Applied Soft Computing 9 (2009) 896–905

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Real options approach to evaluating genetic algorithms

Sunisa Rimcharoen [a], Daricha Sutivong [b,*], Prabhas Chongstitvatana [a]

[a] Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand
[b] Department of Industrial Engineering, Chulalongkorn University, Bangkok 10330, Thailand

ARTICLE INFO

Article history:
Received 8 February 2008
Received in revised form 30 October 2008
Accepted 15 November 2008
Available online 25 November 2008

Keywords:
Real options
Estimation of distribution algorithm
Optimal stopping time

ABSTRACT

The real options technique has emerged as an evaluation tool for investment under uncertainty. It explicitly recognizes future decisions, and the exercise strategy is based on the optimal decisions in future periods. This paper employs the optimal stopping policy derived from real options approach to analyze and evaluate genetic algorithms, specifically for the new branches namely Estimation of Distribution Algorithms (EDAs). As an example, we focus on their simple class called univariate EDAs, which include the population-based incremental learning (PBIL), the univariate marginal distribution algorithm (UMDA), and the compact genetic algorithm (cGA). Although these algorithms are classified in the same class, the characteristics of their optimal stopping policy are different. These observations are useful in answering the question "which algorithm is suitable for a particular problem". The results from the simulations indicate that the option values can be used as a quantitative measurement for comparing algorithms.

1. Introduction

The real options approach has been applied to many economic and financial problems. It helps investors evaluate investment risk and guides them when to take an opportunity. Its advantages in managerial flexibility have been widely recognized in the literature. The novelty of this work lies in applying real options to a computational problem, namely to analyze an optimal stopping policy of the evolutionary algorithms.

Evolutionary algorithms are becoming a common technique to solve difficult real-world problems. In spite of many useful practical applications, there are little knowledge about their behavior. Many approaches have been presented in order to understand how evolutionary algorithms work. The analysis is usually based on Markov chain model [34]. Time complexity has been studied [16,22] and the first hitting time are derived [23,24]. The results lead to the question of what kinds of problems are easy or hard for the evolutionary algorithms. Various techniques have been proposed to measure their difficulties, such as epistasis variance [12], fitness distance correlation [26,38], NK landscapes [28], fitness distribution [9] and information landscape [8]. Unfortunately, these predictive measures still have a problem in reliability, which are reported in [1,27,39,43]. The comparison results from the study of Naudts and Kallel [32] show that the values of the measures can be completely unreliable. A few years later, He et al. [21] show rigorous proof that finding a general difficulty measure is impossible.

The problem of GA-easy and GA-hard is closely related to the stopping problem. The first hitting time analysis [24] yields an important insight in more understanding what makes a problem hard for an evolutionary algorithm. The two conditions that lead evolutionary algorithms to an exponential time are presented to characterize what problems are hard. Similarly, the stopping time analysis gives bounds on running an evolutionary algorithm. Aytug and Koehler [4,5] estimated an upper bound of the number of iterations required to achieve a level of confidence to guarantee that a simple genetic algorithm converges. However, characterizing the hard problem is not mentioned in the paper. A critical review of the state-of-the-art in the design of termination conditions can be found in Safe et al. [44].

Theoretical bounds on running an evolutionary algorithm give us a large picture of the ability to solve a problem. In practice, evolutionary algorithms may stop early or they may not have enough computational effort to achieve it. We typically accept a good result with a given effort. With limited resources, the efficiency of computation is essential. Using the real options technique, it facilitates analyzing an optimal stopping time using an economic approach. The analysis offers us two things: a stopping criterion based on the bound of fitness value in each generation and a quantitative value indicates the efficiency of the

* Corresponding author. Tel.: +66 2 2186830; fax: +66 2 2186813.
E-mail address: daricha.s@chula.ac.th (D. Sutivong).

1568-4946/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2008.11.002

algorithm under investigation. In a complex algorithm, which is hard to analyze analytically, this approach gives us a method to investigate its behavior in searching for a solution. We propose the real options technique as a tool to evaluate algorithms by analyzing an optimal stopping time. It focuses on a computational approach rather than the theoretical analysis. In computational approach, the algorithm is run several times and its profile is collected. From these data, the benefit of an algorithm can be calculated. It takes a computational cost, time and the possibility to find a solution into account. The obtained value can be used to compare different evolutionary algorithms for their efficiency.

The optimal stopping problem is an important class of a stochastic control problem that arises in economics and finance, such as finding optimal exercise rules for financial options. Fortunately, there are similarities in the problem of finding an optimal stopping time in genetic algorithms and finding optimal exercise rules for financial options. The concept behind this technique is that finding an optimal stopping time of the algorithm can be viewed as deciding when to exercise a call option. Note that a call option is the right to buy an asset at a certain price. In this case, exercising a call option is analogous to stopping an algorithm, or buying an asset, same as quitting an algorithm, ignores all future possibilities. To explore this approach, Rimcharoen et al. [42] proposed finding an optimal stopping time in the compact genetic algorithm. Using the compact genetic algorithm, a special class of genetic algorithms, the underlying uncertainty can be viewed as a probability distribution. This distribution automatically captures the underlying uncertainty of the problem, which can be simulated to obtain an evolutionary process of the algorithm. This forms a basis in using the real options valuation in order to determine when it is worth stopping the algorithm. The extensions of this work which improved solution's quality on the deceptive problem were published in [40,41].

In this paper, the analysis and evaluation of univariate EDAs are presented as an example. They include the population-based incremental learning (PBIL) [6], the univariate marginal distribution algorithm (UMDA) [30], and the compact genetic algorithm (cGA) [19]. The different behaviors among these algorithms are also discussed. Although they belong to the same class, they have their own characteristic in searching for solution, which can be specified by their optimal stopping policies.

2. Estimation of distribution algorithm

Genetic algorithms (GAs) have been developed by Holland [25], who was motivated to study the behavior of complex and adaptive systems. The genetic algorithms, the branches of evolutionary computation, are based upon the principle of natural evolution and the principle of survival of the fittest. Evolutionary computation techniques abstract these evolutionary principles into algorithms. In an evolutionary algorithm, a representation scheme is chosen by a researcher to define a set of solutions which form the search space for the algorithm. The representation of genetic algorithm is a fixed-length bit string. A number of candidate solutions are created and evaluated using a fitness function that is specific to the problem being solved. A number of solutions are chosen using their fitness values to be parents for creating new individuals or offspring to form a new population of the next generation.

Goldberg [17] introduced a simple genetic algorithm (sGA), which is a simple binary coding using two genetic operators: mutation and one-point crossover. A selection operator is applied to the population and the appropriate solutions will survive. There have been numerous extensions and modifications of the simple genetic algorithm thus far.

Recently, the probabilistic model-building genetic algorithms (PMBGAs) or the estimation of distribution algorithms (EDAs) have been proposed. These models generalize genetic algorithms by replacing the crossover and mutation operators with the probability model estimation. The probability distribution of the solutions is estimated by adjusting the model according to the good solutions. New solutions are generated from the constructed model. The simplest way to design the distribution of promising solutions is to assume that the variables are independent, which is called univariate EDAs. These models include the PBIL, the UMDA, and the cGA.

The population-based incremental learning was introduced by Baluja [6]. It uses a probability vector to represent its population. At each generation, using the probability vector, M individuals are obtained. Each of these M individuals is evaluated and the N best of them are selected to update the probability vector. The pseudo code of the PBIL is shown below. The parameter is the learning rate (α) where $\alpha \in (0, 1]$, and x_k is a value of each position in the bit string (0 or 1).

1. Initialize probability vector (p) with 0.5 at each position.
2. Generate M individuals from the vector.
3. Select N best individuals, where $N \leq M$.
4. Update the probability vector p.
   for $i = 1$ to l do
   $$p_i = (1 - \alpha)\, p_i + \alpha \frac{1}{N} \sum_{k=1}^{N} x_k$$
5. Go to step 2 until a termination criterion is met.

The univariate marginal distribution algorithm was proposed by Mühlenbein and Paaß [30]. It maintains a population and creates a new population based on the frequency of each gene. The pseudo code of UMDA is shown below.

1. Randomly generate M individuals.
2. Select N individuals according to a selection method, where $N \leq M$.
3. Estimate univariate marginal probabilities (p_i) for each x_k.
   for $i = 1$ to l do
   $$p_i = \frac{1}{N} \sum_{k=1}^{N} x_k$$
4. Go to step 2 until a termination criterion is met.

Another type of this class, the cGA, was proposed by Harik et al. [19]. It represents the population as a probability distribution over the set of solutions. In each generation, the compact genetic algorithm samples individuals according to the probabilities specified in the probability vector. The individuals are evaluated and the probability vector is updated towards the better individual. The compact genetic algorithm has an advantage of using a small amount of memory and achieving comparable quality with approximately the same number of fitness evaluations as the simple genetic algorithm. The pseudo code of the cGA is shown below. The parameters are the updating step size (n) and chromosome length (l). Notice that the parameter n is related to the population size in the simple genetic algorithm. The detail is provided in the original paper [19].

1. Initialize probability vector (p).
   for $i := 1$ to l do $p[i] := 0.5$;
2. Generate two individuals from the vector.
   $a :=$ generate (p);
   $b :=$ generate (p);
3. Let them compete.
   winner, *loser* := compete(a, b);

4. Update the probability vector towards the better one.
   for $i := 1$ to $l$do
      if $winner[i] \neq loser[i]$ then
         if $winner[i] = 1$ then $p[i] := p[i] + 1/n$
                            else $p[i] := p[i] - 1/n$;
5. Check if the vector has converged.
   for $i := 1$ to $l$do
      if $p[i] > 0$ and $p[i] < 1$ then
         return to step 2;

Harik et al. also proposed a modification of cGA that used larger population. A tournament selection, which is one of many selection methods in GA, is used in this modification. A few individuals are chosen at random from the population and compete, after which only the winner survives. It allows the algorithm to simulate higher selection pressure, which adds an intensity of a selection mechanism. Selection pressure can be easily adjusted by changing the tournament size, i.e. the number of individuals chosen to compete. The larger the tournament size, the smaller chance weak individuals have to survive.

For the modified cGA, if we would like to simulate a tournament of size s, steps 2–4 of the above cGA's pseudo code would be replaced by the following procedures.

1. Generate s individuals from the vector and store them in S.
   for $i := 1$ to s do
      $S[i] :=$ generate(p);
2. Rearrange S so that $S[1]$ is the individual with higher fitness, and let $S[1]$ compete with the other individuals.
   for $j := 2$ to s do
   begin
      $winner, loser =$ compete($S[1], S[j]$);
      update probability vector $p$using the method
      in step 4 of the original code
   end

3. Real options approach

Real options approach is a financial concept that applies a financial option theory to investments in real assets (as opposed to financial assets that are traded in the market). A financial option is the right, but not an obligation, to buy or sell an asset. An option that gives the holder the right to purchase an asset at a specified price is a call option, while an option that gives the holder the right to sell an asset at a specified price is a put option. The financial options are useful for managing risks in the financial world. For example, a call option limits possible loss by paying an upfront premium to have this right, and it opens the possibility to unlimited gains. Black and Scholes [7] and Merton [29] have inspired the rapid development in financial option pricing. For example, the two widely used methods for pricing financial options are the binomial lattice [11] and the Black–Scholes formula [7].

The financial option concept was extended to real assets when Myers [31] identified the fact that many corporate real assets can be viewed as call options. The real options approach addresses an investment decision problem by analyzing not only the expected net present value (NPV), but also considering the value of an option to wait, expand, abandon, etc.

One of the techniques to find an option value is a dynamic programming method. The idea of dynamic programming is to split a whole sequence of decisions into two parts: the immediate choice and the remaining decisions. The detailed technique is described in Dixit and Pindyck [13].

The value $F_t(x_t)$ is the expected NPV when the firm makes all the decisions optimally from this point onwards. The value function called Bellman equation or the fundamental of optimality is shown in Eq. (1).

$$F_t(x_t) = \max_{u_t}\left\{\pi_t(x_t, u_t) + \frac{1}{1+\rho}\varepsilon_t[F_{t+1}(x_{t+1})]\right\} \quad (1)$$

At each period t, choices available to the firm are represented by the control variable(s) u_t. The value u_t must be chosen using only the information available at the time t, namely x_t. When the firm chooses the control variables u_t, it gets an immediate profit flow $\pi_t(x_t, u_t)$. The discount factor between any two periods is $1/(1+\rho)$, where ρ is the discount rate. The term $\varepsilon_t[F_{t+1}(x_{t+1})]$ is the expected value from time $t+1$ on called a continuation value.

An optimal stopping time is found by selecting the maximum value between the termination payoff $\Omega(x)$ and the continuation value. The Bellman equation becomes

$$F(x) = \max\left\{\Omega(x), \pi(x) + \frac{1}{1+\rho}\varepsilon[F(x')|x]\right\}. \quad (2)$$

From Eq. (2), there is a payoff value as a function of x achieved by termination and a payoff value as a function of x achieved through continuation. The x values that produce the boundary payoff values, where termination is optimal on one side and continuation is on the other, form an exercise region. This also provides a guideline for making decision optimally called an exercise policy.

4. Proposed option-based methodology

We employ the real options approach to determine when to stop running a genetic algorithm, which is analogous to deciding when to exercise a call option. In each generation, the algorithm can stop or continue running. If the algorithm decides to stop, the payoff from stopping is obtained. If the algorithm decides to continue, further computation may add the value, while it must incur a computational cost. To determine when to terminate, the algorithm needs to know the probability distribution of the fitness value (underlying uncertainty) and the payoff model (value function of option). At every generation, we compute the expected payoff from stopping and continuing using the underlying uncertainty and the value function of option. The algorithm should continue if the expected payoff from continuing is higher than that of stopping. The stop or continue decision is solved starting from the last time step and working backward to the first generation, as in dynamic programming.

The methodology of finding an optimal stopping time in genetic algorithm described above can be summarized in the following process.

4.1. Modeling underlying uncertainty

In this step, we need to know the movement of fitness values in each generation. We can obtain this distribution by running the genetic algorithms many times. For example, suppose the average fitness value in the first generation is 5.0. Assume that the fitness value increases to 7.0 in the first run and falls to 4.0 in the second run. The fitness movement of these two runs can be shown in Fig. 1. From this example, it means that the fitness value in the second generation is 7.0 with probability 0.5 and 4.0 with probability 0.5.

By running the genetic algorithms many times, we have fitness values in each generation (time step). We accumulate the possible changes of fitness values in each generation over many runs and

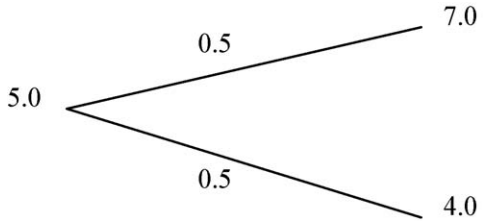

Fig. 1. An example of fitness movement.

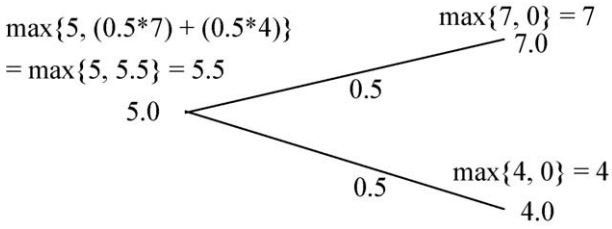

Fig. 3. An example of the option value calculation.

then calculate the probability of all possible values in each state. For example, running the compact genetic algorithm with a 5 × 3-Trap problem, the possible average values are 0.0, 0.5, 1.0, . . ., 14.0, 14.5, and 15.0. Fig. 2 shows the lattice of all possible values along with their associated transitional probability. Note that in other algorithms and problems we can discretize these values into an appropriate interval as well.

4.2. Defining the value function of option

In this step, we formulate a function that indicates value of a solution in each generation. The termination payoff and the computational cost is defined specific to the problem.

Let $\Omega(x)$ denote the termination payoff. The termination payoff is shown in (3)

$$\Omega(x) = g(x) \tag{3}$$

where $g(x)$ is the fitness value of x. The profit term $\pi(x)$ can be discarded because the genetic algorithm does not produce any immediate profit flow. The solution value is obtained from the fitness value at the time the algorithm terminates. Therefore, the optimal stopping equation becomes

$$F(x) = \max\left\{g(x), \frac{1}{1+\rho}\varepsilon[F(x')|x]\right\}. \tag{4}$$

Note that we also assume the discount factor to be zero because in each state the genetic algorithm takes a few milliseconds to run. In this case, the optimal stopping equation becomes quite simple as shown in Eq. (5).

$$F(x) = \max\{g(x), \varepsilon[F(x')|x]\} \tag{5}$$

The first term of the maximization is the value if the algorithm stops now; thus, we receive the outcome that is the value of the current fitness value. The second term is the value if the algorithm continues. We choose the maximum of the two, as a policy to stop or continue the algorithm, when we reach x.

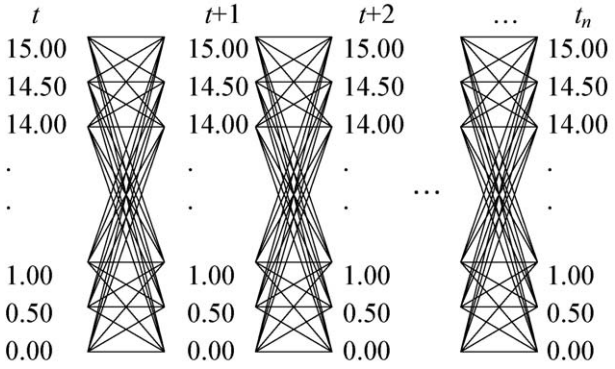

Fig. 2. Lattice of a 5 × 3-Trap problem.

4.3. Calculating the option value according to the value function of option

Using the probability distribution of the fitness value in step 1 and the value function of option in step 2, we can calculate the option value in each generation by working backward from the last time step. The option value of the above example is shown in Fig. 3.

Given that the termination payoffs in the last time step are 7 and 4, we work backward one time step. In this generation, the termination payoff is 5 for the fitness value of 5.0 whereas the continuation value is 5.5. Therefore, the algorithm should continue because the continuation value is greater than the termination value.

4.4. Summarizing an option value and an exercise policy

From step 3, we obtain the maximum values that may arise from stopping or continuing the algorithm. The underlying values, where the termination is optimal on one side and continuation is on the other, produce the boundary which forms the exercise region.

The option value of this algorithm is an option value at the first generation. From the example in Fig. 3, the option value is 5.5.

5. Experimental setting

We will explore the behaviors of cGA, PBIL and UMDA on these five test problems: 30-bit OneMax, 3-Trap × 10, 5-Trap × 6, 27-bit HTrap and 32-bit HIFF. These benchmark problems have been widely used for evaluating the performance of GAs [2,3,35,36,46]. They are also used in the analysis of algorithm and problems [10,15,33] because they are good representatives of easy and hard problems for GAs. For the OneMax problem, it is almost always a starting point for empirical verification. If an equation fails in such an uncomplicated setting, it is not likely to perform well in a more complex situation. For a variety of deceptive problems, they are difficult test functions that are used to test performance of algorithms. If an algorithm performs well in these benchmark problems, it is more likely to perform well in more complex setting. For example, there is an algorithm called hierarchical Bayesian optimization algorithm (hBOA) [37] that can efficiently solve the hierarchically decomposable functions such as HTrap, and it can be applied to solve real-world problems such as ising spin glasses[1] and MAXSAT[2] as well [20]. The definitions of those benchmark problems are as follows.

5.1. OneMax problem

OneMax problem is a well-known simple test problem for GA. The problem is to find a maximum value which occurs when all bits

[1] Ising spin glasses problem is a problem of statistical physics to find the value for each pair, formed in 2D or 3D that minimizes the energy.

[2] MAXSAT is a problem to find maximum satisfiability of predicate calculus formulas in conjunctive normal form.

are one. The fitness value is assigned according to the number of bits that are one in the chromosome. Thus, the maximum value is equal to the chromosome length. Formally, this problem can be described as finding a string $\vec{x} = \{x_1, x_2, \ldots, x_N\}$, with $x_i \in (0,1)$, that maximizes the following equation:

$$F(\vec{x}) = \sum_{i=1}^{N} x_i \tag{6}$$

5.2. Trap problem

The trap function [18] is a difficult test problem for GA. The general k-bit trap function is defined as

$$F_k(b_0 \ldots b_{k-1}) = \begin{cases} f_{\text{high}}; & \text{if } u = k \\ f_{\text{low}} - u\dfrac{f_{\text{low}}}{k-1}; & \text{otherwise} \end{cases} \tag{7}$$

where $b_i \in \{0,1\}$, $u = \sum_{i=0}^{k-1} b_i$, and $f_{\text{high}} > f_{\text{low}}$. Usually, f_{high} is set at k and f_{low} is set at $k-1$. The test function $F_{k\times m}$ is defined as

$$F_{k\times m}(B_0 \ldots B_{m-1}) = \sum_{i=0}^{m-1} F_k(B_i), \qquad B_i \in \{0,1\}^k \tag{8}$$

This function fools gradient-based optimizers to favor zeroes, but the optimal solution is composed of all ones. The k and m may vary to produce a number of test functions.

5.3. HTrap problem

The HTrap function [36] is a kind of hierarchically decomposable functions, which are defined on multiple levels where the input to each level is based on the solutions found on lower levels. The HTrap function represents a solution as a tree. An example is shown in Fig. 4. The solution is a 9-bit string placed at the leaf nodes. Triple zeroes are interpreted as zero in the higher level, and triple ones are interpreted as one. Otherwise, the interpretation is "−". The contribution of node i is c_i which can be calculated from the following equation:

$$c_i = \begin{cases} 3^h F_3(b_0 b_1 b_2); & \text{if } b_j \neq \text{“}-\text{”} \text{ for all } 0 \le j \le 2 \\ 0; & \text{otherwise} \end{cases} \tag{9}$$

where h is the height of node i, and b_0, b_1, and b_2 are the interpretations in the left, middle, and right of child node of node i.

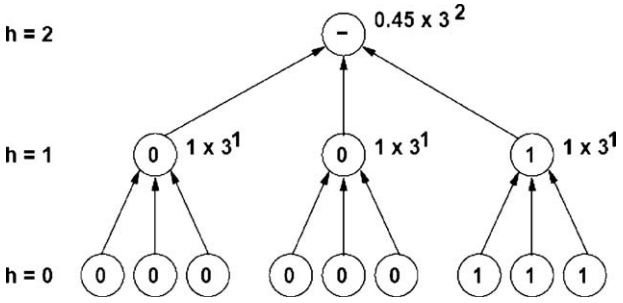

Fig. 4. An example of calculating fitness value of HTrap problem.

At the root node, the contribution is given by a 3-Trap function with parameters $f_{\text{high}} = 1$ and $f_{\text{low}} = 0.9$ multiplied by 3^h. The other nodes use $f_{\text{high}} = 1$ and $f_{\text{low}} = 1$. Fig. 4 shows the calculation of fitness value. The HTrap function returns $\sum c_i = 13.05$.

5.4. HIFF problem

The HIFF function [45] is also a kind of hierarchically decomposable functions. A solution is interpreted as a binary tree. An example is shown in Fig. 5. The sample solution is an 8-bit string "00001101" placed at the leaf nodes of the binary tree. The leaf nodes force the higher levels of the tree. A pair of zeroes and a pair of ones are interpreted as zero and one in the higher level, respectively. Otherwise, the interpretation result is "−." The HIFF function returns the sum of values calculated from each node. The value of node i is c_i which can be calculated from the following equation:

$$c_i = \begin{cases} 2^h; & \text{if node } i \text{ is “0” or “1”} \\ 0; & \text{if node } i \text{ is “}-\text{”} \end{cases} \tag{10}$$

where h is the height of node i. In the following example, the fitness of "00001101" is $\sum c_i = 18$. The HIFF functions do not bias an optimizer to favor zeroes rather than ones or vice versa. There are two optimal solutions: the string composed of all zeroes and the string composed of all ones.

The following experiments use these problems as test functions for comparing the behaviors among the univariate EDAs. The numerical results are averaged over 100 runs. In the experiments, we simulate the univariate EDAs with minimum

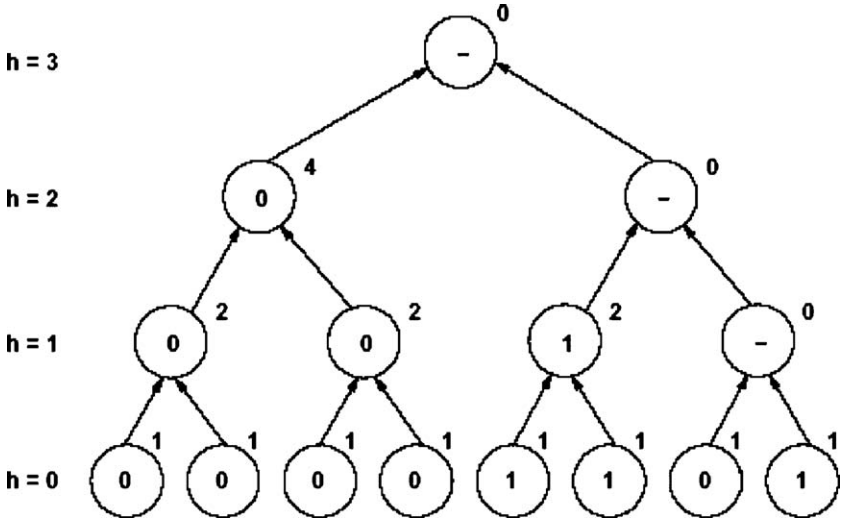

Fig. 5. An example of calculating fitness value of HIFF problem.

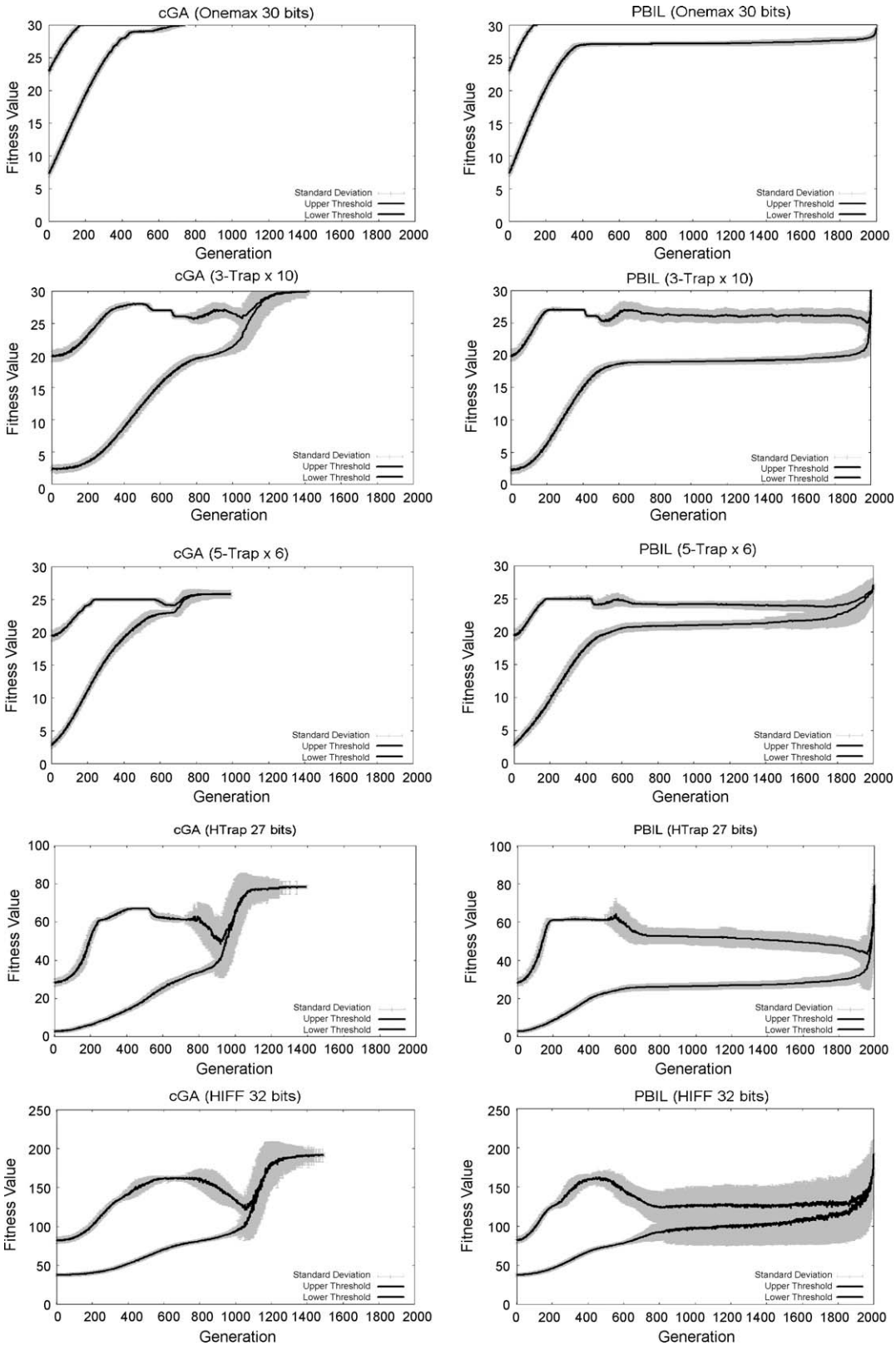

Fig. 6. Exercise regions from simulating cGA and PBIL with minimum population size. The left column shows the results from simulating the cGA and the right column is the results from simulating the PBIL. The curves are plotted using an average value of 100 runs. The standard deviations are shown in gray color.

population size in order to study behaviors of algorithm in the simplest setting. The results are presented in Section 6. The behaviors when using larger population size are also provided in Section 7.

6. Univariate EDAs with minimum population

Experimenting with minimum population helps us understand the basic behavior of algorithms. The original cGA employs

Table 1
Option values from simulating cGA and PBIL with minimum population size.

Problems	Algorithms			
	cGA		PBIL	
	Option value (f)	Difficulty index (f/F)	Option value (f)	Difficulty index (f/F)
30-bit OneMax ($F = 30$)	30.00	1.00	30.00	1.00
3-Trap × 10 ($F = 30$)	27.00	0.90	26.53	0.88
5-Trap × 6 ($F = 30$)	24.03	0.80	24.70	0.82
27-bit HTrap ($F = 81$)	77.74	0.96	61.53	0.76
32-bit HIFF ($F = 192$)	138.34	0.72	126.01	0.66

population of size two. Therefore, in order to compare with cGA, we also run PBIL with population of two, while UMDA is ignored in this experiment because it requires large population to estimate the distribution. The comparison among these three algorithms will be provided in the next section with large population.

In all experiments, the learning rate (α) in PBIL is set as 0.05, and the updating step size in cGA is 0.02. Fig. 6 shows the exercise policies of those algorithms on various test functions. Option values of the algorithms are summarized in Table 1.

In Fig. 6, the left column shows exercise regions of the cGA, while the right column shows those of the PBIL. Each column shows the results from OneMax, 3-Trap, 5-Trap, HTrap and HIFF problems, respectively.

As shown in Fig. 6, there are two lines in each graph. The line shown in the upper position is called the upper threshold while the lower line is the lower threshold. These lines form exercise regions. An optimal decision is determined by these exercise regions. The exercise regions are divided into three areas. The areas above the upper threshold and under the lower threshold are called the stopping region, while the area between the upper and lower threshold is called the continuation region.

The algorithm should stop the search when the fitness value rises above the upper threshold because the fitness value is already high. If the fitness value is lower than the lower threshold, the algorithm should also stop because with the current population, it is unlikely to achieve a better result.

Note that the exercise regions of the cGA and PBIL on the OneMax problem are quite similar, and the option values of these algorithms are equal. Both of them can achieve the global optimum. However, the continuation region of the PBIL is bigger than that of the cGA during the last part of evolution. It means that the PBIL allow more lower fit candidates to continue evolving when compared to the cGA.

From Fig. 6, in the graphs of OneMax, 3-Trap and 5-Trap problems (their fitness values are in the same range [0,30]), the exercise regions suggest that, at the beginning, the OneMax problem requires a higher solution quality for stopping than the trap problems. This is because good solutions abound in the OneMax problem. On the other hand, good solutions in the trap problems are rare. The OneMax problem has a large area of lower stopping region than the trap problem. This denotes that for a relatively easy problem, if the population cannot improve its quality fast enough, the algorithm should not continue.

From the upper thresholds of all algorithms, there are two main characteristics of the exercise policies. In the OneMax problem which known as an easy problem, the upper threshold is gradually improving, and when it reaches the upper bound, it remains stable. This behavior is different from the other test problems, which have local optima. The exercise regions of those problems have some ripples in the upper thresholds. These fluctuations in the upper threshold reveal an uncertainty in finding a good solution in hard problems. He and Yao [24] presented that one of the conditions that makes a problem hard for evolutionary algorithms is a "wide gap"—a situation when the probability to move to higher fitness value is very small. The fluctuations in the upper threshold of hard problems confirm this behavior. They occur when an algorithm is deceived into a local optimum; therefore, there is little chance in finding the global optimal solution.

The continuation regions of the PBIL show that this algorithm allows lower fit candidates to evolve even to later generations. These promising areas are larger than those of the cGA, whose upper and lower thresholds quickly join together. When the upper and lower thresholds join together before the optimal solution is reached, the algorithm decides to stop because the current fitness value exceeds the expected fitness value of continuing.

Table 1 shows the option values and the difficulty level of running the cGA and the PBIL on various problems. The results show that almost all problems solved by the cGA have higher values than the PBIL, except in the 5-Trap problem. The graphs of 5-Trap problem in the third row of Fig. 6 show that the upper bound of the PBIL reaches higher fitness value than that of the cGA. This better result may arise from the fact that the PBIL allow more candidates to continue, so they may get a better solution eventually. The algorithm that has a higher option value is better than the algorithms with lower values. This is because the option value is the expected fitness values based on optimal decision-making.

To determine the difficulty of the problems, the ratio of the option value, which is the expected fitness value, to the optimal solution is proposed as a measure, called the difficulty index. Note that when the optimal value is unknown, this ratio can be calculated using the best known value instead.

Specifically, let f_i and f_j be option values of the same algorithm running on the problem i and j, respectively, and F_i and F_j be their optimal values (or the best known value). The difficulty index of solving problem i using a particular algorithm is f_i/F_i. We say that the problem i is easier to solve than the problem j, if $(f_i/F_i) > (f_j/F_j)$.

By considering the values of f/F, both cGA and PBIL perform well in a OneMax problem. They have a ratio of 1, which means that they can reach the global optimum. The HIFF problem is the hardest benchmark for both of them because they have the smallest ratio, which means the solutions are far from the best value. For the trap problems, it is obvious that 3-Trap is easier than 5-Trap.

It is interesting that the cGA is better in solving the HTrap problem (difficulty index = 0.96) than solving the trap problems (difficulty index = 0.90 and 0.80) while the PBIL is opposite (HTrap's difficulty index = 0.76, trap problems' difficulty index = 0.88 and 0.82). The reason comes from differences in updating method of both algorithms. The cGA updates the vector according to the winner bit by bit, while the PBIL updates using a distribution of selected individuals. In the hierarchical problem, the method of the cGA to update bit by bit is more suitable than the method of PBIL because it considers a group of bits and assigns fitness according to their relationships in each level. If we update the vector according to the estimated distribution, like the PBIL, it

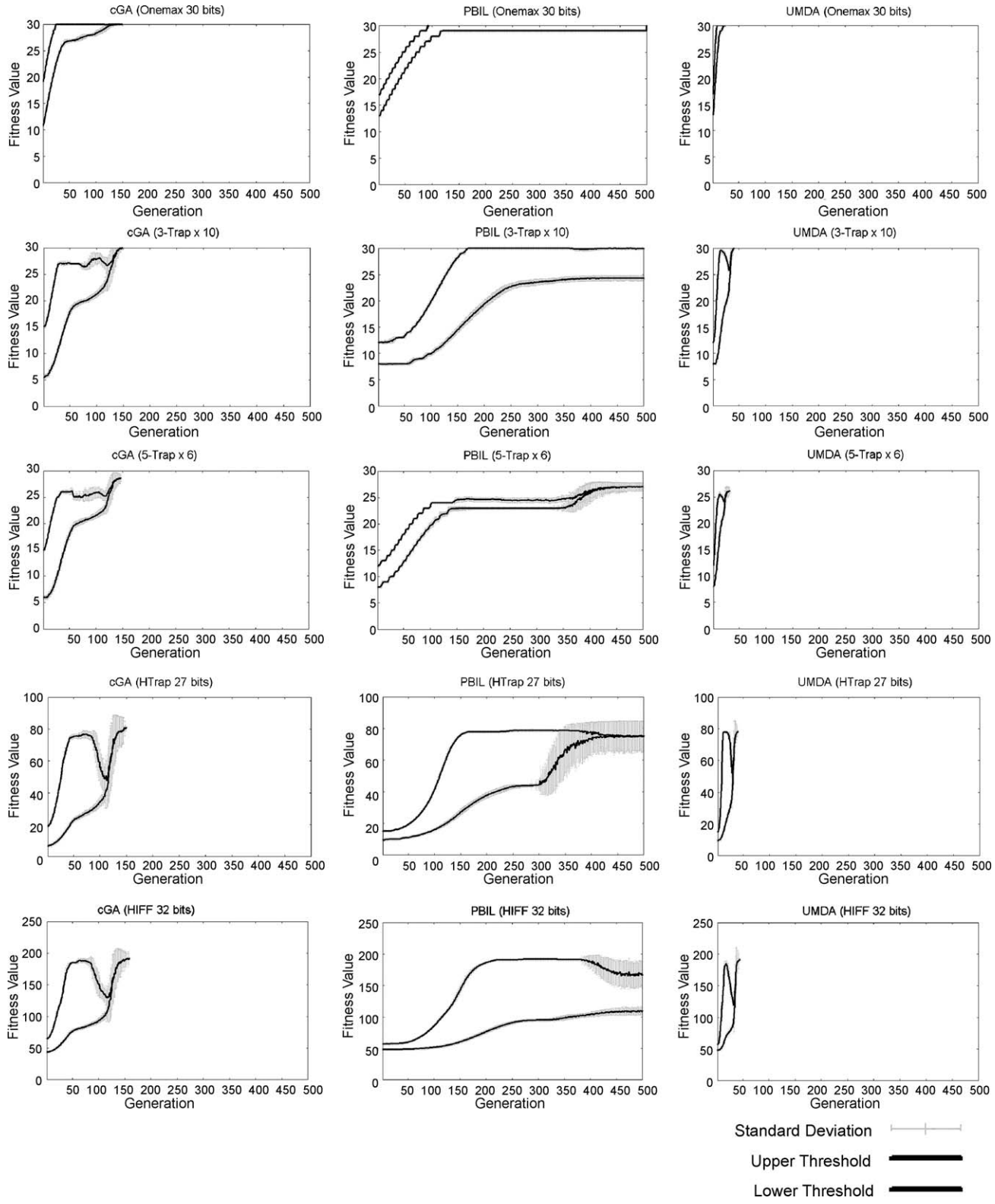

Fig. 7. Exercise regions from simulating cGA, PBIL and UMDA with larger population. The results from cGA, PBIL and UMDA are presented in the left, middle and right columns, respectively. The curves are plotted using an average value of 100 runs. The standard deviations are shown in gray color.

is more likely to guide every bit toward that distribution, so it loses diversity. Note that, in this case where the population size is two, only the best individual is used to estimate the distribution, the vector is biased by this solution. The experiments using larger population are provided in the next section.

7. Simulating with larger population

The behaviors of the univariate EDAs with large population are provided in this section. We simulate cGA and PBIL with larger population size in order to compare with UMDA, which is a

Table 2
Option values from simulating cGA, PBIL and UMDA with large population.

Problems	Algorithms					
	cGA		PBIL		UMDA	
	Option value (f)	Difficulty index (f/F)	Option value (f)	Difficulty index (f/F)	Option value (f)	Difficulty index (f/F)
30-bit OneMax ($F = 30$)	30.00	1.00	30.00	1.00	29.99	1.00
3-Trap × 10 ($F = 30$)	26.06	0.87	29.94	1.00	24.41	0.81
5-Trap × 6 ($F = 30$)	24.93	0.83	24.00	0.80	23.98	0.80
27-bit HTrap ($F = 81$)	45.19	0.56	76.27	0.94	51.44	0.64
32-bit HIFF ($F = 192$)	116.58	0.61	132.82	0.69	108.92	0.57

Table 3
Option values from simulating cGA, PBIL and UMDA with a discount rate of 5% and 10%.

Problems	Algorithms					
	cGA		PBIL		UMDA	
	Discount 5%	Discount 10%	Discount 5%	Discount 10%	Discount 5%	Discount 10%
30-bit OneMax	14.52	13.37	13.86	13.19	18.35	14.18
3-Trap × 10	9.63	8.87	9.13	8.65	10.93	8.65
5-Trap × 6	9.80	9.02	9.32	8.77	13.54	9.45
27-bit HTrap	11.70	10.73	11.00	10.48	23.40	12.09
32-bit HIFF	49.68	47.18	49.31	47.06	49.39	47.07

population-based algorithm. The population size used in the experiments is 50, and the tournament selection of size 8 is used. The exercise policies are shown in Fig. 7, and the option values are provided in Table 2.

With larger population size, the main characteristics of exercise regions do not change. In an easy problem, OneMax, the exercise thresholds are still gradually improving. Also, in harder problems, there still exist some fluctuations in the upper thresholds.

From Fig. 7, it is obvious that the graphs of each algorithm have an individual characteristic. In the cGA, the upper and lower thresholds meet up during early generations of evolution. In the PBIL, the thresholds seem to be parallel until the end. The exercise region of the UMDA is quite similar to the cGA, but it converges much faster. These characteristics can be an indicator of the type of algorithm used.

For the option values shown in Table 2, the PBIL mostly achieves higher values than other algorithms, and the UMDA seems to be the worst. This is because the UMDA uses the whole selected population to estimate the univariate marginal distribution that causes it to converge too fast, which may not be good for deceptive-type problems.

As proposed earlier, the difficulty index (f/F) is used to indicate the difficulty of solving problems. From Table 2, it shows that the hardest problem for the cGA in the experiments is the HTrap problem, while the HIFF problem is the hardest benchmark for the PBIL and the UMDA.

The HIFF problem is the hardest problem for the PBIL and the UMDA because it has two optimal values: all zeroes and all ones. As the two algorithms construct a model using marginal distribution and the samples usually contain both one and zero in their chromosomes, it is unlikely to achieve an all one or all zero bit pattern. The cGA has more chance to escape this situation because it updates the probability vector bit by bit according to the good sampling. When using a larger population and higher selection pressure, the HTrap problem becomes the hardest problem for the cGA because it deceives the algorithm to fall into trap. More selection pressure leads the cGA to quickly come close to the best winner and loses diversity.

The insights from this study suggest that if we have limited resources, for example, small population size, the cGA is a promising method to solve the problem because it has the highest option values among univariate EDAs when using small population size. When we have more samples to construct the model, the PBIL may be a good choice. It mostly provides the highest option values when simulating with larger population.

Note that if time is a major constraint, the UMDA is a method that converges fast. To account for the time value, we can set the discount term in Eq. (4) in order to highlight the solutions that quickly converge. In general, a discount rate is used to discount future cash flows into the present value. We incorporate this factor in the experiments in order to study the behaviors of algorithms when time plays an important role in searching for a solution.

For experimental purposes, we set the discount rate to 5% and 10%. The results are shown in Table 3. The UMDA generally provides higher option values than the other algorithms for both 5% and 10% discount rates, followed by cGA. This confirms that if we want to get a solution quickly, the UMDA is a promising technique.

8. Conclusions

This paper has proposed new optimal stopping policies for the univariate EDAs using real options approach. The exercise policies suggest the optimal stopping time of the algorithms, and their option values are presented as a quantitative measurement for evaluating algorithms. The option value is also useful in measuring effectiveness of running a particular algorithm on the problem. The higher option value shows higher fitness value that we can expect from a particular algorithm.

The insights from the experimental analysis suggest that among the three univariate EDAs, the cGA is a promising method for solving a problem with small population, whereas the PBIL should be used with large population. Moreover, when time plays an important role in obtaining a solution, the UMDA offers a faster convergence.

The data from the experiments show the use of the real options approach to analyze the variable independent EDAs. A different stopping characteristic of each algorithm is presented. This method can be applied to other algorithms. For more complex models of EDAs, such as bivariate and multivariate, the stopping behavior may be different, and requires further study. There are

also non-classical EDAs such as the eigen decomposition EDA (ED-EDA) [14]. Its procedure on tuning eigenvalue to influence the evolution process is complicated. The optimal stopping time analysis of these classical and non-classical EDAs is currently an open problem. As we mentioned earlier, in the situation that analytical method is hard, analyzing the optimal stopping time using the real options approach is an alternative. It does not require prior knowledge about algorithms and problems, and uses only the fitness movement to analyze the optimal stopping time. Any algorithms that have a fitness value in each time step can utilize this technique, while the main evolution process remains untouched.

The proposed method can be used as an analysis tool to investigate the behavior of an algorithm. As a practical tool, it is hard to accept a large number of runs required in collecting the fitness data. As shown in the experiments, the optimal stopping time and its policy are only obtained after performing many runs. Future work will focus on incorporating this approach directly into the evolutionary process so that there is no need to perform many runs beforehand.

Nonetheless, the proposed method helps us understand the behavior of genetic algorithms. From the experiments, the exercise regions are the characteristics of the algorithm type. The option value can also be used as a quantitative measurement for comparing algorithms in terms of their effectiveness in solving problems. The sensitivity analysis can be studied by adding costs into Eq. (5). The analysis on discount rates can be performed as well without requiring additional runs. We can use the obtained fitness-movement profile and re-calculate option value with various costs and discount rates. This opens up a new way to explore behaviors of the algorithm in various situations.

References

[1] L. Altenberg, Fitness distance correlation analysis: An instructive counterexample, in: Proceedings of the 7th International Conference on Genetic Algorithms, 1997, pp. 57–64.
[2] C. Aporntewan, P. Chongstitvatana, Building-block identification by simultaneity matrix, Soft Computing 11 (2007) 541–548.
[3] C. Aporntewan, P. Chongstitvatana, Chi-square matrix: an approach for building-block identification, ASIAN (2004) 63–67.
[4] H. Aytug, G.J. Koehler, New stopping criterion for genetic algorithm, European Journal of Operational Research 126 (2000) 662–674.
[5] H. Aytug, G.J. Koehler, Stopping criterion for finite length genetic algorithms, INFORMS Journal on Computing 8 (1996) 183–191.
[6] S. Baluja, Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning, Technical Report CMU-CS-95-163, Carnegie Mellon University, 1994.
[7] F. Black, M. Scholes, The pricing of options and corporate liabilities, Journal of Political Economy 81 (1973) 637–654.
[8] Y. Borenstein, R. Poli, Information landscapes and problem hardness, in: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, 2005, pp. 1425–1431.
[9] Y. Borenstein, R. Poli, Fitness distributions and GA hardness, in: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature, 2004, pp. 11–20.
[10] M. Clergue, P. Collard, GA-hard functions built by combination of trap functions, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2002, pp. 249–254.
[11] J.C. Cox, S.A. Ross, M. Rubinstein, Option pricing: a simplified approach, Journal of Financial Economics 7 (1979) 229–263.
[12] Y. Davidor, Epistasis variance: a viewpoint on GA-hardness, in: G.J.E. Rawlins (Ed.), Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991 , pp. 23–35.
[13] A.K. Dixit, R.S. Pindyck, Investment Under Uncertainty, Princeton University Press, NJ, 1994.
[14] W. Dong, X. Yao, Unified eigen analysis on multivariate Gaussian based estimation of distribution algorithms, Information Sciences 178 (2008) 3000–3023.
[15] S. Droste, A rigorous analysis of the compact genetic algorithm for linear functions, Natural Computing 5 (2006) 257–283.
[16] S. Droste, T. Jansen, I. Wegener, On the analysis of the (1 + 1) evolutionary algorithm, Theoretical Computer Science 276 (2002) 51–81.
[17] D.E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison Wesley, 1989.
[18] D.E. Goldberg, Simple genetic algorithms and the minimal deceptive problem, in: Genetic Algorithms and Simulated Annealing, Morgan Kaufmann Publisher, 1987.
[19] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Transactions on Evolutionary Computation 3 (1999) 287–297.
[20] M. Hauschild, M. Pelikan, C.F. Lima, K. Sastry, Analyzing probabilistic models in hierarchical BOA on traps and spin glasses, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2007, pp. 523–530.
[21] J. He, C. Reeves, C. Witt, X. Yao, A note on problem difficulty measures in black-box optimization: classification, realizations and predictability, Evolutionary Computation 15 (2007) 435–443.
[22] J. He, X. Yao, Drift analysis and average time complexity of evolutionary algorithms, Artificial Intelligence 127 (2001) 57–85.
[23] J. He, X. Yao, From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (2002) 495–511.
[24] J. He, X. Yao, Towards an analytic framework for analysing the computation time of evolutionary algorithms, Artificial Intelligence 145 (2003) 59–97.
[25] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.
[26] T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: Proceedings of the 6th International Conference on Genetic Algorithms, 1995, pp. 184–192.
[27] L. Kallel, B. Naudts, M. Schoenauer, On functions with a fixed fitness-distance relation, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1998, pp. 1910–1916.
[28] S. Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, Oxford, 1993.
[29] R.C. Merton, Theory of rational option pricing, Bell Journal of Economics and Management Science 4 (1973) 141–183.
[30] H. Mühlenbein, G. Paaß, From recombination of genes to the estimation of distributions. I. Binary parameters, in: Parallel Problem Solving from Nature—PPSN IV, 1996, 178–187.
[31] S.C. Myers, Determinants of corporate borrowing, Journal of Financial Economics 5 (1977) 147–175.
[32] B. Naudts, L. Kallel, A comparison of predictive measure of problem difficulty in evolutionary algorithms, IEEE Transactions on Evolutionary Computation 4 (2000) 1–15.
[33] S. Nijssen, T. Back, An analysis of the behaviour of simplified evolutionary algorithms on trap functions, IEEE Transactions on Evolutionary Computation 7 (2003) 11–22.
[34] A.E. Nix, M.D. Vose, Modeling genetic algorithms with Markov chains, Annals of Mathematics and Artificial Intelligence 5 (1992) 79–88.
[35] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, BOA: the bayesian optimization algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 1999, pp. 525–532.
[36] M. Pelikan, D.E. Goldberg, Escaping hierarchical traps with competent genetic algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2001, pp. 511–518.
[37] M. Pelikan, Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms, Springer, 2005.
[38] R.J. Quick, V.J. Rayward-Smith, G.D. Smith, Fitness distance correlation and ridge functions, in: Proceedings of the 5th Conference on Parallel Problem Solving from Nature, 1998, pp. 77–86.
[39] C. Reeves, C. Wright, Epistasis in genetic algorithms: an experimental design perspective, in: Proceedings of the 6th International Conference on Genetic Algorithms, 1995, pp. 217–230.
[40] S. Rimcharoen, D. Sutivong, P. Chongstitvatana, A synthesis of optimal stopping time in compact genetic algorithm based on real options approach, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2007, p. 630.
[41] S. Rimcharoen, D. Sutivong, P. Chongstitvatana, Optimal stopping time of compact genetic algorithm on deceptive problem using real options analysis, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2007, pp. 4668–4675.
[42] S. Rimcharoen, D. Sutivong, P. Chongstitvatana, Real option approach to finding optimal stopping time in compact genetic algorithm, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, 2006, pp. 215–220.
[43] S. Rochet, G. Venturini, M. Slimane, E.M. El Kharoubi, A critical and empirical study of epistasis measures for predicting GA performances: a summary, Artificial Evolution (1998) 275–285.
[44] M. Safe, J. Carballido, I. Ponzoni, N. Brignole, On stopping criteria for genetic algorithms, SBIA (2004) 405–413.
[45] R.A. Watson, G.S. Hornby, J.B. Pollack, Modeling building-block interdependency, in: Parallel Problem Solving from Nature PPSN V, 1998, 97–106.
[46] R.A. Watson, J.B. Pollack, Hierarchically consistent test problems for genetic algorithms, in: Proceedings of the IEEE Congress on Evolutionary Computation, 1999, pp. 292–297.