Noname manuscript No.
(will be inserted by the editor)

Adaptive-mutation Compact Genetic Algorithm for Dynamic Environments

Chigozirim J. Uzor · Mario Gongora · Simon Coupland · Benjamin N. Passow

Received: date / Accepted: date

Abstract In recent years, the interest in studying nature inspired optimization algorithms for dynamic optimization problems (DOPs) has been increasing constantly due to its importance in real-world applications. Several techniques such as hyper-selection, change prediction, hypermutation and many more have been developed to address DOPs. Among these techniques, the hypermutation scheme has proved beneficial for addressing DOPs but requires that the mutation factors be picked a priori and this is one of the limitation of the hypermutation scheme.

This paper investigates variants of the recently proposed adaptive-mutation compact genetic algorithm (amcGA). The amcGA is made up of a change detection scheme and mutation schemes, where the degree of change regulates the probability of mutation (i.e. the probability of mutation is directly proportional to the degree of change). This paper also presents a change trend scheme for the amcGA so as to boost its performance whenever a change occurs. Experimental results shows that the change trend and mutation schemes has an impact on the performance of the amcGA in dynamic environment and also indicates that the effect of the schemes depends on the dynamics of the environment as well as the dynamic problem being considered.

Keywords Dynamic Optimization Problems (DOPs) · Evolutionary Algorithms (EAs) · Compact Genetic Algorithm (cGA) · Adaptive-mutation Compact Genetic Algorithm (amcGA) · Population-based Incremental Learning (PBIL)

Chigozirim J. Uzor
Centre for Computational Intelligence (CCI)
De Montfort University
Leicester, United Kingdom
E-mail: juzor@dmu.ac.uk

Mario Gongora, Simon Coupland
Centre for Computational Intelligence (CCI)
De Montfort University
Leicester, United Kingdom
E-mail: mgongora, simonc@dmu.ac.uk

Benjamin N. Passow
Centre for Computational Intelligence (CCI)
De Montfort University's ITS Research Group (DIGITS)
De Montfort University
Leicester, United Kingdom
E-mail: benpassow@ieee.org

1 Introduction

Most real-world engineering, economic and information technology problems change over time (i.e. experience uncertain and dynamic changes). The interest in improving the performance of EAs in dynamic environments continues to increase so as to identify promising techniques capable of addressing more complex DOPs. Many studies (such as Gongora et al (2009) and many more) have demonstrated that the standard EA is good at finding the optimum of complex multi-modal functions when the promising region of the search space remains stationary during an optimization process. However, when solving DOPs, the standard EA is not suitable because the algorithm is expected to not only find the optimum but also track the optimum with respect to time.

In fact, realistic applications are more likely to experience uncertain or dynamic changes, in the sense that one or more of the problem specifications (i.e. the target function, constraints and parameters) may vary over time. In such environment, optimization algorithms are not only required to optimize the problem in its actual state, but also adapt to the new optima whenever an

environmental change is detected and then to continuously track the moving optima throughout the whole optimization process.

In EAs, diversity in the population is useful for adapting in a changing environment, since members of the population represents potential solutions that can be applied to different environmental circumstances. Classic EAs have been successful in solving optimization problem in static environment (Passow et al, 2008; Chen and Wu, 2011) but when confronted with DOPs the algorithms performance is limited. Also the standard EAs employs a strong selection policy based on feedback which gradually reduces diversity during an optimization process. In typical applications, the function representing the environment remains static so that the algorithm is mainly limited to finding a solution. If the environment changes, the performance of the standard EA is not guaranteed as it is unable to redirect its region of concentration in the search space.

Recently Uzor et al (2014a) investigated a compact genetic algorithm (cGA) for DOPs known as Adaptive-mutation cGA (amcGA) by introducing a change detection and mutation scheme where the mutation scheme is directly linked with a change detection scheme such that the change detection scheme regulates the mutation rate (i.e degree of change determines the probability of mutation). In (Uzor et al, 2014b), the amcGA was evaluated using a real-world dynamic optimization control problem with some preliminary results. In this paper, variants of the amcGA are presented and further investigated to improve the algorithms adaptability in a dynamic environment. These variants are denoted as amcGA1 (Uzor et al, 2014a), amcGA2, amcGA3, amcGA4 and amcGA5. In amcGA2 and amcGA3, a scaled mutation rate (based on the degree of change) is used to regulate the amount a mutation operation alters the probability vector within the algorithm. The amcGA4 and amcGA5 make use of change patterns exhibited by the current working probability vector to mutate the probability vector held in memory so as to boost the algorithms response to dynamic change.

The rest of this paper is organized as follows; Section 2 reviews existing EAs for dynamic environments, Section 3 introduces a background knowledge of the cGA and details the amcGA as well as its variants, Section 4 describes the setup-scene for all experiments and performance analysis. Finally, Section 5 concludes this paper with discussions on future direction.

2 EAs for DOPs

In real world applications, certain problems arise which can be a search or optimization problem. In order to solve such problems appropriate techniques are required so as to obtain desired/best performance. These problems normally require the consideration of multiple performance criteria and non-proportional control variables. Optimization problems can be found everywhere in science, technology and even daily life activities e.g. planning (Bui et al, 2012), tuning of controllers (Pedersen and Yang, 2006) and many more. Most real-world optimization problems are often influenced by uncertain and dynamic factors (Jin and Branke, 2005) and it is unlikely that a solution found for a particular problem would remain valid for a long period of time. In order to counter these dynamic and uncertain factors an adaptive mechanism is required to introduce changes to the current solution. These types of optimization problems can be referred to as a dynamic optimization problem.

The nature of DOPs presents challenges to traditional optimization algorithm because these problems usually require the tracking of the changing environment with respect to time. In general, addressing DOPs using EAs can be grouped into four categories: 1) Using implicitly or explicitly defined memory to store and reuse useful information so as to adapt the EA whenever a change occurs (Yang and Yao, 2008). 2) Creating a multi-population to distribute the search force in the search space (Zhu et al, 2006). 3) Promote diversity by inserting random immigrants back in the population (Yu et al, 2008) and 4) Adjusting genetic operators adaptively (Eiben et al, 2006). Apart from the approaches mentioned above, for an EA to function properly the genetic operators needs to be tuned/defined properly (as it affects performance and is problem dependent) and this can be achieved in three ways: 1) Deterministic method(this involves adjusting the value of the strategy parameter using a deterministic rule which is fixed). 2) Adaptive method, which makes use of feedback from the optimization process to determine when to change the strategy parameter, which can be in form of an IF-THEN rule and may involve a credit assignment which defines the quality of the solution discovered and 3) Self-adaptive method (where the mechanism for updating the strategy parameter is implicitly defined) (Affenzeller and Wagner, 2003).

When solving DOPs, evolutionary algorithms are considered a good choice because they are inspired from the principles of biological evolution, which takes place in a dynamic environment. But when using classic EAs, once converged, they are unable to adapt to changes in a dynamic environment. In DOPs, values of the optima change with time, thus rendering the problem of optimum finding to optimum tracking and this means the fitness landscape of a given problem is dynamic with possibly both the search space and fitness being time

dependent. In (Yang and Tinos, 2008) a hyper-selection scheme in dynamic environment was proposed for a genetic algorithm (GA) to address DOPs, where the selection pressure is increased whenever an environment change occurs. In standard GAs, individuals in the population converge to an optimal solution in static environments due to selection pressure but in dynamic environments, converging to an optimum becomes a problem for the standard GAs since it does not encourage genetic diversity and hence makes it hard to adapt to a new environment whenever a change occurs. Although the scheme discussed in (Yang and Tinos, 2008) demonstrated the effects of selection pressure for GAs in dynamic environment, adjusting the selection pressure adaptively during an optimization/search process still remains an open question. A forward-looking approach for solving dynamic multi-objective optimization problems using EA was proposed in (Hatzakis and Wallace, 2006), the idea was to implement a forecasting method where the location in variable space of the optimal solution is estimated, the optimization algorithm exploits past information and prepares for the change before it arrives instead of reacting to the change.

In general certain techniques are suitable for certain environments i.e. memory based approaches are suitable for periodic optima, self-adaptation and mutation approaches are suitable for landscapes with fast changes, multi-population approaches are suitable for competing peaks and maintaining diversity is suitable for continuously moving peaks (Woldesenbet and Yen, 2009). In (Yang, 2008), a memory and elitism based immigrants approach for GAs in dynamic environment was presented. The best individual during an optimization process is stored in memory (or elite from previous generation) and is retrieved as a base to create new individuals by mutation so as to ensure diversity and also adapt to a new environment.

Although, these algorithms have been successful in tackling DOPs, none of the authors have considered linking change severity with a diversity scheme such that the degree of change is directly proportional to the diversity scheme used.

3 cGA for DOPs

There are some optimization problems that limits the application of traditional optimization algorithm due to hardware limitations and this is as a result of the complex structure employed by population-based approach (which makes them computationally expensive). In order to overcome hardware limitations, a memory efficient algorithm is required. The compact genetic algorithm (cGA) offers the advantage of being computationally efficient (i.e requires less memory and execution time).

The compact genetic algorithm as proposed by Harik et al (1999) is an estimation of distribution algorithm (EDA) (Larraanaga and Lozano, 2001; Pelikan et al, 2000) that generates offspring population according to an estimated probability model of the parent population. The cGA makes use of a real-valued probability vector $\overrightarrow{P}$ to represent the bit probability of 0 or 1 which models the distribution of the population:

$$\overrightarrow{P} = \{P_1, ..., P_l\} \tag{1}$$

where l is the binary-encoding or chromosome length and $Pi \in \{0, 1\}$, $(i = 1, ..., l)$. The probability vector is initially assigned 0.5 to represent a randomly generated population. In every generation, competing solutions are generated based on the current probability vector and the probabilities P_i are updated to favour a better solution. In a simulated population of size s, the probability of each gene increases or decreases by $\frac{1}{s}$ based on the gene of the best solution i.e.:

$$P_i^{'} = \begin{cases} P_i(t) + 1/s & \text{if } best_i = 1, \quad (2a) \\ P_i(t) - 1/s & \text{if } best_i = 0. \quad (2b) \end{cases}$$

The cGA maintains a probability vector and evolves it towards the best sample solution created from it. The driving force for cGA to solve an optimization problem lies in the update mechanism of the probability vector towards the best sample created from it iteratively. The probability vector of the cGA usually converges to either 0.0 or 1.0 in each element which will produce the optimal solution when sampled in static environments. The performance of the cGA in a dynamic environment is not guaranteed since once the probability vector converges it is unable to adapt to the changed environment. As a result modifications to the original algorithm have been proposed so as to enable it tackle DOPs.

To address the convergence problem, several approaches have been developed to re-introduce diversity after a change occurs e.g. the restart scheme which resets the optimization algorithm back to the default setting when a change occurs (Harik et al, 2006; Sastry et al, 2005), the hypermutation scheme (Cobb, 1990; Morrison and De Jong, 2000) where the probability of mutation is raised from a low mutation rate to a high mutation rate when the environment changes, and many more. The hypermutation creates an adaptive EA with small incremental memory and computational cost but requires the mutation factor to be picked a priori.

Although these algorithms have been successful in tackling DOPs, to the best of the authors knowledge none has considered an adaptive method for controlling the mutation factor and none has tried to link together

the mutation scheme with a change severity scheme (i.e. measuring the degree of change) such that the degree of change is directly proportional to the mutation factor. This section describes the amcGA as well as variants suitable for memory constrained applications.

3.1 Change detection

A Gaussian function is employed so as to detect and determine the degree of change C_d in the environment:

$$C_d = e^{-a} \tag{3}$$

$$a = \frac{(\Delta f - c)^2}{2\sigma^2} \tag{4}$$

where c is the mean (1.0), σ represents variance and Δf is the change in fitness or fitness difference between the elite solution at generation t and same elite solution re-evaluated at generation $t+1$:

$$\Delta f = f(E_s, t) - f(E_s, t+1) \tag{5}$$

It is important to note that change in fitness of the elite solution is considered in this study as a sign of change in the environment (i.e. the algorithm monitors the performance of the elite solution). The algorithm employs the elitism approach, where the best solution from a previous generation is transferred and evaluated in subsequent generations. In order to adaptively control the mutation rate (i.e. probability of mutation) p_m, C_d is converted to the mutation rate such that a high degree of change results in a high mutation rate and a low degree of change results in a low mutation rate but when no change occurs, the algorithm proceeds as a normal cGA. The probability of mutation p_m is defined as follows:

$$p_m = m_l + (C_d - d_l) \times \left(\frac{m_h - m_l}{d_h - d_l}\right), p_m[0.01, 0.5] \tag{6}$$

where $m_l = 0.01$ is low probability of mutation, $m_h = 0.5$ is high probability of mutation, $d_l = 0.0$ is low degree of change and $d_h = 1.0$ is high degree of change.

3.2 Mutation schemes

Unlike the mutation scheme adopted by most cGA variants where mutation is applied directly to candidate solutions to create another solution for selection, the mutation scheme discussed in this paper is applied directly to the probability vector that generated the best solution (elite solution) since the probability vector represents a distribution of the population.

Suppose at generation t an elite solution E_s with fitness $f(E_s, t)$ was obtained, the probability vector that generated the solution is held in a temporary memory $\overrightarrow{mP}$. At generation $t+1$ the elite solution is re-evaluated and a new fitness value is obtained i.e. $f(E_s, t+1)$. If the fitness difference Δf is greater than a defined threshold (e.g. $\Delta f>0$) then a change is said to have occurred which triggers the mutation scheme, which is applied directly to $\overrightarrow{mP}$ to generate a mutated version of the elite solution E_m to compete with E_s.

The cGA makes use of a real valued probability which generates two solutions when sampled. In order to apply the mutation scheme to the probability vector, a random number $r = rand(0.0, 1.0)$ is generated then compared with p_m and $\overrightarrow{mP}$ is mutated as follows:

amcGA1:

$$mP_i^{'} = \begin{cases} rand(C_d, mP_i) & \text{if } r<p_m, \quad (7a) \\ mP_i & \text{if } r>p_m. \quad (7b) \end{cases}$$

amcGA2:

$$mP_i^{'} = \begin{cases} |mP_i + n - p_m| & \text{if } r<p_m, \quad (8a) \\ mP_i & \text{if } r>p_m. \quad (8b) \end{cases}$$

amcGA3:

$$mP_i^{'} = \begin{cases} \left|mP_i + \left(n - \frac{p_m}{2}\right)\right| & \text{if } r<p_m, \quad (9a) \\ \left|mP_i - \left(n - \frac{p_m}{2}\right)\right| & \text{if } r>p_m. \quad (9b) \end{cases}$$

where $n = rand(0, p_m)$.

Sometimes, changes in dynamic environments may exhibit some trends. In such case, it might be beneficial to try to use these change trends to boost the algorithms response to subsequent changes in such dynamic environment. Some studies have been made following this idea but differ in that they exploit the predictability of dynamic environments (Simões and Costa, 2009b,a).

Memory approaches (Branke, 1999; Yu and Suganthan, 2009), which were originally proposed to deal with periodical changes, can also be considered as a type of prediction method. Algorithms following the prediction approach make use of memory scheme to cope with various types of changes (e.g. cyclic, noisy and random) but requires the use of accurate training data and dedicated memory allocation, which makes the respective algorithm computationally expensive.

In this study, the change trend T_{chg} is used to boost the amcGAs performance whenever a change occurs by applying the change trend to $\overrightarrow{mP}$ such that $\overrightarrow{mP}$ learns from past dynamic changes and adapts to future dynamic changes instead of explicitly using stored training data. The change trend T_{chg} is defined as follows:

$$T_{chg} = \overrightarrow{mP_t} - \overrightarrow{P}_{t+1} \tag{10}$$

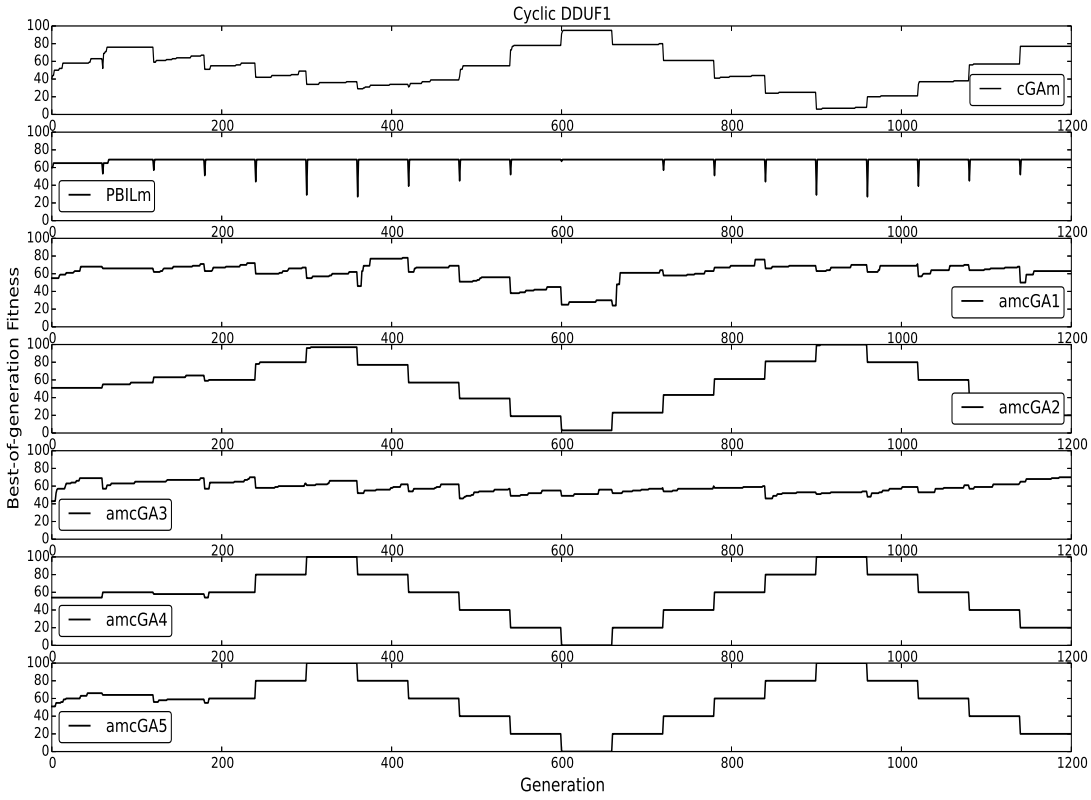

Fig. 1 Dynamic behaviour of all algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment.

where $\overrightarrow{P}_{t+1}$ is the current working probability vector at generation $t+1$. The amcGA4 and amcGA5 makes use of the change trend approach described above and is defined as follows:

amcGA4:

$$mP_i^{'} = \begin{cases} \left| mP_i + \left(n - \frac{T_{chg}}{l}\right) \right| & \text{if } r<p_m, \text{ (11a)} \\ \left| mP_i - \left(n - \frac{T_{chg}}{l}\right) \right| & \text{if } r>p_m. \text{ (11b)} \end{cases}$$

amcGA5:

$$mP_i^{'} = \begin{cases} \left| mP_i + s - \frac{T_{chg}}{l} \right| & \text{if } r<p_m, \quad \text{(12a)} \\ mP_i & \text{if } r>p_m. \quad \text{(12b)} \end{cases}$$

where $s = \left(n - \frac{p_m}{2}\right)$ and l is the binary string length. It is important to state that the change trend scheme was applied to amcGA2 and amcGA3 (which yeilds amcGA5 and amcGA4 respectively) so as to study the effect of T_{chg} on the performance the algorithm. This way, the mutation strategy updates itself based on the change pattern exhibited by the probability vector. Also T_{chg} controls the amount a mutation operation alters the value of each element in $\overrightarrow{mP}$.

After the mutation operation, a mutated version of the elite solution E_m is generated using the mutated temporary probability vector (i.e. $\overrightarrow{mP}$) to compete with the current elite solution E_s, if the mutated elite solution performs better than the current elite, it replaces the elite solution and the mutated probability vector replaces the current probability vector. The mutation scheme is repeated for a defined number of generations similar to the hypermutation scheme. After the mutation operation, the algorithm continues as a standard cGA unless another change occurs.

4 Experiments

4.1 Dynamic Benchmark Generator

For the experiments, the DOP generator proposed in (Yang and Yao, 2005) which constructs a dynamic environment was chosen to test the efficiency of the am-

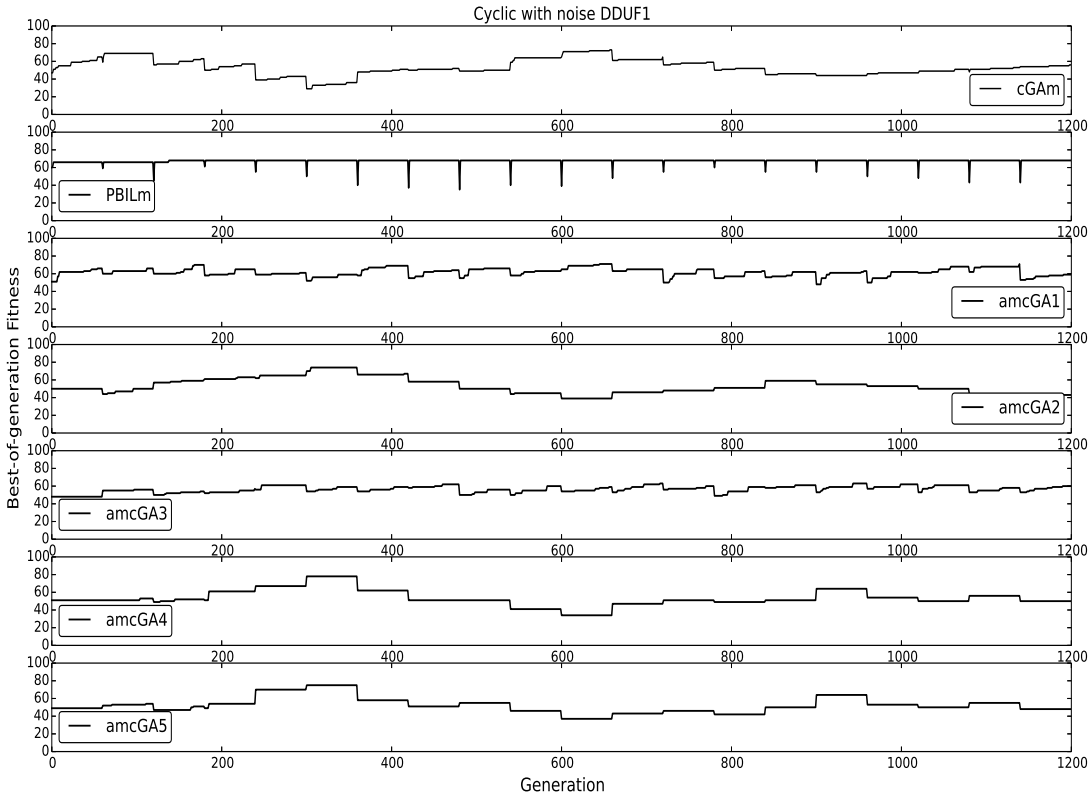

Fig. 2 Dynamic behaviour of all algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment with noise.

cGA variants. The generator can construct a DOP from any binary-encoded static function $f(\overrightarrow{x})$. Given a static optimization problem $f(x)$ $(x \in \{0,1\}^l)$ where l is the binary string length, the dynamic environment is generated by applying a binary XORing mask $\overrightarrow{M}$ to each solution before evaluating at every τ generations.

$$f(x,t) = f(x \oplus \overrightarrow{M}(k)) \tag{13}$$

where $f(x,t)$ is the fitness of solution x, $k = t/\tau$ is the period index at time t, $\oplus$ is a bitwise exclusive-or (XOR) operator which is applied to the x and $M(k)$ according to the following principle:

$$x_i \oplus x_j = \begin{cases} 0 & \text{if } x_i = x_j, \quad (14a) \\ 1 & \text{otherwise.} \quad (14b) \end{cases}$$

For each environment k, $\overrightarrow{M}(k)$ is incrementally generated as follows:

$$\overrightarrow{M}(k) = \overrightarrow{M}(k-1) \oplus \overrightarrow{T}(k) \tag{15}$$

where $\overrightarrow{T}(k)$ is an intermediate binary template generated for environment k. $\overrightarrow{T}(k)$ is generated with ρ x l $(\rho \in (0.0, 1.0])$ random loci set to 1 while the remaining loci set to 0. ρ controls the intensity or severity of change. If ρ is set to 0, the environment is considered stationary since $\overrightarrow{T}$ will contain only $0s$ and no change will occur. On the other hand $p = 1$ guarantees a high degree of change (i.e. high change severity). Also a small τ means faster environment change while a large τ means slow environment change.

4.2 Dynamic Test Problem

4.2.1 Decomposable Unitation-Based Functions (DUFs)

The decomposable unitation-based functions have been used as benchmark functions by the EA community in an attempt to understand what constructs difficult optimization problems for EAs (e.g (Goldberg, 2002)). These type of functions return the number of ones in a binary string (i.e. unitation function of binary string). Two DUFs, denoted as DUF1 and DUF2 are used as static functions to construct dynamic test environments, in order to compare the performance of algorithms discussed in this paper.

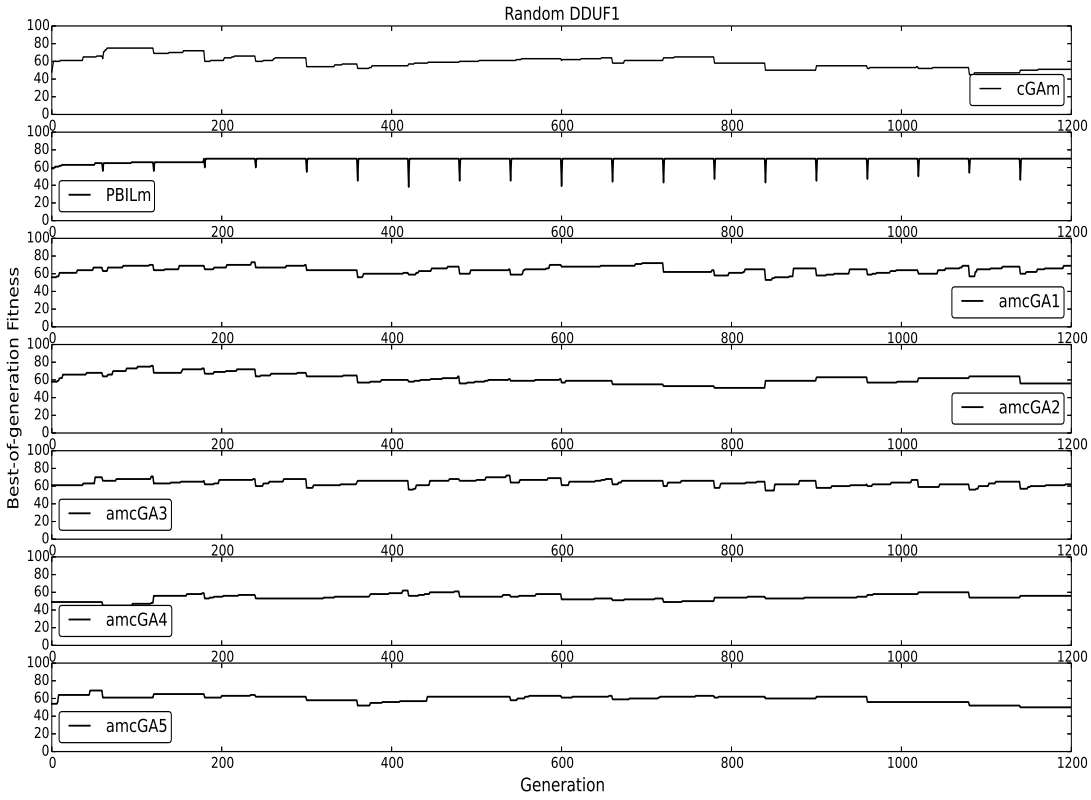

Fig. 3 Dynamic behaviour of all algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in a random environment.

The DUF1 is simply a Onemax function which aims to maximize the number of 1's in a binary string. The fitness of a binary string is the number of 1's contained in the string.

$$f(x) = u(x) \tag{16}$$

DUF2 is a fully deceptive function, which are considered hard problems for EAs because the low-order building blocks inside the functions do not combine to form the higher order optimal building block, instead they combine to form deceptive suboptimal building block.

$$f(x) = \begin{cases} 3 - u(x) & \text{if } u(x)<4 \quad (17a) \\ 4 & \text{otherwise} \quad (17b) \end{cases}$$

Using the dynamic benchmark generator discussed in Section 4.1, dynamic test environments are constructed from the DUFs and are denoted as DDUF1 and DDUF2.

4.2.2 Dynamic Knapsack Problem

The knapsack problem is a classic NP-complete optimization problem that has been rigorously studied by the EA community in the last few decades. The main aim of this problem is to fill a knapsack with the best subset of items among a larger set so as to maximize the value of contents in the knapsack without exceeding the knapsack capacity. This benchmark problem has been studied in both static (e.g. Shah and Reed (2011); Martins et al (2014)) and dynamic environments (e.g. Yang et al (2013) with different modifications. The dynamic property of the knapsack problem is achieved when the problem parameters (such as item weight, value and sack capacity) are time dependent and subject to variation.

Given n items, each of which has a weight $w_i(t)$ and a value $v_i(t)$ and a knapsack of capacity C. The main goal of the knapsack problem is to load the items that guarantees maximum value without exceeding the knapsack capacity C. A dynamic test environment is constructed for the knapsack problem and is denoted as DKP. Mathematically DKP can be described as follows:

$$\text{Maximize } f(x,t) = \sum_{i=1}^{n} p_i(t)x_i(t) \tag{18}$$

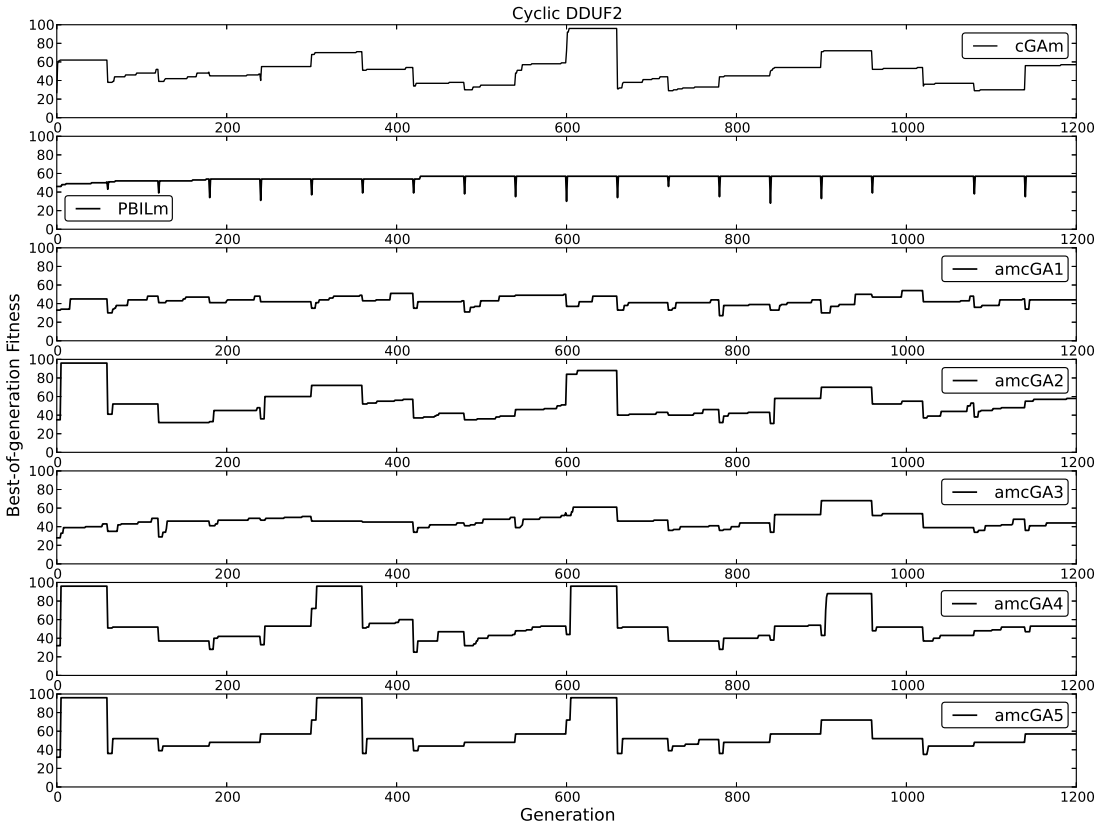

Fig. 4 Dynamic behaviour of all algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment.

$$s.t. \begin{cases} \sum_{i=1}^{n} w_i(t)x_i(t) & \leq C \quad (19a) \\ x_i(t) \in \{0,1\}, \quad i = 1, ..., n & \quad (19b) \end{cases}$$

where x_i is the binary decision variable used to indicate if item i is included or discarded. In this study, all values and weights are positive, also all weights are less than the knapsack capacity $C = 500$. A knapsack problem with 100 items using randomly generated data was constructed as follows:

$$w_i = \text{uniformly random integer}[2, 20] \quad (20)$$

$$p_i = \text{uniformly random integer}[1, 30] \quad (21)$$

The sum of the profits of the selected items is used as the fitness of a candidate solution if the sum of item weight is within the knapsack capacity. However, if a candidate solution selects too many items such that the summed weight exceeds the knapsack capacity then a penalty function is used to judged how much the candidate solution exceeds the knapsack capacity. The penalty function is defined as follows:

$$f(x,t) = \begin{cases} \sum_{i=1}^{n} p_i x_i & \sum_{i=1}^{n} w_i x_i \leq C \quad (22a) \\ f(x,t) - l_f & \text{else} \quad (22b) \end{cases}$$

where $l_f = 7 * (\sum_{i=1}^{n} w_i x_i$ - $C)$ and $n = 100$.

4.3 Parameter Settings and Performance Measures

Experiments were carried out on the selected DOPs to investigate the effect of the change detection, change trend and mutation schemes on the performance of the amcGA. An additional experiment was carried out to compare the performance of the scheme presented in section 3 with a cGA with hypermutation (denoted as cGAm) and a probability based incremental learning algorithm with hypermutation (denoted as PBILm) (Yang and Richter, 2009).

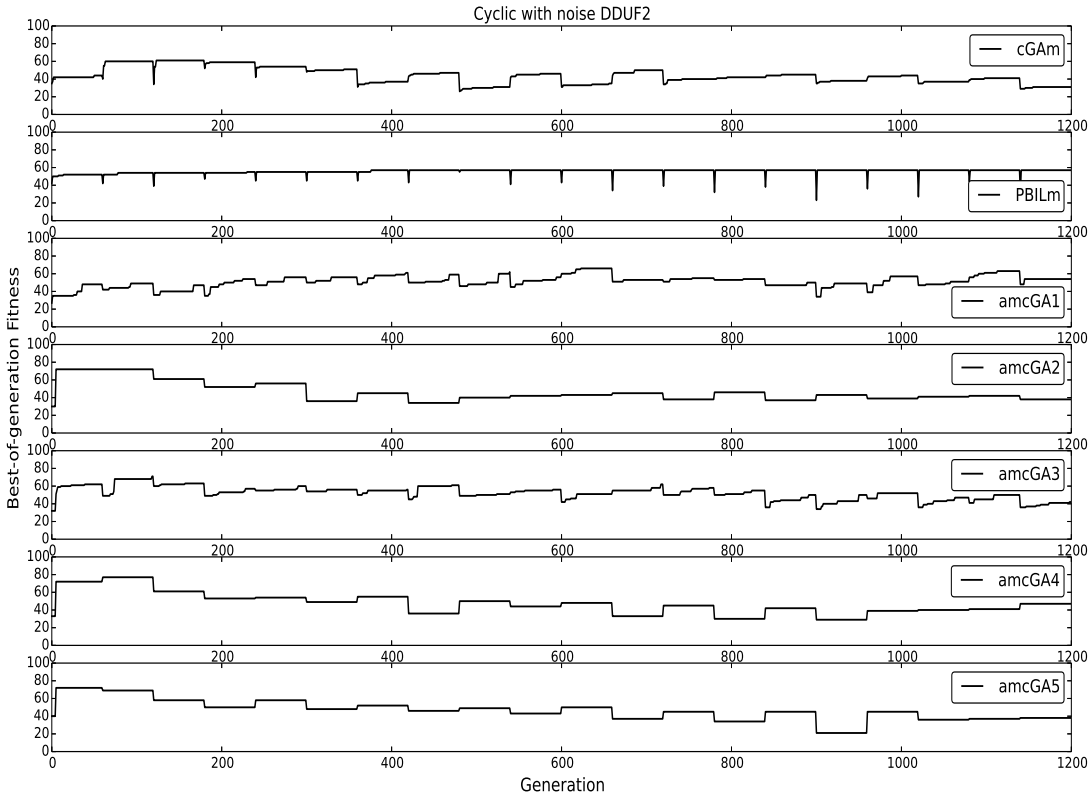

Fig. 5 Dynamic behaviour of all algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment with noise.

For all algorithms some common parameters were set as follows; the population size $n = 100$, speed of change $\tau = 20$, 60 and 100, and change ratio $\rho = 0.1$, 0.2, 0.5 and 1.0. While the sensitivity level for detecting change $\Delta f > 0$ for the scheme described in section 3, probability of mutation (for the PBILm) $p_m = 0.05$ with mutation shift $\delta = 0.05$. Also all algorithms use the elitism approach (in the case of the PBIL an elite of size 1 was used). For the PBILm, the probability of mutation was set to a base level $p_m^l = 0.05$ for normal generations and a high value $p_m^h = 0.3$ for interim generations when the hypermutation scheme is triggered due to change in environment and this lasts for 5 generations i.e. $g_{hm} = 5$.

The DDUFs considered consists of 25 copies of 4-bit building blocks. Each building block of the two DDUFs contributes a maximum value of 4 to the total fitness. The fitness of a bit string is the sum of contributions from all building blocks which gives an optimal fitness of 100 (for DDUF1 and DDUF2). Three kinds of dynamic environments were constructed (i.e. cyclic, cyclic with noise and random) using the dynamic problem generator discussed in section 4.1.

For each experiment using all algorithms on the DOP, 30 independent runs were executed and for each run 20 environmental changes were allowed, which are equivalent to 400, 1200 and 2000 generations for $\tau =$ 20, 60 and 100 respectively. Best-of-generation fitness was recorded every generation and the overall offline performance of all algorithms on each DOP is defined as:

$$\overline{F}_{BOG} = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{1}{N}\sum_{j=1}^{N} F_{BOG_{ij}}\right) \tag{23}$$

where $F_{BOG_{ij}}$ expresses the fitness value of the best solution at generation i of run j, $G = 20$ x τ is the total number of generation for a run, $N = 30$ is the total number of runs and $\overline{F}_{BOG}$ is the overall offline performance, which is the best-of-generation fitness averaged over N and then over the data gathering period.

Experimental results of all algorithms on the selected DOPs based on $\overline{F}_{BOG}$ are presented in Fig. 1- 9 respectively. The corresponding statistical results of the Wilcoxon rank-sum test at 0.05 level of significance are shown in Table 1. In Table 1, the result regarding $Alg.1$ - $Alg.2$ is shown as "+", "−" and " " when $Alg.1$ is sig-

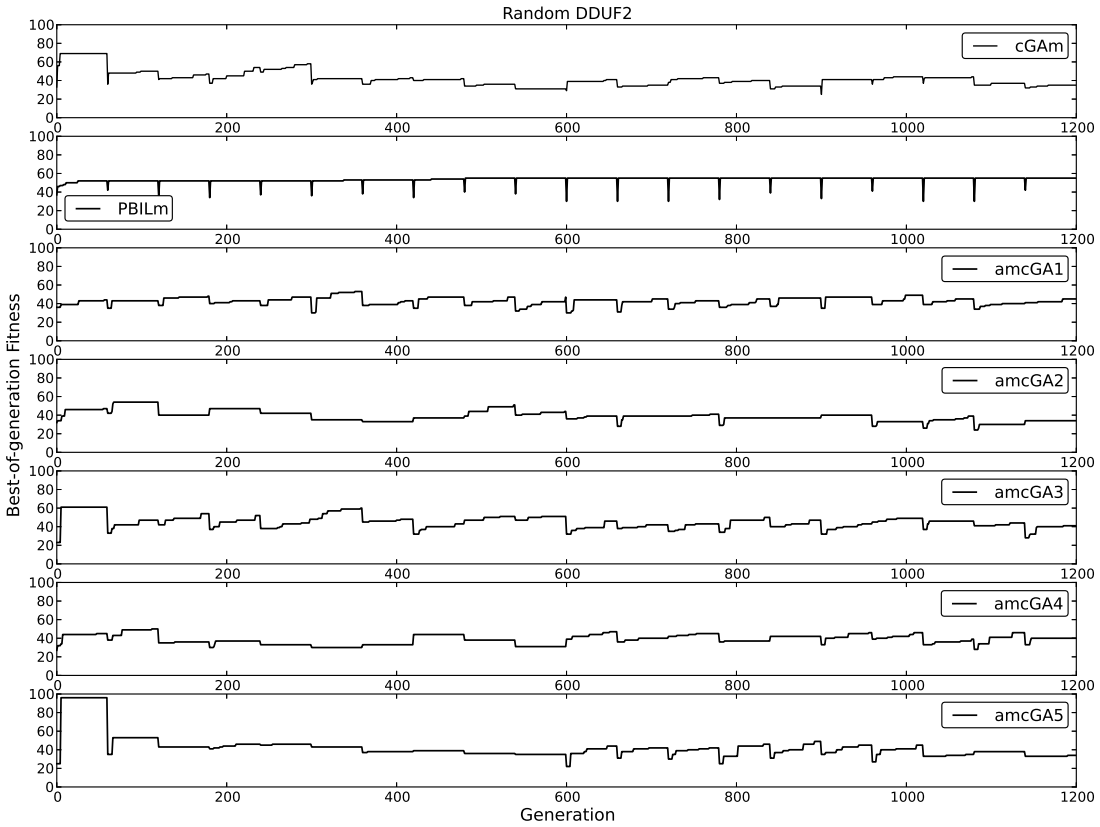

Fig. 6 Dynamic behaviour of all algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in a random environment.

nificantly better than, significantly worse than or statistically equivalent to $Alg.2$. The dynamic performance of all algorithms regarding the best-of-generation fitness against generations on the dynamic problems are plotted in Fig. 1- 9 (with $\tau = 60$ and $\rho = 0.2$).

From Fig. 1- 9 and Table 1, several behaviours can be observed and are discussed in next section from two aspects: 1.) regarding performance based on $\overline{F}_{BOG}$ and 2.) algorithms behaviour on the selected DOPs (i.e. the effect of environmental dynamics on algorithms performance).

4.4 Experimental Study Regarding Overall Performance

The experimental results on dynamic problems and key statistical test results are shown in Fig. 1- 10 and Table 1 respectively.

First, PBILm shows a constant performance across all DOPs regardless of the dynamics of the environment. This is because the PBILm evaluates 100 candidate solutions (every generation) and has a greater chance of finding better solutions than the cGAm and amcGAs' which only evaluates 2 candidate solutions every generation. And with the increasing of τ, PBILm has more time to search for solutions with higher fitness before the next change. However in some environment change ratio ρ, PBILm was outperformed by the amcGA (variants) as can be observed from Figs 10 and Table 1. This is due to the lack of information transfer from the last environment of the last dynamic change. Also, PBILm applies the mutation scheme to the current working $\overrightarrow{P}$ which has no information of the previous environment and this means the PBILm is focused more on preventing premature convergence of $\overrightarrow{P}$.

Second, cGAm outperforms some of the amcGA variants in some of the DOPs. This is due to the fact that whenever a change occurs, the cGAm tries to find a better solution for the current environment (i.e. which is the effect of rapid increase in probability of mutation p_m) but does not ensure diversity as can be seen in Fig. 1. Also for some dynamic settings, cGAm shows similar performance to some the amcGA vari-

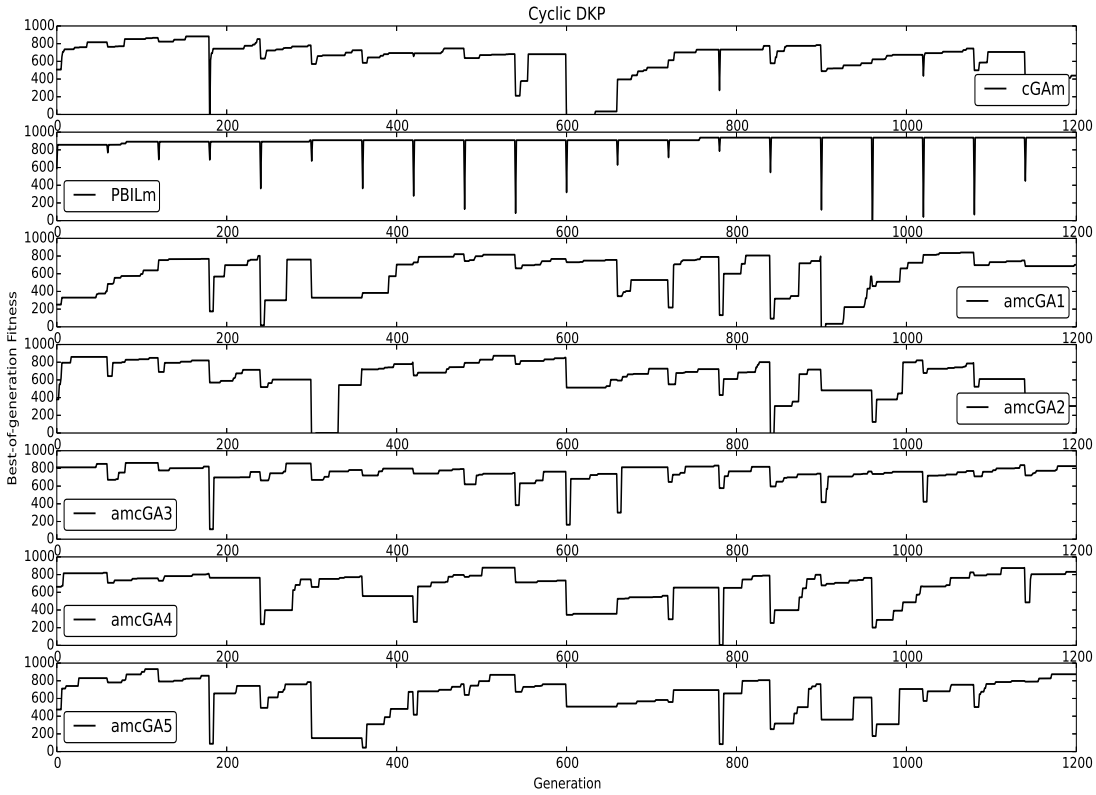

Fig. 7 Dynamic behaviour of all algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment.

ants. Given a value of τ when the environment changes with respect to the change ratio, the performance of the algorithms are approximately the same (see Fig. 3 and 4).

Third, among the five variants of the amcGA, amcGA1 and amcGA3 exhibit interesting performance. From Fig. 1- 10, it can be observed that amcGA3 performs better than amcGA1. Also, amcGA3 shows stable performance across different environment dynamics. Performance of all amcGA variants based on $\overline{F}_{BOG}$ is shown on Fig. 1- 9 for different environment dynamics. Although Fig. 1- 9 shows general performance of all algorithms, it is difficult to draw out conclusions about the final result of the compared algorithms by just visual inspection of the performance curves. Using the Wilcoxon rank-sum test, several conclusions can be observed. On several environment dynamics, the performance of the amcGA1 and amcGA3 are better than that of the cGAm, also when the ρ is low and τ is low to medium, most of the amcGA variant outperformed both the cGAm and the PBILm. This behaviour is as a result of how the amcGA handles its probability vector.

The amcGA maintains a moderate convergence rate as it explores the search space. This can be considered as an advantage over the hypermutation scheme since the amcGA not only carries information from one stage of the problem to the next stage but also retains these information in the form of $\overrightarrow{mP}$, which represents properties and dynamics of a particular environment. Also since the mutation scheme is only applied to $\overrightarrow{mP}$, it ensure that the current working $\overrightarrow{P}$ maintains its diversity unless a solution generated by the mutated $\overrightarrow{mP}$ (whenever a change is detected) outperforms the current best solution generated by $\overrightarrow{P}$, thereby replacing $\overrightarrow{P}$ with $\overrightarrow{mP}$.

Finally, performance of the amcGA4 and amcGA5 in the cyclic environment is better than (or same as) that of the cGAm on some of the DOPs (with respective environment dynamics). This behaviour is as a result of the change trend scheme within the algorithm. Although this scheme does not make used of any external training data, it has a positive effect on the performance of the amcGA4 and amcGA5. The change trend scheme ensures that the amcGA retains information about past environment (i.e. $\overrightarrow{mP}$) while searching

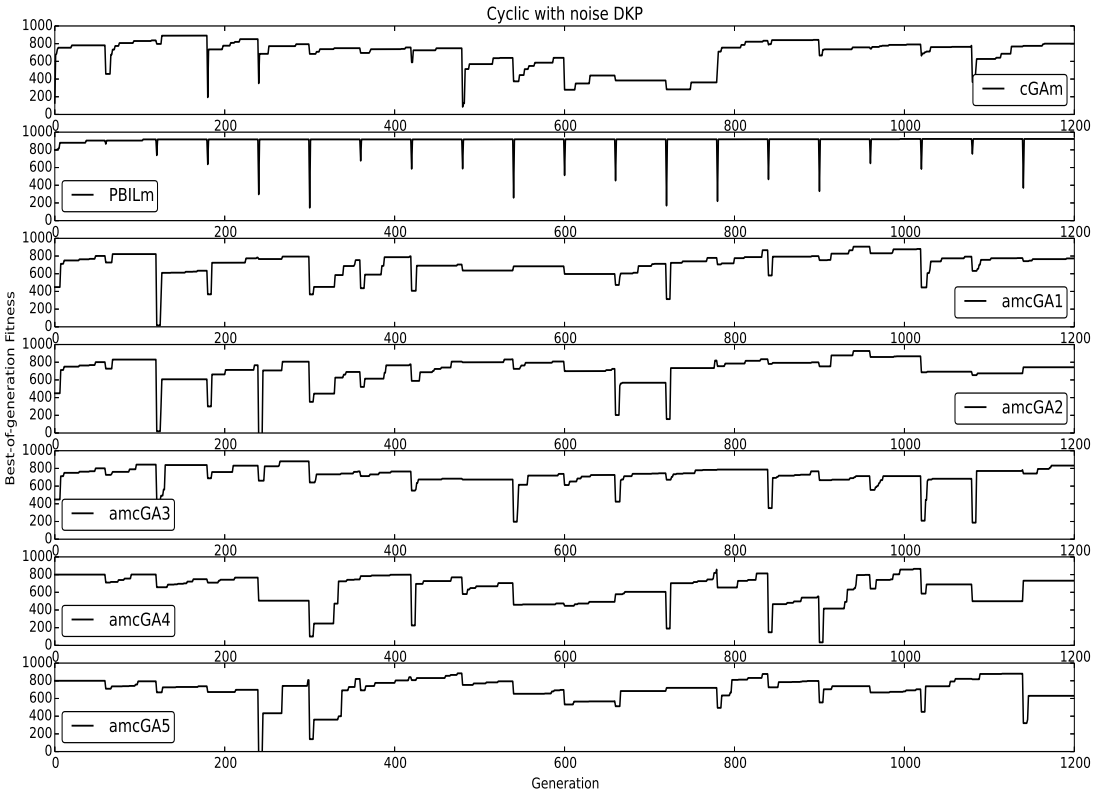

Fig. 8 Dynamic behaviour of all algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a cyclic environment with noise.

for promising region (using $\overrightarrow{P}$) in the search space of a new environment. This can be observed when $\tau = 100$ and ρ is between 0.1 and 0.5 (see Table 1), the algorithms are given more time to search before the next environmental change but experience slow convergence rate. On the other hand, convergence deprives cGAm of the adaptability to changing environments because the $\overrightarrow{P}$ within cGAm learns from the best hyper-mutated solution whenever a change occurs. However, the mutation mechanism and change trend scheme embedded in amcGA4 and amcGA5 grants more diversity than cGAm (and PBILm in some environment) and hence better adaptability to environmental changes.

4.5 Experimental analysis of algorithms behaviour on selected DOPs

In order to better understand the experimental results, we need to look deeper into the dynamic behaviour of all algorithms. The dynamic behaviour all different algorithms on the selected DOPs are shown in Fig. 10, where the data were averaged over 30 runs, τ is set to 60, $\rho = 0.1$, 0.2, 0.5 and 1.0. Several behaviours can be observed when examining the effect of the dynamic environments on the performance of the algorithms investigated.

From Fig. 10, it can be observed that for a fixed τ with increasing value of ρ, PBILm outperform other algorithms on several cases and maintains almost the same performance across the three DOPs. The behaviour is a result of the high adaptability brought in by the hypermutation scheme (and population-based structure) within PBILm. However, the performance of PBILm decrease on the cyclic DDUF2 and random DDUF2. This is due to the fact that, when the environment changes, the deceptive building blocks inside DDUF2 will draw the population into the new environment slowly since the deceptive attractors are not globally optimal but they are suboptimal with relatively high fitness.

An interesting behaviour is that on DDUF1, the performance of the amcGA variants drops when ρ is between 0.1 and 0.5 but soon stabilizes. This is because when $\rho = 1.0$, the environment switches between two landscapes and the algorithm may wait during one environment for the return of the other environment

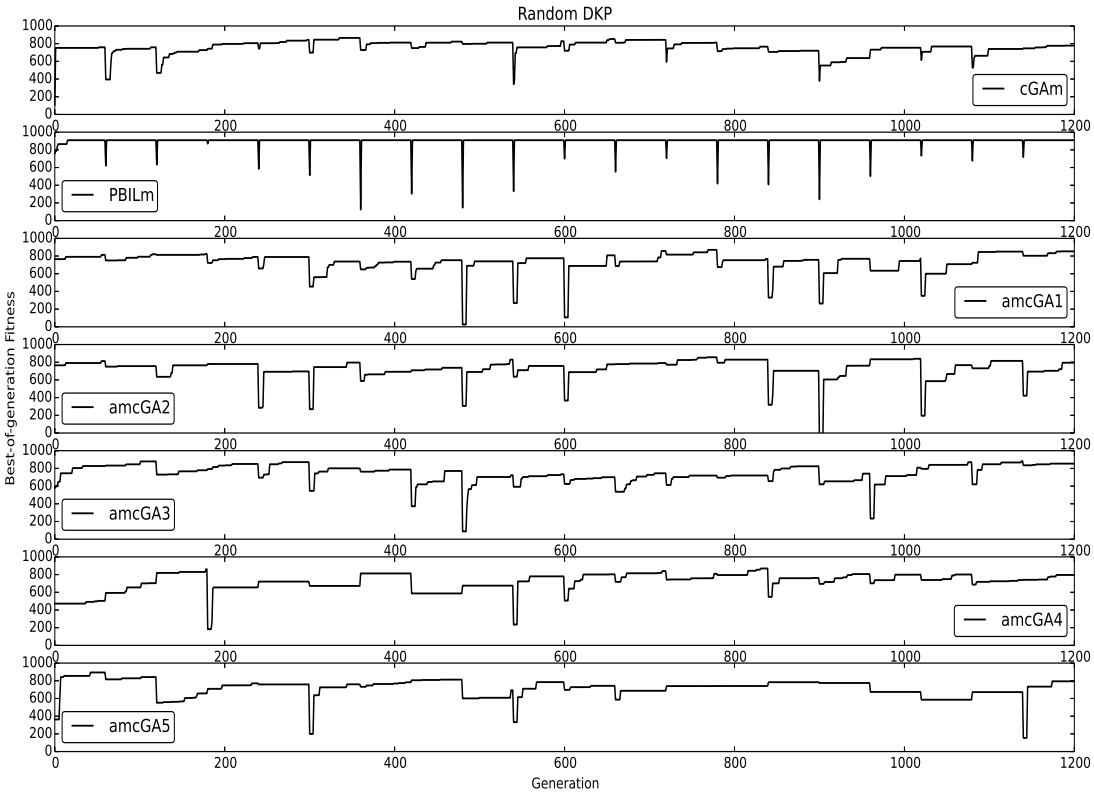

Fig. 9 Dynamic behaviour of all algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in a random environment.

to which they converged well. Also, among the amcGA variants, the amcGA3 shows better performance in DDUF1. The reason lies in the way the mutation scheme operates within amcGA, it ensures that the amcGA3 adapts to the changing environment regardless of the change severity. Also, the mutation scheme only increases (or decreases) $\overrightarrow{mP}$ by $(n - \frac{p_m}{2})$ which is determined by a random number r unlike the mutation scheme in PBILm which is determined by the probabilities in $\overrightarrow{P}$.

In Fig 10, it can be observed that on the cyclic DDUF2 (with and without noise), all amcGA variants show low performance when ρ= 0.1 to 0.5 but exhibit rapid increase in performance when ρ=1.0. This is due to the deceptive nature of DDUF2, since low-order building block inside the function do not clearly lead to a high-order building block and the amcGAs seems to be sensitive to low ρ. However, all amcGA variants cope well with high ρ (i.e. when $\rho = 1.0$) because the environment switches between two states which in turn gives more time for the algorithm to obtain a better solution suitable for the environment before the next change occurs.

Looking at DKP (bottom) of Fig 10, it can be observed that for all dynamic environments, the performance of all variants of the amcGA reduces as ρ increases. This can be considered normal, since an increase in ρ implies more severe environment changes. When the cyclic nature of the dynamic environment increases from cyclic to cyclic with noise, the performance of all amcGA variant (and cGAm) increases slightly. Despite the fact that a cyclic environment with noise is relatively more difficult than a cyclic environment, the amcGA variants showed better performance. But in the random environment, the performance of some of the amcGA variants dropped (when $\rho = 0.5$ and 1.0). This implies that even though the existence of noise in a cyclic environment may over weigh randomness (i.e. in terms of difficulty), it favours the performance of all amcGA variants.

Finally, from Figs. 1 to 10, it can be observed that the amcGA variants (i.e. amcGA to amcGA5) performed better on the DDUF2 problem, especially when τ is large (see Table 1). This implies that the performance of the amcGA not only depends on the dynamics of the environment but also on the DOP being considered.

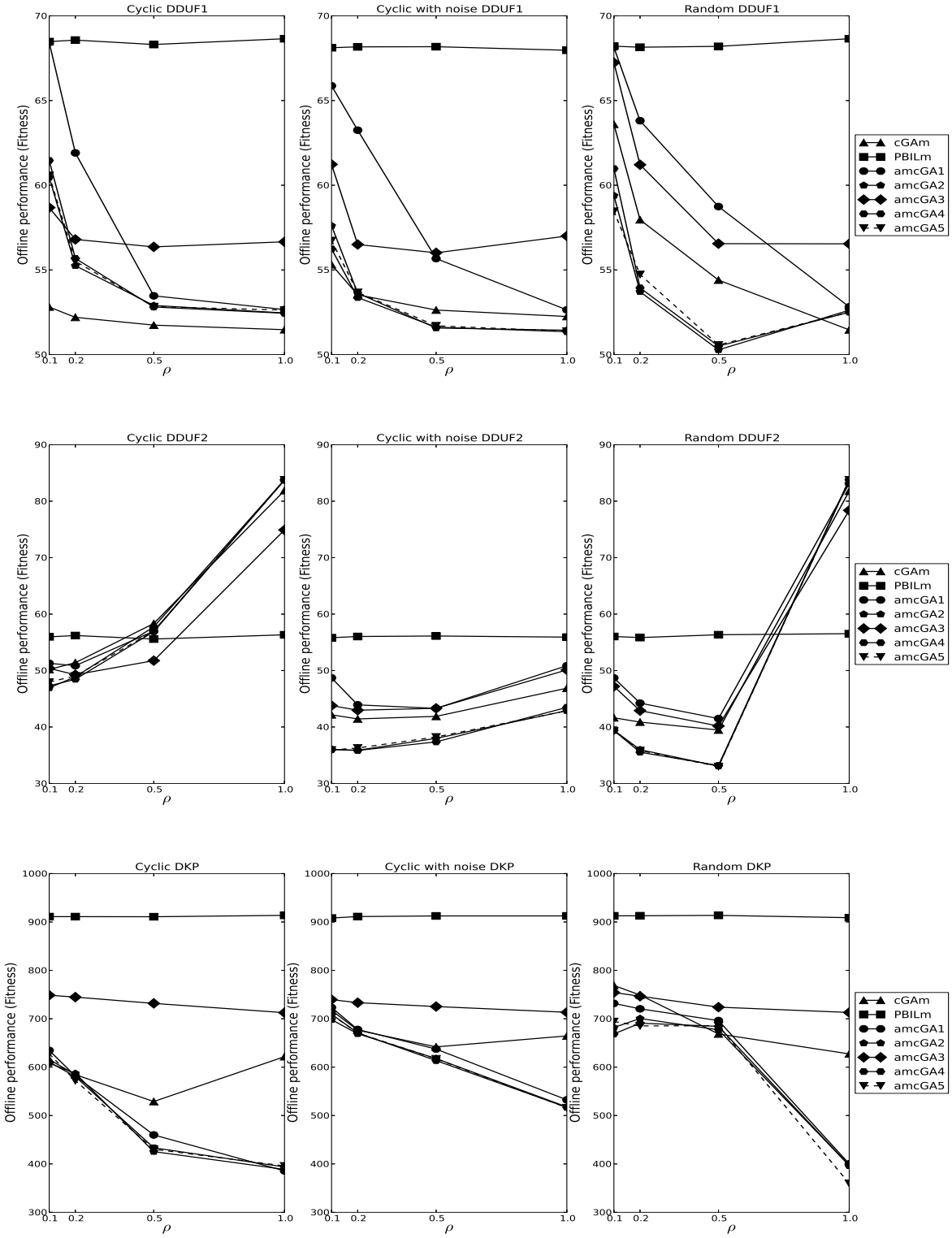

Fig. 10 Experimental results of all algorithms on DOPs in different dynamic environments (i.e. cyclic, cyclic with noise and random) with $\tau = 60$.

Table 1 Statistical results regarding the offline performance of the amcGA variants against other algorithms

Algorithms and DOPs	DDUF1												DDUF2												DKP											
Environment Dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau =60$				$\tau= 100$			
Cyclic ρ	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
amcGA1 - cGAm	+	+	∼	∼	+	+	∼	∼	+	+	+	∼	∼	∼	∼	+	∼	∼	-	+	+	∼	∼	∼	-	-	-	+	+	∼	-	-	+	+	+	-
amcGA1 - PBILm	-	-	-	-	∼	-	-	-	∼	-	-	-	-	-	∼	+	∼	-	∼	+	∼	-	∼	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - cGAm	+	∼	∼	∼	+	+	∼	∼	+	+	∼	∼	∼	∼	∼	+	-	∼	∼	+	∼	∼	∼	∼	-	-	-	-	∼	∼	-	-	+	+	+	-
amcGA2 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	∼	+	-	-	∼	+	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA3 - cGAm	∼	∼	∼	∼	+	+	∼	+	+	+	+	+	-	-	-	∼	∼	∼	-	-	∼	∼	-	-	+	-	+	∼	+	+	+	+	+	+	+	+
amcGA3 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	∼	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-
amcGA4 - cGAm	+	∼	∼	-	+	+	∼	∼	+	+	∼	∼	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼	-	-	-	-	∼	∼	-	-	+	+	+	-
amcGA4 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	∼	+	-	-	∼	+	-	-	∼	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA5 - cGAm	+	∼	∼	-	+	+	∼	∼	+	+	∼	v∼	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼	-	-	-	-	∼	∼	-	-	+	+	+	∼
amcGA5 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	∼	+	-	-	∼	+	-	-	∼	+	-	-	-	-	-	-	-	-	-	-	-	-
Cyclic with noise, ρ	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
amcGA1 - cGAm	+	+	∼	-	+	+	+	∼	+	+	+	+	∼	∼	∼	-	+	∼	∼	+	+	+	+	+	-	∼	-	-	+	+	+	-	+	+	+	-
amcGA1 - PBILm	-	-	-	-	-	-	-	-	∼	∼	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - cGAm	+	+	∼	-	+	∼	∼	∼	+	∼	∼	∼	-	∼	∼	-	∼	∼	∼	+	+	∼	∼	∼	-	-	-	-	+	+	∼	-	+	+	+	-
amcGA2 - PBILm	-	-	-	-	-	-	-	-	∼	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
amcGA3 - cGAm	∼	∼	∼	-	+	+	+	+	+	+	+	+	∼	-	∼	-	-	-	-	-	-	+	+	∼	+	+	+	∼	+	+	+	+	+	+	+	+
amcGA3 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
amcGA4 - cGAm	∼	∼	-	-	∼	∼	∼	∼	+	∼	∼	∼	-	-	∼	-	-	-	-	-	-	∼	∼	-	-	∼	-	-	+	+	∼	-	∼	+	+	-
amcGA4 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
amcGA5 - cGAm	∼	∼	-	-	∼	∼	∼	∼	+	∼	∼	∼	-	-	-	-	-	-	-	-	-	-	-	∼	-	∼	-	-	+	+	∼	-	∼	+	+	-
amcGA5 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Random, ρ	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0	0.1	0.2	0.5	1.0
amcGA1 - cGAm	-	∼	∼	∼	+	+	+	-	+	+	+	+	∼	∼	∼	+	+	+	∼	+	+	+	+	∼	-	-	-	-	∼	∼	+	-	+	∼	+	-
amcGA1 - PBILm	-	-	-	-	∼	-	-	-	+	∼	∼	-	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA2 - cGAm	-	-	∼	∼	∼	∼	-	-	∼	+	+	+	∼	-	-	+	∼	-	-	+	∼	∼	-	∼	-	-	-	-	∼	∼	+	-	+	-	+	-
amcGA2 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA3 - cGAm	-	∼	∼	∼	+	+	+	∼	+	+	+	+	+	∼	-	∼	+	∼	∼	∼	+	+	+	-	∼	∼	∼	∼	-	∼	+	+	+	∼	+	+
amcGA3 - PBILm	-	-	-	-	∼	-	-	-	+	-	-	-	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA4 - cGAm	-	-	-	-	∼	∼	-	-	∼	∼	∼	+	∼	-	-	+	∼	-	-	+	∼	-	-	∼	-	-	-	-	-	-	+	-	+	-	+	-
amcGA4 - PBILm	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-
amcGA5 - cGAm	-	-	-	-	∼	-	-	-	∼	∼	∼	+	∼	-	-	+	∼	-	-	+	∼	-	-	-	-	-	-	-	-	-	+	-	+	-	+	-
amcGA5 - PBILm	-	-	-	-	-	-	-	-	-	-	-		-	-	-	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Generally speaking, the experimental results indicate the amcGA variants can be considered when solving deceptive DOPs.

5 Conclusion and future work

Mutation is a double-edged sword, it ensures diversity and improves an algorithms ability to respond to changes in a dynamic environment. However, mutation can reduce the performance of an optimization algorithm if the mutation rate is too high and not controlled appropriately. The effect of change trend and different mutation schemes on the performance of the amcGA in dynamic environments was studied in this paper. From experimental results shown, several conclusion can be drawn on the overall performance of the algorithms:

First, the mutation schemes has a positive effect on the performance of the amcGA in dynamic environments as it ensures that information about an environment is retained and reused whenever the environment changes (instead of using a dedicated memory space and/or training data).

Second, statistical results highlight that variants of the amcGA display best performance on some DOPs (with respect to environment dynamics) when compared with cGAm and PBILm. On several cases, the change in environment had minimal effect on the performance of the amcGA variants while the algorithms tries to find a suitable solution. Also, the interaction between the change trend and mutation depends on the DOP (see Fig. 10 and Table 1).

Third, the addition of a change trend scheme to the amcGA improves the algorithms performance in dynamic environments. The change trend scheme ensures that the amcGA responds to dynamic changes based on the change pattern exhibited by the current working probability. Also, it allows the algorithm to update its mutation strategy using the change pattern. However, the effect may not be as strong as the effect of the hypermutation on the performance of the PBILm.

Finally, the mutation scheme embedded within all amcGA variants promotes diversity in dynamic environments i.e. it ensures that the population maintains its diversity while tackling the DOP and gradually move towards the optimal solution.

In general, this paper investigated the effects of the change trend and adaptive mutation schemes for the amcGA in dynamic environments. Based on results obtained, there are several future work relevant to this paper:

All amcGA variants are relatively easy to implement, especially in memory constrained application since all variants of the amcGA retains the small footprint of the cGA which allows direct implementation on memory constrained devices thus overcoming the limitations related to typical population-based dynamic optimization algorithms (e.g. PBILm).

The results obtained may be used to guide the design of compact dynamic optimization algorithms for tackling DOPs and compare the algorithms obtained with the amcGA variants as well as other EAs for DOPs. Further research will focus on using the schemes developed in this paper to solve real-world DOPs (implemented in memory constrained applications and embedded hardware) which includes further experimentations to identify possible limitations of the algorithms.

References

Affenzeller M, Wagner S (2003) A self-adaptive model for selective pressure handling within the theory of genetic algorithms. In: Computer Aided Systems Theory: EUROCAST 2003, Lecture Notes in Computer Science

Branke J (1999) Memory enhanced evolutionary algorithms for changing optimization problems. In: In Congress on Evolutionary Computation CEC99, Citeseer

Bui LT, Michalewicz Z, Parkinson E, Abello M (2012) Adaptation in dynamic environments: A case study in mission planning. Evolutionary Computation, IEEE Transactions on 16(2):190–209, DOI 10.1109/TEVC.2010.2104156

Chen Y, Wu Q (2011) Design and implementation of pid controller based on fpga and genetic algorithm. In: Electronics and Optoelectronics (ICEOE), 2011 International Conference on, vol 4, pp V4–308–V4–311, DOI 10.1109/ICEOE.2011.6013491

Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Research Laboratory, Washington, USA, URL citeseer.ist.psu.edu/cobb90investigation.html

Eiben A, Schut M, de Wilde A (2006) Boosting genetic algorithms with self-adaptive selection. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, pp 477–482, DOI 10.1109/CEC.2006.1688348

Goldberg DE (2002) The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell, MA, USA

Gongora M, Passow B, Hopgood A (2009) Robustness analysis of evolutionary controller tuning using real systems. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on, pp 606–613, DOI 10.1109/CEC.2009.4983001

Harik G, Lobo F, Goldberg D (1999) The compact genetic algorithm. Evolutionary Computation, IEEE Transactions on 3(4):287–297, DOI 10.1109/4235.797971

Harik G, Lobo F, Sastry K (2006) Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In: Pelikan M, Sastry K, CantPaz E (eds) Scalable Optimization via Probabilistic Modeling, Studies in Computational Intelligence, vol 33, Springer Berlin Heidelberg, pp 39–61, DOI 10.1007/978-3-540-34954-9 3

Hatzakis I, Wallace D (2006) Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '06, pp 1201–1208, DOI 10.1145/1143997.1144187, URL http://doi.acm.org/10.1145/1143997.1144187

Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. Evolutionary Computation, IEEE Transactions on 9(3):303–317, DOI 10.1109/TEVC.2005.846356

Larraanaga P, Lozano JA (2001) Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers, Norwell, MA, USA

Martins JP, Fonseca CM, Delbem AC (2014) On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem. Neurocomputing 146:17–29

Morrison RW, De Jong KA (2000) Triggered hypermutation revisited. In: Evolutionary Computation, 2000. Proceedings of the 2000 Congress on, IEEE, vol 2, pp 1025–1032

Passow B, Gongora M, Coupland S, Hopgood A (2008) Real-time evolution of an embedded controller for an autonomous helicopter. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, pp 2538–2545, DOI 10.1109/CEC.2008.4631139

Pedersen GK, Yang Z (2006) Multi-objective pid-controller tuning for a magnetic levitation system using nsga-ii. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '06, pp 1737–1744, DOI 10.1145/1143997.1144280, URL http://doi.acm.org/10.1145/1143997.1144280

Pelikan M, Goldberg D, Lobo F (2000) A survey of optimization by building and using probabilistic models. In: American Control Conference, 2000. Proceedings of the 2000, vol 5, pp 3289–3293 vol.5, DOI 10.1109/ACC.2000.879173

Sastry K, Abbass HA, Goldberg D (2005) Substructural niching in non-stationary environments. In: AI 2004: Advances in Artificial Intelligence,

Springer, pp 873–885

Shah R, Reed P (2011) Comparative analysis of multiobjective evolutionary algorithms for random and correlated instances of multiobjective d-dimensional knapsack problems. European Journal of Operational Research 211(3):466–479

Simões A, Costa E (2009a) Improving prediction in evolutionary algorithms for dynamic environments. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, pp 875–882

Simões A, Costa E (2009b) Prediction in evolutionary algorithms for dynamic environments using markov chains and nonlinear regression. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, pp 883–890

Uzor C, Gongora M, Coupland S, Passow B (2014a) Adaptive mutation in dynamic environments. In: Computational Intelligence (UKCI), 2014 14th UK Workshop on, pp 1–7, DOI 10.1109/UKCI.2014.6930175

Uzor CJ, Gongora M, Coupland S, Passow BN (2014b) Real-world dynamic optimization using an adaptive-mutation compact genetic algorithm. In: Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on, pp 17–23, DOI 10.1109/CIDUE.2014.7007862

Woldesenbet Y, Yen G (2009) Dynamic evolutionary algorithm with variable relocation. Evolutionary Computation, IEEE Transactions on 13(3):500–513, DOI 10.1109/TEVC.2008.2009031

Yang S (2008) Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evol Comput 16(3):385–416, DOI 10.1162/evco.2008.16.3.385, URL http://dx.doi.org/10.1162/evco.2008.16.3.385

Yang S, Richter H (2009) Hyper-learning for population-based incremental learning in dynamic environments. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on, pp 682–689, DOI 10.1109/CEC.2009.4983011

Yang S, Tinos R (2008) Hyper-selection in dynamic environments. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, pp 3185–3192, DOI 10.1109/CEC.2008.4631229

Yang S, Yao X (2005) Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Comput 9(11):815–834, DOI 10.1007/s00500-004-0422-3, URL http://dx.doi.org/10.1007/s00500-004-0422-3

Yang S, Yao X (2008) Population-based incremental learning with associative memory for dynamic environments. Evolutionary Computation, IEEE Transactions on 12(5):542–561, DOI 10.1109/TEVC.2007.913070

Yang S, Jiang Y, Nguyen TT (2013) Metaheuristics for dynamic combinatorial optimization problems. IMA Journal of Management Mathematics 24(4):451–480

Yu E, Suganthan P (2009) Evolutionary programming with ensemble of explicit memories for dynamic optimization. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on, pp 431–438, DOI 10.1109/CEC.2009.4982978

Yu X, Tang K, Yao X (2008) An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, pp 1141–1147, DOI 10.1109/CEC.2008.4630940

Zhu H, Jiao L, Pan J (2006) Multi-population genetic algorithm for feature selection. In: Jiao L, Wang L, Gao X, Liu J, Wu F (eds) Advances in Natural Computation, Lecture Notes in Computer Science, vol 4222, Springer Berlin Heidelberg, pp 480–487, DOI 10.1007/11881223 59, URL http://dx.doi.org/10.1007/11881223 59