Dictionary Based Estimation of Distribution Algorithms

Chalermsub Sangkavichitr and Prabhas Chongstitvattana
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Chalermsub.S@hotmail.com and Prabhas.C@chula.ac.th

***Abstract*— This paper proposes a new algorithm in the field of Estimation of Distribution Algorithms. The proposed algorithm combines a data compression algorithm to extract a model into the dictionary. This dictionary is used as a part of the generator to generate the better next generation of population. The proposed method is tested and compared with two well-known Genetic Algorithms, QHFC and BOA. Both algorithms can solve difficult problems and are claimed to scale well. Our algorithm compared well with them. We also report the behaviour of using dictionary as the model for EDAs.**

I. INTRODUCTION

The modern Genetic Algorithm has incorporated model building into the process. The model is then used as a generator to evolve the population (of candidate solutions) to the next generation. This is in contrasted to the traditional Genetic Algorithm which relies on genetic operators to evolve the next generation. The use of model as a generator is therefore the landmark of modern Genetic Algorithms. This class of Genetic Algorithms is named Estimation of Distribution Algorithms [1] after the process of model building in Genetic Algorithms. These new generation of Genetic Algorithms are shown to scale well to large scale problems. However, the process of model building is still an open research.

This work proposes the application of data compression to model building process. The data compression algorithm such as LZW [2] will store the fragments that occur frequently into a dictionary. This dictionary may be implicit (it can be reconstructed on the fly while performing decompression). This is the basis of which we use to build the model for our algorithm. There are some researches using LZW in their work [6][7].

Among good individuals in each generation, there are some parts of chromosomes that appear in the same pattern of data in the same position (e.g. ten bit chromosome no.1: ***1101*** and no.2: ***1101***, they have the same bit pattern at fourth to seventh position). These patterns are regarded as Building Blocks (BBs), so we want to record these data and use them to reproduce better offspring in the next generation. The traditional Genetic Algorithm such as Simple Genetic Algorithm, never exploits this knowledge. However EDA can use probability vector between each bit position to bias the patterns. There are many different methods to build models.

LZW algorithm can help to record the repeated patterns that appear among population. The technique that makes it possible, is sharing dictionary with all individuals in a population, this means that the words that ever appeared in prior individuals will be referred by the following individuals. Thus individuals have been shared fraction parts in theirs chromosome together but the reference position may be different. The problem is how to construct these fraction parts into a new individual. A map is need to guide the position of each fraction part. We use the position that each fraction part begin and sort it in order, so there may be many fraction parts at the same position. In operation, to construct new individual, we begin selecting the first fraction part and then select next in sequence, until the individual is complete.

Generating a new individual from the dictionary can be considered as combinatorial problem. Although, we have a map but we do not know, which is the suitable fraction part for each position in the map. The evolutionary aspect of Genetic Algorithms helps to solve this problem. The better individuals will survive to the next generation and eventually the good model will emerge and also the solution.

In this paper, we compare the results with Hierarchical Fair Competition (QHFC) [3] and Bayesian Optimization Algorithm [4]

II. LEMPEL-ZIV-WELCH (LZW) ALGORITHM [2]

The LZW is a Dictionary-based compression method in lossless data compression algorithm. This method uses a table to record fractional parts of the data (the words) as dictionary. During compression method, the dictionary size and the word length gradually increase. Size of dictionary depends on two factors, the length of data and the regularity in data. Normally, the size of compressed data is shorter.

A main benefit of LZW is that the dictionary can keep the patterns that ever appeared which are referred in the compressed data. So we know the distribution of the words in compressed data that can refer to the positions in uncompressed data. The pseudo code for LZW Compression is shown in Fig. 1.

1-4244-0977-2/07/$25.00 © 2007 IEEE

A sample demonstration of LZW algorithm for encoding the word "BIT_KIT_BIN" which based on ASCII code was shown in Fig. 2. The algorithm creates dictionary table for represent the input data (word) and starts reading the input data one by one. If the word does not exist in the dictionary table, it will be added. If the word is already existed, it will return the entry reference (index in dictionary table). The word "IT" and "BI" were referred twice in the sample then they can reduce into ones index. This shows the compression mechanism. The compressed data were represented into a form of an integer array which referred to reference indexes of the dictionary, e.g. {66,73,84,32,75,130,32,129,78} (See Fig. 2-a)

```
1  Algorithm LZW Compression
2  begin
3       w = NIL ;
4
5       while ( read a character k ) do
6       begin
7               if wk exists in the dictionary
8                       w = wk ;
9               else
10                      add wk to the dictionary ;
11                      output the entry index for w ;
12                      w = k ;
13      end;
14 end.
```

Fig. 1. LZW Compression pseudo code

The variable "w" is a word.
The variable "k" is a character in uncompressed data.
The "wk" means variable w concatenates with variable k

Input Data					
Input	**Code**	**Input**	**Code**	**Input**	**Code**
B	66	K	75	B	
I	73	I		I	129
T	84	T	130	N	78
_	32	_	32	End.	

(a)

Dictionary							
Index	**Word**	**Index**	**Word**	**Index**	**Word**	**Index**	**Word**
0		…		129	BI	134	IT_
…		97	a	130	IT	135	_B
32	_	…		131	T_	136	BIN
…		122	z	132	_K		
65	A	128	End.	133	KI		

(b)

Fig. 2. LZW encoding for word "BIT_KIT_BIN". (a) input data stream and code for each symbol from left to right and top to bottom. (b) ASCII code table was used as reference to create the dictionary table.

III. DICTIONARY BASED ESTIMATION OF DISTRIBUTION ALGORITHM

The process of Dictionary Based Estimation of Distribution Algorithms (DBEDA) begins by random creating binary chromosomes in the first population as Simple GA method [4], and evaluates the fitness score of each chromosome, and selects a number of the good fitness chromosomes by tournament selection method, and uses selected chromosomes to create dictionary by using LZW compression algorithm (this operation like compression a text file, each chromosome is a sentence and compresses each sentence in the same dictionary). We use a special table (we call generative table) to record only the words that were referred in compressed data, and sorts them by position in uncompressed chromosome that they ever appeared. For reproduction new offspring, will begin at first position in chromosome by random selecting one word in the generative table which start at position one, and counts the length of that word to find the next position for concatenating next word which random selecting from the generative table and repeats until complete chromosome length. The algorithm will terminate when a solution is found or limit of evaluation is reached. The pseudocode for DBEDA is shown in Fig. 3.

```
1  Algorithm DBEDA
2  begin
3       Z ← create_first_generation() ;
4       repet
5               evaluate(Z) ;
6               P ← select_good_fitness_chromosomes() ;
7               create_dictionary(P) ;
8               create_generative_table() ;
9               Z ← create_next_generation() ;
10
11      until( termination_condition = true )
12 end.
```

Fig. 3. DBEDA pseudo code

The variable "Z" is the population.
The variable "P" is the selected chromosomes by tournament selection method.

A. Creating Dictionary

Dictionary was created with algorithm that shown in Fig. 1. A number of high fitness chromosome will be selected by tournament selection to create dictionary. Each chromosome will be fed to LZW algorithm to update dictionary. During compression, if the word is already existed in the dictionary, the algorithm will skip it and reads the next word. We want to keep the length of words in dictionary so we cannot concatenate all chromosomes into the big one before compression. If we use the big one chromosome, the length of some words in dictionary may be longer than the chromosome length. In Fig. 4. shows a sample encoding for four of 10-bits chromosomes (1010110011, 0110010100, 1000111011, 0001001100).

Input Data							
Input	Code	Input	Code	Input	Code	Input	Code
1	1	0		1		0	
0	0	1		0		0	10
1		1	7	0	6	0	
0	2	0	0	0		1	
1	1	0		1		0	
1		1	3	1	7	0	12
0	2	0		1		1	
0		1		0		1	5
1	3	0	11	1	4	0	
1	1	0	0	1	1	0	10

(a)

Dictionary							
Index	Word	Index	Word	Index	Word	Index	Word
0	0	6	100	12	0100	18	000
1	1	7	011	13	0	19	01001
2	10	8	1	14	1000	20	110
3	01	9	0110	15	0111	21	00
4	101	10	00	16	1011		
5	11	11	010	17	1		

(b)

Fig. 4. Sample of encoding four chromosomes; (a) input data, each chromosome was represent in each grey column from left to right and top to bottom. (b) dictionary of binary data which has been shared encoding among four chromosomes (the grey block in column "Word" means that it is the last bit of each encoding chromosome).

Generative Table				
Position	Code 1	Code 2	Code 3	Code 4
1	1	7	6	10
2	0			
3	2			12
4		0	7	
5	1	3		
6	2			
7		11	4	5
8	3			
9				10
10	1	0		

Fig. 5. The Generative table which stored only unique compressed code (entry in the dictionary) in each position and sorted by position.

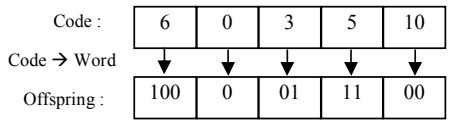

Fig. 6. New Offspring was reproduced by random selecting code from Generative table in order by position then transformed them back to word.

B. *Creating Generative Table*

In a compressed chromosome, each code (index value) referred to the position of the word (fraction part of chromosome before compression) in dictionary. We will record this word into generative table and sort it by position reference to uncompressed chromosome. So there are many words starting in the same position. If the same word already existed in the same position, it will be skipped recording. (See Fig 5.)

The generative table consists of compression code (refers to word) and position reference to uncompressed chromosome.

C. *Creating the Next Generation*

The offspring was created one by one. The process will randomly select words to create offspring by concatenating them together in ordering. For example, if length of the first word is ten, the next word will be selected randomly from generative table at position eleventh. If there is no word in that position, the algorithm will select a word in the prior nearest position and then cut up only desired position. The offspring has a chance to be oversize because the last word may make the offspring loner than the chromosome length due to there are many sizes of words at the same position in generative table. If the length is oversize, it will be cut out to satisfy the limited length of chromosome. In Fig. 6. shows a sample of reproduction method.

D. *Mutating Chromosome*

For the mutation method, we use basic genetic algorithm mutation by change the mutated bit from one to zero or zero to one (each bit has the same mutation probability).

IV. Experiment and Results

We are interested in BBs construction in hard problems with a large number of local optima by using three widely-used deceptive GA test problems to evaluate performance of DBEDA and comparing to the QHFC, which performances have been deemed excellent in these problems.

The three benchmark problems used include:

1) f3-deceptive: order-3 deceptive problem [4], with problem sizes n=60, 120, 180, 240, 300
2) f6-bipolar: order-6 bipolar deceptive problem [4], with problem size n= 60, 120, 180, 240, 300
3) trap-5: order-5 trap problem [4], with problem size n= 60, 120, 180, 240, 300

The deceptive problems represent hard problems that they are difficult to find the optimal solutions because they are guided by fitness values that lead to the trap as shown in Fig. 7-9. These problems lead the fitness climb in the wrong way; however the evolution mechanism can help to survive from these traps.

The algorithm is implemented in Object Pascal language and run on Pentium 4HT 3GHz with 512 MBytes of RAM. The operating system is Windows XP Professional. We repeated the experiment for 30 times and reported the results.

The Parameters sets for each problem are shown in table 1. For terminated condition, an optimal solution must be found or number of evaluation reaches the limit (1,000,000).

A. *f3-deceptive*

The deceptive function of order 3 is defined as

$$f_{3\text{-deceptive}}(u) = \begin{cases} 0.9 & \text{if } u = 0 \\ 0.8 & \text{if } u = 1 \\ 0 & \text{if } u = 2 \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Where u is the number of bit '1' in an input string

This deceptive function is composed of separable building blocks of order 3 and has one global optimum for all ones and a deceptive attractive to all zeros.

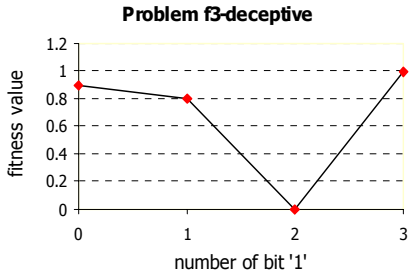

Fig. 7. fitness value of f3-deceptive problem.

B. *Trap*

The general k-bit trap functions are defined as

$$F_k(b_1 \ldots b_k) = \begin{cases} f_{high} & ; \text{if } u\text{=k} \\ f_{low} - (u \times f_{low})/(k-1) & ; \text{otherwise} \end{cases} \quad (2)$$

Where b_i is in $\{0,1\}$, $u = \sum_{i=1}^{k} b_i$ and $f_{high} > f_{low}$. Usually, f_{high} is set at k and f_{low} is set at k-1. The Trap functions denoted by $F_{m \times k}$ are defined as

$$F_{m \times k}(k_1 \ldots k_m) = \sum_{i=1}^{m} F_k(k_i), k_i \in \{0,1\}^k \quad (3)$$

The m and k are varied to produce a number of test functions. The Trap functions fool the gradient-based optimizers to favor zeros, but the optimal solution is composed of all ones. The Trap function is a fundamental unit for designing test functions that resist hill-climbing algorithms.

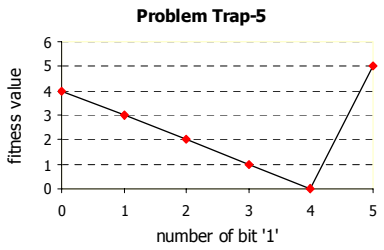

Fig. 8. fitness value of trap-5 problem.

C. *f6-bipolar*

The bipolar deceptive function of order 6 is defined with the use of the f3-deceptive function as follows

$$f_{6\text{-bipolar}}(u) = f_{3\text{-deceptive}}(|3 - u|) \quad (4)$$

The f6-bipolar function is highly multimodal. It has $\binom{6}{3}^{\frac{n}{6}}$ local optima besides $2^{(n/6)}$ global ones. The optimal solutions of this problem are all zero and all one.

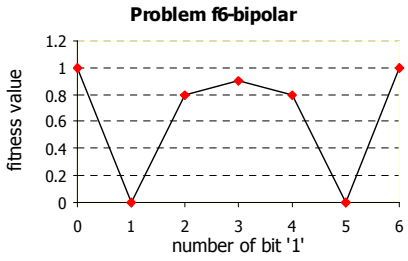

Fig. 9. fitness value of f6-bipolar problem.

The experimental results are shown in Fig. 10.

V. Discussion

The DBEDA bases on LZW algorithm to create dictionary so it needs a number of good fitness individual to extract building blocks. The population size for problem size larger than 50 should not less than 1,000 and the number of individual for generating dictionary should not less than 300, and mutation rate normally is not exceed 0.01 (these values observed from experiments). The mutation parameter is very sensitive in deceptive problems. Main operation for this algorithm depends on dictionary creation process therefore if uses the large number of individual to create dictionary the time will be taken.

VI. Conclusions and Future Work

We proposed the new method Dictionary Based Estimation of Distribution Algorithm (DBEDA) and tested with the

widely-used benchmark problems. The results show that DBEDA can solve the problem efficiently.

Next phase, we will improve the reproduction process because we do not know about the suitable length of each word that concatenating together. This will lead to proper Building Blocks and better results.

TABLE I
DICTIONARY BASED EDAS, PARAMETERS FOR EACH PROBLEM

Problem size = 60			
Parameter	f3-deceptive	trap-5	f6-bipolar
Population size	1,000	1,000	1,500
Pop. to create dict.	500	500	700
Tournament size	4	4	4
Mutation rate	0.005	0.005	0.01
Problem size = 120			
Parameter	f3-deceptive	trap-5	f6-bipolar
Population size	1,500	1,500	3,000
Pop. to create dict.	500	700	1,000
Tournament size	4	4	16
Mutation rate	0.005	0.005	0.005
Problem size = 180			
Parameter	f3-deceptive	trap-5	f6-bipolar
Population size	2,000	2,000	5,000
Pop. to create dict.	700	700	1,000
Tournament size	8	8	32
Mutation rate	0.005	0.005	0.002
Problem size = 240			
Parameter	f3-deceptive	trap-5	f6-bipolar
Population size	3,000	2,500	7,000
Pop. to create dict.	700	1,000	1,500
Tournament size	8	8	64
Mutation rate	0.005	0.005	0.002
Problem size = 300			
Parameter	f3-deceptive	trap-5	f6-bipolar
Population size	4,000	4,500	10,000
Pop. to create dict.	1,000	700	2,000
Tournament size	32	16	128
Mutation rate	0.005	0.01	0.002

"Pop. to create dict." means a number of chromosome that is used to create dictionary

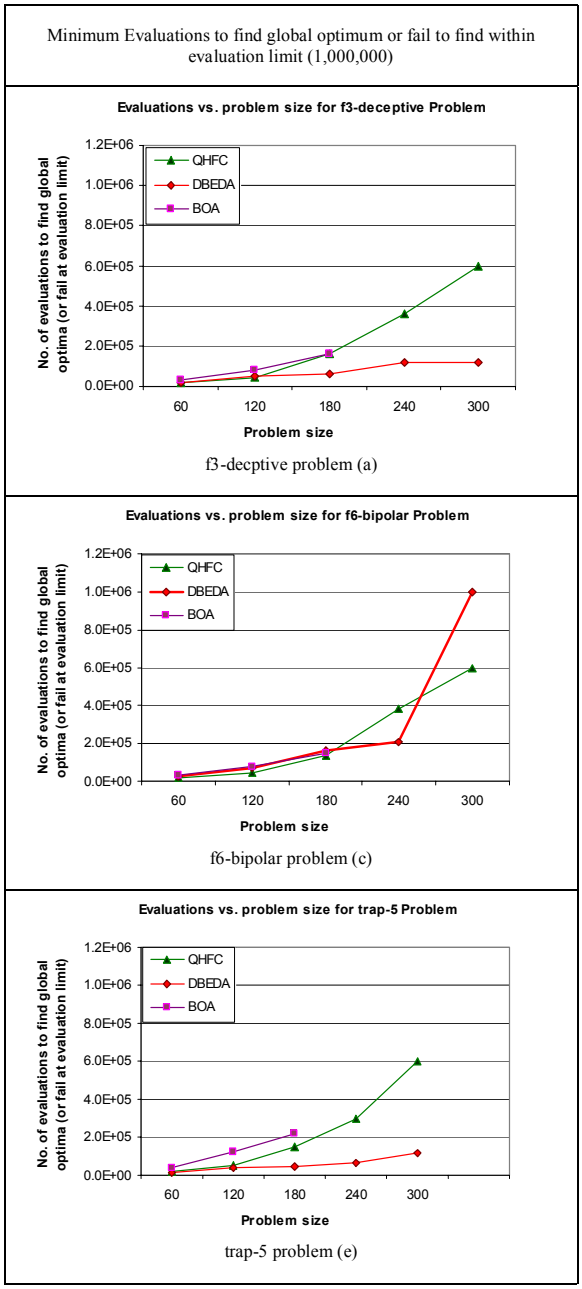

Fig. 10. Comparison of dictionary based estimation of distribution algorithm (DBEDA), hierarchical fair competition (QHFC), and bayesian optimization algorithm (BOA) in term of scalability and efficiency

REFERENCES

[1] P. Larranaga, J.A. Lozano, *Estimation of Distribution Algorithms*, Kluwer Academic Publisher, 2002

[2] David Solomon, *Data Compression* , Springer, 2004.

[3] J. Hu, E. Goodman, "Robust and Efficient Genetic Algorithms with Hierarchical Niching and Sustainable Evolutionary Computation Model," Proc. Genetic and Evolutionary Computation Conference, 2004.

[4] Pelikan, M, Goldberg, D.E. & Erick Cantu-Paz, E. BOA: The Bayesian optimization algorithm. Proc. of the Genetic and Evolutionary Computation Converence (GECCO),pages 525-532, 1999

[5] David E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison Wesley, 1989.

[6] W. Suwannik, N. Kunasol, P. Chongstitvatana, "Compressed Genetic Algorithm," Proceeding of Northeastern Computer Science and Engineering Conference, pages 203-211, March 31, 2005. (abstract in english)

[7] N. Kunasol, W. Suwannik, P. Chongstitvatana, "LZW Encoding in Genetic Algorithm," Proceedings of Electrical Engineering Conference (EECON-28), pages 861-864, October 20, 2005. (abstract in English)