Soft Comput (2016) 20:3097–3115
DOI 10.1007/s00500-016-2195-x

FOCUS

Adaptive-mutation compact genetic algorithm for dynamic environments

Chigozirim J. Uzor[1] · **Mario Gongora**[1] · **Simon Coupland**[1] · **Benjamin N. Passow**[2]

Published online: 7 June 2016

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 hyperselection, 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 limitations 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 show that the change trend and mutation schemes have an impact on the performance of the amcGA in dynamic environment and also indicate 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)

Communicated by D. Neagu.

✉ Chigozirim J. Uzor
juzor@dmu.ac.uk

[1] Centre for Computational Intelligence (CCI), De Montfort University, Leicester, UK

[2] Centre for Computational Intelligence (CCI), De Montfort University's ITS Research Group (DIGITS), De Montfort University, Leicester, UK

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 Nelson et al. (2009) and 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 represent potential solutions that can be applied to different environmental circumstances (Branke et al. 2000; Yang 2008). Standard EAs have been successful in solving

optimization problem in static environments, e.g. Deb et al. (2002), Juang (2004) and Passow et al. (2008), but when confronted with DOPs the algorithms performance is limited (see Yang and Tinós (2007)). Also the standard EAs employ a strong selection policy based on feedback which gradually reduces diversity during an optimization process (Cobb and Grefenstette 1993). 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 (Grefenstette et al. 1992).

Recently, Uzor et al. (2014a) investigated a compact genetic algorithm (cGA) for DOPs known as adaptive-mutation cGA (amcGA). They introduced a mechanism whereby the mutation rates are directly linked to the detected rate of change (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; Sect. 2 reviews existing EAs for dynamic environments, Sect. 3 introduces a background knowledge of the cGA and details the amcGA as well as its variants, and Sect. 4 describes the set-up scene for all experiments and performance analysis. Finally, Sect. 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 (e.g. Higuchi et al. (1999) and Bektas (2006)). 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. For more than 20 years, static optimization has been an active area of research. However, the inclusion of time dependency by Goldberg and Smith (1987) created a distinct degree of difficulty. 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:

- Using implicitly or explicitly defined memory to store and reuse useful information so as to adapt the EA whenever a change occurs (Goldberg and Smith 1987; Dasgupta and McGregor 1993, 1992; Yang and Yao 2008).
- Creating a multi-population to distribute the search force in the search space (Branke et al. 2000; Zhu et al. 2006).
- Promote diversity by inserting random immigrants back in the population (Grefenstette et al. 1992; Yu et al. 2008) and
- Adjusting genetic operators adaptively (Morrison and De Jong 2000; Eiben et al. 2006).

Apart from the approaches mentioned above, for an EA to function properly the genetic operators need to be tuned/defined properly as it affects performance and is problem dependent. This can be achieved in three ways: (1) Deterministic method, which 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. (3) Self-adaptive method, where the mechanism for updating the strategy parameter is implicitly defined (see Eiben et al. (1999) and Eiben and Smit (2012)).

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 (Goldberg 1989). 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. This means that 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 hyperselection 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). They implemented 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 Yang (2008), a memory and elitism-based immigrants approach for GAs in dynamic environment were 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. In general, certain techniques are suitable for certain environments. 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).

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. Numerous dynamic techniques have been proposed to tackle dynamic problems effectively (Nguyen et al. 2012). However, the complex structure of some the existing dynamic optimization algorithms makes them unsuitable for solving real-time DOPs on-board embedded hardware system with limited memory. Therefore, a compact dynamic approach is investigated in this paper.

3 cGA for DOPs

There are some optimization problems that limit the application of traditional optimization algorithm due to hardware limitations. 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 compact algorithm is required. The compact genetic algorithm (cGA) offers the advantage of being computationally efficient (i.e requires less memory and execution time) (see Harik et al. (1999) and Mininno et al. (2008)).

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:

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

where l is the binary-encoding or chromosome length and $Pi \in \{0, 1\}, (i = 1, \ldots, l)$. 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 (Ahn and Ramakrishna 2003). As a result, modifications to the original algorithm have been proposed so as to enable it to tackle DOPs.

To address the convergence problem, several approaches have been developed to reintroduce 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.

3.1 Change detection

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

$$C_\mathrm{d} = \mathrm{e}^{-a} \tag{3}$$

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

where c is the mean (and is set to 1.0), σ represents standard deviation of the change detection 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 $t+1$:

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

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:

$$p_\mathrm{m} = m_\mathrm{l} + (C_\mathrm{d} - d_\mathrm{l}) \times \left(\frac{m_\mathrm{h} - m_\mathrm{l}}{d_\mathrm{h} - d_\mathrm{l}}\right), \quad p_\mathrm{m} \in [0.01, 0.5] \tag{6}$$

where $m_\mathrm{l} = 0.01$ is low probability of mutation, $m_\mathrm{h} = 0.5$ is high probability of mutation, $d_\mathrm{l} = 0.0$ is low degree of change and $d_\mathrm{h} = 1.0$ 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 (i.e. elite) since the probability vector represents a distribution of the population.

Suppose at generation t an elite solution E_s with fitness $f(E_\mathrm{s}, t)$ 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. $f(E_\mathrm{s}, t+1)$. 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 $r = \mathrm{rand}(0.0, 1.0)$ is generated and then compared with p_m and $\overrightarrow{mP}$ is mutated as follows:

amcGA1

$$mP_i' = \begin{cases} \mathrm{rand}(C_\mathrm{d}, mP_i) & \text{if } r < p_\mathrm{m}, \quad (7a) \\ mP_i & \text{if } r > p_\mathrm{m}. \quad (7b) \end{cases}$$

amcGA2

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

amcGA3

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

The diversity techniques by Vavak et al. (1996) differ in that it makes use of a variable local search around the locations in the search space which are represented by chromosomes before a change occurs. It is triggered when the time-average best performance of the population falls below a defined threshold. When triggered, all mutation and crossover operations are suspended.

However, the amcGA detects and measures the degree of change whenever there is a drop in the fitness of an elite solution. Then, the probability of mutation is directly linked to the measured degree of change. The amcGA has been design to be suitable for systems and applications with limited computational resources and time constraints. This is because the amcGA maintains the small footprint of the cGA without a significant increase in memory requirements, unlike the algorithm presented by Vavak et al. (1996), which is not efficient in such systems and applications.

Sometimes, changes in dynamic environments may exhibit some trends (or pattern). In such case, it might be beneficial to try to use these change trends to improve the algorithms

response to subsequent changes in such dynamic environment. Some studies have been made following this idea by exploiting the predictability of dynamic environments (e.g. 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 require 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:

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

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

amcGA4

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

amcGA5

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

where $s = \left(n - \frac{p_{\text{m}}}{2}\right)$ and l is the binary string length. It is important to state that the change trend scheme was applied to amcGA2 and amcGA3 (which yields amcGA5 and amcGA4, respectively) to study the effect of T_{chg} on the performance of 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.

The adaptive mutation scheme presented in this section is somewhat similar to the well-established covariance matrix adaptation evolution strategy (CMA-ES) (Hansen and Ostermeier 2001, 1996; Hansen and Kern 2004), which is a de facto standard in continuous domain evolutionary optimization (in static environments). Both algorithms share same idea of storing information about a candidate solution which is used to update/bias the solution sampling process. However, in the CMA-ES evolution cycle, after the evaluation and ranking of the solutions, their relative steps are used to update the parameters of the sampling distribution. This process is repeated through out the evolution cycle until a termination criterion is met. On the other, the amcGA only updates $\overrightarrow{mP}$ when an elite is discovered and samples a mutated elite from it when a dynamic change occurs. This sampling and update process is only repeated for a defined number of iterations. Also, the amcGA is developed for combinatorial optimization problems and the embedded adaptive mutation scheme can be considered as an improved form of the popular hypermutation scheme.

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 amcGA 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

The dynamic test problems used in the experiments are presented in this section. Three 100-bit binary-encoded problems are selected as stationary functions and are described below:

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 (Goldberg 2002). These type of functions return the number of ones in a binary string (i.e. unitation function of binary string). Two decomposable unitation-based functions 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.

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.

$$f_{\mathrm{DUF1}}(x) = u(x) \tag{16}$$

where $u(x)$ is number of $1's$ in a building block.

DUF2 is a fully deceptive function (Goldberg 2002), which is 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 (Whitley 1991). This property makes it much more difficultly for EAs to solve DUF2 than DUF1.

$$f_{\mathrm{DUF2}}(x) = \begin{cases} 3 - u(x) & \text{if } u(x) < 4 \quad (17a) \\ 4 & \text{otherwise} \quad (17b) \end{cases}$$

Using the dynamic benchmark generator discussed in Sect. 4.1, dynamic test environments are constructed from the DUFs and are referred to as dynamic DUFs (i.e. denoted as DDUF1 and DDUF2). The DDUFs 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).

4.2.2 Dynamic knapsack problem

The knapsack problem is a classic NP-complete combinatorial optimization problem that has been rigorously studied by the EA community in the last few decades (Branke et al. 2006; Wang et al. 2009; Rohlfshagen and Yao 2009). The main aim of this problem is to fill a knapsack with the best subset of items among a larger set 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:

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

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

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:

$$w_i = \text{uniformly random integer}[2, 20] \tag{20}$$

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

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 judge how much the candidate solution exceeds the knapsack capacity. The penalty function is defined as follows:

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

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

These dynamic test problems have been chosen to evaluate the performance of the amcGA in different dynamic scenarios. Specifically the DDUFs and DKP are academic combinatorial problems suitable for evaluating dynamic (binary-encoded) optimization algorithms. The DDUFs has an increasing difficulty for the amcGA in the order from DDUF1 to DDUF2. This is due to the deceptive property inherent in DDUF2 which makes it a difficult problem for dynamic EAs (Whitley 1991; Goldberg 2002). The DKP is another difficult problem, because the profit and weight of each item selected change over time based on the XOR mask (i.e. $\overrightarrow{M}$) in Eq. 13.

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 schemes presented in Sect. 3 with a probability-based incremental learning algorithm with hypermutation (denoted as PBILm) (Yang and Richter 2009) and a cGA with hypermutation (denoted as cGAm). The hypermutation scheme embedded in cGAm generates a mutated version of an elite solution for tournament selection whenever a dynamic change occurs. The elite solution is replaced if it is outperformed by the mutated elite. This way the $\overrightarrow{P}$ in cGAm is only updated by the elite solution.

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 Sect. 3, probability of mutation (for the PBILm) $p_{\mathrm{m}} = 0.05$ 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 $p_{\mathrm{m}}^{l} = 0.05$ for normal generations and a high value $p_{\mathrm{m}}^{h} = 0.3$ for interim generations when the hypermutation scheme is triggered due to change in environment and this lasts for five generations, i.e. $g_{hm} = 5$. Three kinds of dynamic environments were constructed (i.e. cyclic, cyclic with noise and random) using the dynamic problem generator discussed in Sect. 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:

$$\overline{F}_{\mathrm{BOG}} = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{N} \sum_{j=1}^{N} F_{\mathrm{BOG}_{ij}} \quad (23)$$

where $F_{\mathrm{BOG}_{ij}}$ expresses the fitness value of the best solution at generation i of run j, $G = 20 \times \tau$ is the total number of generation for a run, $N = 30$ is the total number of runs and $\overline{F}_{\mathrm{BOG}}$ 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 Figs. 1, 2, 3 and 4, respectively. The corresponding statistical tests are shown in Tables 1 and 2, where Kruskal–Wallis tests were applied followed by pairwise comparisons using Wilcoxon rank-sum tests with the Bonferroni correction. In Tables 1 and 2, the result regarding Alg. 1–Alg. 2 is shown as "+", "−" and "∼" when Alg. 1 is significantly 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 is plotted in Figs. 1, 2 and 3 (with $\tau = 60$ and $\rho = 0.2$).

From Figs. 1, 2, 3 and 4, Tables 1, 2, 3, several behaviours can be observed and are discussed in next section from two aspects: (1) regarding performance based on $\overline{F}_{\mathrm{BOG}}$ 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

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 two candidate solutions every generation. With the increasing 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 Fig. 4 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 $\overrightarrow{P}$. The performance of the PBILm was expected to be better than that of the amcGA and cGAm. Due to its algo-

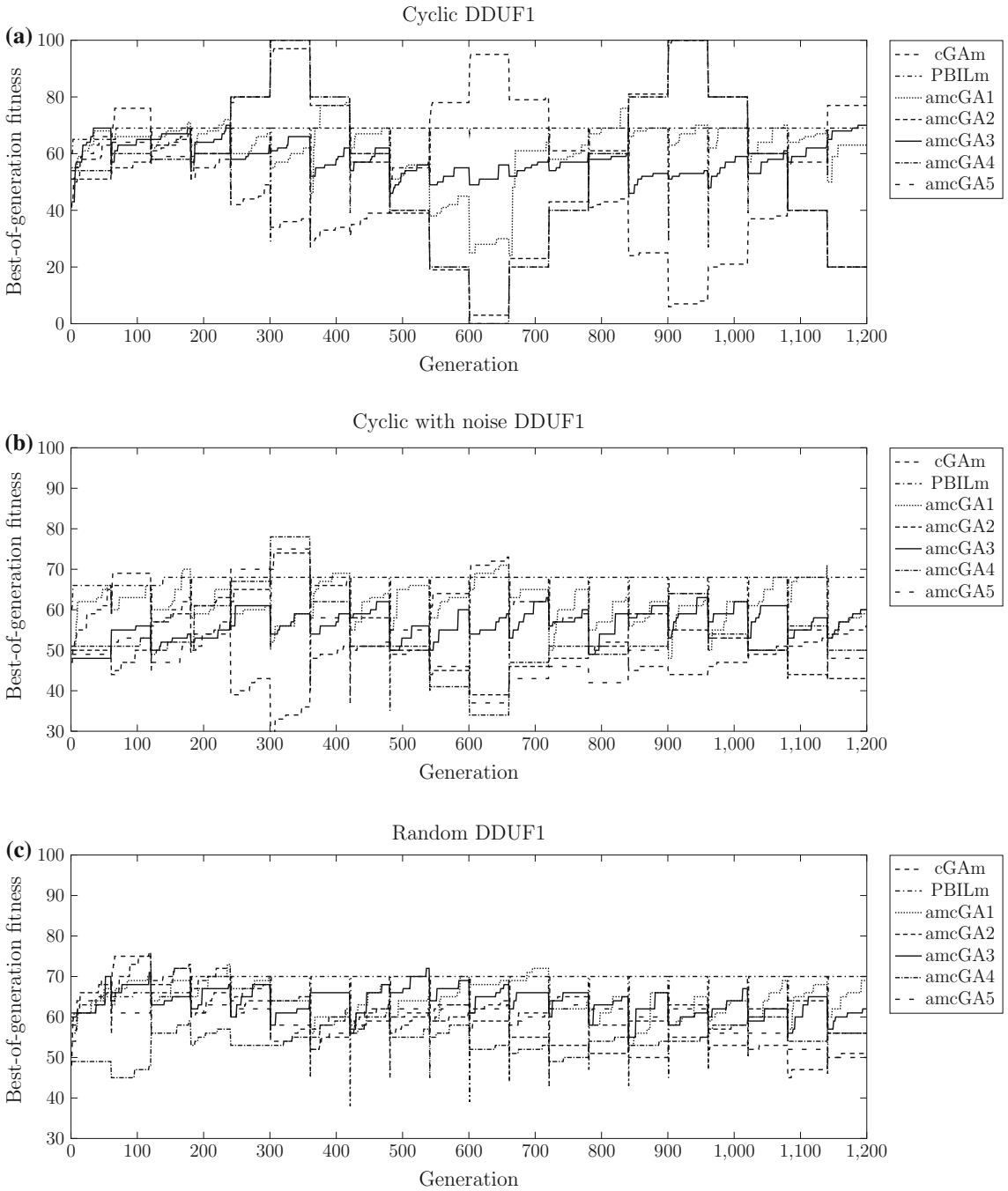

Fig. 1 Dynamic behaviour of all algorithms on DDUF1 with $\tau = 60$ and $\rho = 0.2$ in different dynamic environment, i.e. **a** cyclic, **b** cyclic with noise and **c** random environments

rithmic structure, the PBILm generates series of solutions (population) every generation until a solution with better fitness is discovered. Therefore, it is important to state that this comparison was carried out in order to see which of the competing algorithm was less behind the PBILm in terms of the overall performance (see Table 3).

Second, cGAm outperforms some of the amcGA variants in some of the DOPs (see Fig. 1b, c). 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. 1a. Also for some dynamic settings, cGAm shows similar performance to some of the amcGA variants (see Figs. 1c, 2a). It must be noted that the cGAm and amcGAs are modified variants of the cGA suitable for solving DOPs.

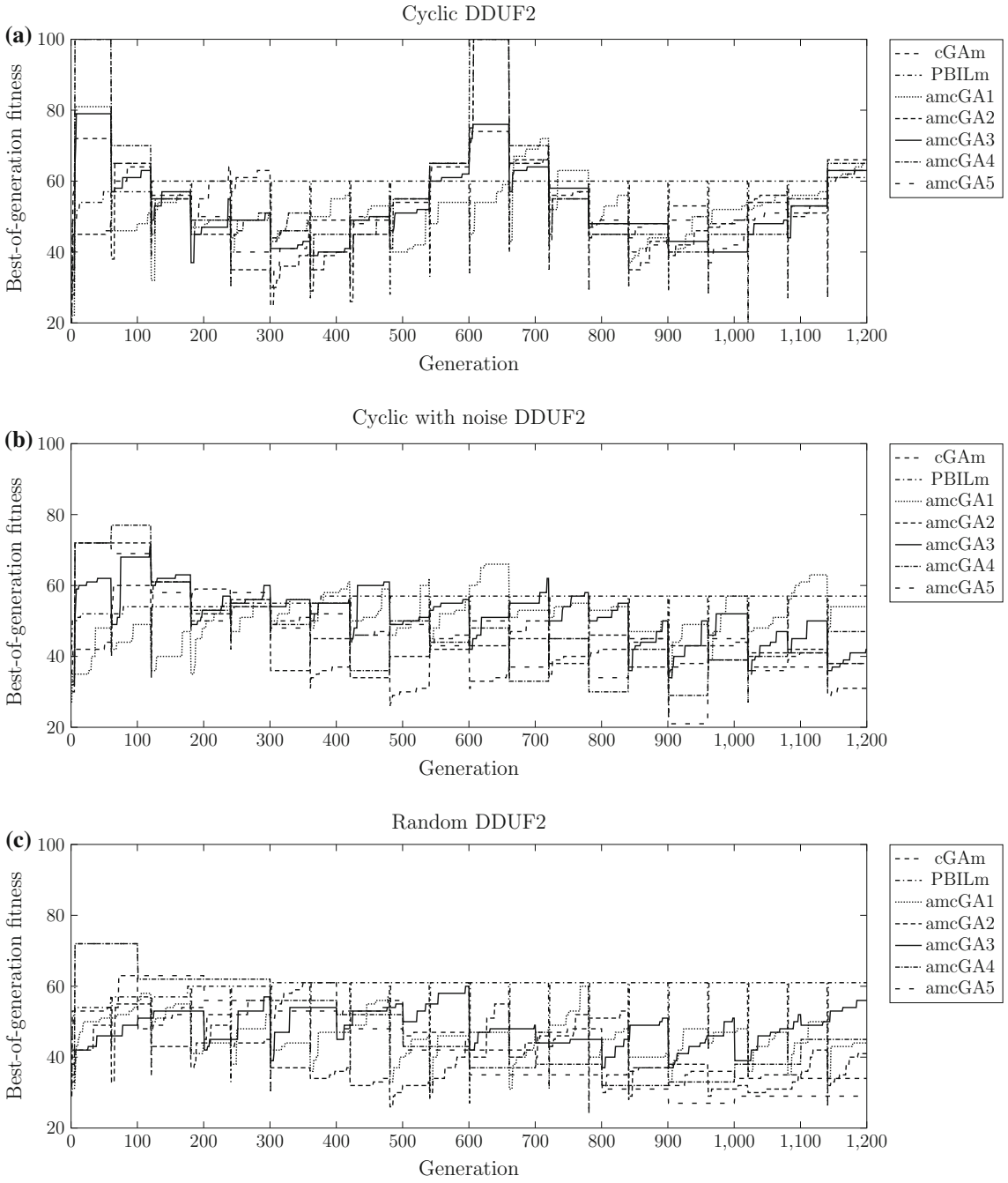

Fig. 2 Dynamic behaviour of all algorithms on DDUF2 with $\tau = 60$ and $\rho = 0.2$ in different dynamic environment, i.e. **a** cyclic, **b** cyclic with noise and **c** random environments

Third, among the five variants of the amcGA, amcGA1 and amcGA3 exhibit better performance (see Table 2). From Figs. 1, 2 and 3, it can be observed that amcGA3 performs better than amcGA1. This is as a result of how the mutation scheme within both algorithms alters $\overrightarrow{mP}$. The amcGA3 mutation scheme either increase or decrease $\overrightarrow{mP}$ at a reduced scale (see Eq. 9), while amcGA1 is a randomized mutation based on the current values of elements of $\overrightarrow{mP}$ and the degree of change C_d [see Eq. (7)]. Also, amcGA3 shows stable performance across different environment dynamics. Performance of all amcGA variants based on $\overline{F}_{BOG}$ is shown on Figs. 1, 2 and 3 for different environment dynamics. Although figures show 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 Kruskal–Wallis tests followed by

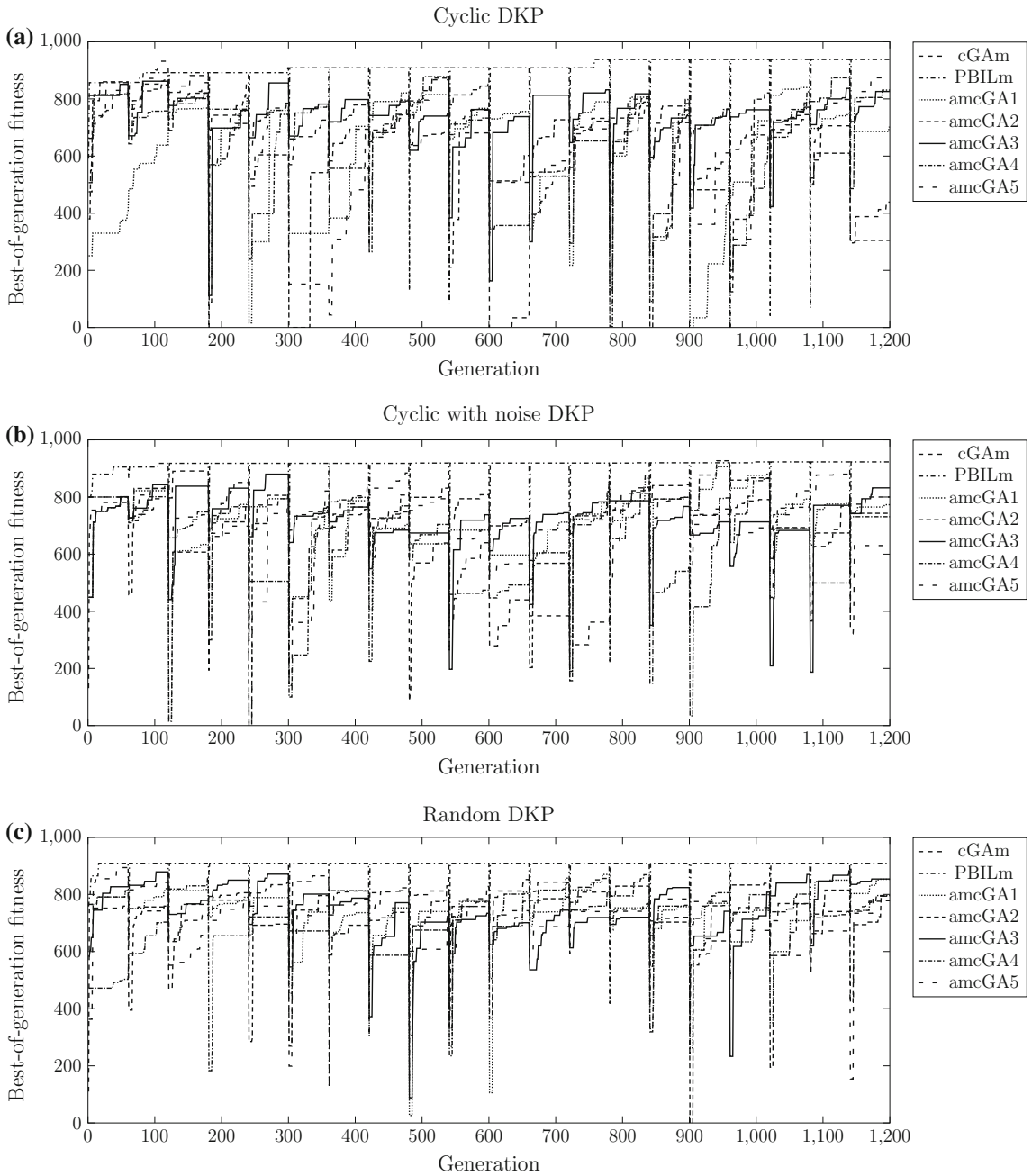

Fig. 3 Dynamic behaviour of all algorithms on DKP with $\tau = 60$ and $\rho = 0.2$ in different dynamic environment, i.e. **a** cyclic, **b** cyclic with noise and **c** random environments

pairwise comparisons using Wilcoxon rank-sum tests with the Bonferroni correction, several results can be observed. On several environment dynamics, the performance of the amcGA1 and amcGA3 is better than that of the cGAm (see Tables 1 and 2). Also when the ρ is low and τ is low to medium, most of the amcGA variants outperformed the cGAm (and the PBILm in some environment dynamics). This behaviour is as a result of how the amcGA handles its probability vector.

The amcGA3 maintains a moderate convergence rate as it explores the search space (see Figs. 1b, c, 3a). 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

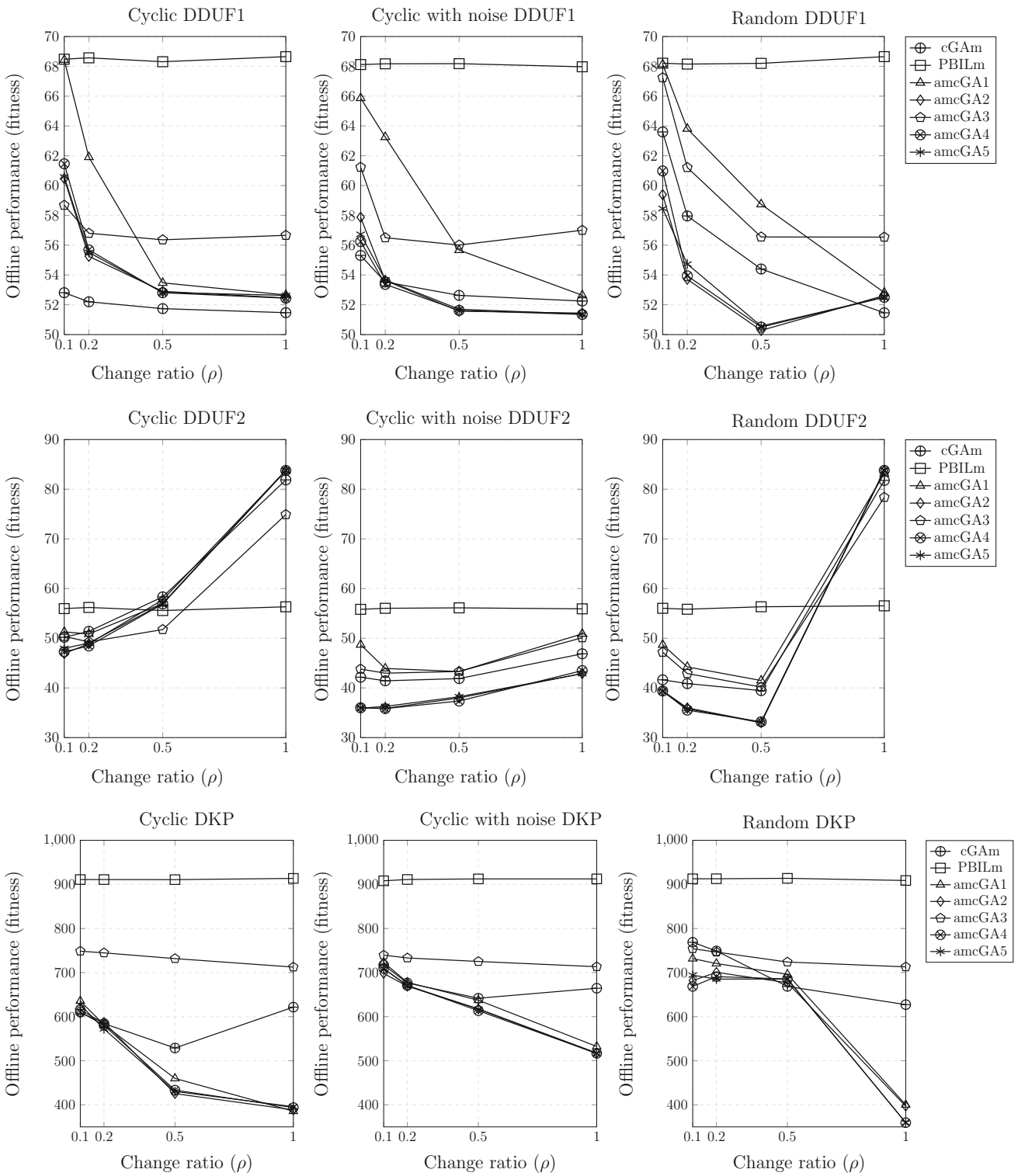

Fig. 4 Experimental results of all algorithms on DOPs in different dynamic environments (i.e. cyclic, cyclic with noise and random) with $\tau = 60$

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

Algorithms and DOPs	DDUF1												DDUF2												DKP											
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
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
Cyclic																																				
amcGA1–cGAm	+	+	∼	∼	+	+	∼	∼	+	+	+	∼	∼	∼	∼	+	∼	∼	−	+	+	∼	∼	∼	−	−	−	+	+	∼	−	−	+	+	+	−
amcGA1–PBILm	−	−	−	−	∼	−	−	−	∼	−	−	−	−	−	∼	+	∼	−	∼	+	∼	−	∼	+	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–cGAm	+	∼	∼	∼	+	+	∼	∼	+	+	∼	∼	∼	∼	∼	+	−	∼	∼	+	∼	∼	∼	∼	−	−	−	−	∼	∼	−	−	+	+	+	−
amcGA2–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	+	+	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3–cGAm	∼	∼	∼	∼	+	+	∼	+	+	+	+	+	−	−	−	∼	∼	∼	−	−	∼	∼	−	−	+	−	+	∼	+	+	+	+	+	+	+	+
amcGA3–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	∼	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−
amcGA4–cGAm	+	∼	∼	−	+	+	∼	∼	+	+	∼	∼	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼	−	−	−	−	∼	∼	−	−	+	+	+	−
amcGA4–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+	−	−	−	−	−	−	−	−	−	−	−	−
amcGA5–cGAm	+	∼	∼	−	+	+	∼	∼	+	+	∼	∼	∼	∼	∼	+	∼	∼	∼	+	∼	∼	∼	∼	−	−	−	−	∼	∼	−	−	+	+	+	∼
amcGA5–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	+	−	−	∼	+	−	−	∼	+	−	−	−	−	−	−	−	−	−	−	−	−
Cyclic with noise																																				
amcGA1–cGAm	+	+	∼	−	+	+	+	∼	+	+	+	+	∼	∼	∼	−	+	∼	∼	+	+	+	+	+	−	∼	−	−	+	+	+	−	+	+	+	−
amcGA1–PBILm	−	−	−	−	−	−	−	−	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–cGAm	+	+	∼	−	+	∼	∼	∼	+	∼	∼	∼	−	∼	∼	−	∼	∼	∼	+	+	∼	∼	∼	−	−	−	−	+	+	∼	−	+	+	+	−
amcGA2–PBILm	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA3–cGAm	∼	∼	∼	−	+	+	+	+	+	+	+	+	∼	−	∼	−	−	−	−	−	−	+	+	+	∼	+	+	∼	+	+	+	+	+	+	+	+
amcGA3–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA4–cGAm	∼	∼	−	−	∼	∼	∼	∼	+	∼	∼	∼	−	−	∼	−	−	−	−	−	−	∼	∼	−	−	∼	−	−	+	+	∼	−	∼	+	+	−
amcGA4–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA5–cGAm	∼	∼	−	−	∼	∼	∼	∼	+	∼	∼	∼	−	−	−	−	−	−	−	−	−	−	−	∼	−	∼	−	−	+	+	∼	−	∼	+	+	−
amcGA5–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
Random																																				
amcGA1–cGAm	−	∼	∼	∼	+	+	+	−	+	+	+	+	∼	∼	∼	+	+	+	∼	+	+	+	+	∼	−	−	−	−	∼	∼	+	−	+	∼	+	−
amcGA1–PBILm	−	−	−	−	∼	−	−	−	+	∼	∼	−	−	−	−	+	−	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–cGAm	−	−	∼	∼	∼	∼	−	−	∼	+	+	+	∼	−	∼	+	∼	−	−	+	∼	∼	−	∼	−	−	−	−	∼	∼	+	−	+	−	+	−
amcGA2–PBILm	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	−	−	−	+	−	−	−	+	−	−	−	−	−	−	−	−	−	−	−	−

Table 1 continued

Algorithms and DOPs	DDUF1										DDUF2										DKP									
Environment dynamics	$\tau = 20$					$\tau = 60$					$\tau = 100$					$\tau = 20$					$\tau = 60$					$\tau = 100$				
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	
amcGA3–cGAm	−	≈	≈	≈		+	+	+	≈		+	+	+	+		+	≈	−	≈		+	≈	≈	≈		+	≈	+	+	
amcGA3–PBILm	−	−	−	−		≈	−	−	−		+	−	−	−		−	−	−	+		−	−	−	−		−	−	−	−	
amcGA4–cGAm	−	−	−	−		≈	≈	−	−		≈	≈	≈	+		≈	−	−	+		−	−	−	−		−	−	+	−	
amcGA4–PBILm	−	−	−	−		−	−	−	−		−	−	−	+		−	−	−	≈		−	−	−	−		−	−	−	−	
amcGA5–cGAm	−	−	−	−		≈	−	−	−		≈	≈	≈	+		≈	−	−	+		−	−	−	−		−	−	+	−	
amcGA5–PBILm	−	−	−	−		−	−	−	−		−	−	−	+		−	−	−	≈		−	−	−	−		−	−	−	−	

the current best solution generated by $\overrightarrow{P}$, thereby replacing $\overrightarrow{P}$ with $\overrightarrow{mP}$. This can be considered as an advantage over the hypermutation scheme. The $\overrightarrow{P}$ in cGAm is mainly updated based on the solutions sampled from it. This implies that genetic diversity is encouraged at individual level since the mutation scheme is applied directly to a candidate solution to create another for selection.

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, i.e. cyclic DDUF1 and DKP (with respective environment dynamics). This behaviour is as a result of the change trend scheme within the algorithm. Although this scheme does not make use 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 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) and 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 hypermutated 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 (see Fig. 3a, b). The change trend scheme within amcGA4 and amcGA5 makes use of change patterns exhibited by the current working probability vector (i.e. $\overrightarrow{P}$) to mutate/alter $\overrightarrow{mP}$. This way, both amcGA4 and amcGA5 respond to dynamic changes based on how often new elites are obtained. One limitation of the amcGA4 and amcGA5 is the lack of an explicitly defined memory allocation to store change trends such that various change patterns can be reused. This means the performance of the amcGA4 and amcGA5 is problem dependent. In addition, the structure of the amcGA4 and amcGA5 permits their application to DOPs with limited computational resources specifically limited in the amount of available memory.

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 is shown in Fig. 4, 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

Table 2 Statistical results regarding the offline performance of the amcGA variants

Algorithms and DOPs	DDUF1												DDUF2												DKP											
Environment dynamics	$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$				$\tau = 20$				$\tau = 60$				$\tau = 100$			
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
Cyclic																																				
amcGA1–amcGA2	+	+	+	+	+	∼	∼	+	+	+	+	+	∼	+	+	−	−	∼	+	+	+	−	+	+	+	+	+	+	+	+	+	−	+	+	+	+	+
amcGA1–amcGA3	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	∼	−
amcGA1–amcGA4	+	−	−	+	+	−	−	−	−	+	−	∼	+	∼	−	+	−	+	−	+	−	∼	∼	+	+	∼	∼	−	∼	∼	−	−	+	−	−	+
amcGA1–amcGA5	−	+	∼	−	+	−	+	−	+	∼	−	+	+	−	−	+	+	−	+	+	+	−	+	−	−	∼	∼	−	+	−	∼	−	+	+	∼	∼
amcGA2–amcGA3	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–amcGA4	−	∼	−	+	∼	−	∼	∼	−	∼	∼	+	∼	∼	+	−	−	−	−	+	−	+	−	−	−	+	−	−	−	∼	−	−	∼	∼	−	−
amcGA2–amcGA5	+	−	−	−	−	−	∼	−	−	+	−	−	∼	−	−	−	∼	−	−	+	−	∼	−	+	−	∼	∼	−	∼	+	−	−	+	−	∼	−
amcGA3–amcGA4	+	+	+	+	+	+	+	−	+	+	+	∼	+	+	+	∼	∼	+	+	∼	−	∼	+	+	+	+	+	∼	+	+	+	+	∼	+	+	+
amcGA3–amcGA5	+	+	∼	+	+	+	∼	+	+	+	+	+	+	+	∼	+	+	+	∼	+	+	+	+	∼	+	+	+	∼	∼	+	+	+	+	+	+	∼
amcGA4–amcGA5	∼	∼	∼	+	∼	+	∼	+	∼	∼	∼	−	∼	+	∼	+	+	−	+	+	∼	+	∼	+	+	+	+	∼	∼	∼	+	+	∼	∼	+
Cyclic with noise																																				
amcGA1–amcGA2	−	+	+	−	+	−	+	−	+	+	∼	+	+	∼	∼	+	+	+	∼	−	+	+	+	+	+	∼	+	+	+	+	+	∼	−	+	+	−
amcGA1–amcGA3	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	∼	−	−
amcGA1–amcGA4	−	+	−	∼	+	−	+	−	+	−	∼	−	∼	−	+	−	∼	−	−	+	+	∼	∼	∼	+	−	∼	+	+	−	∼	∼	+	∼	+	∼
amcGA1–amcGA5	+	+	∼	−	∼	−	∼	−	+	−	−	−	+	−	−	−	+	−	−	+	−	∼	∼	−	∼	−	−	∼	+	−	∼	∼	+	−	+	−
amcGA2–amcGA3	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–amcGA4	−	∼	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA2–amcGA5	∼	−	−	−	−	−	−	∼	+	−	∼	−	−	∼	∼	−	−	∼	−	−	−	−	∼	−	−	−	−	−	−	−	−	∼	−	∼	−	−	−
amcGA3–amcGA4	+	+	∼	−	∼	−	+	−	+	+	∼	+	+	−	∼	−	+	−	∼	−	−	∼	−	+	−	∼	+	+	−	∼	+	+	−	+	∼	∼
amcGA3–amcGA5	+	+	∼	∼	+	+	∼	∼	+	+	∼	+	∼	+	+	−	+	∼	+	∼	+	∼	∼	−	+	+	∼	−	∼	+	∼	+	∼	−	+	∼
amcGA4–amcGA5	−	∼	∼	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	∼	−	−	−	−	∼	−	−	−	−	−	−	∼	−	−	−	∼	−	−
Random																																				
amcGA1–amcGA2	+	+	+	+	+	−	+	∼	+	−	+	+	+	+	∼	+	+	−	−	+	+	−	+	+	+	+	∼	∼	+	+	−	∼	+	+	+	−
amcGA1–amcGA3	−	−	−	−	−	−	−	∼	−	−	∼	−	−	−	−	−	−	∼	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−
amcGA1–amcGA4	+	−	−	−	−	∼	−	−	∼	+	+	+	+	+	∼	−	+	−	−	+	+	∼	+	∼	−	∼	+	−	−	∼	+	−	+	−	+	−
amcGA1–amcGA5	∼	−	−	∼	−	−	−	∼	−	∼	−	−	∼	−	−	+	−	−	−	∼	+	−	∼	−	−	∼	−	−	∼	∼	−	−	−	−	∼	∼
amcGA2–amcGA3	−	−	−	∼	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	∼	−	−	−	−	∼	−	−

Table 2 continued

Algorithms and DOPs	DDUF1						DDUF2						DKP					
Environment dynamics	$\tau = 20$			$\tau = 60$			$\tau = 20$			$\tau = 60$			$\tau = 20$			$\tau = 60$		
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
amcGA2–amcGA4	−	≀	−	−	−	−	−	≀	−	−	−	≀	−	−	≀	−	−	−	≀	−
amcGA2–amcGA5	−	−	−	−	≀	−	−	−	≀	−	≀	−	−	−	−	−	≀	−	≀	−
amcGA3–amcGA4	+	−	≀	+	+	−	+	+	+	≀	+	+	+	≀	+	+	+	+	+	+
amcGA3–amcGA5	+	+	+	+	+	+	≀	+	+	+	≀	+	+	+	+	+	+	≀	+	+
amcGA4–amcGA5	+	−	+	+	−	≀	−	+	−	−	−	+	+	−	≀	+	−	−	≀	+

effect of the dynamic environments on the performance of the algorithms investigated.

From Fig. 4, it can be observed that for a fixed τ with increasing value of ρ, PBILm outperforms 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 decreases 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. This means 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 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. 4, it can be observed that on the cyclic DDUF2 (with and without noise), all amcGA variants show low performance when $\rho = 0.1$ to 0.5, but exhibit rapid increase in performance when $\rho = 1.0$. This is due to the deceptive nature of DDUF2, since low-order building block inside the function does 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. 4, 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 variants (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 (i.e. when $\rho = 0.5$ and 1.0). This implies

Table 3 Overall offline fitness difference of the cGAm and amcGAs against the PBILm when $\tau = 60$ and $\rho = 0.1, 0.2, 0.5$ and 1. The table shows how far off the performance of the cGAm and amcGAs is from the PBILm

DOPs	Cyclic DDUF1				Cyclic with noise DDUF1				Random DDUF1			
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
cGAm	−22.79	−20.72	−19.85	−19.84	−12.81	−14.64	−15.55	−15.72	−4.61	−10.19	−13.80	−17.19
amcGA1	−7.17	−11.01	−18.12	−18.65	−2.25	−4.92	−12.51	−15.33	−0.97	−4.34	−9.46	−15.83
amcGA2	−15.13	−17.67	−18.69	−18.85	−10.51	−14.54	−16.61	−16.53	−8.82	−14.44	−17.92	−16.03
amcGA3	−16.92	−16.12	−15.23	−14.65	−6.89	−7.67	12.17	−10.97	−0.90	−6.44	−11.65	−12.11
amcGA4	−14.14	−17.24	−18.78	−18.87	−11.88	−14.80	−16.56	−16.62	−7.23	−14.21	−17.70	−16.15
amcGA5	−15.02	−17.41	−18.76	−18.68	−11.39	−14.50	−16.48	−16.59	−9.75	−13.42	−17.63	−16.14
DOPs	Cyclic DDUF2				Cyclic with noise DDUF2				Random DDUF2			
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
cGAm	−5.78	−4.81	+2.75	+25.55	−13.66	−14.60	−14.24	−9.05	−14.40	−14.97	−16.88	+25.24
amcGA1	−4.69	−5.34	+1.40	+27.42	−7.12	−12.11	−12.82	−5.06	−7.34	−11.61	−14.85	+26.61
amcGA2	−9.04	−7.31	+2.16	+27.44	−19.78	−20.15	−18.14	−13.03	−16.61	−19.86	−23.28	+27.25
amcGA3	−5.59	−6.94	+3.83	+18.57	−12.03	−13.06	−12.81	−5.81	−8.84	−12.94	−16.17	+21.84
amcGA4	−8.69	−7.75	+1.31	+27.45	−19.82	−20.19	−18.76	−12.44	−16.65	−20.31	−23.16	+27.26
amcGA5	−8.01	−7.16	+1.38	+27.45	−19.87	−19.75	−17.89	−13.04	−16.75	−20.02	−23.35	+27.24
DOPs	Cyclic DKP				Cyclic with noise DKP				Random DKP			
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
cGAm	−354.33	−394.69	−381.88	−292.01	−190.91	−234.80	−270.66	−248.00	−143.91	−163.83	−245.08	−281.53
amcGA1	−334.61	−297.42	−450.90	−528.02	−184.93	−233.33	−275.14	−380.12	−181.01	−192.34	−217.43	−508.74
amcGA2	−361.61	−323.99	−485.36	−525.08	−210.17	−242.15	−294.71	−394.82	−230.77	−212.36	−236.6	−511.96
amcGA3	−221.57	−166.30	−179.06	−201.12	−168.65	−177.97	−187.26	−199.17	−158.61	−166.32	−189.69	−195.89
amcGA4	−359.76	−330.42	−477.52	−519.84	−198.85	−240.39	−298.96	−395.17	−244.04	−222.01	−229.37	−512.18
amcGA5	−342.86	−338.89	−480.47	−517.86	−199.4	−240.73	−295.32	−396.36	−218.47	−227.63	−228.14	−549.28

The "$-A$" indicates that the overall fitness of the respective algorithm is less than the performance of the PBILm by "A" while "+" indicates the overall fitness of the respective algorithm is greater than the performance of the PBILm by "A"

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, 2, 3 and 4, 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. Generally speaking, the experimental results indicate the amcGA variants can be considered when solving DOPs with deceptive properties. These are functions in which low-order building block does not combine to form higher-order building block. Instead, low-order building block may mislead an optimization algorithm towards local optima (Whitley 1991; Fernandes et al. 2009).

5 Conclusion and future work

Mutation is a double-edged sword; it ensures diversity and improves an algorithm's 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, several conclusions can be drawn on the overall performance of the algorithms:

First, the mutation schemes have 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, on all but a very small niche of the DOPs with extreme environmental change, the proposed amcGA algorithms were statistically significantly outperformed by the PBILm. On several cases, the change in environment had minimal effect on the performance of the amcGA variants while the algorithms try to find a suitable solution. Also, the interaction between the change trend and mutation depends on the DOP (see Fig. 4; Tables 1, 2, 3).

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 moves 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 works relevant to this paper:

All amcGA variants are relatively easy to implement, especially in memory-constrained application since all variants of the amcGA retain the small footprint of the cGA. This 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 include further experimentations to identify possible limitations of the algorithms.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Ahn CW, Ramakrishna RS (2003) Elitism-based compact genetic algorithms. IEEE Trans Evol Comput 7(4):367–385

Bektas T (2006) The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34(3):209–219

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

Branke J, Kaußler T, Smidt C, Schmeck H (2000) A multi-population approach to dynamic optimization problems. In: Parmee IC (ed) Evolutionary design and manufacture. Springer, pp 299–307

Branke J, Orbayı M, Uyar Ş (2006) The role of representations in dynamic knapsack problems. In: Rothlauf F, Branke J, Cagnoni S, Costa E, Cotta C, Drechsler R, Lutton E, Machado P, Moore JH, Romero J, Smith GD, Squillero G, Takagi H (eds) Applications of evolutionary computing. Springer, pp 764–775

Bui LT, Michalewicz Z, Parkinson E, Abello M (2012) Adaptation in dynamic environments: a case study in mission planning. IEEE Trans Evol Comput 16(2):190–209

Cobb H, Grefenstette J (1993) Genetic algorithms for tracking changing environments, Morgan Kaufmann Publishers Inc, San Francisco, CA, pp 523–530

Cobb HG (1990) An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory, Washington

Dasgupta D, McGregor DR (1993) SGA: a structured genetic algorithm. University of Strathclyde, Department of Computer Science

Dasgupta D, McGregor DR (1992) Nonstationary function optimization using the structured genetic algorithm. In: PPSN, pp 145–154
Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
Eiben A, Smit S (2012) Evolutionary algorithm parameters and methods to tune them. In: Youssef H, Eric M, Frédéric S (eds) Autonomous search. Springer, pp 15–36
Eiben A, Schut M, de Wilde A (2006) Boosting genetic algorithms with self-adaptive selection. In: IEEE congress on evolutionary computation. CEC 2006. pp 477–482
Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. IEEE Trans Evol Comput 3(2):124–141
Fernandes CM, Guervós JJM, Rosa AC (2009) Using dissortative mating genetic algorithms to track the extrema of dynamic deceptive functions. CoRR abs/0904.3063
Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning, 1st edn. Addison-Wesley, Boston
Goldberg DE (2002) The design of innovation: lessons from and for competent genetic algorithms. Kluwer Academic Publishers, Norwell
Goldberg DE, Smith RE (1987) Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: ICGA, pp 59–68
Gongora M, Passow B, Hopgood A (2009) Robustness analysis of evolutionary controller tuning using real systems. In: IEEE Congress on evolutionary computation. CEC '09, pp 606–613
Grefenstette JJ et al (1992) Genetic algorithms for changing environments. PPSN, vol 2. San Francisco, CA, pp 137–144
Hansen N, Kern S (2004) Evaluating the cma evolution strategy on multimodal test functions. In: Yao X, Burke EK, Lozano JA, Smith J, Merelo-Guervos JJ, Bullinaria JA, Rowe JE, Tivno P, Kaban A, Schwefel H-P (eds) Parallel problem solving from nature-PPSN VIII. Springer, pp 282–291
Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of IEEE international conference on evolutionary computation, pp 312–317. doi:10.1109/ICEC.1996.542381
Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
Harik G, Lobo F, Goldberg D (1999) The compact genetic algorithm. IEEE Trans Evol Comput 3(4):287–297
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, pp 39–61
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, GECCO '06, pp 1201–1208
Higuchi T, Iwata M, Keymeulen D, Sakanashi H, Murakawa M, Kajitani I, Takahashi E, Toda K, Salami N, Kajihara N, Otsu N (1999) Real-world applications of analog and digital evolvable hardware. IEEE Trans Evol Comput 3(3):220–235
Jin Y, Branke J (2005) Evolutionary optimization in uncertain environments-a survey. IEEE Trans Evol Comput 9(3):303–317
Juang CF (2004) A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. IEEE Trans Syst Man Cybern B Cybern 34(2):997–1006
Larraanaga P, Lozano JA (2001) Estimation of distribution algorithms: a new tool for evolutionary computation. Kluwer Academic Publishers, Norwell
Martins JP, Fonseca CM, Delbem AC (2014) On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem. Neurocomputing 146:17–29
Mininno E, Cupertino F, Naso D (2008) Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Trans Evol Comput 12(2):203–219
Morrison RW, De Jong KA (2000) Triggered hypermutation revisited. In: Proceedings of the 2000 congress on evolutionary computation, IEEE, vol 2, pp 1025–1032
Nelson AL, Barlow GJ, Doitsidis L (2009) Fitness functions in evolutionary robotics: a survey and analysis. Robot Auton Syst 57(4):345–370
Nguyen TT, Yang S, Branke J (2012) Evolutionary dynamic optimization: a survey of the state of the art. Swarm Evol Comput 6:1–24
Passow B, Gongora M, Coupland S, Hopgood A (2008) Real-time evolution of an embedded controller for an autonomous helicopter. In: IEEE congress on evolutionary computation. CEC 2008 (IEEE World Congress on Computational Intelligence), pp 2538–2545
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, GECCO '06, pp 1737–1744
Pelikan M, Goldberg D, Lobo F (2000) A survey of optimization by building and using probabilistic models. In: Proceedings of the 2000 American control conference, vol 5, pp 3289–3293
Rohlfshagen P, Yao X (2009) The dynamic knapsack problem revisited: a new benchmark problem for dynamic combinatorial optimisation. In: Giacobini M, Brabazon A, Cagnoni S, Di Caro G, Ekrt A, Esparcia-Alczar A, Farooq M, Fink A, Machado P (eds) Applications of evolutionary computing. Lecture notes in computer science, vol 5484. Springer, Berlin, pp 745–754
Sastry K, Abbass HA, Goldberg D (2005) Sub-structural niching in non-stationary environments. In: Geoffrey IW, Xinghuo Y (eds) 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. Eur J Oper Res 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: 14th UK workshop on computational intelligence (UKCI), pp 1–7
Uzor CJ, Gongora M, Coupland S, Passow BN (2014b) Real-world dynamic optimization using an adaptive-mutation compact genetic algorithm. In: 2014 IEEE Symposium on computational intelligence in dynamic and uncertain environments (CIDUE), pp 17–23
Vavak Frank, Fogarty Terence C, Jukes Ken (1996) A genetic algorithm with variable range of local search for tracking changing environments. In: Voigt H-M, Ebeling W, Rechenberg I, Schwefel H-P (eds) Parallel problem solving from nature PPSN IV. Springer, pp 376–385
Wang H, Yang S, Ip W, Wang D (2009) Adaptive primal-dual genetic algorithms in dynamic environments. IEEE Trans Syst Man Cybern B Cybern 39(6):1348–1361
Whitley D (1991) Fundamental principles of deception. In: Foundations of genetic algorithms 1991 (FOGA 1), pp 221–241
Woldesenbet Y, Yen G (2009) Dynamic evolutionary algorithm with variable relocation. IEEE Trans Evol Comput 13(3):500–513

Yang S (2008) Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evol Comput 16(3):385–416

Yang S, Richter H (2009) Hyper-learning for population-based incremental learning in dynamic environments. In: IEEE congress on evolutionary computation. CEC '09, pp 682–689

Yang S, Tinós R (2007) A hybrid immigrants scheme for genetic algorithms in dynamic environments. Int J Autom Comput 4(3):243–254

Yang S, Tinos R (2008) Hyper-selection in dynamic environments. In: IEEE congress on evolutionary computation. CEC 2008 (IEEE World Congress on Computational Intelligence), pp 3185–3192

Yang S, Yao X (2005) Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Comput 9(11):815–834

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

Yang S, Jiang Y, Nguyen TT (2013) Metaheuristics for dynamic combinatorial optimization problems. IMA J Manag Math 24(4):451–480

Yu E, Suganthan P (2009) Evolutionary programming with ensemble of explicit memories for dynamic optimization. In: IEEE congress on evolutionary computation. CEC '09, pp 431–438

Yu X, Tang K, Yao X (2008) An immigrants scheme based on environmental information for genetic algorithms in changing environments. In: IEEE congress on evolutionary computation. CEC 2008 (IEEE World Congress on Computational Intelligence), pp 1141–1147

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, pp 480–487