called RIGA, which combines the random immigrants technique within SGA is also studied as a peer algorithm. RIGA differs from SGA only in that when the population has undergone the crossover and mutation operations and just before it is put to evaluation, a subset of randomly selected individuals (10% of the population) is replaced by randomly created individuals. Then the population is evaluated and put to next evolution cycle. The genetic operators and relevant parameter settings for RIGA are all the same as those for SGA.

6.2 Experimental Results

The experimental settings for RIGA, PBILc, PPBIL3 and DPBIL3 are the same as previous settings. The experimental results are shown in Fig. 14 (where the environmental parameter setting is indexed the same way by Table 1) and Table 4. Some key statistical test results are given in Table 5. From Table 4, Table 5 and Fig. 14 the following results can be observed.

First, generally speaking PBILc, PPBIL3, and DPBIL3 outperform their peers PBIL, PPBIL2, and DPBIL2 respectively, especially when the environmental dynamics parameter τ is large and ρ is set to medium values of 0.4 and 0.6 or $rand(0.01, 0.99)$. When $\tau = 10$, the effect of introducing the central probability vector is not significant in many cases or is negative in some

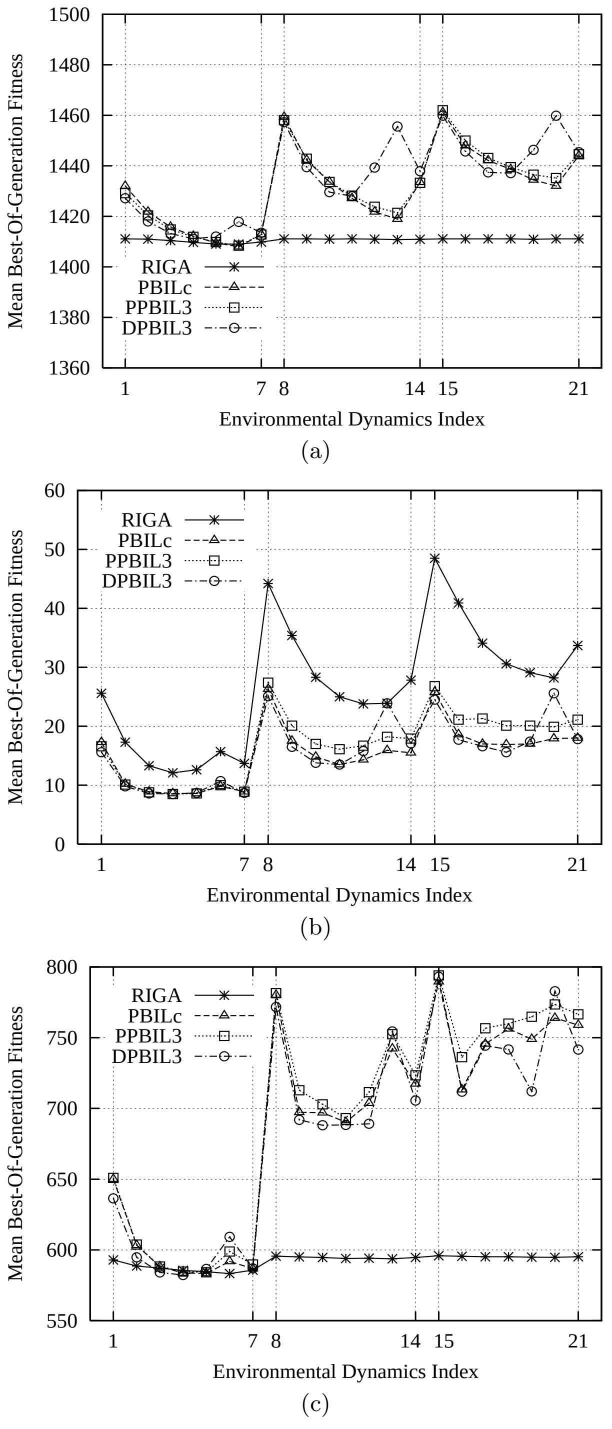


Fig. 14 Experimental results of RIGA, PBILc, PPBIL3, and DPBIL3 with respect to mean best-of-generation fitness against different environmental dynamics parameter settings on dynamic problems: (a) Knapsack, (b) Royal Road, and (c) Deceptive.

cases. This is because convergence is not very serious when the environment changes quickly. When ρ is set to medium values, PBILc, PPBIL3, and DPBIL3 achieve statistically significantly better performance over their peers respectively, see Table 5 for the t-test results with respect to $PBILc - PBIL$, $PPBIL3 - PPBIL2$, and $DPBIL3 - DPBIL2$. This happens because the central probability vector works well under dynamic environments with medium degree of changes. In the genotype space with ρ set to a medium value each time when the environment changes an optimal solution is shifted about halfway away from its original point toward its complementary point in terms of Hamming distance, and falls into the very area represented by the central vector.

Second, both PPBIL3 and DPBIL3 now outperform RIGA on most dynamic knapsack and deceptive problems, see the t-test results with respect to $PPBIL3 - RIGA$ and $DPBIL3 - RIGA$ in Table 5. This result happens because the advantage of introducing diversity by the mutation mechanism in RIGA is now overrun by the effect of the central probability vector in PPBIL3 and DPBIL3. However, on dynamic royal road functions it seems that the central probability vector in PPBIL3 and DPBIL3 is not strong enough for them to beat RIGA.

Third, comparing PBILc with PPBIL2 and DPBIL2, from the t-test results in Table 5 with respect to $PBILc - PPBIL2$ and $PBILc - DPBIL2$ it can be seen that when τ is large PBILc significantly outperforms PPBIL2 on almost all dynamic problems and it outperforms DPBIL2 on most dynamic problems except when ρ is set to 0.95. This means when convergence becomes a problem, the central probability vector is more helpful than just an extra or dual probability vector. However, when the environment suffers significant changes, e.g., $\rho = 0.95$, introducing the dual probability vector is more helpful than the central probability vector.

Finally, from Table 5 an interesting and sort of confusing observation is that RIGA outperforms SGA on several dynamic royal road functions when ρ is set to medium values and on dynamic knapsack problems when $\tau = 10$ and $\rho = 0.05$ while it is beaten by SGA on most other dynamic problems. The reason lies in the interactive effect between SGA and the problems. According to our extra experimental results (not shown in this paper) decreasing the mutation probability p_m in SGA from 0.01 to 0.001 increases SGA's performance on the stationary knapsack problem and deceptive function while decreasing its performance on the stationary royal road function. This means strengthening the mutation and hence the diversity may not be beneficial for the knapsack and deceptive problems. This also sort of explains that the effect of introducing random immigrants into SGA is problem-dependent.

Similarly, in order to better understand the effect of the central probability vector, we give the dynamic behavior of RIGA, PBILc, PPBIL2, DPBIL2, PPBIL3, and DPBIL3 with respect to best-of-generation fitness against generations on dynamic problems in Fig. 15 to Fig. 17 respectively, where the value of τ equals 200 and ρ equals 0.05, 0.4, and 0.95 from top to bottom row respectively. From these figures it can be seen that when ρ is small or large, the central probability vector

Table 4 Experimental results of RIGA, PBILc, PPBIL3, and DPBIL3 on dynamic problems with respect to overall mean best-of-generation fitness.

Dynamics		Knapsack Problem				Royal Road Function				Deceptive Function			
τ	ρ	*RIGA*	*PBILc*	*PPBIL3*	*DPBIL3*	*RIGA*	*PBILc*	*PPBIL3*	*DPBIL3*	*RIGA*	*PBILc*	*PPBIL3*	*DPBIL3*
10	0.05	1411.1	1432.0	1429.2	1427.2	25.6	17.3	16.6	15.6	592.9	650.1	650.9	636.4
10	0.2	1411.0	1421.8	1420.4	1418.0	17.3	10.2	10.1	9.8	588.6	603.3	603.8	594.7
10	0.4	1410.4	1415.8	1414.8	1413.2	13.3	8.9	8.8	8.6	587.1	588.2	588.3	584.1
10	0.6	1409.7	1412.2	1411.9	1411.1	12.1	8.6	8.5	8.5	585.5	583.7	584.7	582.2
10	0.8	1409.0	1409.5	1409.8	1412.0	12.6	8.6	8.6	8.7	584.6	583.4	584.2	586.5
10	0.95	1409.0	1408.0	1408.4	1417.8	15.7	9.8	10.0	10.7	583.3	591.9	598.8	609.2
10	*rand*	1409.8	1412.8	1412.8	1413.5	13.7	8.9	8.9	8.7	585.7	586.7	589.7	586.9
100	0.05	1411.1	1459.3	1458.1	1457.3	44.2	26.3	27.4	25.1	595.6	780.2	781.5	771.7
100	0.2	1411.1	1442.3	1442.8	1439.5	35.4	17.5	20.1	16.5	595.1	697.3	712.9	691.9
100	0.4	1411.0	1433.6	1433.6	1429.6	28.3	14.8	17.0	13.8	594.6	697.0	702.9	688.1
100	0.6	1411.1	1427.1	1428.2	1428.0	25.0	13.5	16.1	13.5	593.9	690.3	693.2	688.3
100	0.8	1411.0	1421.8	1423.8	1439.3	23.8	14.3	16.7	15.8	594.1	703.7	711.5	689.2
100	0.95	1410.8	1418.9	1421.4	1455.6	23.9	15.9	18.2	23.9	593.7	742.2	752.5	754.4
100	*rand*	1410.9	1433.5	1433.3	1437.9	27.8	15.5	17.9	17.1	594.6	717.4	723.4	705.6
200	0.05	1411.1	1461.2	1462.0	1459.8	48.5	25.8	26.8	24.5	595.9	789.6	794.0	793.1
200	0.2	1411.1	1448.0	1450.0	1445.7	40.9	18.6	21.1	17.7	595.5	713.3	736.3	711.8
200	0.4	1411.1	1442.1	1443.1	1437.4	34.1	17.0	21.3	16.6	595.2	745.6	756.7	744.4
200	0.6	1411.1	1438.6	1439.5	1437.1	30.6	16.9	20.1	15.6	595.2	756.3	760.0	741.7
200	0.8	1410.9	1434.4	1436.5	1446.3	29.1	17.0	20.1	17.4	594.8	749.0	764.8	712.0
200	0.95	1411.1	1432.0	1435.2	1459.8	28.2	17.9	19.9	25.6	594.7	763.9	773.5	782.9
200	*rand*	1411.1	1444.0	1444.6	1445.4	33.7	18.0	21.1	17.8	595.2	758.9	766.5	741.6

Table 5 Statistical comparison of algorithms on dynamic problems by one-tailed t-test with 98 degrees of freedom at a 0.05 level of significance. The t-test result regarding Alg. 1 − Alg. 2 is shown as "+", "−", or "∼" when Alg. 1 is significantly better than, significantly worse than, or statistically equivalent to Alg. 2 respectively.

t-test Result	Knapsack Problem							Royal Road Function							Deceptive Function						
$\tau = 10$, $\rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$PPBIL3 - PPBIL2$	∼	∼	+	+	+	+	+	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼	∼	−	−	∼
$DPBIL3 - DPBIL2$	−	−	∼	∼	∼	−	∼	∼	∼	∼	∼	∼	−	−	∼	−	−	−	∼	∼	−
$PPBIL3 - RIGA$	+	+	+	+	+	−	+	−	−	−	−	−	−	−	+	+	∼	∼	∼	+	+
$DPBIL3 - RIGA$	+	+	+	+	+	+	+	−	−	−	−	−	−	−	+	+	−	−	+	+	∼
$PBILc - PBIL$	−	∼	∼	+	+	+	∼	∼	∼	∼	∼	∼	∼	∼	−	−	∼	∼	−	−	∼
$PBILc - PPBIL2$	+	+	+	+	+	+	+	∼	∼	∼	∼	∼	∼	∼	∼	−	∼	−	−	−	−
$PBILc - DPBIL2$	+	+	+	+	−	−	−	+	∼	+	+	−	−	∼	+	+	+	∼	−	−	−
$RIGA - SGA$	−	−	∼	∼	+	+	∼	−	∼	+	+	+	∼	+	−	−	−	∼	∼	∼	−
$\tau = 100$, $\rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$PPBIL3 - PPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+	+
$DPBIL3 - DPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+
$PPBIL3 - RIGA$	+	+	+	+	+	+	+	−	−	−	−	−	−	−	+	+	+	+	+	+	+
$DPBIL3 - RIGA$	+	+	+	+	+	+	+	−	−	−	−	−	∼	−	+	+	+	+	+	+	+
$PBILc - PBIL$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+	+
$PBILc - PPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+	+
$PBILc - DPBIL2$	+	+	+	+	+	−	+	+	+	+	+	+	−	+	∼	+	+	+	+	−	+
$RIGA - SGA$	−	−	−	−	−	−	−	−	−	−	∼	+	∼	∼	−	−	−	−	−	−	−
$\tau = 200$, $\rho \Rightarrow$	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*	.05	.2	.4	.6	.8	.95	*rand*
$PPBIL3 - PPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+	+
$DPBIL3 - DPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	−	+	+	+	+	−	+
$PPBIL3 - RIGA$	+	+	+	+	+	+	+	−	−	−	−	−	−	−	+	+	+	+	+	+	+
$DPBIL3 - RIGA$	+	+	+	+	+	+	+	−	−	−	−	−	−	−	+	+	+	+	+	+	+
$PBILc - PBIL$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	∼	+
$PBILc - PPBIL2$	+	+	+	+	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+	+
$PBILc - DPBIL2$	+	+	+	+	+	−	+	+	+	+	+	+	−	+	−	+	+	+	+	−	+
$RIGA - SGA$	−	−	−	−	−	−	−	−	−	∼	+	+	+	∼	−	−	−	−	−	−	−

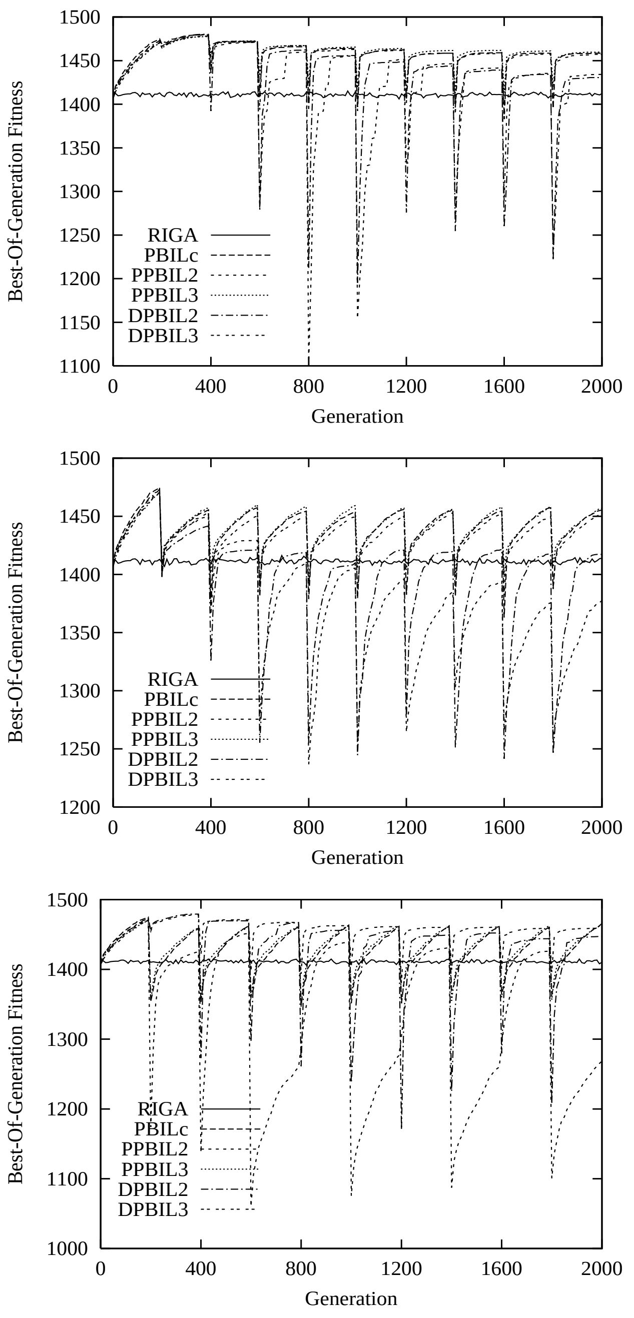


Fig. 15 Dynamic behavior of algorithms on dynamic knapsack problems. The environmental dynamics parameter τ is set to 200 and ρ is set to 0.05, 0.4, and 0.95 from top to bottom respectively.

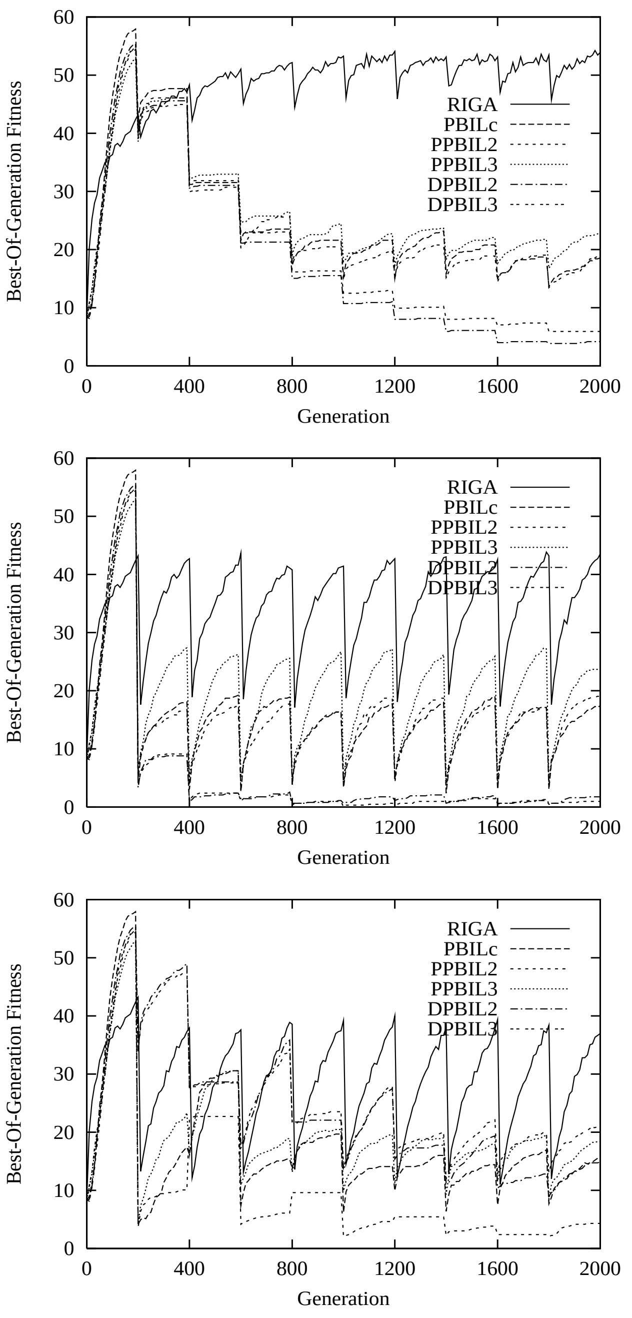


Fig. 16 Dynamic behavior of algorithms on dynamic royal road functions. The environmental dynamics parameter τ is set to 200 and ρ is set to 0.05, 0.4, and 0.95 from top to bottom respectively.

does not help much. However, when ρ is set to medium values (e.g., 0.4), the performance of PBILc, PPBIL3 and DPBIL3 is greatly improved in comparison to their peers (the performance of PBIL is not shown) respectively during dynamic periods. The central probability vector stops their performance from significant dropping. From Fig. 15 and Fig. 17 it can also be seen that for RIGA due to the random immigrants scheme its performance is degraded heavily during the stationary period and hence the following dynamic periods on dynamic knapsack problems and dynamic deceptive functions.

7 Conclusions

In this paper we investigate the application of Population-Based Incremental Learning (PBIL) algorithms for dynamic optimization problems. We study the effect of in-

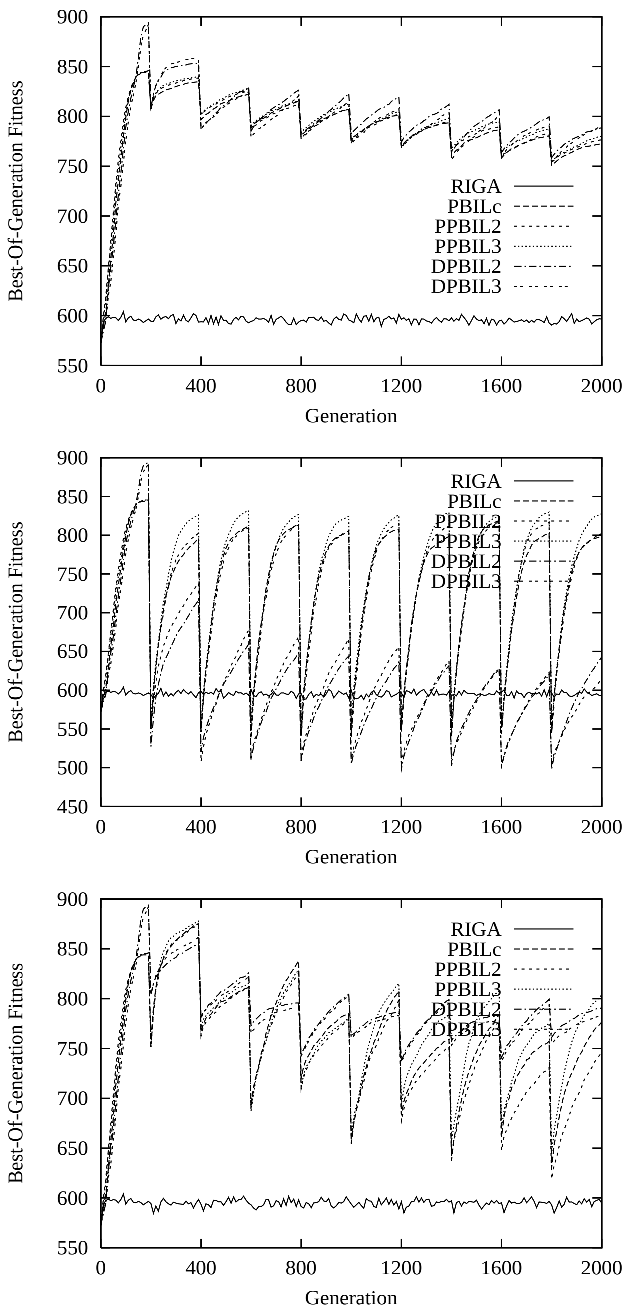


Fig. 17 Dynamic behavior of algorithms on dynamic deceptive functions. The environmental dynamics parameter τ is set to 200 and ρ is set to 0.05, 0.4, and 0.95 from top to bottom respectively.

troducing several approaches, such as the re-start, multi-population, and random immigrants methods, from EA's community into PBIL to improve its performance in dynamic environments. Inspired by the complementarity mechanism broadly existing in nature, we propose a Dual PBIL that operates on a pair of probability vectors that are dual to each other with respect to the central point in the genotype space. In order to counterbalance the problem caused by the convergence of probability vectors, the central probability vector is also introduced into PBILs.

This paper also formalizes a new dynamic problem generator that can generate required dynamics from any binary encoded stationary problem. This generator is genotype-based, easy to realize required dynamics, and convenient for theoretical analysis. Based on the new dynamic problem generator, a series of dynamic problems are systematically constructed from several benchmark stationary problems. These dynamic problems are used as the test base for the experimental study to compare the investigated PBILs and two variants of standard GA.

From the experimental results, the following conclusions can be achieved on the tested dynamic problems.

First, on the stationary problems introducing extra probability vector into PBIL may not be beneficial.

Second, if it is feasible to timely detect environmental changes, the re-start scheme is a good choice for PBIL in dynamic environments, especially when the environment changes slowly and hence convergence becomes a problem. However, it is usually not possible to detect environmental changes timely, which greatly degrades the re-start scheme for PBIL in dynamic environments.

Third, when the environment is subject to significant changes in the sense of genotype space, introducing the dual probability vector into PBIL can achieve very high performance improvement.

Fourth, introducing the central probability vector can improve PBIL's performance under dynamic environments, especially when the environment is subject to medium degree of changes in the genotype space.

Finally, the effect of introducing the random immigrants scheme into SGA is problem dependent.

Generally speaking, the experimental results indicate that PBILs with dual and central probability vectors seem to be a good choice as EAs for dynamic problems.

8 Future Work

This paper starts an interesting work on applying PBILs for dynamic optimization problems. Based on this paper there are several works to be carried out in the future.

First, PBILs investigated in this paper are relatively simple. It is an interesting work to investigate more mechanisms, such as mutation, population interaction schemes [4], and the hypermutation technique from EA's community into PBILs and compare their performance for dynamic optimization problems.

Second, it is also an interesting future work to extend the results in this paper to other Estimation of Distribution Algorithms (EDAs) [19], [25], of which PBILs are a sub-class, and compare obtained algorithms with other GAs or EAs for dynamic optimization problems.

Finally, based on the new dynamic problem generator it is an important work to carry out theoretical analysis of the performance of PBILs and other EAs for dynamic optimization problems, e.g, with respect to the environmental change speed and change severity.

Acknowledgment

The authors would like to thank Dr. Jürgen Branke and the anonymous reviewers for their thoughtful suggestions and helpful comments. Shengxiang Yang was supported by UK EPSRC under Grant GR/S79718/01.

References

1. T. Bäck (1998). On the Behavior of Evolutionary Algorithms in Dynamic Fitness Landscape. *Proc. of the 1998 IEEE Int. Conf. on Evolutionary Computation*, 446-451.
2. J. E. Baker (1987). Reducing Bias and Inefficiency in the Selection Algorithms. In J. J. Grefenstelle (ed.), *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, 14-21. Lawrence Erlbaum Associates.
3. S. Baluja (1994). Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. *Technical Report CMU-CS-94-163*, Carnegie Mellon University, USA.
4. S. Baluja and R. Caruana (1995). Removing the Genetics from the Standard Genetic Algorithm. In *Proc. of the 12th Int. Conf. on Machine Learning*, 38-46.
5. J. Branke (1999). Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. *Proc. of the 1999 Congress on Evolutionary Computation*, Vol. 3, 1875-1882.
6. J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck (2000). A Multi-Population Approach to Dynamic Optimization Problems. In *Adaptive Computing in Design and Manufacturing*.
7. J. Branke (2001). Evolutionary Approaches to Dynamic Optimization Problems - Updated Survey. *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, 134-137.
8. J. Branke (2002). *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers.
9. H. G. Cobb (1990). An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments. *Technical Report AIC-90-001*, Naval Research Laboratory, Washington, USA.
10. H. G. Cobb and J. Grefenstette (1993). Genetic Algorithms for Tracking Changing Environments. In *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 523-530.
11. D. Dasgupta and D. McGregor (1992). Nonstationary Function Optimization Using the Structured Genetic Algorithm. *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, 145-154.
12. D. E. Goldberg and R. E. Smith (1987). Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy. In J. J. Grefenstelle (ed.), *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, 59-68. Lawrence Erlbaum Associates.
13. D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.
14. C. González, J. A. Lozano, and P. Larrañaga (2000). Analyzing the Population Based Incremental Learning Algorithm by Means of Discrete Dynamical Systems. *Complex Systems*, 12(4): 465-479.
15. J. J. Grefenstette (1992). Genetic Algorithms for Changing Environments. In R. Männer and B. Manderick (eds.), *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, 137-144.
16. J. J. Grefenstette (1999). Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach. *Proc. of the 1999 Congress on Evolutionary Computation*, Vol. 3, 2031-2038.
17. M. Höhfeld and G. Rudolph (1997). Towards a Theory of Population-Based Incremental Learning. *Proc. of the 4th IEEE Conference on Evolutionary Computation*, 1-5.
18. J. H. Holland (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, University of Michigan Press.
19. P. Larrañaga and J. A. Lozano (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.
20. J. Lewis, E. Hart and G. Ritchie (1998). A Comparison of Dominance Mechanisms and Simple Mutation on Non-Stationary Problems. *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature*, 139-148.
21. M. Mitchell, S. Forrest and J. H. Holland (1992). The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. *Proc. of the 1st European Conf. on Artificial Life*, 245-254.
22. N. Mori, H. Kita and Y. Nishikawa (1997). Adaptation to Changing Environments by Means of the memory Based Thermodynamical Genetic Algorithm. In T. Bäck (ed.), *Proc. of the 7th Int. Conf. on Genetic Algorithms*, 299-306. Morgan Kaufmann Publishers.
23. R. W. Morrison and K. A. De Jong (1999). A Test Problem Generator for Non-Stationary Environments. *Proc. of the 1999 Congress on Evolutionary Computation*, Vol. 3, 2047-2053.
24. R. W. Morrison and K. A. De Jong (2000). Triggered Hypermutation Revisited. *Proc. of the 2000 Congress on Evolutionary Computation*, 1025-1032.
25. H. Mühlenbein and G. Paab (1996). From Recombination of Genes to the Estimation of Distributions I. Binary Parameters. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (eds.), *Proc. of the 4th Int. Conf. on Parallel Problem Solving from Nature*, 178-187.
26. K. P. Ng and K. C. Wong (1995). A New diploid Scheme and Dominance Change Mechanism for Non-Stationary Function Optimisation. In L. J. Eshelman (ed.), *Proc. of the 6th Int. Conf. on Genetic Algorithms*.
27. M. P. Servais, G. de Jaer, and J. R. Greene (1997). Function Optimization Using Multiple-Base Population Based Incremental Learning. *Proc. of the 8th South African Workshop on Pattern Recognition*.
28. K. Trojanowski and Z. Michalewicz (2000). Evolutionary Optimization in Non-Stationary Environments. *Journal of Computer Science and Technology*, **1**(2): 93-124.
29. L. D. Whitley (1991). Fundamental Principles of Deception in Genetic Search. In G. J. E. Rawlins (ed.), *Foundations of Genetic Algorithms 1*, 221-241.
30. S. Yang (2003). Non-Stationary Problem Optimization Using the Primal-Dual Genetic Algorithm. *Proc. of the 2003 Congress on Evolutionary Computation*, Vol. 3, 2246-2253.