Real Options Approach to Finding Optimal Stopping Time in Compact Genetic Algorithm

Sunisa Rimcharoen, Daricha Sutivong and Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok Thailand
suni16@hotmail.com, daricha.s@chula.ac.th, prabhas@chula.ac.th

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. The real options approach has been applied to many economic and financial problems, but few are in computer science and engineering. The novelty of this work lies in applying real options to a computational problem. This paper proposes using the real options technique to find an optimal stopping decision for the compact genetic algorithm. The compact genetic algorithm, a kind of genetic algorithms, represents the population as a probability distribution over a set of solutions. This distribution automatically captures the underlying uncertainty of the problem, which can be simulated to obtain an evolutionary process of the algorithm. The experiments show preliminary results of employing the real options approach to determine the optimal stopping time for the compact genetic algorithm. The proposed technique can be applied to analyze other machine-learning algorithms, such as neural networks or other variations of genetic algorithms.*

Keywords: real options, optimal stopping time, compact genetic algorithm

1 Introduction

Genetic algorithms are becoming a common technique to solve difficult real-world problems. In spite of many useful practical applications, there are very few studies on an optimal stopping time in genetic algorithms. For example, Aytug and Koehler [1-2] estimated an upper bound of the number of iterations required to achieve a level of confidence to guarantee that a simple genetic algorithm converges. Meyer and Feng [3] proposed a fuzzy stopping criterion for genetic algorithm to establish the termination condition. A critical review of the state-of-the-art in the design of termination conditions can be found in Safe, Carballido, Ponzoni and Brignole [4].

This paper presents a different approach to analyze an optimal stopping time by using the real options approach. 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. Thus, this paper proposes using the real options analysis to address uncertainty in the compact genetic algorithm, namely to find an optimal stopping policy of the algorithm. 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. Using the special class of genetic algorithms, the compact genetic algorithm, the underlying uncertainty can be viewed as a probability distribution. This forms a basis in using the real options approach in order to find values to determine when it is worth stopping running the algorithm.

There are many research works in genetic algorithms and real options. Most earlier research used genetic algorithms or genetic programming as a computational technique in the option pricing model. For example, Chen and Lee [5] studied the application of genetic algorithms to option pricing. Chidambaran, Lee and Trigueros [6] proposed a new methodology that used genetic programming to approximate the relationship between the price of a stock option and the properties of the underlying stock price. Chen, Yeh and Lee [7] also provided some initial evidences of the empirical relevance of genetic programming to option pricing. The pricing formulas are derived from genetic programming and then compared with the Black-Scholes model. In the next year, Chen, Lee and Yeh [8] proposed an extended version of hedging derivative securities with genetic programming. Chidambaran [9] used Monte Carlo simulations to generate stock and option price data to develop a genetic option pricing program. Lazo, Pacheco and Vellasco [10] proposed finding an optimal decision rule for oil field development. In this research, the Monte Carlo simulation was employed in the genetic algorithm for simulating the possible paths of oil prices. Contrary with those research works, this paper proposes applying real options to a computational problem. The

approach opens up a new direction of analyzing the optimal stopping time in terms of investment.

The paper is organized as follows. Section 2 introduces the concept of real options, and section 3 describes detailed techniques of the compact genetic algorithm. Section 4 defines the test problems used in the experiments. Section 5 shows how to model the underlying uncertainty of the problems. The real options valuation function is formulated in section 6. Section 7 presents the results and analysis. Finally, the concluding remarks of this study are in section 8.

2 The Real Options Approach

Real options are a financial concept that applies financial options 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 is a put option. The financial options are useful for managing risks in the financial world. The financial option concept was extended to real assets when Myers [11] 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 decision into two parts: the immediate choice and the remaining decision. The detailed technique is described in Dixit and Pindyck [12].

The value $F_t(x_t)$ is the expected net present value (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 equation (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. 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

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

From equation (2), there are some payoff values as a function of x achieved by termination, and other payoff values as a function of x achieved through continuation. The x values that produce the boundary payoff values form an exercise region with termination being optimal on one side and continuation on the other.

3 The Compact Genetic Algorithm

This section gives an overview of the compact genetic algorithm, its characteristics and its algorithm. The genetic algorithms, the branches of evolutionary computation, are based upon the principle of natural evolution and the principle of the 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 that form the search space for the algorithm. The representation of genetic algorithm is a fixed-length bit string and that of the compact genetic algorithm is a probability vector. In general genetic algorithm, 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 to be parents for creating new individuals or offspring. The survivors are selected from the original population and the offspring to form a new population of the next generation using their fitness values.

The compact genetic algorithm (cGA), proposed by Harik, Lobo and Goldberg [13], is a special class of genetic algorithms. The pseudocode of cGA is shown in figure 1.

```
1) initialize probability vector
   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] ≠ 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;
```

Figure 1. Pseudocode of the cGA

The parameters are population size(n) and chromosome length(l). The cGA represents the population as a probability distribution over the set of solution; thus, the whole population needs not to be stored. In each generation, cGA samples individuals according to the probabilities specified in the probability vector. The individuals are evaluated and the probability vector is updated towards the better individuals. The cGA has an advantage of using a small amount of memory and achieving comparable quality with approximately the same number of fitness evaluations as a simple genetic algorithm.

For example, the update method of the compact genetic algorithm is shown in figure 2, assuming a step size of 0.25.

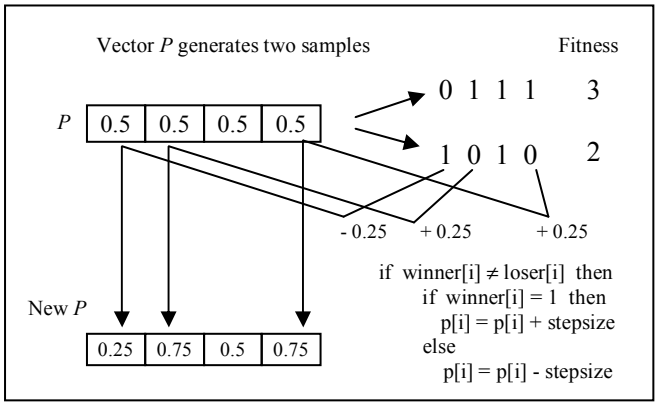

Figure 2. Updating method in the cGA

4 Description of Test Problems

In these experiments, we choose a 10-bit one-max problem and a 3x5 bit trap problem as the test problems. One-max is a simple test problem or a toy problem for a genetic algorithm. This problem finds a maximum value where all bits 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. An example is shown in table 1.

Table 1. Example of one-max problem

Chromosome String	Fitness
1011100010	5
1110010101	6
0010001011	4
1111100000	5
1111111111	10

The trap problem [14] is a difficult test problem for a genetic algorithm. The general k-bit trap function is defined as:

$$F_k(b_0 \ldots b_{k-1}) = \begin{cases} f_{\text{high}} & ; \quad \text{if } u = k \\ f_{\text{low}} - u\dfrac{f_{\text{low}}}{k-1} & ; \quad \text{otherwise} \end{cases} \tag{3}$$

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:

$$F_{k \times m}(B_0 \ldots B_{m-1}) = \sum_{i=0}^{m-1} F_k(B_i),\ B_i \in \{0,1\}^k \tag{4}$$

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. For example, a 3x5 bit trap function is shown in table 2.

Table 2. Example of the 3x5 bit trap function

Ind.	$b_0b_1b_2$	$b_3b_4b_5$	$b_6b_7b_8$	$b_9b_{10}b_{11}$	$b_{12}b_{13}b_{14}$	Fit.
1	111	111	000	111	000	13.0
2	000	000	111	000	111	12.0
3	111	111	011	111	111	12.0
4	111	000	000	111	000	12.0
5	111	001	010	111	111	11.0
6	000	000	000	000	111	11.0
7	111	001	110	111	111	10.0
8	000	000	000	000	000	10.0

5 Modeling Underlying Uncertainty

The underlying uncertainty of the compact genetic algorithm is naturally its fitness value. According to the algorithm, when a candidate solution is sampled from the probability distribution, it is evaluated and the fitness value is assigned. This value is associated with the distribution. In order to characterize change in the fitness value in the compact genetic algorithm, the algorithm is simulated many times, statistics of the fitness movement are collected. Generally, the fitness value will increase over time, as the probability distribution is evolved.

To model the uncertainty in the real options application, the general process is to identify the key uncertainties and to model them using a stochastic process that fits the problem, such as a geometric brownian motion or a mean-reverting process. In the compact genetic algorithm, however, the uncertainty can be viewed as the change of the fitness value in each step. At the beginning, the average fitness value of a 10-bit one-max problem is 5.0 because we initialize the probability vector with a uniform distribution. In the next step, this fitness may rise from 5.0 to 6.0, 7.0, …, 10.0 or

fall to 4.0, 3.0, …, 0.0. We can find the probability of occurrence of these values and use it to characterize the underlying uncertainty of the compact genetic algorithm.

In this work, we model the uncertainty of the compact genetic algorithm by observing the fitness values from many runs and keeping track of them over time. Because the underlying uncertainty of this problem can be automatically obtained by simulating the compact genetic algorithm, we do not need to employ any particular stochastic process, such as a geometric brownian motion or a mean-reverting process. We can construct a probability tree of the compact genetic algorithm straightforwardly. Using this method, real options can be applied to a wide variety of applications that use the learning method including the genetic algorithm.

By running the compact genetic algorithm, we have fitness values in each generation (time step). We accumulate the possible changes of fitness values in each generation over many runs and then calculate the probability of all possible values in each state. For example, in a 10-bit one-max problem, the possible average values are 0.0, 0.5, 1.0, .., 9.0, 9.5, and 10.0. The range of these values is increased by 0.5 because the compact genetic algorithm has two populations, so the average fitness of two individuals ends with .0 or .5. Therefore, a 10-bit one-max problem has 21 possible values. Figure 3 shows the lattice of all possible values along with their associated probability.

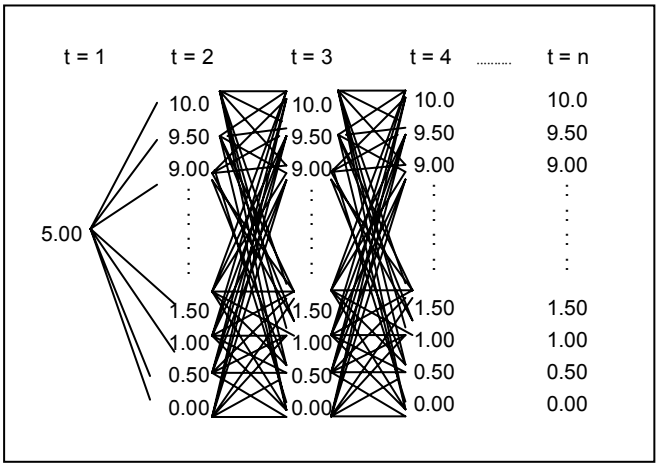

Figure 3. Lattice of a 10-bit one-max problem

6 Value Function of Option

This paper proposes applying the real options analysis to the compact genetic algorithm. We select the compact genetic algorithm because the uncertainty can be directly represented. The underlying uncertainty depends on a probability vector. In each time step, two individuals are sampled from the distribution and fitness values of these candidates are assigned by the evaluation routine. The probability vector drives these values, and the algorithm uses these values to update the probability vector according to the best candidate. A certain cost per one sampling is assigned in order to account for an effort spent in running the algorithm. The average fitness of these candidates is used as a representative fitness value. As the candidate solutions are sampling from the probability vector, there is a chance that one sampling is good and the other is bad. Therefore, we use the average value to be a representative of the information in order to neutralize the event.

Let $\pi(x)$ denote the profit, and $\Omega(x)$ is the termination payoff. We apply the Bellman equation, where the value function is

$$F(x) = \max \left\{ \Omega(x),\ \pi(x) + \frac{1}{1+\rho} \varepsilon[F(x') \mid x] \right\}. \quad (5)$$

The termination payoff is shown in equation (6).

$$\Omega(x) = g(x) * v \quad (6)$$

Where $g(x)$ is the fitness value of x, and v is the price. We illustrate the method with a simple example. In this case, there is no profit and discounting. Equation (5) becomes

$$F(x) = \max \left\{ \Omega(x),\ \varepsilon[F(x') \mid x] \right\}. \quad (7)$$

Note that in this work we do not use the discount factor because in each state the compact genetic algorithm takes a few milliseconds to run; thus, the future value is not distinguishable from the present value. We also ignore the profit term $\pi(x)$ because the compact genetic algorithm does not produce any immediate profit flow. The solution value is obtained from the fitness value at the time the algorithm terminates.

To implement this idea, we assume that one sampling costs one dollar and one fitness value is worth 100 dollars. The compact genetic algorithm samples two individuals, so it must pay two dollars in each generation. Here, the termination payoff is the fitness value multiplied by 100 dollars and the continuation must pay two dollars for a new sampling because the compact genetic algorithm requires two evaluations per time step. We formulate the option value of this case as below:

$$F(x) = \max \{ g(x)*100,\ \varepsilon[F(x') \mid x] - 2 \}. \quad (8)$$

In these benchmark problems, we assume artificial cost and price in order to test the model. However, in the real-world problem, the fitness value's worth and the algorithm cost can be determined according to the application. For example, in a bin packing problem, we know how a profit

depends on the number of pieces packed into the bin. Thus, equation (8) can be adapted to real-world parameter values.

7 Results and Analysis

For preliminary studies, we use a 10-bit one-max problem and a 3x5 bit trap problem as the test examples. First, we solve these problems with the compact genetic algorithm and keep track of the probability distribution of each fitness value over time. The probabilities are averaged over 10,000 runs. The number of generations or time steps is set to 100. We use these data to construct a lattice of the fitness distribution. Second, we calculate an option value according to equation (5) using a dynamic programming approach. The option values are averaged over 100 runs. Finally, we summarize an option value and an exercise policy. We also calculate the standard deviation (sd) of the option values obtained over 100 runs and plot the values of ± 1sd in the graph to illustrate the confidence level in the answer. Figure 4 shows the exercise region of a 10-bit one-max problem.

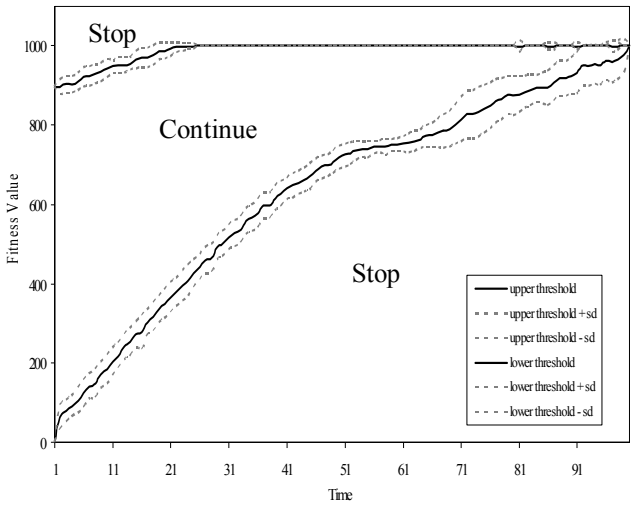

Figure 4. Exercise region of a 10-bit one-max problem

As shown in figure 4, the algorithm should decide to stop the search when the fitness value rises above the upper threshold because the fitness value is already high and it is not worth the sampling cost to continue. If the fitness value is lower than the lower threshold, the algorithm should also decide to stop because with the current population, it is unlikely to achieve a better result relative to the cost required. In this problem, the option values play an important role for an algorithm in deciding whether to stop or continue. If an option has a positive value, the algorithm will decide to put an effort to search for more samplings. On the other hand, if an option has no value, the algorithm will terminate or reset. This decision helps the algorithm to avoid useless effort and time spent on a valueless sampling.

Most people use the genetic algorithm to find an optimal solution regardless of time and effort used to achieve it. However, from efficiency standpoint, the algorithm should trade off between effort and solution improvement in deciding when to stop. The real options approach shows that we can analyze the optimal stopping time for running the algorithm. In the above experiment, we show a preliminary result that employs the real options approach in determining the optimal stopping time of the compact genetic algorithm in a simple problem. Next, we also show the experiment in a more difficult problem, namely the trap problem.

In the trap problem, a nearly optimal solution deludes the compact genetic algorithm into the trap. Therefore, the probability of success is less than the one-max problem. We test the proposed technique with the trap problem besides the one-max problem. First, we run the problem with the compact genetic algorithm and keep the probability distribution of each fitness value over time. The probabilities are averaged over 10,000 runs. Then, we use these data to construct a lattice of fitness distribution. Using this lattice, we can calculate an option value according to equation (5) using a dynamic programming approach. The option values are averaged over 100 runs. Finally, the option value and the exercise policy are obtained. Figure 5 illustrates the exercise region of the 3x5 bit trap problem.

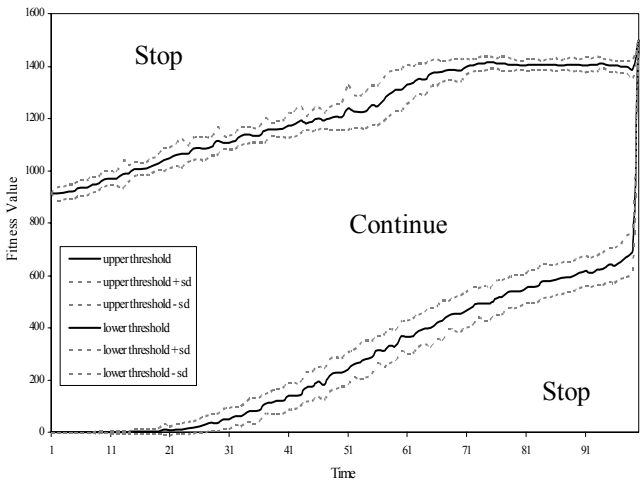

Figure 5. Exercise region of a 3x5 bit trap problem

There are many differences between the exercise regions of the two problems. In a one-max problem, the threshold is clearer than the trap problem. The exercise threshold of the one-max problem confirms that the solution quality in this problem is gradually improving. It guarantees that the compact genetic algorithm will achieve the optimal solution if the fitness value lies within a continuation region. On the other hand, the fitness value in the trap problem can be fluctuating. The solution may fall in a trap. Thus, although the fitness value falls into the continuation region,

it does not guarantee to achieve the optimality. The upper bound threshold of the trap problem does not reach the optimal solution. This characteristic also shows a rarity of finding an optimal solution in a hard problem.

From figure 4 and figure 5, the exercise regions suggest that, at the beginning, the one-max problem requires a higher solution quality for stopping than the trap problem. This is because good solutions abound in the one-max problem. On the other hand, good solutions in the trap problem are rare. The one-max 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. On the contrary, the harder trap problem attempts to keep a lower-fitness solution in order to maintain diversity.

Note that the standard deviation of the upper threshold in the one-max problem drops to zero toward the end. Since the upper bound is at the optimal fitness and the continuation region converges toward this optimal value, this explains an important insight that the compact genetic algorithm guarantees the optimal solution in the one-max problem, given that the evaluations stay in the continuation region.

8 Concluding Remarks

This paper proposes applying the real options technique to finding an optimal stopping decision for running the compact genetic algorithm. The novelty of this work lies in introducing a new methodology to determine an optimal stopping time of a machine-learning algorithm. In the experiment, we show preliminary results from employing the real options approach to analyze the 10-bit one-max problem and the 3x5 trap problem. The results illustrate that the proposed technique can provide a stopping strategy for the algorithm. For the studied problems with the compact genetic algorithm, the exercise regions are broken into three areas. Specifically, the algorithm should stop the search when the fitness value rises above the upper threshold or when the fitness value falls below the lower threshold. This methodology can also be used to analyze the characteristic of other learning algorithms, such as neural networks or other variations of genetic algorithms.

References

[1] H. Aytug and G. J. Koehler, Stopping criterion for finite length genetic algorithms, in INFORMS Journal on Computing, 1996.

[2] H. Aytug and G. J. Koehler, New stopping criterion for genetic algorithm, in European Journal of Operational Research, 2000.

[3] L. Meyer and X. Feng, A fuzzy stop criterion for genetic algorithms using performance estimation, In proceedings of the Third IEEE Conference on Fuzzy Systems, 1994.

[4] M. Safe, J. Carballido, I. Ponzoni and N. Brignole, On stopping criteria for genetic algorithms, in SBIA, 2004.

[5] S-H Chen and W-C Lee, Option pricing with genetic algorithms: separating out-of-the-money from in-the-money, in Proceeding of the IEEE International Conference on Intelligent Processing Systems, 1997.

[6] N. K. Chidambaran, C. H. J. Lee and J. R. Trigueros, An adaptive evolutionary approach to option pricing via genetic programming, in Proceeding of the Third Annual Genetic Programming Conference, 1998.

[7] S-H Chen, A-H Yeh and W-C Lee, Option pricing with genetic programming, in Proceeding of the Third Annual Genetic Programming Conference, 1998.

[8] S-H Chen, W-C Lee and C-H Yeh, Hedging derivative securities with genetic programming, in International Journal of Intelligent Systems in Accounting, Finance and Management, 1999, 8(4): 237-251.

[9] N. K. Chidambaran, Genetic programming with monte carlo simulation for option pricing, in Proceeding of the 2003 Winter Simulation Conference, 2003.

[10] J. G. L. Lazo, M. A. C. Pacheco and M. M. B. R. Vellasco, Real option decision rules for oil field development under market uncertainty using genetic algorithms and monte carlo simulation, in the Seventh Annual Real Options Conference, 2003.

[11] S. C. Myers, Determinants of corporate borrowing, in Journal of Financial Economics, 1977, 5(2): 147-175.

[12] A. K. Dixit and R. S. Pindyck, Investment under uncertainty, Princeton University Press, Princeton, NJ, 1994.

[13] G. R. Harik, F. G. Lobo and D. E. Goldberg, The compact genetic algorithm, in IEEE Transactions on Evolutionary Computation, 1999, 3(4): 287-297.

[14] D. H. Ackley, A connectionist machine for genetic hillclimbing, Kluwer Academic Publishers, Boston, MA, 1987.