Soft Comput (2009) 13(8-9): 763–780
DOI 10.1007/s00500-008-0347-3

A Memetic Algorithm with Adaptive Hill Climbing Strategy for Dynamic Optimization Problems

Hongfeng Wang[1], **Dingwei Wang**[1], **Shengxiang Yang**[2]

[1] School of Information Science and Engineering, Northeastern University, Shenyang 110004, P. R. China
`{hfwang, dwwang}@mail.neu.edu.cn`
[2] Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, United Kingdom
`s.yang@mcs.le.ac.uk`

Published online: 5 August 2008

Abstract Dynamic optimization problems challenge traditional evolutionary algorithms seriously since they, once converged, cannot adapt quickly to environmental changes. This paper investigates the application of memetic algorithms, a class of hybrid evolutionary algorithms, for dynamic optimization problems. An adaptive hill climbing method is proposed as the local search technique in the framework of memetic algorithms, which combines the features of greedy crossover-based hill climbing and steepest mutation-based hill climbing. In order to address the convergence problem, two diversity maintaining methods, called adaptive dual mapping and triggered random immigrants respectively, are also introduced into the proposed memetic algorithm for dynamic optimization problems. Based on a series of dynamic problems generated from several stationary benchmark problems, experiments are carried out to investigate the performance of the proposed memetic algorithm in comparison with some peer evolutionary algorithms. The experimental results show the efficiency of the proposed memetic algorithm in dynamic environments.

Key words Genetic algorithm, memetic algorithm, local search, crossover-based hill climbing, mutation-based hill climbing, dual mapping, triggered random immigrants, dynamic optimization problems

1 Introduction

Many real-world optimization problems are dynamic optimization problems (DOPs), where the function landscapes may change over time and, thus, the optimum of these problems may also change over time. DOPs require powerful heuristics that account for the uncertainty present in the real world. Since evolutionary algorithms (EAs) draw their inspiration from the principles of natural evolution, which is a stochastic and dynamic process, they also seem to be suitable for DOPs. However, traditional EAs face a serious challenge for DOPs because they cannot adapt well to the changing environment once converged.

In order to address DOPs, many approaches have been developed [41] and can be grouped into four categories: 1) increasing population diversity after a change is detected, such as the adaptive mutation methods [4, 33]; 2) maintaining population diversity throughout the run, such as the immigrants approaches [11,39,40]; 3) memory approaches, including implicit [10,32] and explicit memory [2,35,38,43] methods; 4) multi-population [3,24] and speciation approaches [27]. A comprehensive survey on EAs applied to dynamic environments can be found in [14].

In recent years, there has been an increasing concern from the evolution computation community on a class of hybrid EAs, called memetic algorithms (MAs), which hybridize local search (LS) methods with EAs to refine the solution quality. So far, MAs have been widely used for solving many optimization problems, such as scheduling problems [13,19,20], combinatorial optimization problems [8,30,31], multi-objective problems [9,12,18] and other applications [44,45]. However, these problems for which MAs have been applied are mainly stationary problems. MAs have rarely been applied for DOPs [6,7,36]. During the running course of general MAs, they may always exhibit very strong exploitation capacity due to executing efficient local refinement on individuals, but they may lose the exploration capacity as a result of the population converging to one optimum, which needs to be avoided in dynamic environments. Therefore, it becomes an interesting research issue to examine the performance of MAs, which are enhanced by suitable diversity methods, for DOPs.

In this paper, we investigate the application of a MA with an adaptive hill climbing strategy, which combines the features of crossover-based hill climbing and mutation-based hill climbing in both cooperative and

```
Procedure General GA-based MA:
begin
  parameterize($pop\_size, pc, pm$);
  $t := 0$;
  initializePopulation($P(0)$);
  evaluatePopulation($P(0)$);
  $elite$ := selectForLocalSearch($P(0)$);
  localSearch($elite$);
  repeat
    $P'(t)$ := selectForReproduction($P(t)$);
    $P''(t)$ := crossover($P'(t)$);
    mutate($P''(t)$);
    evaluatePopulation($P''(t)$);
    $P(t+1)$ := selectForSurvival($P(t)$,$P''(t)$);
    $elite$ := selectForLocalSearch($P(t+1)$);
    localSearch($elite$);
    $t := t + 1$;
  until a stop condition is met
end;
```

Fig. 1 Pseudo-code for a general GA-based MA.

competitive fashions, to address DOPs. In order to address the convergence problem, two diversity maintaining approaches, an adaptive dual mapping and a triggered random immigrants scheme, are introduced into the proposed MA to improve its performance in dynamic environments. The effect of introducing these two diversity maintaining approaches into our MA for DOPs is experimentally investigated.

The rest of this paper is organized as follows. Section 2 describes the proposed MA with two hill climbing strategies investigated in this paper. Section 3 introduces a series of DOPs generated by a dynamic problem generator from the stationary test suite. Section 4 reports the experimental results and relevant analysis. Finally, Section 5 concludes this paper with some discussions on relevant future work.

2 Investigated Algorithms

2.1 Framework of GA-based Memetic Algorithms

The memetic algorithms investigated in this paper are a class of GA-based MAs, which can be expressed by the pseudo-code in Fig. 1, where *pop_size*, *pc*, and *pm* are the population size, crossover probability, and mutation probability respectively. Within these MAs, a population of *pop_ size* individuals are generated randomly and then evaluated at the initialization step. Then, an elite individual, i.e., an individual with the best fitness, is improved by a local search (LS) strategy. At each subsequent generation, individuals are selected randomly or proportionally from the current population and undergo the uniform crossover operation with a probability *pc*. Uniform crossover is the generalization of n-point crossover which creates offspring by deciding, for each bit of one parent, whether to swap the allele of that bit with the corresponding allele of the other parent. After crossover is executed, the bit-wise mutation operator is performed for each newly generated offspring individual, which may change the allele in each locus of an offspring bitwise (0 to 1 and vice versa) with a probability *pm*. Then, the *pop_size* best individuals among all parents and offspring are selected to proceed into the next generation and an elite individual in the newly generated population is improved by the LS strategy.

The convergence problem must be addressed when an EA or MA is applied for DOPs. Some diversity maintaining approaches, such as the dualism and immigrants methods, have proved to be good choices for EAs to address this problem. However, more economical diversity methods should be introduced to MAs for DOPs, given that the LS operations always cost a number of extra evaluations. Hence, two diversity maintaining approaches will be introduced to our MA, which utilizes the adaptive hill climbing strategy as the LS operator, to address DOPs in the next section.

2.2 Hill Climbing

Within MAs, LS operators perform directive local refinements to ensure sufficient exploitation during the course of evolving the population. Among many LS methods available in the literature, hill climbing (HC) is a common strategy. In the context of GAs, HC methods may be divided into two ways: crossover-based hill climbing [17,26] and mutation-based hill climbing [16,25]. The basic idea of HC methods is to use stochastic iterative hill climbing as the move acceptance criterion of the search (i.e. move the search from the current individual to a candidate individual if the candidate has a better fitness) and use crossover or mutation as the move operator in a local area.

Here, we propose two HC methods, a greedy crossover-based hill climbing (GCHC) and a steepest mutation-based hill climbing (SMHC), in this section. They are specially designed for MAs with binary encoding scheme, which are our concern in this paper. We will consider a class of binary encoded DOPs in our experiments to be discussed later. The two HC methods are described as follows.

1) GCHC: In this strategy, the elite individual is taken as one parent and another parent is selected from the current population using a roulette wheel. Then, a special uniform crossover is executed between these two parent individuals to generate an offspring. The offspring will replace the elite individual in the current population if it has a better fitness than the elite. This procedure is outlined in Fig. 2, where a maximization optimization problem is assumed.

2) SMHC: The steepest mutation means that the chromosome only changes several bits randomly when

```
Procedure GCHC(elite):
begin
  calculate($\xi, pc_{ls}$);
  for $i := 1$ to $ls\_size$ do
    $par\_chr$ := selectParentForCrossover($P$);
    for $j := 1$ to $n$ do
      if $random() < pc_{ls}$ then
        $chi\_chr[j] := par\_chr[j]$;
      else
        $chi\_chr[j] := elite[j]$;
    endfor
    evaluate($chi\_chr$);
    if $f(chi\_chr) > f(elite)$ then $elite := chi\_chr$;
  endfor
end;

GCHC's parameters:
  $\xi$: a population index used to renew the value of $pc_{ls}$
  $pc_{ls}$: the crossover probability in GCHC
  $ls\_size$: the step size of hill climbing
  $par\_chr$: the proportionally selected parent individual
  $P$: current population
  $n$: individual length (problem dependent)
  $random()$: a random number between 0 and 1
  $chi\_chr$: the new individual generated by performing a
    uniform crossover between $par\_chr$ and $elite$
  $elite$: the elite chromosome for hill climbing
```

Fig. 2 Pseudo-code for the GCHC operator.

```
Procedure SMHC(elite):
begin
  calculate($\xi, nm_{ls}$);
  for $i := 1$ to $ls\_size$ do
    for $j := 1$ to $n$ do
      $chi\_chr[j] := elite[j]$;
    endfor
    for $k := 1$ to $nm_{ls}$ do
      $loc := random(1, n)$;
      $chi\_chr[loc] := 1 - chi\_chr[loc]$;
    endfor
    evaluate($chi\_chr$);
    if $f(chi\_chr) > f(elite)$ then $elite := chi\_chr$;
  endfor
end;

SMHC's parameters:
  $nm_{ls}$: the number of bits mutated in SMHC
  $chi\_chr$: the new individual generated by performing
    steepest mutation upon the elite
  $loc$: a random selected location for flipping
  $random(1, n)$: a random integer between 1 and $n$
  Other parameters are the same as those for GCHC
```

Fig. 3 Pseudo-code for the SMHC operator.

executing one mutation operation on it. In SMHC, the elite individual is picked out from the current population and several random bits are changed. If the newly mutated individual has a better fitness, it will replace the elite individual. The SMHC strategy is outlined in Fig. 3.

From Fig. 2 and Fig. 3, it can be seen that two important parameters, pc_{ls} in GCHC and nm_{ls} in SMHC respectively, may affect the results of the local search. In GCHC the smaller the value of pc_{ls}, the more the offspring inherits from the elite individual. This means executing one step LS operation in a smaller area around *elite*. Similar results can be obtained for nm_{ls} in SMHC. When the value of nm_{ls} is larger, SMHC will perform the LS operation within a wider range around *elite*.

Therefore, the question that remains to be answered here is how to set the two parameters. Generally speaking, the methods of setting strategy parameters in GAs can be classified into three categories [5]: *deterministic mechanism* where the value of the strategy parameter is controlled by some deterministic rules without any feedback from the search, *adaptive mechanism* where there is some form of feedback information from the search process that is used to direct the setting of a strategy parameter, and *self-adaptive mechanism* where the parameter to be adapted is encoded into the chromosomes and undergoes genetic operators.

Two different parameter-setting methods will be discussed for pc_{ls} and nm_{ls} in the later experiments. In the *deterministic* method, both pc_{ls} and nm_{ls} are set to constant values, which means that the LS operation will always be executed in a local area of a certain fixed range. In the *adaptive* method, a population index ξ which can measure the diversity of the population is considered as the feedback information to direct the change of the values of pc_{ls} and nm_{ls}.

Let the normalized Hamming distance between two individuals $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$ and $\mathbf{x}_j = (x_{j1}, \ldots, x_{jn})$ be defined by:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{k=1}^{n} |x_{ik} - x_{jk}|}{n} \tag{1}$$

and ξ is calculated by the following formula:

$$\xi = \frac{\sum_{i=1}^{pop\_size} d(\mathbf{x}^*, \mathbf{x}_i)}{pop\_size}, \tag{2}$$

where $\mathbf{x}^*$ denotes the best individual achieved so far. Obviously, the index ξ can measure the convergence state of the population via the Hamming distance calculation. When ξ decreases to zero, it means that the population has lost its diversity absolutely.

With the definition of ξ, pc_{ls} and nm_{ls} can be calculated as follows:

$$pc_{ls} = min\{\alpha \cdot \xi \cdot (pc_{ls}^{max} - pc_{ls}^{min}) + pc_{ls}^{min}, pc_{ls}^{max}\} \tag{3}$$

$$nm_{ls} = min\{\beta \cdot \xi \cdot (nm_{ls}^{max} - nm_{ls}^{min}) + nm_{ls}^{min}, nm_{ls}^{max}\}, \quad (4)$$

where pc_{ls}^{max} and pc_{ls}^{min} are the preset maximum and minimum value of pc_{ls} respectively, nm_{ls}^{max} and nm_{ls}^{min} are the preset maximum and minimum value of nm_{ls} respectively, and α and β are the predefined constants to control the decreasing or increasing speed of pc_{ls} and nm_{ls} respectively. From these formulae, it is easy to understand that both GCHC and SMHC exhibit a wide range LS operations in the presence of a high population diversity (i.e., when $\xi \rightarrow 1$) as a result of $pc_{ls} \rightarrow pc_{ls}^{max}$ and $nm_{ls} \rightarrow nm_{ls}^{max}$. This may help algorithms find the optimum (maybe local optimum) more quickly. However, when the population is converging (i.e., when $\xi \rightarrow 0$), $pc_{ls} \rightarrow pc_{ls}^{min}$ and $nm_{ls} \rightarrow nm_{ls}^{min}$, which limits the LS operations in a very small range in order to perform more efficient local improvement for the elite individual.

2.3 Adaptive Hill Climbing (AHC)

It has been reported that multiple LS operators can be employed in a MA framework [21,28,29]. This is because each LS operator makes a biased search, which makes a method efficient for some classes of problems but not efficient for others. That is, LS is problem-dependent. Therefore, how to achieve improved LS operators and avoid utilizing inappropriate LS methods becomes a very important issue. In order to address this problem, many researchers have used multiple LS methods in their MAs. In comparison with traditional MAs that use a single LS operator throughout the run, MAs with multiple LS methods can usually obtain a better performance.

The key idea of using multiple LS operators in MAs is to promote the cooperation and competition of different LS operators, enabling them to work together to accomplish the shared optimization goal. Some researchers [15,23] have suggested that multiple LS operators should be executed simultaneously on those individuals that are selected for local improvements and that a certain learning mechanism should be adopted to give the efficient LS methods greater chances to be chosen in the later stage. However, Neri et al. [22] have also proposed a multiple LS based MA with a non-competitive scheme, where different LS methods can be activated during different population evolution periods. Inspired by these researches, an adaptive hill climbing (AHC) strategy that hybridizes the GCHC and SMHC methods described in Section 2.2 is proposed in this paper.

In AHC, the GCHC and SMHC operators are both allowed to work in the whole LS loop and are selected by probability to execute one step LS operation at every generation when the MA is running. Let p_{gchc} and p_{smhc} denote the probabilities of applying GCHC and SMHC to the individual that is used for a local search respectively, where $p_{gchc} + p_{smhc} = 1$. At the start of this strategy, p_{gchc} and p_{smhc} are both set to 0.5, which means

```
Procedure AHC($elite$):
begin
  if $p_{gchc}$ and $p_{smhc}$ are not initialized then
    set $p_{gchc} = p_{smhc} = 0.5$;
  calculate($\xi, pc_{ls}, nm_{ls}$);
  set $\eta_{gchc} = \eta_{smhc} = 0$;
  for $i := 0$ to $ls\_size - 1$ do
    if $random() < p_{gchc}$ then // GCHC is selected
      GCHC($elite$);
      update($\eta_{gchc}$);
    else // SMHC is selected
      SMHC($elite$);
      update($\eta_{smhc}$);
  endfor
  recalculate($p_{gchc}, p_{smhc}$);
end;

AHC's parameters are defined the same as those for
  GCHC and SMHC
```

Fig. 4 Pseudo-code for the AHC operator.

giving a fair competition chance to each LS operator. As each LS operator always makes a biased search, the LS operator which produces more improvements should be given a greater selection probability. Here, an adaptive learning approach is used to adjust the value of p_{gchc} and p_{smhc} for each LS operator. Let η denotes the improvement degree of the selected individual when one LS operator is used to refine it and η can be calculated by:

$$\eta = \frac{|f_{imp} - f_{ini}|}{f_{ini}}, \quad (5)$$

where f_{imp} is the final fitness of the elite individual after applying the local search and f_{ini} is its initial fitness before the local search. At each generation, the degree of improvement of each LS operator is calculated when a predefined number (ls_size) of iterations is achieved and then p_{gchc} and p_{smhc} are re-calculated to proceed with the local improvement in the next generation.

Suppose $\eta_{gchc}(t)$ and $\eta_{smhc}(t)$ respectively denote the total improvement of GCHC and SMHC at generation t. The LS selection probabilities $p_{gchc}(t+1)$ and $p_{smhc}(t+1)$ at generation $(t+1)$ can be calculated orderly by the following formulae:

$$p_{gchc}(t+1) = p_{gchc}(t) + \Delta \cdot \eta_{gchc}(t), \quad (6)$$

$$p_{smhc}(t+1) = p_{smhc}(t) + \Delta \cdot \eta_{smhc}(t), \quad (7)$$

$$p_{gchc}(t+1) = \frac{p_{gchc}(t+1)}{p_{gchc}(t+1) + p_{smhc}(t+1)}, \quad (8)$$

$$p_{smhc}(t+1) = 1 - p_{gchc}(t+1), \quad (9)$$

where Δ signifies the relative influence of the degree of the improvement on the selection probability. The AHC operator can be expressed by the pseudo-code in Fig. 4.

From the above discussion, the two different HC strategies, GCHC and SMHC, may not only cooperate to improve the quality of individuals, but also compete with each other to achieve a greater selection probability in the running process of AHC. To promote competition between them, the selection probability of LS operators can be re-calculated according to an adaptive learning mechanism where the LS operator with a higher fitness improvement is rewarded with more chance of being chosen for the subsequent individual refinement.

2.4 Population Diversity

So far, almost all MAs are used for solving stationary optimization problems, where the fitness landscape or objective function does not change during the course of computation. The LS operators are designed for exploiting information in the current population and the genetic operators, for example, mutation, are mostly responsible for enhancing the diversity of population in order to make an efficient jump from a local optimum. Generally speaking, the population will converge to a small area in the whole search space as a result of keeping the sufficient exploitation for the global optimum. Therefore, MAs may gradually loose their population diversity during the running. However, in dynamic environments, the fitness landscape may change over time. That is, the current optimum point may become a local optimum and the past local optimum may become a new global optimum point. Considering that a spread-out population can adapt to these changes more easily, it is very important and necessary to maintain a sufficient diversity of the population for MAs all the time.

Obviously, a simple mutation operator can not maintain sufficient population diversity in MAs since LS operators can make the population rapidly converge into an optimum. In order to address this converge problem, two diversity-maintaining methods, called *adaptive dual mapping* (ADM) and *triggered random immigrants* (TRI), are introduced into our algorithm framework of MAs, as shown in Fig. 5, for DOPs.

1) The ADM method: Dualism and complementarity are quite common in nature, such as double-stranded chromosome in DNA molecules. Inspired by the complementarity mechanism in nature, a primal-dual genetic algorithm has been proposed and applied for DOPs [37]. In this paper, we investigate the application of dualism [34,42] into MAs. For the convenience of description, we first introduce the definition of a dual individual here. Given an individual $\mathbf{x} = (x_1, \ldots, x_n) \in I = \{0,1\}^n$ of a fixed length n, its dual individual is defined as $\mathbf{x}' = (x'_1, \ldots, x'_n) \in I$ where $x'_i = 1 - x_i$ $(i = 1, \ldots, n)$. With this definition, the dual ($elite'$) of an individual ($elite$) is first evaluated before executing a LS on it. If its dual is evaluated to be fitter ($f(elite') > f(elite)$), $elite$ is replaced with $elite'$ before the local search is executed; Otherwise, $elite$ will be refined by LS directly.

```
**Procedure** Proposed GA-based MA:
**begin**
  parameterize($pop\_size, pc, pm$);
  $t := 0$;
  initializePopulation($P(0)$);
  evaluatePopulation($P(0)$);
  calculate($\xi, ls\_size, pc_{ls}, nm_{ls}$);
  $elite$ := selectForLocalSearch($P(0)$);
  **if** *ADM is used* **then**
    $elite' := dual(elite)$;
    evaluateDualIndividual($elite'$);
    **if** $f(elite') > f(elite)$ **then** $elite := elite'$;
  AHC($elite$);
  **repeat**
    $P'(t)$ := selectForReproduction($P(t)$);
    $P''(t)$ := crossover($P'(t)$);
    mutate($P''(t)$);
    evaluatePopulation($P''(t)$);
    $P(t+1)$ := selectForSurvival($P''(t)$, $P(t)$);
    calculate($\xi, ls\_size, pc_{ls}, nm_{ls}$);
    $elite$ := selectForLocalSearch($P(t+1)$);
    **if** *ADM is used* **then**
      $elite' := dual(elite)$;
      evaluateDualIndividual($elite'$);
      **if** $f(elite') > f(elite)$ **then** $elite := elite'$;
    AHC($elite$);
    **if** *TRI is used* **then**
      **if** $\xi < \theta_0$ **then**
        replace the worst $im\_size$ individuals in
          $P(t+1)$ with random immigrants;
    $t := t + 1$;
  **until** a stop condition is met
**end**;
```

Fig. 5 Pseudo-code for the proposed GA-based MA with diversity maintaining techniques.

2) The TRI method: The *random immigrants* approach was first introduced by Grefenstette [11] where in every generation the population is partly replaced by randomly generated individuals. Though this method introduces a constant diversity into the population, it is more helpful that the random individuals migrate into a converging population than a spread-out one. Thus, it is not necessary that the population is always injected by random individuals at every generation. Here, we introduce a *triggered random immigrants* method via combining a trigger mechanism with the random immigrants scheme. In the triggered method, the random individuals will be immigrated into the population only when its convergence degree is below a threshold. A triggered generator may be designed using the index ξ (see Section 2.2). When the value of ξ is less than a certain threshold θ_0, the *random immigrants* strategy will be triggered and im_size (here, $im\_size = 10\% \times pop\_size$) lowest fitness individuals in the population are replaced by the same amount of randomly generated ones.

Based on the above description, the ADM method can introduce a certain degree of diversity to the current population if the *elite* individual makes a long jump to its complement in the search space. Although just one individual in the current population is allowed to execute the dual mapping operation, this diversity method may affect the algorithm very explicitly. This is because the selected individual is the best fitness individual which plays an important role in the running of investigated MAs. The TRI method can bring a high degree of diversity when the current population has converged below a certain level. It is obvious that the TRI method may just make an implicit influence on the performance of algorithms as the fitness level of randomly generated individuals is usually very low.

One main problem that follows when the diversity methods are introduced into MA is how much they affect the LS operation. Just as introduced in Section 2.1, the number of evaluations must be considered in the framework of MAs for DOPs while the LS operation and diversity maintaining operation both cost a number of additional evaluations (ls_size in LS, one in ADM, and im_size in TRI) per generation of the MA running. As a generation index is used to set the change period of environment in the later experiments, it is necessary to maintain a constant number of evaluations in each generation in order to have fair comparisons among our investigated MAs and other peer EAs. Therefore, the LS step size ls_size will be re-calculated when ADM or TRI or both are used. Let Num_epg denote the number of evaluations per generation, $ls\_size = Num\_epg - pop\_size - 1 - im\_size$ if ADM and TRI techniques are both introduced into MA. In fact, it is why we use the ADM and TRI methods in our investigated MAs, with a view to decrease the evaluations of useless diversity maintaining operations as much as possible.

3 Dynamic Test Environments

In this paper, a series of dynamic test environments are constructed by a specific dynamic problem generator from a set of well studied stationary problems. Four 100-bit binary-coded functions, denoted OneMax, Plateau, RoyalRoad, and Deceptive respectively, are selected as the stationary functions to construct dynamic test environments. Each stationary function consists of 25 copies of 4-bit building blocks and has an optimum value of 100. Each building block for the four functions is a unitation-based function, as shown in Fig. 6. The unitation function of a bit string returns the number of ones inside the string. The building block for OneMax is an OneMax subfunction, which aims to maximize the number of ones in a bit string. The building block for Plateau contributes 4 (or 2) to the total fitness if its unitation is 4 (or 3); otherwise, it contributes 0. The building block for RoyalRoad contributes 4 to the total fitness if all

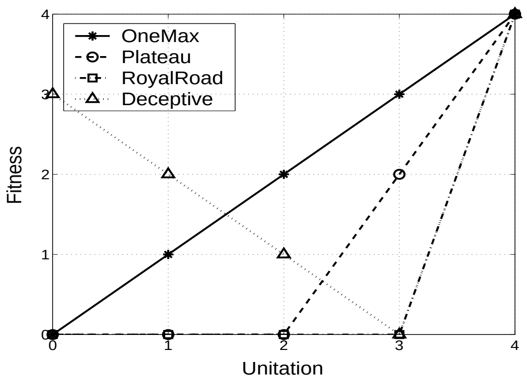

Fig. 6 The building blocks for the four stationary functions selected to construct dynamic test problems in this paper.

its four bits are set to one; otherwise, it contributes 0. The building block for Deceptive is a fully deceptive subfunction. Generally speaking, the four functions have an increasing difficulty for GAs in the order from OneMax to Plateau, RoyalRoad to Deceptive.

In [37,42], a DOP generator was proposed. The DOP generator can generate dynamic environments from any binary-encoded stationary function $f(\mathbf{x})$ ($\mathbf{x} \in \{0,1\}^l$) by a bitwise exclusive-or (XOR) operator. The environment is changed every τ generations. For each environmental period k, an XOR mask $\mathbf{M}(k)$ is incrementally generated as follows:

$$\mathbf{M}(k) = \mathbf{M}(k-1) \oplus \mathbf{T}(k), \tag{10}$$

where "$\oplus$" is the XOR operator (i.e., $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$) and $\mathbf{T}(k)$ is an intermediate binary template randomly created with $\rho \times l$ ones for the k-th environmental period. For the first period $k = 1$, $\mathbf{M}(1) = \mathbf{0}$. Then, the population at generation t is evaluated as:

$$f(\mathbf{x}, t) = f(\mathbf{x} \oplus \mathbf{M}(k)), \tag{11}$$

where $k = \lceil t/\tau \rceil$ is the environmental index. One advantage of this XOR generator lies in that the speed and severity of environmental changes can be easily tuned. The parameter τ controls the speed of changes while $\rho \in (0.0, 1.0)$ controls the severity of changes. A bigger ρ means more severe changes while a smaller τ means more frequent changes.

The dynamic test environments used in this paper are constructed from the four stationary functions using the aforementioned XOR DOP generator. The change severity ρ parameter is set to 0.1, 0.2, 0.5, and 0.9 respectively in order to examine the performance of algorithms in dynamic environments with different severities: from slight change ($\rho = 0.1$ or 0.2) to moderate variation ($\rho = 0.5$) to intense change ($\rho = 0.9$). The change speed parameter τ is set to 10, 50, and 100 respectively, which means that the environment changes very fast, in the moderate speed, and slowly respectively.

Table 1 The index table for dynamic parameter settings

τ	Environmental Dynamics Index			
10	1	2	3	4
50	5	6	7	8
100	9	10	11	12
$\rho \rightarrow$	0.1	0.2	0.5	0.9

In total, a series of 12 different dynamic problems are constructed from each stationary test problem. The dynamics parameter settings are summarized in Table 1.

4 Experimental Study

4.1 Experimental Design

In this section, experiments are carried out in order to study the major features of our proposed MAs and to compare their performance with several existing peer algorithms where similar dualism and immigrants methods are also used. The following abbreviations represent GAs or MAs considered in this paper:

- CHMA: MA with the GCHC operator;
- MHMA: MA with the SMHC operator;
- AHMA: MA with the AHC operator;
- SGA: Standard GA;
- SGAr: SGA with restart from scratch whenever the environment changes;
- RIGA: GA with the random immigrants scheme;
- EIGA: GA with the elitism-based immigrants scheme [39]. In EIGA, a set of immigrant individuals are generated by bitwise mutating the elitist (the best individual) from the previous generation to replace the worst individuals in the population at each generation;
- DPBIL3: The population-based incremental learning (PBIL) algorithm is a combination of evolutionary computation and competitive learning [1]. At each generation PBIL first generates a population of samples (solutions) according to a real valued probability vector and then retrieves the best sample generated to update (learn) the probability vector. With the progress of such iterations, each element in the probability vector will eventually converge to either 0.0 or 1.0 and PBIL can always achieve high quality solution with a high probability. In order to improve the performance of PBIL in dynamic environments, a PBIL variant, denoted DPBIL3, was investigated in [42]. DPBIL3 integrates the dualism and immigrants approaches. Instead of using only one probability vector as in the standard PBIL, DPBIL3 uses three different probability vectors, a pair of probability vectors that are dual to each other and one central probability vector. The three probability vectors generate their own sets of samples respectively and the number of samples they generate changes adaptively according to their relative performance. More details on DPBIL3 can be found in [42].

The following parameters are used in all algorithms: the total number of evaluations per generation Num_{epg} is always set to 120 for all algorithms, and the population size (*pop_size*) is set to 100 for all MAs, RIGA and EIGA, but is set to 120 for SGA, SGAr and DPBIL3 because the LS operation in MAs may be executed $ls\_size = 20$ steps at most and the immigrant ratio is set to 0.2 in RIGA and EIGA per generation. The uniform crossover probability pc equals to 0.6 and the bit-wise mutation probability pm is set to 0.01 for all GAs and MAs. The specific parameters in our MAs are set as follows: $\alpha = \beta = 1$, $\Delta = 4$ and $\theta_0 = 0.1$. Other parameters in the studied peer algorithms are always the same as their original settings.

For each experiment of an algorithm on a test problem, 20 independent runs were executed with the same set of random seeds. For each run of an algorithm on a DOP, 10 environmental changes were allowed and the best-of-generation fitness was recorded per generation.

The overall offline performance of an algorithm is defined as the best-of-generation fitness averaged across the number of total runs and then averaged over the data gathering period, as formulated below:

$$\overline{F}_{BG} = \frac{1}{G}\sum_{i=1}^{G}(\frac{1}{N}\sum_{j=1}^{N}F_{BG_{ij}}), \tag{12}$$

where G is the number of generations (i.e., $G = 10 * \tau$), $N = 20$ is the total number of runs, and $F_{BG_{ij}}$ is the best-of-generation fitness of generation i of run j.

In order to measure the behavior of an algorithm during the course of running, another numeric measure is defined as the best-of-generation fitness averaged across the number of total runs and then averaged from the last change generation τ' to the current generation t. More formally, the running offline performance is defined as:

$$\overline{F}_{BG_t} = \frac{1}{t-\tau'}\sum_{i=\tau'}^{t-\tau'}(\frac{1}{N}\sum_{j=1}^{N}F_{BG_{ij}}) \tag{13}$$

4.2 Experimental Study on the Effect of LS Operators

In the experimental study on LS operators, we first study the influence of different settings of pc_{ls} in CHMA and nm_{ls} in MHMA, with the aim of determining a robust setting for these two parameters. In particular, we have implemented CHMA that hybridizes the ADM and TRI schemes just on stationary test problems. Three different settings for pc_{ls} were used: $pc_{ls} = 0.6$ and $pc_{ls} = 0.1$ in the *deterministic* setting and $pc_{ls}^{max} = 0.6$ and $pc_{ls}^{min} = 0.1$ in the *adaptive* setting scheme (see Section 2.2). For each run of an algorithm on each problem, the maximum allowable number of generations was set

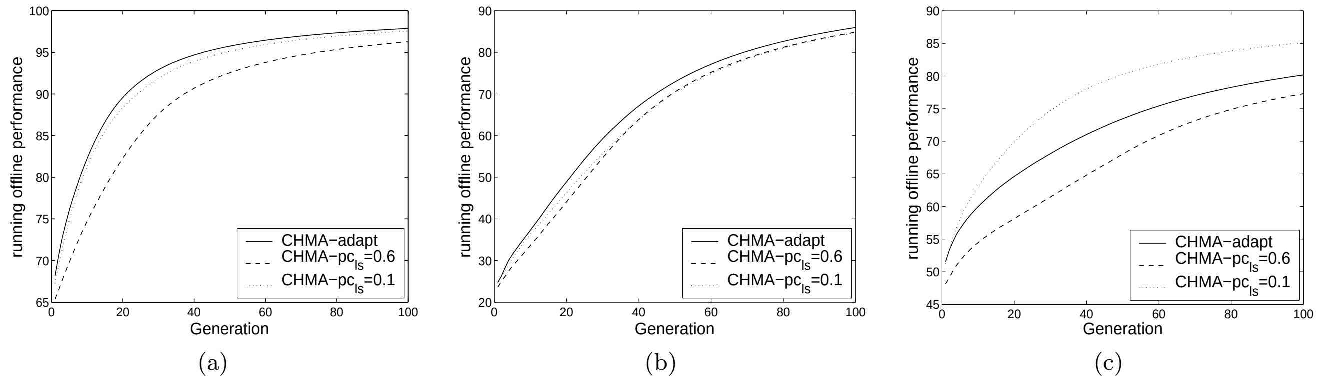

Fig. 7 Experimental results with respect to the running offline performance of CHMAs with different pc_{ls} settings on stationary test problems: (a) OneMax, (c) RoyalRoad, and (d) Deceptive.

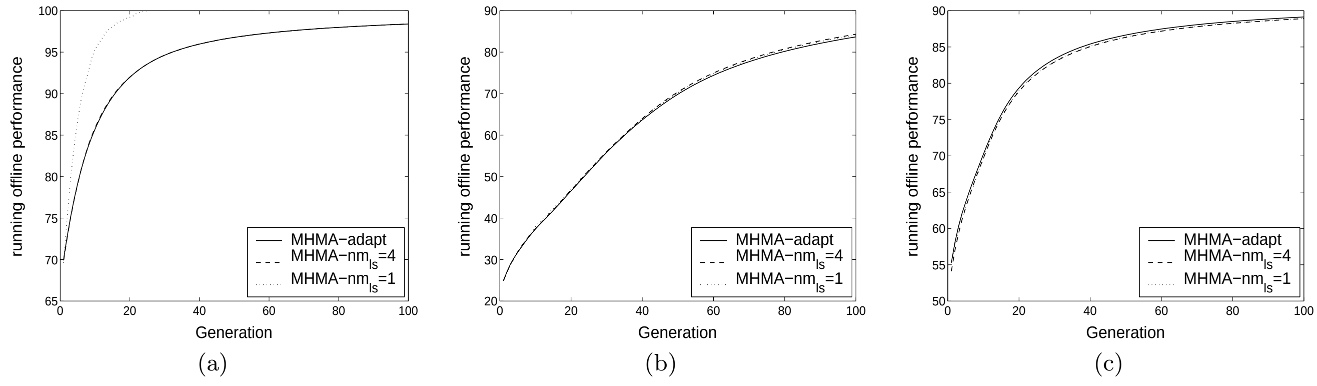

Fig. 8 Experimental results with respect to the running offline performance of MHMAs with different nm_{ls} settings on stationary test problems: (a) OneMax, (b) RoyalRoad, and (c) Deceptive.

to 100[1]. The experimental results are shown in Fig. 7, where the data were averaged over 20 runs. The results on the Plateau problem are similar to the results on the RoyalRoad problem and are not shown in Fig. 7.

From Fig. 7, it can be seen that CHMA with *adaptive* pc_{ls} always outperforms CHMAs with the *deterministic* value of pc_{ls} on the OneMax, Plateau and RoyalRoad problems and that a smaller pc_{ls} can help CHMA obtain a better performance on the Deceptive problem. So the *adaptive* setting scheme for pc_{ls} is always used in the following experiments considering that the *deterministic* setting scheme is problem-dependent and the *adaptive* scheme for pc_{ls} always shows a better adaptive capacity on different problems.

Similar experiments were also carried out to test the influence of different settings of nm_{ls} on the performance of MHMA, where the ADM and TRI methods are integrated. The value of nm_{ls} was set to 4 and 1 respectively for the *deterministic* scheme and $nm_{ls}^{max} = 4$ and $nm_{ls}^{min} = 1$ in the *adaptive* setting scheme (see Section 2.2). The experimental results with respect to the running offline performance are presented in Fig. 8.

From Fig. 8, it can be observed that the performance curves of the three MHMAs almost overlap together on the Plateau, RoyalRoad and Deceptive problems except that MHMA with $nm_{ls} = 1$ performs better than MHMA with adaptive nm_{ls} and MHMA with $nm_{ls} = 4$ on the OneMax problem. This indicates that adaptively varying the search range of the SMHC operator may not improve the performance of MHMA remarkably. Hence, the value of nm_{ls} will always be set to 1 in the later experiments.

In the following experiments, we investigate the performance of AHMA, MHMA, CHMA and SGA on the stationary test problems in order to examine the validity of LS operators. The two diversity maintaining methods (ADM and TRI) are both used in all MAs and experimental results with respect to the running offline performance are shown in Fig. 9.

From Fig. 9, it can be seen that all MAs always outperform SGA on all test problems. This shows that the combination of proper LS techniques (here AHC in AHMA, SMHC in MHMA, and GCHC in CHMA) and some diversity methods (here, ADM and TRI) can help MAs obtain a much better performance than SGA.

[1] The number of maximum allowable fitness evaluations is actually 12000 since each algorithm has 120 fitness evaluations per generation.

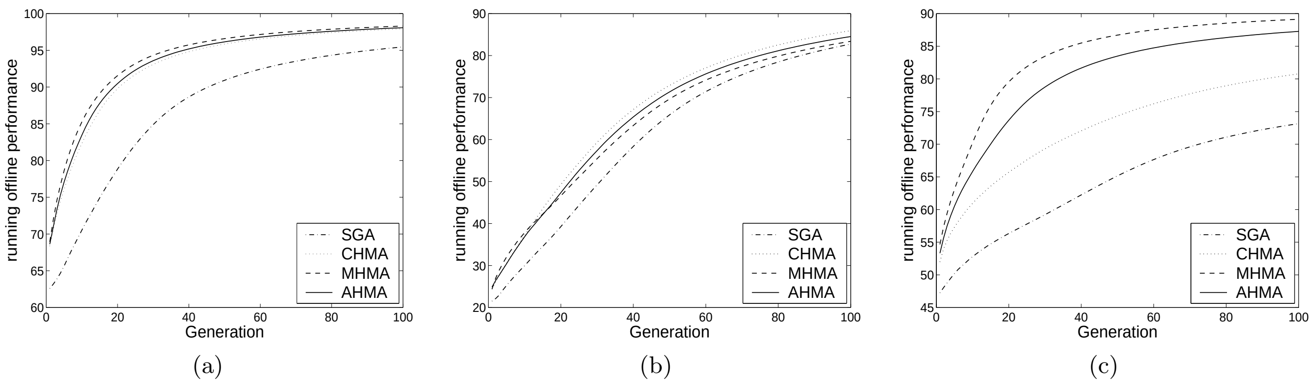

Fig. 9 Experimental results with respect to the running offline performance of MAs and SGA on stationary test problems: (a) OneMax, (b) RoyalRoad, and (c) Deceptive.

Table 2 Experimental results with respect to overall offline performance of AHMAs on dynamic test problems

Dynamics		OneMax Problem				Plateau Problem			
τ	ρ	AHMA1	AHMA2	AHMA3	AHMA4	AHMA1	AHMA2	AHMA3	AHMA4
100	0.1	98.97±0.06	99.28±0.17	99.26±0.18	99.02±0.06	97.49±0.26	98.14±0.28	98.11±0.20	97.60±0.29
100	0.2	99.07±0.07	98.77±0.42	98.72±0.39	98.07±0.10	96.66±0.46	96.19±0.35	96.12±0.32	95.53±0.53
100	0.5	97.80±0.08	97.57±0.07	97.42±0.09	97.87±0.09	89.69±1.06	84.46±0.55	84.53±0.39	90.34±0.98
100	0.9	99.23±0.07	95.89±0.11	99.25±0.06	97.86±0.11	98.03±0.22	62.88±1.71	98.03±0.22	90.54±0.64
Dynamics		RoyalRoad Problem				Deceptive Function			
τ	ρ	AHMA1	AHMA2	AHMA3	AHMA4	AHMA1	AHMA2	AHMA3	AHMA4
100	0.1	93.48±0.82	94.33±0.68	94.19±0.61	93.86±0.78	85.22±1.32	79.50±1.57	88.06±0.44	79.49±2.11
100	0.2	87.79±1.21	88.03±1.10	88.07±0.85	87.06±0.92	89.17±0.47	77.82±1.12	88.19±0.44	78.24±1.40
100	0.5	71.23±1.59	67.34±0.88	67.13±1.10	70.62±1.23	80.18±0.68	77.86±0.55	88.64±0.51	79.26±0.66
100	0.9	94.42±0.51	52.52±1.27	94.08±0.62	57.31±3.12	87.75±1.23	85.75±0.27	88.90±1.09	85.54±0.29

Of course, these conclusions have been drawn by many researchers. On the OneMax problem, AHMA always performs better than CHMA but is always beaten by MHMA with a lower degree. On the Plateau and RoyalRoad problems, CHMA outperforms MHMA after a period of early running and AHMA behaves similar as CHMA but with a little inferior performance. On the Deceptive problem, MHMA always performs better than AHMA and CHMA significantly while AHMA always performs better than CHMA with a much high degree.

The results indicate that LS operators are problem-dependent and AHC always does well although it needs to take some time to adjust its local search strategy. Since it is almost impossible for an algorithm to achieve all the characters of a problem in advance, the combination of multiple LS operators within a single MA framework is a good choice for solving optimization problems.

4.3 Experimental Study on the Effect of Diversity Maintaining Schemes

There are two diversity maintaining schemes, ADM and TRI, within the investigated MAs. In the above experiments, we used both of them in MAs. In order to investigate the effect of different diversity methods on the performance of MAs, we further carry out experiments on AHMAs on DOPs with $\tau = 100$ and ρ set to 0.1, 0.2, 0.5 and 0.9 respectively. In order to make a convenient description of the experiments, AHMA1, AHMA2, AHMA3 and AHMA4 are used to denote AHMA with both the ADM and TRI methods, AHMA without any diversity scheme, AHMA with only the ADM scheme, and AHMA with only the TRI method respectively.

The experimental results with respect to the overall offline performance are presented in Table 2. The corresponding statistical results of comparing algorithms by the one-tailed t-test with 38 degrees of freedom at a 0.05 level of significance are given in Table 3. In Table 3, the t-test results regarding Alg. 1−Alg. 2 are shown as "+", "−", or "∼" when Alg. 1 is significantly better than, significantly worse than, or statistically equivalent to Alg. 2 respectively. From Table 2 and Table 3, several results can be observed and are analyzed below.

First, AHMA1 always performs a little worse than other AHMAs on most dynamic problems just except on the Deceptive problem when the change severity ρ is very small ($\rho = 0.1$). This is because a new environment is close to the previous one when the value of ρ is very small. For such instances, executing sufficient LS operations for the elite individual in the current population

Table 3 The t-test results of comparing the overall offline performance of AHMAs on dynamic test problems

t-test Result	OneMax Problem				Plateau Problem				RoyalRoad Problem				Deceptive Function			
$\tau = 100$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA1 − AHMA2	−	+	+	+	−	+	+	+	−	∼	+	+	+	+	+	+
AHMA1 − AHMA3	−	+	+	∼	−	+	+	∼	−	∼	+	+	−	+	−	−
AHMA1 − AHMA4	−	+	−	+	∼	+	−	+	∼	+	∼	+	+	+	+	+
AHMA2 − AHMA3	∼	∼	+	−	∼	∼	∼	−	∼	∼	∼	−	−	−	−	−
AHMA2 − AHMA4	+	+	−	−	+	+	−	−	+	+	−	−	∼	∼	−	+
AHMA3 − AHMA4	+	+	−	+	+	+	−	+	+	+	−	+	+	+	+	+

may be more beneficial than introducing some population diversity. As AHMA1 requires more "energy" than other AHMAs in maintaining the population diversity, it is beaten when the value of ρ is very small. However, AHMA1 begins to exhibit a better performance with the increasing of the value of ρ. When $\rho = 0.2$, AHMA1 always outperforms other AHMAs on most dynamic problems. When ρ increases to 0.5 or 0.9, AHMA1 also does well except being beaten by AHMA3 on the Deceptive problems and by AHMA4 on the OneMax and Plateau problems with $\rho = 0.5$. Obviously, these results confirm our expectation of introducing diversity maintaining methods into AHMA.

Second, AHMA2 performs better just when the change severity ρ is not very large because its converging population can only adapt to such changes. However, the situation seems a little different on the Deceptive problem where AHMA2 performs worse than other AHMAs when ρ is small but performs a little better than AHMA4 when $\rho = 0.9$. The reason lies in that the deceptive attractor in the Deceptive problem may mislead the direction of hill climbing in AHMA2, which may be escaped from by other AHMAs via the diversity maintaining technique. When the environment is subject to significant changes ($\rho = 0.9$), the XOR operation may enable AHMA2 jump out from the deceptive attractor.

Third, AHMA3 always performs better than AHMA2 and AHMA4 on the OneMax, Plateau and RoyalRoad problems when $\rho = 0.9$ and on all Deceptive problems. This is because the dualism mechanism may help AHMA3 react to significant environmental changes rapidly and also enable it to escape from the deceptive attractor in the Deceptive problem.

Fourth, AHMA4 always exhibits a better performance on most dynamic problems when a random environmental change occurs ($\rho = 0.5$). This is easy to understand. When the environment changes with $\rho = 0.5$, almost all building blocks found so far are demolished. Obviously, AHMA4 can adapt to this environmental change more easily as the TRI method ensures AHMA4 to always maintain a certain population diversity level.

In order to understand the effect of the two diversity schemes on the population diversity during the running of an algorithm, we also recorded the diversity of the population every generation. The diversity of the population at generation t in the k-th run of an MA on a DOP is defined as:

$$Div(k,t) = \frac{\sum_{i=0}^{pop\_size} \sum_{j \neq i}^{pop\_size} d(\mathbf{x}_i, \mathbf{x}_j)}{n \cdot pop\_size(pop\_size - 1)}, \qquad (14)$$

where n is the encoding size and $d(\mathbf{x}_i, \mathbf{x}_j)$ is the normalized Hamming distance between the i-th ($\mathbf{x}_i$) and j-th ($\mathbf{x}_j$) individuals in the population. The overall diversity level of a MA on a DOP over 20 runs is calculated as follows.

$$\overline{Div(k,t)} = \frac{1}{G} \sum_{t=1}^{G} \frac{1}{20} \sum_{k=1}^{20} Div(k,t), \qquad (15)$$

where $G = 10 \times \tau = 500$ is the total number of generations for a run.

The overall diversity of MAs on DOPs with $\tau = 50$ and different values of ρ is plotted in Fig. 10. Form Fig. 10, it can be seen that AHMA4 maintains the highest diversity level on most DOPs since the random immigrants may be introduced into the population once converged. AHMA3 maintain a higher diversity level than AHMA2 only on dynamic Plateau and RoyalRoad problems with $\rho = 0.9$ and all dynamic Deceptive problems. This is because whether its population diversity is improved depends on whether the elite individual makes a long jump to its dual successfully. However, whether the diversity schemes are helpful or not depends on the MAs and DOPs. As analyzed before, AHMA1 outperforms other AHMAs on most dynamic problems though it just maintains a middle diversity level, while AHMA2 performs well only in the slight changing environments because of its poor population diversity.

4.4 Experimental Study on Comparing AHMA with Several Peer EAs on DOPs

In the final experiments, we compare the performance of AHMA, combining ADM and TRI methods, with several other existing peer EAs proposed in the literature on the DOPs constructed in Section 3. These peer EAs are SGAr, RIGA, EIGA and DPBIL3, as described in Section 4.1. The experimental results are plotted in Fig. 11

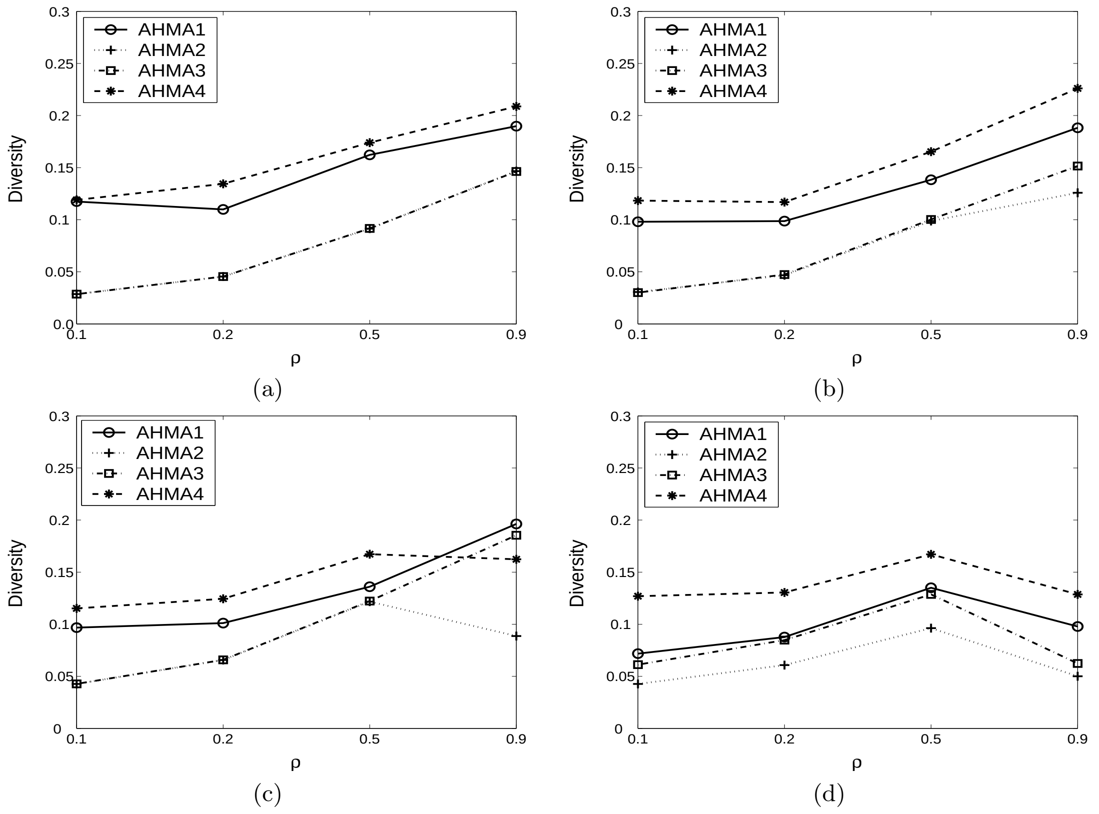

Fig. 10 Experimental results with respect to the diversity of population of AHMAs on DOPs with $\tau = 50$: (a) OneMax, (b) Plateau, (c) RoyalRoad, and (d) Deceptive.

and the corresponding statistical results are given in Table 4. From Fig. 11 and Table 4, several results can be observed and are analyzed as follows.

First, AHMA always outperforms other peer EAs on most dynamic problems and underperforms some of these EAs on some dynamic problems when the environment changes slowly, i.e., when $\tau = 50$ or 100. When the environment changes quickly, i.e., when $\tau = 10$, AHMA can always locate the optimum (maybe local optimum) more quickly than other EAs because the LS operator may have a strong exploitation capacity. This is why AHMA performs best on all dynamic problems with $\tau = 10$. When $\tau = 50$ or 100, AHMA performs a little worse than EIGA on dynamic OneMax problems with $\rho = 0.1$ or 0.2. This is because EIGA can especially fit such a dynamic environment that changes slowly and slightly for one thing and the elitism-based immigrants can maintain a very high fitness level on the OneMax problem for the other. AHMA is also beaten by SGAr on the dynamic Plateau and RoyalRoad problems with $\rho = 0.5$ and $\tau = 50$ or 100. This happens because the random environment always requires algorithms to maintain a sufficient population diversity (see the relevant analysis in Section 4.3) and the restart scheme in SGAr can introduce the maximum diversity into the population. The reason why SGAr outperforms AHMA only on the Plateau and RoyalRoad problems lies in the intrinsic characteristics of these problems. The OneMax problem is simply unimodal, which is very suitable for a HC search in AHMA. Both the Plateau and RoyalRoad problems have higher-order building blocks, which take a HC search much more time to achieve. The Deceptive problem may mislead SGAr's evolution due to the existence of deceptive attractor, which can be escaped from by AHMA. The good performance of AHMA over other peer EAs shows that our investigated AHMA has a strong robustness and adaptivity in dynamic environments.

Second, on dynamic OneMax and Plateau problems EIGA always outperforms SGAr and RIGA when ρ is set to 0.1 or 0.2, but underperforms them when the value of ρ is set to 0.5 or 0.9. On dynamic RoyalRoad and Deceptive problems, the situations become a little different. EIGA performs better than RIGA on dynamic RoyalRoad problems just when $\tau = 10$ and better than both SGAr and RIGA on all dynamic Deceptive problems. This happens because the elitism-based immigrants scheme can introduce higher fitness individuals, which can adapt better to the current environment, into EIGA's population on dynamic OneMax and Plateau problems when the environment changes slightly, on dynamic RoyalRoad problems when the environment changes quickly, and on all dynamic Decep-

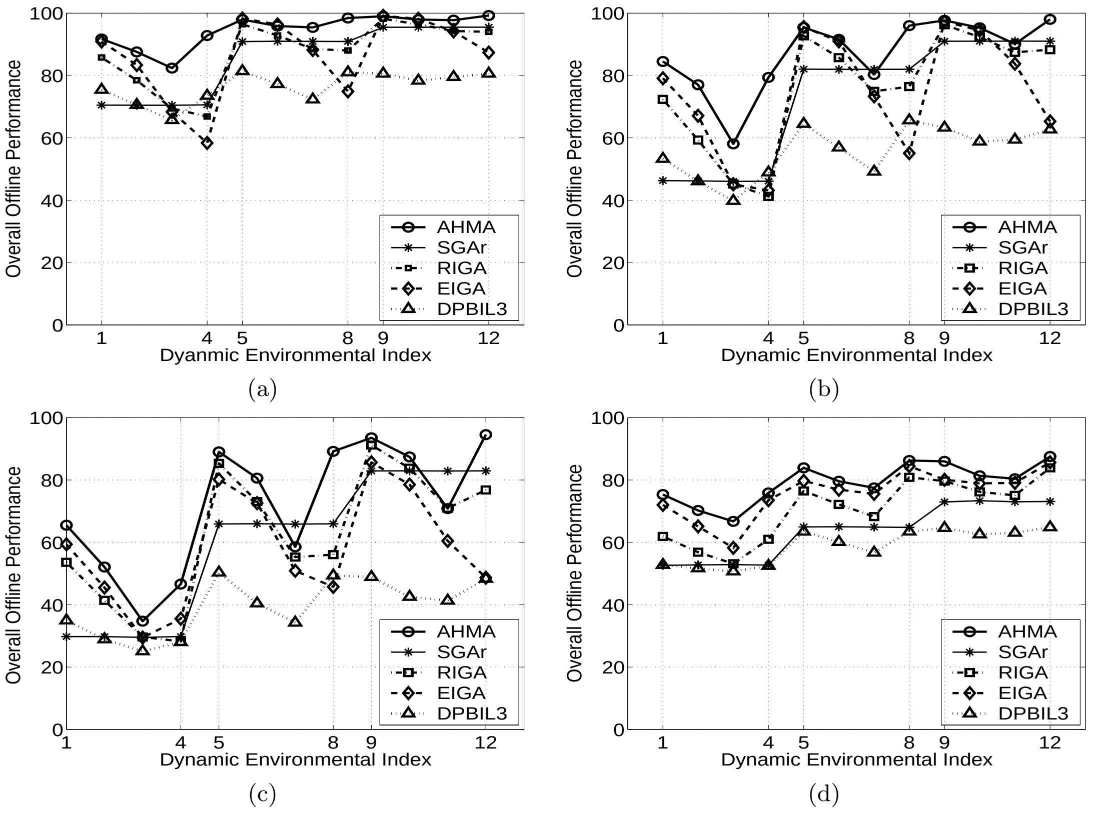

Fig. 11 Experimental results with respect to the overall offline performance of AHMA and peer EAs on dynamic test problems: (a) OneMax, (b) Plateau, (c) RoyalRoad, and (d) Deceptive.

tive problems due to the intrinsic characteristics of these four kinds of functions.

Third, the performance of DPBIL3 is exciting only when the environment is subject to significant changes. This also confirms the expectation of the dualism scheme for our algorithms in dynamic environments. Of course, the similar results have been obtained and relevant analysis were also given in the literature [42]. However, DPBIL3 performs worse than other peer EAs on most other DOPs. The reason will be explained in the following experimental analysis.

Fourth, the performance of other peer EAs is different on different dynamic problems. Generally speaking, RIGA always performs better than SGAr on most dynamic problems when the value of ρ is small. The performance of SGAr increases with the value of τ but does not change with the value of ρ. Similar results have also been observed in [39].

Finally, the environmental parameters affect the performance of algorithms. The performance of all algorithms increases when the value of τ increase from 10 to 50 to 100. It is easy to understand when τ becomes larger, algorithms have more time to find better solutions before the next change. The effect of the changing severity parameter ρ is different. For example, when τ is fixed, the performance curve of AHMA always declines when ρ increases from 0.1 to 0.2 to 0.5, but rises when ρ increases from 0.5 to 0.9.

In order to better understand the experimental results, we make a deeper look into the dynamic behavior of these algorithms. The dynamic behavior of different algorithms with respect to the running offline performance is shown in Fig. 12 to Fig. 15, where τ is set to 50 and ρ is set to 0.1, 0.2, 0.5 and 0.9 respectively. From these figures, it can be easily observed that for the dynamic periods SGAr always performs almost the same as it did for the stationary period (the first 50 generations) and AHMA always outperforms other peer EAs for the stationary period on all test problems while their dynamic behaviors are different on different dynamic problems.

On the OneMax problem (see Fig. 12), the dynamic behavior of AHMA for each dynamic period is almost the same as that for the stationary period when ρ is not very large. When ρ increases to 0.9, AHMA performs better for the dynamic periods than it does for the stationary period. This is because that on the OneMax problem the LS operator can help AHMA trace the changing optimum quickly during one change period of environment while LS's effect is enhanced greatly by the ADM operation when $\rho = 0.9$. The dynamic behavior of both RIGA and EIGA is affected by the value of ρ. With the increment of dynamic periods, their perfor-

Table 4 The *t*-test results of comparing the overall offline performance of AHMA and peer EAs on dynamic test problems

t-test Result	OneMax Problem				Plateau Problem				RoyalRoad Problem				Deceptive Function			
$\tau = 10$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA − SGAr	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AHMA − RIGA	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AHMA − EIGA	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AHMA − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
SGAr − RIGA	−	−	+	+	−	−	+	+	−	−	∼	+	−	−	−	−
SGAr − EIGA	−	−	+	+	−	−	+	+	−	−	∼	−	−	−	−	−
SGAr − DPBIL3	−	∼	+	−	−	∼	+	−	−	+	+	+	∼	+	+	∼
RIGA − EIGA	−	−	+	+	−	−	∼	−	−	−	∼	−	−	−	−	−
EIGA − DPBIL3	+	+	+	−	+	+	+	−	+	+	+	+	+	+	+	+
$\tau = 50$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA − SGAr	+	+	+	+	+	+	−	+	+	+	−	+	+	+	+	+
AHMA − RIGA	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AHMA − EIGA	−	−	+	+	∼	+	+	+	+	+	+	+	+	+	+	+
AHMA − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
SGAr − RIGA	−	−	+	+	−	−	+	+	−	−	+	+	−	−	−	−
SGAr − EIGA	−	−	+	+	−	−	+	+	−	−	+	+	−	−	−	−
SGAr − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
RIGA − EIGA	−	−	+	+	−	−	+	+	+	∼	+	+	−	−	−	−
EIGA − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
$\tau = 100$, $\rho \Rightarrow$	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9	0.1	0.2	0.5	0.9
AHMA − SGAr	+	+	+	+	+	+	−	+	+	+	−	+	+	+	+	+
AHMA − RIGA	+	+	+	+	+	+	+	+	+	+	∼	+	+	+	+	+
AHMA − EIGA	−	−	+	+	+	+	+	+	+	+	+	+	+	+	+	+
AHMA − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
SGAr − RIGA	−	−	+	+	−	−	+	+	−	−	+	+	−	−	−	−
SGAr − EIGA	−	−	+	+	−	−	+	+	−	+	+	+	−	−	−	−
SGAr − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
RIGA − EIGA	−	−	+	+	−	−	+	+	+	+	+	+	−	∼	−	−
EIGA − DPBIL3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

mance upgrades consistently when $\rho = 0.1$, while their behavior for the dynamic periods underperforms that for stationary period when $\rho = 0.5$ or 0.9. For DPBIL3, its performance curve rises continuously on the first several environmental periods but drops heavier and heavier in the later environmental periods when the value of ρ is set to 0.1, 0.2 or 0.9. The reason lies in the convergence problem of probability vectors in DPBIL3. When the environment changes slightly or significantly, a pair of dual probability vectors in DPBIL3 can always keep its evolutionary process to achieve a high fitness solution. However, this pair of probability vectors may converge during this course. Once converged, DPBIL3 can not adapt well to the changing environment. This is the reason why DPBIL3 is beaten by other peer EAs on most DOPs.

On the Plateau and RoyalRoad problems (see Figs. 13 and 14), with the increment of dynamic periods, AHMA's performance drops a little when $\rho = 0.5$, while rises when $\rho = 0.1$, 0.2 and 0.9. The reason is that AHMA does not find the optimum in the stationary period on these two problems. When the environment changes slightly or very significantly, AHMA always reruns from the starting points with a higher fitness in the dynamic periods than that in the stationary period, while when $\rho = 0.5$, AHMA can only obtain worse starting points in the dynamic periods. The dynamic behaviors of RIGA, EIGA and DPBIL3 on these problems are similar to that on the OneMax problem.

On the Deceptive problem (See Fig. 15), with the increment of dynamic periods, AHMA's performance maintains the same when $\rho = 0.1$ or drops a little when $\rho = 0.2$ or 0.5. When ρ is set to 0.9, AHMA's dynamic behavior is sort of switching between odd and even environmental periods. The reason is that after the stationary period for the following odd period the environment is in fact greatly returned or repeated from previous odd period given $\rho = 0.9$.

5 Conclusions and Future Work

In this paper, the application of memetic algorithms with an adaptive hill climbing (AHC) strategy for dynamic optimization problems is investigated. In the proposed memetic algorithm, two local search methods, the greedy crossover-based hill climbing (GCHC) and the steepest mutation-based hill climbing (SMHC), are used to refine

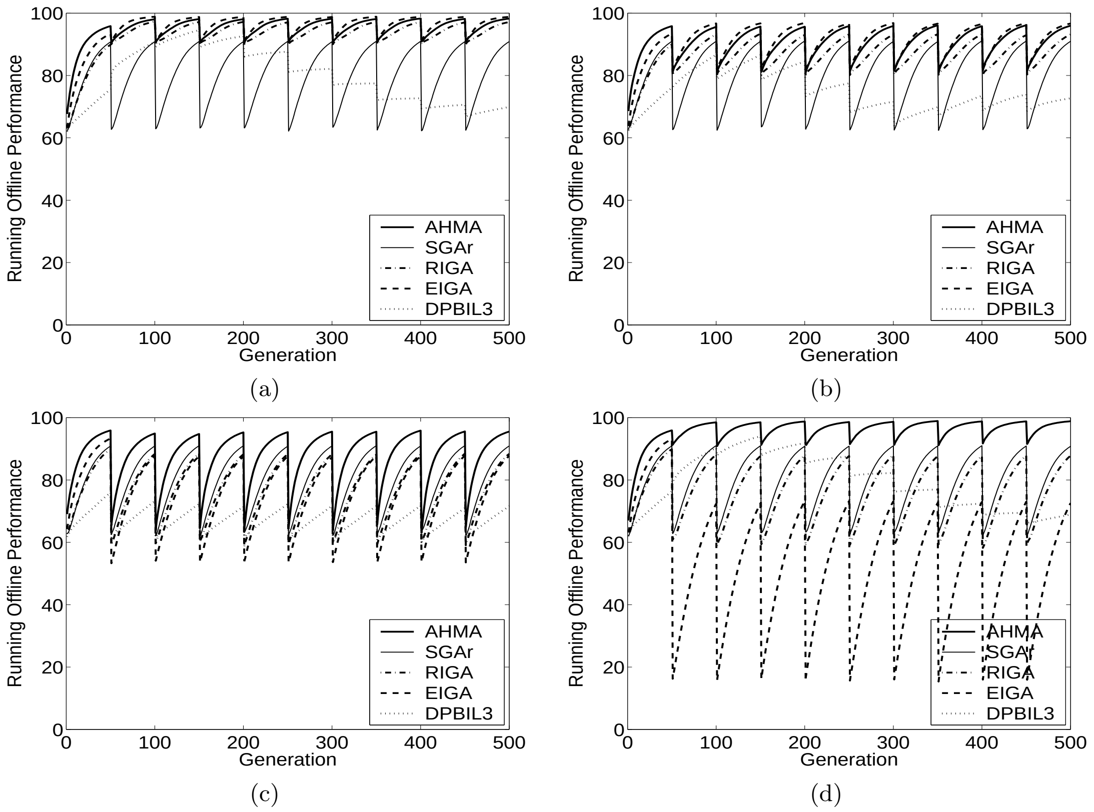

Fig. 12 Dynamic behavior of AHMA and peer EAs on dynamic OneMax problems with $\tau = 50$ and ρ is set to: (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.5$, and (d) $\rho = 0.9$.

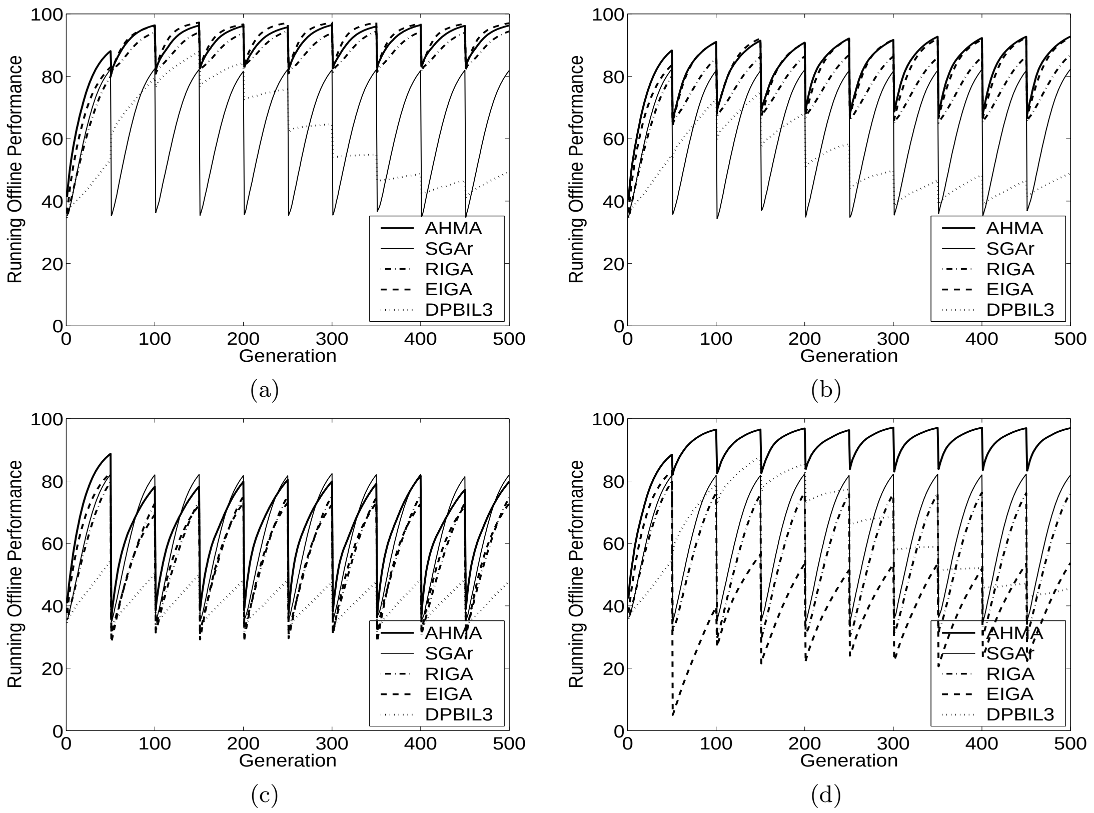

Fig. 13 Dynamic behavior of AHMA and peer EAs on dynamic Plateau problems with $\tau = 50$ and ρ is set to: (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.5$, and (d) $\rho = 0.9$.

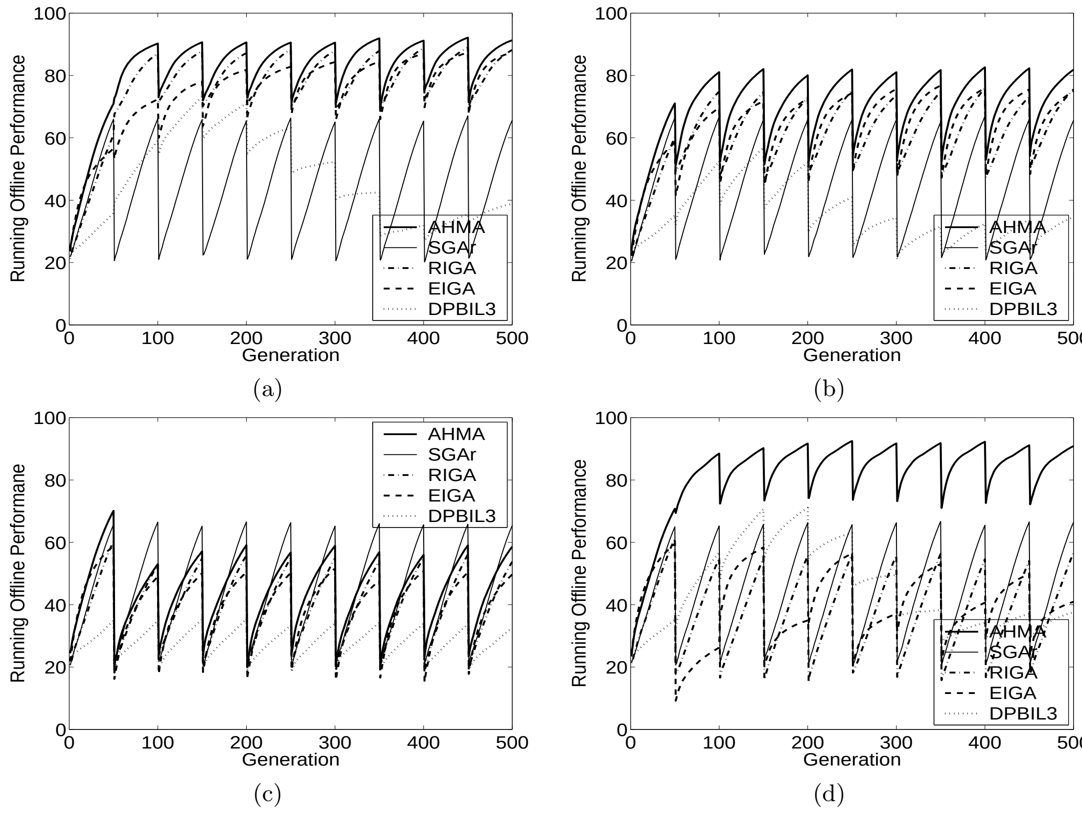

Fig. 14 Dynamic behavior of AHMA and peer EAs on dynamic RoyalRoad problems with $\tau = 50$ and ρ is set to: (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.5$, and (d) $\rho = 0.9$.

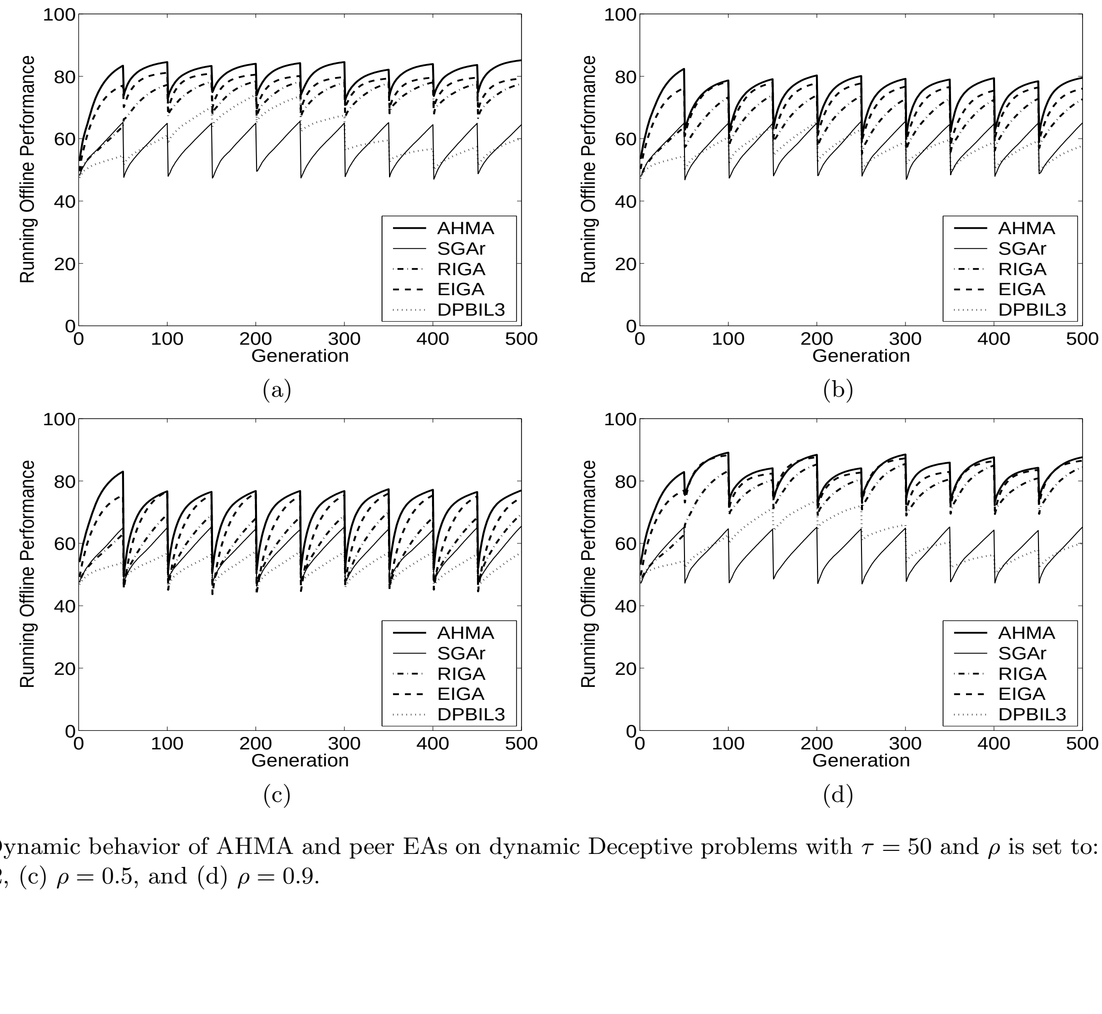

Fig. 15 Dynamic behavior of AHMA and peer EAs on dynamic Deceptive problems with $\tau = 50$ and ρ is set to: (a) $\rho = 0.1$, (b) $\rho = 0.2$, (c) $\rho = 0.5$, and (d) $\rho = 0.9$.

the individual that is selected for local improvements. A learning mechanism, which gives the more effective LS operator greater chance for the later individual refinement, is introduced in order to execute a robust local search. To maintain a sufficient population diversity for the algorithms to adapt well to the environmental changes, two diversity maintaining methods, adaptive dual mapping (ADM) and triggered random immigrants (TRI), are introduced into our proposed MA.

From the experimental results, we can draw the following conclusions on the dynamic test problems.

First, MAs enhanced by suitable diversity methods can exhibit a better performance in dynamic environments. For most dynamic test problems, our MA always outperforms other peer EAs.

Second, the ADM and TRI approaches are both efficient for improving the performance of MAs in dynamic environments. However, the two diversity methods have different effect in different dynamic environments. The ADM method does better when the environment involves significant changes (i.e., $\rho = 0.9$) and the TRI method performs better when the environmental severity $\rho = 0.5$. It is a good choice that the two diversity methods are both introduced into MAs in dynamic environments.

Third, the LS operator is problem dependent. The AHC strategy can help MAs execute a robust individual refinement since it employs multiple LS operators under the mechanism of cooperation and competition.

Fourth, the difficulty of DOPs depends on the environmental dynamics, including severity and speed of changes and the difficulty of the base stationary problems. In our experiments, MAs perform better with the increasing of the frequency of changes and the effect of the severity of changes is problem dependent.

Generally speaking, the experimental results indicate that the proposed MA, where the adaptive hill climbing operator is used as a local search technique for individual refinement, with adaptive dual mapping and triggered random immigrants schemes seems a good EA optimizer for dynamic optimization problems.

For the future work, it is straightforward to introduce other mechanisms, such as memory-based methods [38] and multi-population approaches [27], into MAs for dynamic optimization problems. Another interesting research work is to extend the triggered immigrants and dual mapping scheme to other EAs and examine their performance in dynamic environments. In addition, it is also valuable to carry out the sensitivity analysis on the effect of parameters, e.g., θ_0, α, β, and δ, on the performance of MAs in the future.

Acknowledgments

The authors would like to thank Dr Yew-Soon Ong and anonymous reviewers for their thoughtful suggestions and constructive comments. The work by Hongfeng Wang and Dingwei Wang was supported by the National Nature Science Foundation of China (NSFC) under Grants No. 70431003 and No. 70671020, the National Innovation Research Community Science Foundation of China under Grant No. 60521003, and the National Support Plan of China under Grant No. 2006BAH02A09. The work by Shengxiang Yang was supported by the Engineering and Physical Sciences Research Council (EPSRC) of UK under Grant EP/E060722/01.

References

1. 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.
2. J. Branke (1999). Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems. *Proc. of the 1999 Congress on Evolutionary Computation*, pp. 1875-1882.
3. J. Branke, T. Kaubler, C. Schmidt and H. Schmeck (2000). A multi-population approach to dynamic optimization problems. *Adaptive Computing in Design and Manufacturing*, pp. 299-308.
4. H. G. Cobb (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environment. *Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA.*
5. A. E. Eiben, R. Hinterding and Z. Michalewicz (1999). Parameter Control in Evolutionary Algorithms. *IEEE Trans. on Evol. Comput.*, 3(2): 124-141.
6. R. Eriksson and B. Olsson (2002). On the Behaviour of Evolutionary Global-Local Hybrids with Dynamic Fitness Functions. *Parrallel Problem Solving From Nature VII*, pp. 13-22.
7. R. Eriksson and B. Olsson (2004). On the Performance of Evolutionary Algorithms with Life-time Adaptation in Dynamic Fitness Landscapes. *Proc. of the 2004 Congress on Evolutionary Computation*, pp. 1293-1300.
8. J. E. Gallardo, C. Cotta and A. J. Ferndez (2007). On the Hybridization of Memetic Algorithms with Branch-and-Bound Techniques. *IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics*, 37(1): 77-83.
9. C. K. Goh and K. C. Tan (2008). A competitive-cooperation coevolutionary paradigm for dynamic multi-objective optimization. *IEEE Trans. on Evol. Comput.*.
10. D. E. Goldberg and R. E. Smith (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pp. 59-68.
11. J. J. Grefenstette (1992). Genetic algorithms for changing environments. *Parallel Problem Solving From Nature II*, pp. 137-144.
12. I. Hatzakis and D. Wallace (2006). Dynamic Multi-Objective Optimization with Evolutionary Algorithms: a Forward-Looking Approach. *Proc. of the 2006 Genetic and Evol. Comput. Conference*, pp. 1201-1208.

13. H. Ishibuchi, T. Yoshida and T. Murata (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Trans. on Evol. Comput.*, 7(2): 204-223.
14. Y. Jin and J. Branke (2005). Evolutionary Optimization in Uncertain Environments-A Survey. *IEEE Trans. on Evol. Comput.*, 9(3): 303-317.
15. N. Krasnogor and J. Smith (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Trans. on Evol. Comput.*, 9(5): 474-487.
16. T. L. Lau and E. P. K. Tsang (1996). Applying a Mutation-Based Genetic Algorithm to Processor Configuration Problems. *Proc. of the 8th IEEE Conf. on Tools with Artificial Intelligence*, pp. 17-24.
17. M. Lozano, F. Herrera, N. Krasnogor and D. Molina (2004). Real-coded Memetic algorithms with crossover hill-climbing. *Evolutionary Computation*, 12(3):273-302.
18. D. Liu, K. C. Tan, C. K. Goh, and W. K. Ho (2007). A Multiobjective Memetic Algorithm Based on Particle Swarm Optimization. *IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics*, 37(1): 42-50.
19. B. Liu, L. Wang and Y. H. Jin (2007). An Effective PSO-Based Memetic Algorithm for Flow Shop Scheduling. *IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics*, 37(1): 18-27.
20. S. Man, Y. Liang, K. S. Leung, K. H. Lee and T. S. K. Mok (2007). A Memetic Algorithm for Multiple-Drug Cancer Chemotherapy Schedule Optimization. *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics*, 37(1): 84-91.
21. F. Neri, J. Toivanen, G. L. Cascella and Y.-S. Ong (2007). An Adaptive Multimeme Algorithm for Designing HIV multidrug therapies. *IEEE-ACM Trans. on Comput. Biology and Bioinformatics*, 4(2): 264-278.
22. F. Neri, J. Toivanen, and R. A. E. Makinen (2007). An Adaptive Evolutionary Algorithm with Intelligent Mutation Local Searchers for Designing Multidrug Therapies for HIV. *Applied Intelligence*.
23. Y.-S. Ong and A. J. Keane (2004). Meta-lamarckian learning in memetic algorithms. *IEEE Trans. on Evol. Comput.*, 8(2): 99-110.
24. F. Oppacher and M. Wineberg (1999). The shifting balance genetic algorithm: improving the GA in a dynamic environment. *Proc. of Genetic and Evolutionary Computation Conference*, vol. 1, pp. 504-510.
25. U. M. O'Reilly and F. Oppacher (1994). Program search with a hierarchical variable length representation: genetic programming, simulated annealing and hill climbing. *Parallel Problem Solving from Nature III*, pp. 397-406.
26. U. M. O'Reilly and F. Oppacher (1995). Hybridized Crossover-Based Search Techniques for Program Discovery. *Proc. of the 1995 IEEE Int Conf on Evolutionary Computation*, pp. 573-578.
27. D. Parrott and X. Li (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. on Evol. Comput.*, 10(4): 440-458.
28. J. E. Smith (2007). Coevolving Memetic Algorithms: A Review and Progress Report. *IEEE Trans. on Systems, Man, and Cybernetics-Part B: Cybernetics*, 37(1): 6-17.
29. E. G. Talbi and V. Bachelet (2006). Cosearch: A Parallel Cooperative Metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5: 5-22.
30. J. Tang, M. H. Lim and Y.-S. Ong (2007). Diversity-Adaptive Parallel Memetic Algorithm for Solving Large Scale Combinatorial Optimization Problems. *Soft Computing*, 11(10): 957-971.
31. M. Tang and X. Yao (2007). A Memetic Algorithm for VLSI Floorplanning. *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics*, 37(1): 62-69.
32. A. S. Uyar and A. E. Harmanci. A new population based adaptive dominance change mechanism for diploid genetic algorithms in dynamic environments. *Soft Computing*, 9(11): 803-815, 2005.
33. F. Vavak, T. C. Fogarty, and K. Jukes (1996). Adaptive combustion balancing in multiple burner boilers using a genetic algorithm with variable range of local search. *Proc. of the 7th Int. Conf. on Genetic Algorithms*, pp. 719-726.
34. H. Wang and D. Wang (2006). An Improved Primal-Dual Genetic Algorithm for Optimization in Dynamic Environments. *Proc. of the 13th Int. Conf. on Neural Information Processing, Part III*, pp. 836-844.
35. H. Wang, D. Wang, and S. Yang (2007). Triggered Memory-based Swarm Optimization in Dynamic Environments. *Applications of Evolutionary Computing*, LNCS 4448, pp. 637-646.
36. E. H. William, N. Krasnogor and J. E. Smith(eds.), *Recent Advances in Memetic Algorithms*, Springer-Verlag, Berlin Heidelberg, 2005.
37. S. Yang (2003). Non-Stationary Problem Optimization Using the Primal-Dual Genetic Algorithm. *Proc. of the 2003 Congress on Evolutionary Computation*, vol. 3, pp. 2246-2253.
38. S. Yang (2006). Associative memory scheme for genetic algorithms in dynamic environments. *Applications of Evolutionary Computing*, LNCS 3907, pp. 788-799.
39. S. Yang (2007). Genetic algorithms with elitism-based immigrants for changing optimization problems. *Applications of Evolutionary Computing*, LNCS 4448, pp. 627-636.
40. S. Yang (2008). Genetic algorithms with memory and elitism based immigrants in dynamic environments. *Evolutionary Computation*, 16(3).
41. S. Yang, Y.-S. Ong, and Y. Jin (eds.), *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer-Verlag, Berlin Heidelberg, 2007.
42. S. Yang and X. Yao (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11): 815-834.
43. S. Yang and X. Yao (2008). Population-based incremental learning with associative memory for dynamic environments. *IEEE Trans. on Evol. Comput.*, to appear.
44. Z. Zhou, Y.-S. Ong and M. H. Lim (2007). Memetic Algorithm Using Multi-surrogates for Computationally Expensive Optimization Problems. *Soft Computing*, 11(9): 873-888.
45. Z. Zhu, Y.-S. Ong and M. Dash (2007). Wrapper-Filter Feature Selection Algorithm Using a Memetic Framework. *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics*, 37(1): 70-76.