Analysis of LZW Differential Evolution for Binary Encoding

Orawan Watchanupaporn and Worasait Suwannik

***Abstract*— Differential Evolution (DE) is a fast and robust real vector optimizer. This paper applies DE to discrete problems by converting a real chromosome to an integer chromosome and then decompress to a binary chromosome using LZW algorithm. Experimental result shows that this approach is better than the previous work and the evolution time is very fast. Analysis result shows that the fitness landscape of LZW encoding is less complex than the original encoding for each test problem.**

***Index Terms*— Differential Evolution, LZW, Discrete optimization, Fitness Landscape**

I. Introduction

DIFFERENTIAL Evolution (DE) is an evolutionary algorithm designed for solving real value optimization problems [1]. DE is very fast and efficient. It was ranked the third in the First International Contest on Evolutionary Optimization in 1996. However, it is more robust than those optimizers finished before [2]. In addition, DE is very compact. The core of the algorithm can be implemented in less than 20 lines of C code, which is available on-line [3].

DE performs very well in continuous optimization. However, for discrete optimization, there are a few works that investigate DE's effectiveness [4]. This paper presents two alternative methods for adapting DE for discrete optimization. The first method directly maps a real value chromosome to a binary chromosome. The second method combines compressed chromosome encoding with DE.

Compressed encoding enables evolutionary algorithm to solve very large problems [5][6][7]. For example, LZW encoding in Genetic Algorithm can solve one-million-bit problems. To use compression with GA, the individual is in a compressed form and has to be decompressed before the fitness evaluation. Another advantage of this approach is low memory requirement.

The motivation for using compress encoding is to reduce the size of the search space so that the solution can be found faster. However, in some cases, LZW encoding can solve problem faster even when the size of the search space is equal to the original encoding. This means the LZW encoding not only can reduce the search space, it also aid the evolutionary search process. While the effect of search space reduction can be measure easily by comparing the size of the search space, the effect of using LZW encoding is difficult to be explained. Another contribution of this paper is to analyze LZW compressed encoding.

We use the method proposed by Uludag and Uyar [8] to analyze the fitness landscape of DE. The idea of the analysis method is to random walk on the fitness landscape. A new step is obtained from the proposed neighborhood function which is suitable for DE. In addition, we define a new distance metric that is suitable for LZW encoding.

The organization of this paper is as follows. Section 2 describes DE. Section 3 explains how LZW compressed encoding is applied to DE. Section 4 explains benchmark problems. Section 5 describes the experiment. Section 6 discusses the result. Section 7 analyzes LZW encoding and its interaction with DE. Finally, Section 8 summarizes the paper.

II. Differential Evolution

Differential Evolution (DE) is an evolutionary optimization method. The first generation of real vectors is created by randomly filled the values in the vectors. Each vector has D values. A population consists of NP vectors. There are two schemes (i.e., DE1 and DE2) presented in [4]. In this paper, DE1 is used.

A new generation is created by the following method. Each vector competes with its trial vector. The one with less cost survives to the next generation. A trial vector is created by combining the vector with a mutant vector. The combination is similar to crossover in Genetic Algorithm [9]. A mutant vector is created by adding a random vector with a weight difference of other two random vectors (hence the name Differential Evolution). The mathematical formula for creating a mutant vector is as follows:

$$X'_c = X_c + F(X_a - X_b) \quad (1)$$

The parameters in DE are listed below.

- NP (or population size) should be 5-10 times the number of parameters D.
- F (i.e., the weight) should start with 0.5. F and NP should be increased if the algorithm converges prematurely.
- CR (or the crossover rate) should be 0.9, 0.1, or 0.

Manuscript received March 20, 2013.

O. Watchanupaporn is with Department of Computer Science, Kasetsart University, Bangkok, Thailand (phone: +66-8-9148-6740; e-mail: orawan.liu@gmail.com).

W. Suwannik is with Department of Computer Science, Kasetsart University, Bangkok, Thailand (e-mail: worasait.suwannik@gmail.com).

DOI: 10.5176/2251-3043_3.1.233

III. LZW Differential Evolution

Lempel-Ziv-Welch Algorithm (LZW) is a lossless dictionary-based data compression/decompression algorithm [10]. The input of the compression algorithm is a character string. The output of the compression algorithm (also the input of the decompression algorithm) is an array of integer codes. The output of the decompression algorithm is the original character string. In LZWDE, only decompression algorithm is used. The pseudo code of LZW decompression is given in Fig. 1.

The compression/decompression algorithms start with a dictionary which the number of entries is equal to the number of characters. Each entry contains one character. For example, when using LZW to compress/decompress an English text, the dictionary is initialized with all English characters and symbols. However, when LZW is used to compress or decompress a binary chromosome in GA, the dictionary is initialized with the number 0 and 1. Fig. 2 shows an example of decompressing an array of integer to a binary string. During the compression, the algorithm dynamically expands the dictionary and outputs codes that refer to strings in the dictionary. Normally, the number of bits of the code is less than that of the variable length string in the dictionary. Data is compressed when the algorithm replaces the whole string with its code. The dictionary does not have to be stored because the algorithm can construct the dictionary during the compression or decompression process.

To use LZW compressed encoding with DE, we add a conversion and decompressing step before a fitness evaluation. The real value chromosome is converted to an array of integers. After that, the array is decompressed to a binary string. Because LZW cannot decompress arbitrary input, each code in an integer array must satisfy the following constraint [6].

$0 \le a_i \le i+1$, where i is a zero-based array index

Any positive integer can be changed to satisfy the constraint by modulo with $i+2$. An example of converting a real vector to a binary string is shown in Fig. 3.

Implementing an LZW chromosome encoding in object-oriented language is easy. The core algorithm does not have to be modified to support LZW encoding. Rather, for each benchmark problem, we implemented the interface for fitness evaluation using two classes: one for a normal chromosome and the other is for a compressed chromosome. For DE's point of view, it still evolves real vectors. It does not know that it is evolving compressed encoding chromosomes.

Note that LZWDE evolves a direct representation of an individual as a "compressed" string. There is no compression step involved in LZWDE.

IV. Benchmark Problems

We use synthetic problems to assess the strengths and weaknesses of LZW encoding. The advantage of using a synthetic problem is that its structures (i.e., relationship between variables) are known. Thus, we can assume that if an algorithm can solve the problem, it can also solve a class of problems that has the same structure. Moreover, an algorithm that can solve problems with more complex structures is more sophisticated and is likely to solve a problem with a simpler structure.

```
Algorithm LZW Decompress
    add entries 0 and 1 to the dictionary
    read one code from input to c
    output str(c)
    p = c
    while input are still left
        read one code from input to c
        if the code c is not in the dictionary
            add str(p) + fc(str(p)) to the dictionary
            output str(p) + fc(str(p))
        else
            add str(p) + fc(str(c)) to the dictionary
            output str(c)
        else if
        p = c
    end while

The variable c stores a code read from input.
The variable p is the previous value of c.
The function str(code) returns a string associated with code.
The function fc(string ) returns the first character in string.
```

Fig. 1. The pseudo code of LZW decompression.

Decode		Dictionary	
Input	Output	Index (c) *Initial table*	Full string
		0	0
		1	1
		Start enter string to dictionary	
0	0	-	-
2	00	2	00
1	1	3	001
3	001	4	10
1	1	5	0011
1	1	6	11

Fig. 2. Example of decompressing an array of integers to a binary string.

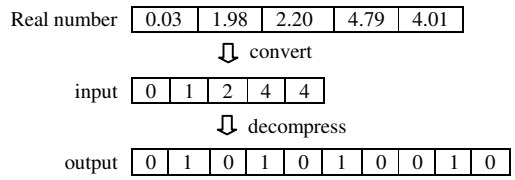


Fig. 3. Converting a real value chromosome to an integer chromosome and decompressing it to a binary chromosome.

In [4], the author applied DE to solve the following discrete optimization problems: OneMax, Royal Road, Order-3 Deceptive, and Long Path problems. We test the performance of our algorithm using the same benchmark. Every benchmark problem is a maximization problem. However, since DE is a global minimizer, the fitness is transformed by multiplying the cost function with −1.

A. OneMax Problem

The OneMax problem [11] (or bit counting) is a widely used problem for testing the performance of various genetic algorithms. Formally, this problem can be described as finding a string $X = \{x_1, x_2, \dots, x_k\}$, where $x \in \{0,1\}$, that maximizes the following equation:

$$F(X) = \sum_{i=1}^{k} x_i \quad (2)$$

B. Royal Road Problem

Royal Road problem [12] is designed to investigate the role of GA crossover and building block hypothesis. The problem can be solved using GA which uses crossover. However, it is difficult for a hill climbing algorithm or GA with a single-bit mutation to solve the problem.

This function involves a set of schemas S and is defined as:

$$F(X) = \sum_{s \in S} c_s \sigma_s \quad (3)$$

where x is a bit string, each c_s is a value assigned to the schema s.

C. Deceptive Order-3 Problem

In Deceptive problem [13], an individual composes of several blocks. Each of the blocks is evaluated by a deceptive function. The deceptive function can fool the gradient-based optimizers to favor zeros, but the optimal solution is composed of all ones. It is a fundamental unit for designing test functions that resist hill-climbing algorithms. The order-3 deceptive function is defined as:

$f(000) = 28$
$f(001) = 26$
$f(010) = 22$
$f(100) = 14$
$f(011) = 0$
$f(101) = 0$
$f(110) = 0$
$f(111) = 30$

The deceptive problem can be decomposed to several deceptive functions. The problem, denoted by f_m, is defined as:

$$f_m(K_1 \cdots K_m) = \sum_{i=1}^{m} f(K_i), K_i \in \{0,1\}^3 \quad (4)$$

D. Long Path Problem

Long Path problem [14] is a problem that can be solved by a hill-climbing algorithm. However, it is not practical to solve this problem using hill climbing algorithm. This is because climbing the hill (or the path) takes exponential time. Each point in the path is differed by one bit. The path is constructed such that is exponentially long. The height from the bottommost of the hill to the top is equal to:

$$HillHeight\ (l) = 3 \times 2^{\lfloor (l-1)/2 \rfloor} + l - 2 \quad (5)$$

where l is a chromosome length.

V. Experiment

We conducted the experiment to compare the performance of LZWDE with Gong and Tuson's binary adapted DE operators [4] and with simple real to binary conversion DE. The latter scheme, which is simply called DE, converts a real value to a binary using the rule ($X_i < 0.5$? 0 : 1)

Table I shows the experimental parameters. The length of an LZWDE chromosome is less than DE chromosome which are 1/5 of OneMax problem size, 1/4 of Royal Road problem size, 1/12 of Deceptive order-3 problem size, and about 1/3 of Long Path problem size. Before a fitness evaluation, the compressed chromosome is decoded and decompressed with LZW decompression algorithm. The length of the decompressed binary chromosome is varied depending on the code in the integer array. If the length is more than the size of the problem size, the excess bits are discarded. However, if the length is less than the problem size, LZWDE will evaluate the fitness of available bits. All experimental results are the average performance obtained from 30 runs.

TABLE I
Experimental Parameters

Parameter	OneMax	Royal Road	Deceptive order-3	Long Path
Population size	50	30	100	30
Problem size	500	80	300	29
LZW chromosome length	100	20	25	10
Maximum generation	500	500	2000	300

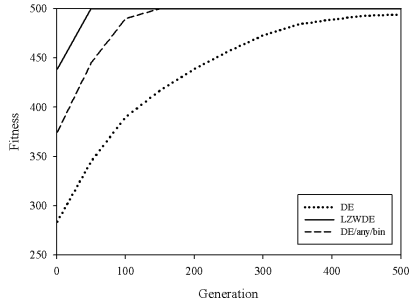
VI. Results

Gong and Tuson [4] used different sets of parameters for OneMax, Royal Road, Deceptive Order-3 and Long Path problems. They reported the result of 4 DE strategies which are: 1) any-change mutation and exponential crossover-DE/any/exp, 2) any-change mutation and binomial crossover-DE/any/bin, 3) restricted-change mutation and exponential crossover-DE/res/exp, and 4) restricted-change mutation and binomial crossover-DE/res/bin for each problem. We choose the best of their experimental results and compare them with our best parameters for each problem.

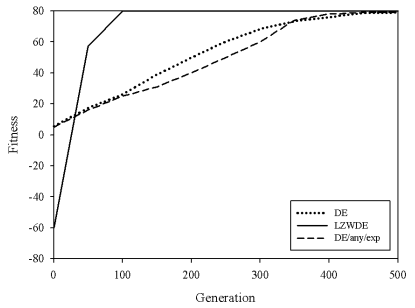
For each benchmark problem, we compare the performance of binary-adapted DE, DE (simple real to binary conversion), and LZWDE. The result is shown in Figure 4. The X-axis shows the number of generations and the Y-axis shows the average-best fitness. LZWDE outperforms both DE and binary-adapted DE. Moreover, it is interesting to see that the performance of simple conversion is comparable to binary-adapted DE in Royal Road problem and better than binary-adapted DE in Long Path problem.

Table II shows the average evolution time. We ran the experiment on Intel Core i5 with 4GB of RAM. In this table, we report only the time that DE successfully finds the solution. The number in the parenthesis is the success rate. LZWDE can find the solution for every run. We do not have the data for binary-adapted DE. Therefore, we only compare

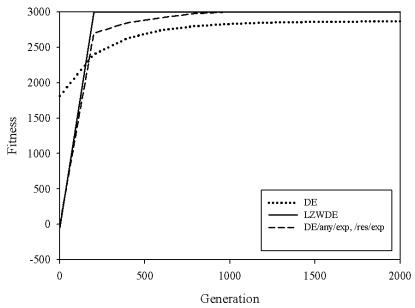
the time of DE and LZWDE. In LZWDE, there is an LZW decompression step. Even with an additional step, the algorithm can still find a solution faster than DE. Simple DE cannot find a solution for Deceptive Order-3 problem.



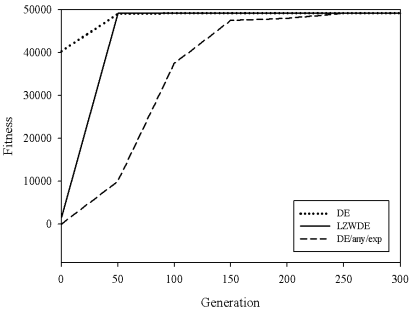
(a) OneMax



(b) Royal Road



(c) Deceptive order-3



(d) Long Path

Fig. 4. The average-best fitness plotted against generation

TABLE II
AVERAGE EVALUATION TIME (IN MILLISECONDS)

Problem	Algorithm			
	DE		*LZWDE*	
OneMax	388.43	(100)	6.50	(100)
Royal Road	36.42	(87)	29.10	(100)
Deceptive order-3	-	(0)	113.97	(100)
Long Path	13.53	(100)	45.69	(98.84)

VII. FITNESS LANDSCAPE ANALYSIS

In this section, we tried to explain why LZW encoding help improve the performance of DE.

A. Binary Fitness Landscape

The difficulty of a problem depends on two factors: the size of search space and the shape of fitness landscape. A problem with a larger search space is usually more difficult to solve. In addition, a problem with a more complex fitness landscape is more difficult. Example of complex fitness landscape is the one with many local minima or the one that leads evolutionary search away from the global minima.

To visualize the fitness landscape for binary optimization problem, we enumerate every possible chromosomes, evaluate their fitness and measure distance from the solution, then plot the graph using the fitness and the distance. Fig. 5(a) shows the fitness landscape of a 9-bit OneMax problem. The X-axis is the number of bits by which a chromosome differs from the solution. The Y-axis is the chromosome's fitness value. The darker area indicates a higher chromosome density. As shown in Fig. 5(a), as the fitness increases, the chromosome is closer to the OneMax solution. Since evolutionary algorithm use fitness value to guide a search process, OneMax is an easy problem because the fitness value can guide the search to the correct direction.

Fig. 5(c) shows the fitness landscape of a 9-bit Trap problem. The problem is more difficult to solve than OneMax because the fitness landscape deceives the search into moving away from the global optima. As the fitness increase, the chromosome is more different from the solution. If we try to solve the Trap problem using a local search which produces a neighbor with 1 bit different from the current position, the search will not be able to find the optimal solution.

Fig. 5(a) and (c) visualize the fitness landscape of two extreme. We can easily tell from the graph which problem is easier. However, for a problem with difficulty in between, a subjective judgment should not be used to judge the complexity of fitness landscape. Therefore, we quantify the shape of a fitness landscape as one single number called fitness-distance correlation (*fdc*). We compute *fdc* or a correlation between fitness and distance using the formula given below.

$$fdc = \frac{cov(F,D)}{\sigma(F)\sigma(D)} \tag{6}$$

where $cov(F,D)$ is a covariance of fitness F and distance D. $\sigma(F), \sigma(D)$ is a standard deviation of F and D respectively.

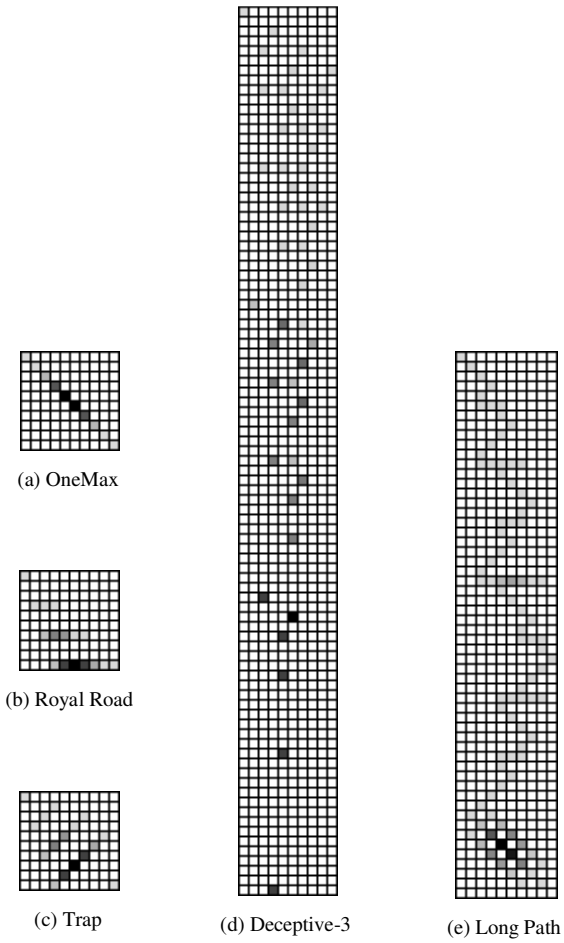


Fig. 5. The fitness landscape for binary optimization problems which are (a) OneMax, (b) Royal Road, (c) Trap, (d) Deceptive-3, and (e) Long Path. All problem sizes are 9-bit.

The *fdc* of the test problem is shown in Table III. For GA, OneMax's *fdc* is −1, which is the lowest. Deceptive problem has positive *fdc* which means that as the fitness increase, the chromosome is getting further from a solution.

TABLE III
THE FDC OF THE TEST PROBLEM

Problem	Algorithm		
	GA	***DE***	***LZWDE***
OneMax	-1.00	-0.63132	-0.78203
Royal Road	-0.65	-0.44755	-0.65961
Deceptive order-3	0.32	0.16866	-0.08296
Long Path	0.02	-0.00091	-0.07498

B. Real-value Fitness Landscape

Our paper use DE to solve binary problem. DE use real value vectors. The *fdc* cannot be calculated using the same method as in the previous subsection because of we cannot enumerate all possible real-value vectors as we enumerate all possible binary chromosome. For a binary optimization problem, there are finite amount of chromosomes given a fixed length binary string. A problem size n bit has 2^n possible chromosome. However, a single real-value in a DE vector, in theory, can have infinitely uncountable possible values.

Since we cannot enumerate all possible chromosomes, we instead explore the fitness landscape using random walk. While an analysis procedure performs random walk, it records a fitness and distance to a solution. Each step of random walk imitates a trial vector generation process in DE.

$$vector_{t+1} = vector_t + F(random\_unit\_vector) \quad (7)$$

In this paper, we set the value of F equals to 0.1 in order to make the step not too long. For each problem, an analysis procedure explores 100 random starting points. For each starting point, the procedure random walks for one million steps. A real value in the vector is constrained within the range [0, 1].

Another difference between binary and real value analysis is as follows. For a binary problem, we calculate a Hamming distance from a chromosome to an optimal solution. In DE, Euclidean distance is calculated. The distance calculation depends on how real-to-binary conversion is done. In this paper, the rule for converting is $X_i < 0.5 \; ? \; 0 : 1$. Therefore, if a one bit of binary solution is 1, and the corresponding real value is in the range [0.5, 1), the distance would be zero. Otherwise, the distance would be $0.5 - X_i$. If a binary solution is 0, the distance would be zero when the corresponding real value is in the range [0, 0.5). Otherwise, the distance would be $X_i - 0.5$.

Table III shows *fdc* for each problem. Real value *fdc* and binary *fdc* are different due to the way we measure the distance and perform the random walk.

C. LZW Real-value Fitness Landscape

Although both DE and LZWDE use real value vectors, the procedure to calculate the distance is different. In LZWDE, a real-value vector has to be converted to an array of integers before decompression and fitness evaluation. Thus, the distance calculation depends on how real to integer conversion is done. In this paper, conversion is done simply by truncating a fraction part of a real number. An example of measuring the distance is as follows. Suppose that one integer in a solution array is 3. If the corresponding real value Xi is in the range [3, 4), the distance would be zero. If X_i is less than 3, then the distance would be $3 - X_i$. Otherwise, the distance would be $X_i - 4$. To calculate a distance of a vector to a solution vector, the Euclidean distance formula is used.

The random walk process is similar to the previous subsection. The difference is that each real value X_i is constrained to the range [0, i+2). The value within this range can be converted to a valid input for LZW decompression algorithm.

For some problem such as OneMax, the original binary encoding has only one solution. However, when the problem is encoded with LZW, there might be more than one solution. For example, an LZW chromosome of length 4 has 2 solutions for 9-bit OneMax problem. In that case, the minimum distance from a vector to both solutions is used to compute *fdc*.

Table III shows *fdc* for each problem. For each test problems, LZW encoding has lower *fdc* than the original encoding. This explains why LZWDE performs better than DE.

VIII. Conclusion

This paper proposes two methods to apply DE to solve discrete optimization problem. The first is simple real-to-binary conversion. The second is using LZW encoding. We compared the result with binary-adapted DE using the same benchmark problems. The result shows that LZWDE outperforms binary-adapted DE and DE with simple real-to-binary conversion. In addition, in term of computation time, LZWDE is very fast even it has to decompress the chromosome. It can solve all benchmark problems in less than one second using a mid-range computer.

Using LZW can speed up evolutionary search because of reduction in search space and transformation of fitness landscape. The latter points are backed up by the analysis. This paper proposed two distance metrics, one for DE and another for LZWDE, to analyze simple real-to-binary conversion and LZW encoding. These metrics in used with a neighborhood function to compute fitness distance correlation (*fdc*). The result shows that, in the benchmark problems, LZW encoding can simplify the fitness landscape.

References

[1] R. Storn and K. Price, "Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces," 1995.

[2] K. V. Price, R. M. Storn and J. A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization," Springer, 2005.

[3] K. Price and R. Storn, Differential Evolution, http://www.drdobbs.com/architecture-and-design/184410166, 1997.

[4] T. Gong and A. Tuson, "Differential Evolution for Binary Encoding," Soft Computing in Industrial Applications, vol. 39, pp. 251-262, 2007.

[5] O. Watchanupaporn, N. Soonthornphisaj, and W. Suwannik, "A Performance Analysis of Compressed Compact Genetic Algorithm," ECTI Transactions on Computer and Information Technology, vol. 2, no. 1, 2006, pp. 16-24.

[6] N. Kunasol, W. Suwannik, and P. Chongstitvatana, "Solving One-Million-Bit Problems Using LZWGA," Proceedings of International Symposium on Communications and Information Technologies (ISCIT), 2006, pp. 32-36.

[7] W. Suwannik and P. Chongstitvatana, "Solving One-Billion Bit Noisy OneMax Problem using Estimation Distribution Algorithm with Arithmetic Coding," Proceedings of IEEE Congress on Evolutionary Computation (CEC 2008), 2008, pp. 1203-1206.

[8] G. Uludag and A. S. Uyar, "Fitness landscape analysis of differential evolution algorithms," Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (ICSCCW 2009), 2009, pp. 1-4.

[9] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison- Wesley, 1989.

[10] T.A. Welch, "A Technique for High-Performance Data Compression," IEEE Computer, vol. 17, no. 6, 1984, pp. 8-19.

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

[12] M. Mitchell, S. Forrest, and J. H. Holland, "The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance," in Proc. The First European Conference on Artificial Life, Cambridge, MA, MIT Press, 1991, pp. 245-254.

[13] D. E. Goldberg, "Genetic algorithms and Walsh functions: Part I, a gentle introduction," Complex Systems, vol. 3, 1989, pp. 129-152.

[14] J. Horn, D. E. Goldberg, and K. Deb, "Long Path Problems," Lecture Notes in Computer Science, vol. 866, 1994, pp. 149-158.

Orawan Watchanupaporn is a Ph.D. student at the Department of Computer Science, Kasetsart University, Thailand. She obtained a bachelor's degree in Computer Science from Bangkok University in 2004 and master's degree from Kasetsart University in 2006. Her research interests include compressed compact genetic algorithms and Estimation of Distribution Algorithms.

Worasait Suwannik received a Ph.D. in computer engineering from Chulalongkorn University, Thailand, in 2006. At present, he is a lecturer at the Department of Computer Science, Kasetsart University, Bangkhen Campus, Thailand. His research interests include compressed genetic algorithms and GPU programming.