Dual Population-Based Incremental Learning for Problem Optimization in Dynamic Environments

Shengxiang Yang[1], Xin Yao[2]

In recent years there is a growing interest in the research of evolutionary algorithms for dynamic optimization problems since real world problems are usually dynamic, which presents serious challenges to traditional evolutionary algorithms. In this paper, we investigate the application of Population-Based Incremental Learning (PBIL) algorithms, a class of evolutionary algorithms, for problem optimization under dynamic environments. Inspired by the complementarity mechanism 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. Using a dynamic problem generating technique we generate a series of dynamic knapsack problems from a randomly generated stationary knapsack problem and carry out experimental study comparing the performance of investigated PBILs and one traditional genetic algorithm. Experimental results show that the introduction of dualism into PBIL improves its adaptability under dynamic environments, especially when the environment is subject to significant changes in the sense of genotype space.

Key words: dynamic optimization, population-based incremental learning, dualism, evolutionary algorithms

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 been widely applied for solving stationary optimization problems where the fitness landscape does not change during the course of computation [8]. However, the environments of real world optimization problems are usually dynamic, i.e., the problem fitness landscape changes over time. For example, in production scheduling problems available resources may change over time. The intrinsic dynamic nature of problems being solved presents serious challenge to traditional EAs since they cannot adapt well to the 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], [11]. Over the past years, a number of researchers have developed many approaches into EAs to address this problem. Branke [6] has grouped them into four categories: 1) increasing diversity after a change [7], [12]; 2) maintaining diversity throughout the run [9]; 3) memory-based methods [10], [14]; and 4) multi-population approaches [5].

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

In this paper we investigate the application of Population-Based Incremental Learning (PBIL) algorithms, a class of EAs, for solving dynamic optimization problems. We study the effect of introducing multi-population approach into PBIL to address dynamic optimization problems. Inspired by the complementarity mechanism that broadly exists 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. Based on a dynamic problem generating technique [15] that can generate dynamic environments from any binary encoded stationary problem, we systematically construct a series of dynamic knapsack problems from a randomly generated stationary knapsack problem and carry out experimental study to compare the performance of investigated PBILs and one variant of traditional genetic algorithm.

In the rest of this paper, we first detail several investigated PBILs including our proposed dual PBIL, next present the algorithmic test environments that consist of a stationary knapsack problem and relevant dynamic problems, then provide the experimental results with analysis, and finally give out our conclusions with discussions on future work.

2. Population-Based Incremental Learning Algorithms

2.1 Population-Based Incremental Learning (PBIL)

The PBIL algorithm, first proposed by Baluja [3], is a combination of evolutionary optimization and competitive learning. It is an abstraction of the genetic algorithm (GA) that explicitly maintains the statistics contained in a GA's population. PBIL has proved to be very successful when compared to standard GAs and hill-climbing algorithms on an amount of benchmark and real-world problems [4]. PBIL

Procedure *PBIL*:

begin
$t := 0$;
// initialize the probability vector
for $i := 1$ **to** L **do**
$P^0[i] := 0.5$;
endfor;
repeat
$S^t := generateSamplesFromProbVector(P^t, n)$;
$evaluateSamples(S^t)$;
$B^t := selectBestSolutionFrom(S^t)$;
// learn the 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;

Fig.1 Pseudo-code of PBIL with one probability vector.

aims to generate a real probability vector, which creates high quality solutions with high probability when sampled. PBIL starts from an initial probability vector with values of each entry set to 0.5. This means when sampling by this initial 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 those values that represent high evaluation solutions. The evolution process is as follows.

During each iteration, a set of samples (solutions) is created according to the current probability vector as follows. 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 corresponding probability value in the probability vector, it is set to 1 (or 0); otherwise it is set to 0 (or 1 respectively). 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 parameter of learning rate. 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 pseudo-code for the PBIL studied 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 fixed at 0.05.

Procedure *PPBIL2*:

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
$n_1^0 := n_2^0 := 0.5 \times n$;
repeat
$S_1^t := generateSamplesFromPro\ bVector(P_1^t, n_1^t)$;
$S_2^t := generateSamplesFromPro\ bVector(P_2^t, n_2^t)$;
$evaluateSamples(S_1^t, S_2^t)$;
$B_1^t := selectBestSolutionFrom(S_1^t)$;
$B_2^t := selectBestSolutionFrom(S_2^t)$;
// learn probability vectors toward best solutions
for $i := 1$ **to** L **do**
$P_1^t[i] := (1-\alpha) \times P_1^t[i] + \alpha \times B_1^t[i]$;
$P_2^t[i] := (1-\alpha) \times P_2^t[i] + \alpha \times B_2^t[i]$;
endfor;
// adjust 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;

Fig.2 Pseudo-code of the Parallel PBIL (PPBIL2).

2.2 Parallel Population-Based Incremental Learning

Using multi-population instead of one population has proved to be a good approach for improving the performance of EAs for dynamic optimization problems [5]. Similarly, we can introduce multi-population into PBIL by using multiple probability vectors. Each probability vector is sampled to generate solutions independently, and is learnt according to the best solution(s) generated by it. For the sake of simplicity, in this paper we investigate a Parallel PBIL with two parallel probability vectors, denoted by PPBIL2. The pseudo-code for PPBIL2 is shown in Fig. 2.

Within PPBIL2 one probability vector P_1 is initialized to be 0.5 for each probability value (in order to compare its

performance with PBIL) and the other P_2 is randomly initialized. The probability vectors P_1 and P_2 are sampled and updated independently. Both P_1 and P_2 have equal initial sample size, half of the total number of samples $n = 120$. However, in order to give the probability vector that performs better more chance to generate samples, the sample sizes are slightly adapted within the range of $[n_{\min}, n_{\max}] = [0.4 \times n, 0.6 \times n] = [48, 72]$ according to their relative performance. If one probability vector outperforms the other, its sample size is increased by $\Delta = 0.05 \times n = 6$ while the other's sample size is decreased by Δ; otherwise, if the two probability vectors tie, there is no change to their sample sizes. The learning rate for both P_1 and P_2 is the same as that for the PBIL.

2.3 Dual Population-Based Incremental Learning

Dualism or complementarity is 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, in this paper we propose a Dual PBIL, denoted by DPBIL2. For the convenience of description, here we first introduce the definition of dual probability vector. Given a probability vector $P = (P[1], \cdots, P[L]) \in I = [0.0, 1.0]^L$ 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 point in the search space. With this definition, DPBIL2 consists of a pair of probability vectors that are dual to each other. The pseudo-code of DPBIL2 is given in Fig. 3.

From 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 solution generated since P_2 changes with P_1 automatically. If the best overall solution is sampled by P_1^t, i.e. $f(B_1^t) \geq f(B_2^t)$, 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 dual probability vector into PBIL lies in two aspects: increasing diversity of samples and fighting significant environment 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 doesn't happen 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.

Procedure *DPBIL2*:

begin
  $t := 0$;
  // initialize probability vectors
  for $i := 1$ **to** L **do**
    $P_1^0[i] := P_2^0[i] := 0.5$;
  endfor;
  // initialize sample sizes for probability vectors
  $n_1^0 := n_2^0 := 0.5 \times n$;
  repeat
    $S_1^t := generateSamplesFromProbVector(P_1^t, n_1^t)$;
    $S_2^t := generateSamplesFromProbVector(P_2^t, n_2^t)$;
    $evaluateSamples(S_1^t, S_2^t)$;
    $B_1^t := selectBestSolutionFrom(S_1^t)$;
    $B_2^t := selectBestSolutionFrom(S_2^t)$;
    // learn 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
        $P_1^t[i] := (1-\alpha) \times P_1^t[i] + \alpha \times B_1^t[i]$;
      else // learn P_1^t away from B_2^t
        $P_1^t[i] := (1-\alpha) \times P_1^t[i] + \alpha \times (1.0 - B_2^t[i])$;
      $P_2^t[i] := 1.0 - P_1^t[i]$;
    endfor;
    // adjust 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;

Fig.3 Pseudo-code of the Dual PBIL (DPBIL2).

3. Algorithm Test Environments

In order to compare the performance of different PBILs, a stationary knapsack problem is randomly constructed as the test problem. A series of dynamic knapsack problems is then constructed from this stationary knapsack problem by a dynamic problem generating technique.

3.1 Stationary Knapsack Problem

The knapsack problem is a well-known NP-complete combinatorial optimization problem and has been well studied in EA's community. 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 0-1knapsack problem can be described as follows:

$$\max \quad p(x) = \sum_{i=1}^{i=m} p_i x_i$$

subject to the weight constraint:

$$\sum_{i=1}^{i=m} w_i x_i \le C$$

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

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

$$w_i = uniformly\ random\ integer[1, 50]$$

$$p_i = w_i + uniformly\ random\ integer[1, 5]$$

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

And given a solution *x*, its fitness $f(x)$ 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 *x* is evaluated as follows:

$$f(x) = \begin{cases} \sum_{i=1}^{i=100} p_i x_i, & if \ \sum_{i=1}^{i=100} w_i x_i \le C \\ 10^{-10} \times (\sum_{i=1}^{i=100} w_i - \sum_{i=1}^{i=100} w_i x_i), & otherwise \end{cases}$$

3.2 Constructing Dynamic Knapsack Problems

In this paper, we construct dynamic test environments from above stationary knapsack problem using a dynamic problem generating technique proposed in [15]. This technique is characterized by two environmental dynamics parameters: the speed of change and the degree of change in the genotype space. The first parameter is referred to as the environmental change period, denoted by τ, and is defined as the number of EA generations between two changes. That is, every τ generations the fitness landscape is changed.

The second parameter is measured by the ratio of ones in a binary template $T(k) \in \{0,1\}^L$ (where *L* is the chromosome length) created for each environmental change period, denoted by ρ. For each environmental change period *k* we first create a binary mask $M(k) \in \{0,1\}^L$ incrementally as:

$$M(k) = M(k-1) \oplus T(k)$$

where $T(k)$ is randomly created for period *k* with $\rho \times L$ ones and " $\oplus$ " is a bitwise exclusive-or (XOR) operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 1$). For the first period, $M(1)$ is initialized to be a zero vector. When evaluating an individual $x \in \{0,1\}^L$ in the population, we first perform the operation $x \oplus M(k)$ on it. The XORed result is then evaluated to obtain a fitness value for the individual *x*. It can be seen that the parameter ρ controls the degree of environmental change. The bigger the value of ρ, the more significant the environmental change.

Putting things together, the environment dynamics can be formulated as follows:

$$f(x,t) = f(x \oplus M(k))$$

where $k = \lceil t / \tau \rceil$ is the period index, $t = [(k-1)\tau, k\tau]$ is the generation counter.

In this paper, we construct dynamic knapsack problems as follows. For each run of an algorithm on each knapsack problem the fitness landscape is periodically changed every τ generations. Based on our preliminary experimental results on the stationary knapsack problem (see Section 4.2), τ is set to 10, 50, 100, 150 and 200 generations respectively in order to 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 the degree of environmental change on the performance of algorithms ρ is set to 0.05, 0.25, 0.5, 0.75, and 1.0 respectively. These values represent different environmental change level, from light shifting ($\rho = 0.05$), to medium variation ($\rho = 0.25, 0.5$), to heavy change ($\rho = 0.75$), and to the extreme case ($\rho = 1.0$) of oscillating between two reversed fitness landscapes.

Totally, we systematically construct a series of 25 dynamic problems, 5 values of τ combined with 5 values of ρ, from the stationary knapsack problem.

4. Computer Experimental Study

4.1 Design of Experiments

Experiments were carried out to compare the performance of investigate PBILs. In order to compare PBILs as a whole with other evolutionary algorithms, we also included a standard genetic algorithm, denoted by SGA, as a peer algorithm in the experiments. SGA has following typical configuration: generational, uniform crossover with a crossover probability $p_c = 0.6$, traditional bit mutation with a mutation probability $p_m = 0.001$, fitness proportionate selection with the Stochastic Universal Sampling (SUS) scheme [2] and without elitist model, and a population size of $n = 120$.

For each experiment of combining algorithm and problem (stationary or dynamic), 50 independent runs were executed with the same 50 random seeds. For each run of an algorithm on each problem, 10 periods of environmental changes were

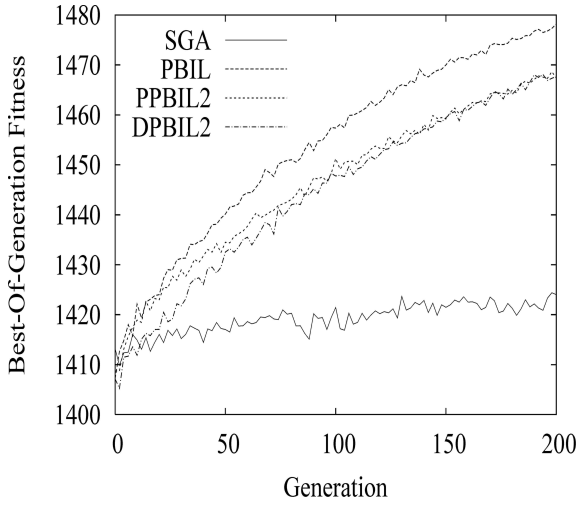

Fig.4 Experimental results on the stationary knapsack problem.

allowed and the best-of-generation fitness was recorded every generation. 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 runs and then averaged over the data gathering period. More formally this is:

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

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., $G = 10 \times \tau$), $N = 50$ is the total number of runs, and $F_{BG_{ij}}$ is the best-of-generation fitness of generation i of run j of an algorithm on a problem.

4.2 Experimental Results on the Stationary Knapsack Problem

In order to help analyze the experimental results on dynamic problems, preliminary experiments were carried out on the stationary knapsack problem. For each run of different algorithm the maximum allowable number of generations was set to 200. The preliminary experimental results with respect to best-of-generation fitness against generations are shown in Fig. 4, where the data were averaged over 50 runs.

From Fig. 4, it can be seen that in general all PBILs outperform SGA. This result is consistent with other researchers' study [4]. PBIL outperforms PPBIL2 and DPBIL2 while PPBIL2 performs as well as DPBIL2. This result shows that on the stationary knapsack problem introducing extra probability vector may not be beneficial since the existence of extra probability vector may slow down the learning speed of the other probability vector, whichever of the two vectors performs better.

4.3 Experimental Results on Dynamic Knapsack Problems

Table 1 Experimental results with respect to the overall mean best-of-generation fitness $\overline{F}_{BG}$ of different algorithms on dynamic knapsack problems.

Param. Setting Index	Parameter Settting (τ, ρ)	Algorithms			
		SGA	PBIL	PPBIL2	DPBIL2
1	(10, 0.05)	1416.6	1432.7	1429.1	1425.2
2	(10, 0.25)	1406.7	1420.3	1418.9	1414.1
3	(10, 0.50)	1402.4	1413.0	1412.4	1411.8
4	(10, 0.75)	1396.9	1409.1	1408.4	1413.1
5	(10, 1.00)	1372.5	1406.8	1404.7	1428.1
6	(50, 0.05)	1424.9	1453.4	1450.8	1450.1
7	(50, 0.25)	1415.8	1426.4	1427.3	1424.1
8	(50, 0.50)	1397.7	1413.3	1416.5	1418.6
9	(50, 0.75)	1377.2	1401.7	1405.4	1423.8
10	(50, 1.00)	1337.3	1393.1	1390.8	1465.8
11	(100, 0.05)	1429.5	1416.2	1442.3	1442.8
12	(100, 0.25)	1421.4	1392.1	1410.7	1406.7
13	(100, 0.50)	1412.1	1381.7	1392.2	1405.7
14	(100, 0.75)	1387.8	1368.8	1383.4	1408.6
15	(100, 1.00)	1338.4	1367.0	1365.0	1474.9
16	(150, 0.05)	1432.1	1403.9	1434.6	1439.7
17	(150, 0.25)	1424.6	1385.4	1401.5	1401.5
18	(150, 0.50)	1418.0	1363.6	1373.9	1400.6
19	(150, 0.75)	1400.7	1340.5	1367.2	1402.7
20	(150, 1.00)	1348.2	1295.3	1311.3	1477.3
21	(200, 0.05)	1433.6	1392.9	1420.6	1439.2
22	(200, 0.25)	1426.7	1363.6	1393.4	1399.6
23	(200, 0.50)	1420.3	1333.2	1362.0	1399.9
24	(200, 0.75)	1407.4	1316.1	1342.5	1405.8
25	(200, 1.00)	1362.1	1233.5	1253.4	1479.1

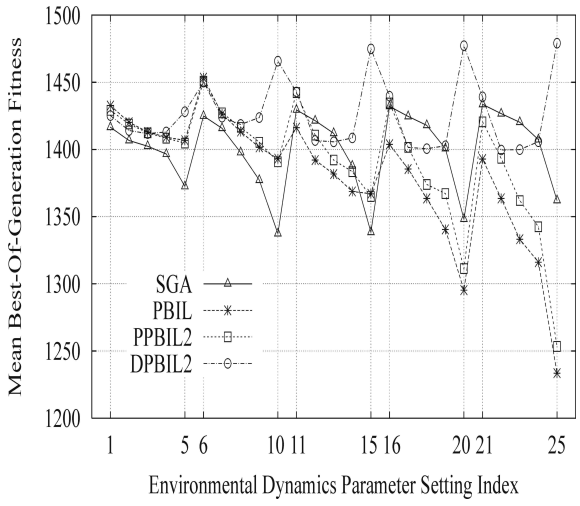

Fig.5 Experimental results on dynamic knapsack problems.

The experimental results on dynamic problems are summarized in Table 1 and plotted in Fig. 5 where the environmental dynamics parameter setting is indexed according to Table 1. From Table 1 and Fig. 5 several results

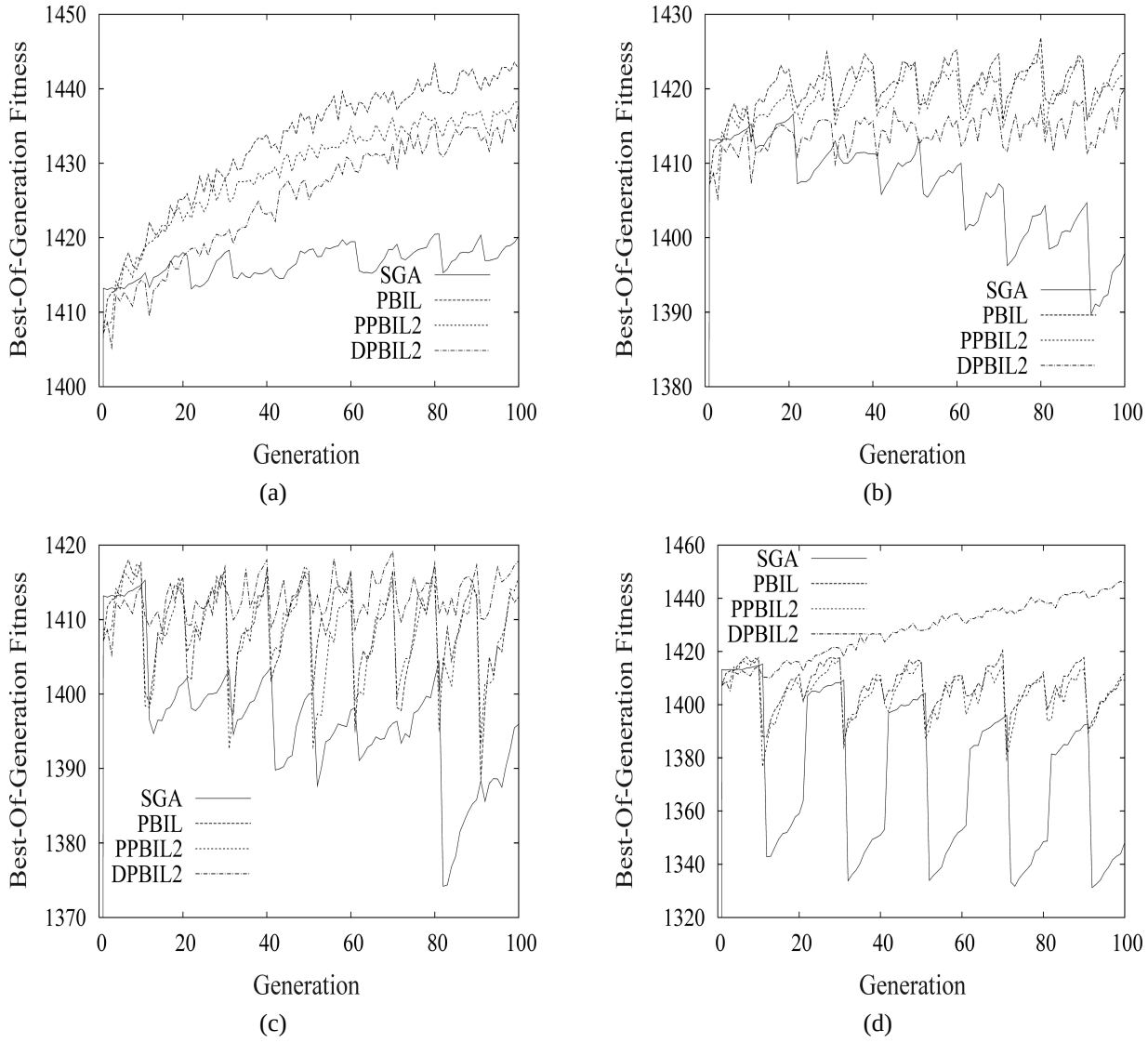

Fig.6 Experimental results with respect to best-of-generation fitness against generations of investigated algorithms on dynamic knapsack problems. The environmental dynamics parameter $\tau = 10$ and ρ is set to (a) 0.05, (b) 0.25, (c) 0.75, and (d) 1.0.

can be observed and are discussed as follows.

First, an obvious result is that for each value of τ DPBIL2 performs consistently with the increasing of the value of ρ. With each fixed τ, when ρ increases from 0.05 to 0.25, 0.50, 0.75 to 1.0 the performance curve of DPBIL2 looks like a big "U" while the other algorithms have a performance curve of "falling stone". This happens because increasing the value of ρ increases the magnitude of environmental changes, which degrades the performance of SGA, PBIL and PPBIL2 persistently. However, in DPBIL2 the introduction of the dual probability vector stops DPBIL2's performance from dropping when the value of ρ reaches 0.5. Thereafter, DPBIL2's performance rises with the increasing of the value of ρ. For each fixed value of τ when $\rho = 1.0$ DPBIL2 achieves the highest performance point. This result confirms our expectation of introducing the dual probability vector into DPBIL2. When the magnitude of environmental change is large, the dual probability vector takes effect quickly to adapt the DPBIL2 to the changed environment.

Second, PBIL is now beaten by both PPBIL2 and DPBIL2 on most situations except for when ρ is small. When ρ is small, the dynamic knapsack problems are closer to the stationary knapsack problem where introducing an extra probability vector may not work well. This is verified by above preliminary experimental results on the stationary knapsack problem as shown in Fig.4. However, when ρ increases the introduction of extra probability vector becomes more and more beneficial. This is because extra probability vector helps improving diversity in the samples.

Third, as opposed to the stationary knapsack problem, SGA now outperforms PBILs on many dynamic knapsack problems, especially when the value of τ is large. When τ is large the algorithms are given more time to search before environment changes and hence they are more likely to converge. Convergence deprives PBILs of the adaptability to

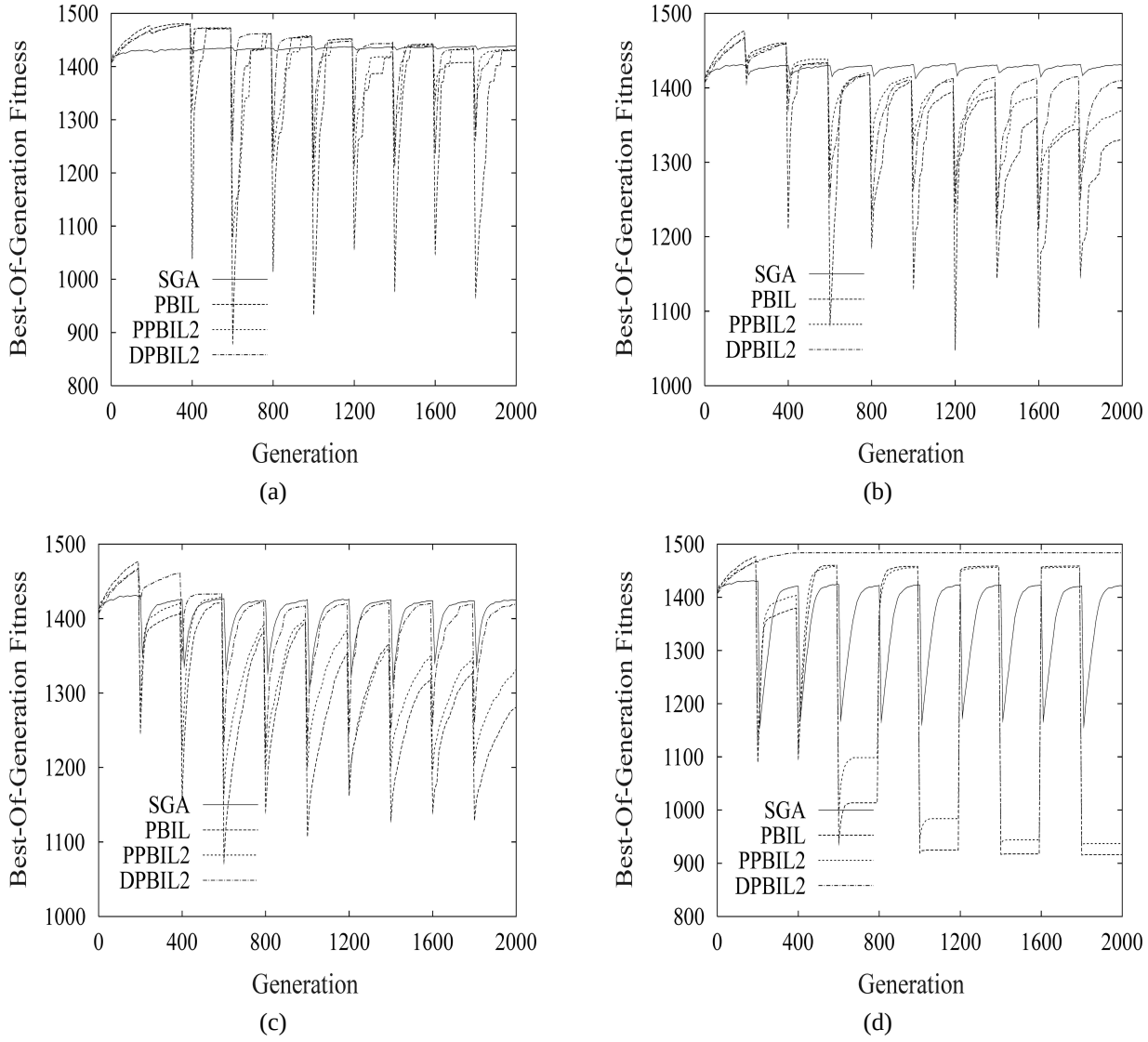

Fig.7 Experimental results with respect to best-of-generation fitness against generations of investigated algorithms on dynamic knapsack problems. The environmental dynamics parameter $\tau = 200$ and ρ is set to (a) 0.05, (b) 0.25, (c) 0.75, and (d) 1.0.

changed environment. However, the mutation mechanism embedded in SGA gives SGA more diversity than PBILs and better adaptability to environment changes. Hence, SGA outperforms PBILs.

In order to better understand the experimental results, we give out the dynamic performance of tested algorithms with respect to best-of-generation fitness against generations on several dynamic problems in Fig. 6 and Fig. 7, where the data were averaged over 50 runs. In Fig. 6 and Fig. 7 the value of τ is set to 10 and 200 respectively, and within both figures the value of ρ is set to 0.05, 0.25, 0.75 and 1.0 respectively. From Fig. 6 and Fig. 7, it can be seen that generally speaking, the performance of the algorithms (except for DPBIL2) drops heavier and heavier with the increasing of the value of ρ as well as the value of τ. With DPBIL2 when $\rho = 1.0$ its performance rises instead of drops with the growing of dynamic periods due to the effect of the dual probability vector. This results in the big "U" curve for DPBIL2's overall performance (see Fig. 5).

5. Conclusions and Future Work

In this paper we investigate the application of Population-Based Incremental Learning (PBIL) algorithms for solving optimization problems under dynamic environments. We study the effect of introducing extra probability vector into PBIL to improve its performance under dynamic environments. Inspired by the complementarity mechanism 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.

Using a dynamic problem generating technique we systematically construct a set of dynamic knapsack problems from a randomly created stationary knapsack problem and based on these stationary and dynamic knapsack problems we carry out experimental study comparing investigated PBILs and one traditional GA. From the experimental results the following conclusions can be achieved.

First, on stationary problems introducing extra probability vector into PBIL may not be beneficial. However, under dynamic environments introducing extra probability vector into PBIL improves its performance.

Second, 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.

Third, though the SGA is beaten by PBILs on the stationary problem, the mutation scheme embedded in SGA helps keeping the diversity in the population and hence improves SGA's performance under dynamic environments.

This paper investigated an interesting work of applying PBILs, especially the dual PBIL, for dynamic optimization problems. There are several relevant works to be carried out in the future. First, extending the results in this paper to other Estimation of Distribution Algorithms (EDAs) [13], of which PBILs are a sub-class, is an interesting work. Second, it is also worthy to introduce and develop more approaches, such as the hypermutation technique [7], [12], from EA's community to PBILs or EDAs for dynamic optimization problems. Finally, formally analyzing the performance of investigated PBILs for dynamic optimization problems is also an important future work.

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. IEEE Press.

[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. Proc. of the 12th Int. Conf. on Machine Learning, 38-46.

[5] J. Branke, T. Kaubler, C. Schmidt, and H. Schmeck (2000). A Multi-Population Approach to Dynamic Optimization Problems. In Proc. of the 4th Int. Conf. on Adaptive Computing in Design and Manufacturing.

[6] J. Branke (2001). Evolutionary Approaches to Dynamic Optimization Problems - Updated Survey. GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems, 134-137.

[7] 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.

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

[9] 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.

[10] J. Lewis, E. Hart and G. Ritchie (1998). A Comparison of Dominance Mechanisms and Simple Mutation on Non-Stationary Problems. In A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds.), Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature, 139-148.

[11] 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.

[12] R. W. Morrison and K. A. De Jong (2000). Triggered Hypermutation Revisited. Proc. of the 1999 Congress on Evolutionary Computation, 1025-1032.

[13] H. Mühlenbein and G. Paaß (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.

[14] 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. Morgan Kaufmann Publishers.

[15] S. Yang (2003). Non-Stationary Problem Optimization Using the Primal-Dual Genetic Algorithm. Proc. of the 2003 Congress on Evolutionary Computation.