Solving One-Billion-Bit Noisy OneMax Problem using Estimation Distribution Algorithm with Arithmetic Coding

Worasait Suwannik and Prabhas Chongstitvatana

***Abstract*— This paper presents an algorithm which combines Estimation Distribution Algorithm with a chromosome compression scheme to solve large scale Noisy OneMax problem. The search space reduction resulted from chromosome compression enables the algorithm to solve a one-billion-bit problem. Arithmetic Coding represents a compressed binary string with two real numbers. Using this representation, a model of highly fit individuals can be constructed. This model can be used to evolve the solution in the manner of Estimation Distribution Algorithm. The experimental result shows that the algorithm can solve billion-bit Noisy OneMax problem in about 34 hours using a normal PC-class computer.**

I. Introduction

RECENTLY in genetic algorithm research, there has been a growing interest in solving very large scale problems. Kunasol et. al. solved one-million-bit genetic algorithm benchmark problems using genetic algorithm with LZW compression algorithm encoding (LZWGA) [1]. Sastry et. al. presented a parallel compact genetic algorithm to solve billion-bit problems [2]. Both methods attack very large scale problems from different points of view. The first method solves the problems by search space reduction. The second method solves the problem by parallelization and population modeling.

Various search space reduction are presented in [3-6]. A review of population modeling is given in [7]. Estimation of Distribution Algorithm with Arithmetic Coding (EDAAC) combines search space reduction with population modeling. Search space reduction was accomplished by Arithmetic Coding. Arithmetic Coding is a lossless compression technique. It represents a binary string with two real numbers. Those two numbers form an EDAAC chromosome. The chromosomes aremodeled by assuming bivariate normal distribution. Suwannik and Chongstitvatana used EDAAC to solve one-billion-bit OneMax problem [8].

Worasait Suwannik is with the Department of Computer Science, Kasetsart University, Bangkok, Thailand. 10900 (e-mail: worasait.suwannik@gmail.com).

Prabhas Chongstitvatana is with the Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand. 10330 (e-mail: prabhas@chula.ac.th).

In this paper, we solved one-billion-bit Noisy OneMax problem. Noisy OneMax problem is more difficult than OneMax problem. An exogenous noise disrupts the selection process. We conducted an experiment to see the effect of noise on the population size.

The organization of this paper is as follows. Section 2 gives an overview of Arithmetic Coding and describes Arithmetic Coding Decompression algorithm. Section 3 explains EDAAC. Section 4 describes Noisy OneMax problems. Section 5 explains the experiment on solving one-billion-bit Noisy OneMax. Section 6 discusses about the effect of noise on the population size. Finally, Section 7 concludes the paper.

II. Arithmetic Coding

The compression algorithm represents a binary string by two real numbers ranged between [0, 1). The first number is the probability that zero will occurs in the binary string. The second number is the compressed message. The first number is denoted by p and the second number is denoted by c.

The coding is best explained by an illustration. The following example demonstrates a decompression of (p, c)=(0.4, 0.6) to a 4-bit binary string. As shown in Fig 1, p divides the interval [0,1) into 2 sub-intervals: [0, 0.4) and [0.4, 1). Since the compressed message c is in the second sub-interval, the algorithm outputs 1.

Next, the algorithm partitions the second interval [0.4, 1) into two sub-intervals proportional to p. The resulting sub-intervals are [0.4, 0.64) and [0.64, 1). Since the compressed message c is in the first sub-interval, the algorithm outputs 0.

Next, the algorithm partitions the first interval [0.4, 0.64) into sub-intervals proportional to p. The resulting sub-intervals are [0.4, 0.496) and [0.496, 0.64). Since the compressed message c is in the second sub-interval, the algorithm outputs 1.

Finally, the algorithm partitions the second interval [0.496, 0.64) into sub-intervals proportional to p. The resulting sub-intervals are [0.496, 0.5536) and [0.5536, 0.64). Since the message c is in the second sub-interval, the algorithm outputs 1. The process is summarized in Table I.

EDAAC required only a decompression algorithm. A pseudo code for Arithmetic Coding decompression used in EDAAC is shown in Fig 2. The algorithm runs in O(l) time, where l is the number of bits to be produced.

978-1-4244-1823-7/08/$25.00©2008 IEEE

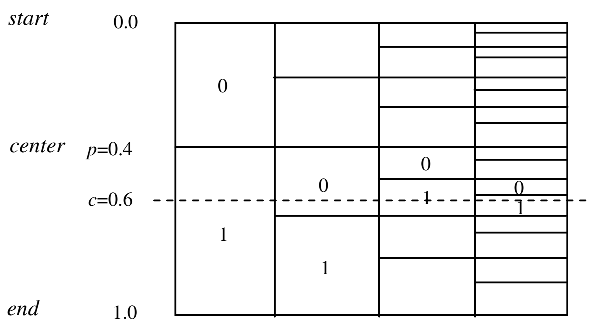

Fig 1. Decompressing 4 bits from (p, c)=(0.4,0.6). The output is 1011.

TABLE I
INTERVAL AND OUTPUT FOR EACH ITERATION

start	*center*	*end*	Output
0.000000	0.400000	1.000000	1
0.400000	0.640000	1.000000	0
0.400000	0.496000	0.640000	1
0.496000	0.553600	0.640000	1

III. Estimation Distribution Algorithm with Arithmetic Coding

The pseudo code of EDAAC is shown in Fig 3. The algorithm begins by creating the first generation of compressed chromosomes, P. The compressed chromosome is decompressed using Arithmetic Coding Decompression algorithm. Then, the fitness of an uncompressed chromosome is evaluated. After all chromosomes are evaluated, highly fit chromosome are selected (Q) and modeled (Π). The model Π is then used to generate offspring R, which will be decompressed and evaluated. The algorithm repeats the process of selecting and modeling highly fit individuals, and generating offspring until the termination criterion is met. The algorithm terminates when a solution is found or a maximum generation is reached.

A. *Creating the First Generation*

Unlike a conventional GA, a chromosome in EDAAC is encoded as two double precision variables (64 bits). The first variable is p and the second variable is c. The value of p and c of the first generation chromosome are uniformly random from the range [0, 1).

B. *Decompressing and Evaluating Individuals*

Because the chromosome in EDAAC is compressed, it has to be decompressed to a binary string before its fitness can be evaluated. A compressed chromosome is decompressed using Arithmetic Coding Decompression algorithm. The de-compression stops when it outputs a binary string with the length equals to that of the problem size. For example, for 100-bit OneMax problem, the decompression stops when the algorithm outputs a 100-bit binary string. However, some value of p and c cannot be decompressed to the intended number of bits. In that case, the remaining bits are filled with 0's.

```
Algorithm: Arithmetic Coding Decompress
  input:    p, c : double
  output:   data : bit array
1:    start ← 0
2:    center ← p
3:    end ← 1.0
4:    for (i ← 0; i < data.length; i++) {
5:      if (c < center) {
6:        data[i] ← 0
7:        end ← center
8:        center ← start + (center – start) × p
9:      } else {
10:       data[i] ← 1
11:       start ← center
12:       center ← start + (end – center) × p
13:     }
14:   }

start is the starting point of the first interval.
center is the starting point of the second interval.
end is the ending point of the second interval.
```

Fig 2. Arithmetic Coding Decompression pseudo code

```
Algorithm: EDAAC
Π is the model
input:    P, Q, R are population
output:   the best individual in P
1:    create n individuals as the first generation, P
2:    decompress and evaluate all individuals
3:    while not terminate
4:        select n/2 individuals, P → Q
5:        model selected individuals, Π
6:        generate n/2 offspring, Π → R
7:        decompress and evaluate all offspring
8:        integrate population, Q ∪ R → P
9:    end while
```

Fig 3. EDAAC pseudo code

C. *Modeling Highly Fit Individuals*

Highly fit chromosomes are selected using any traditional genetic algorithm selection method. The algorithm selects $n/2$ individuals and computes μ_p, μ_c, σ_p, σ_c, and ρ of those individuals, where

- μ_p, μ_c are the means of p's and c's of the selected individuals,
- σ_p, σ_c are the standard deviation of p's and c's of the selected individuals,
- ρ is the correlation coefficient of p's and c's of the selected individuals.

D. *Generating Offsprings*

An offspring (p_g, c_g) is generated by sampling from the model Π. p_g is a normally distributed random variable with the mean μ_p and the standard deviation σ_p. c_g is a normally distributed random variable with the mean μ_g and the standard deviation σ_c. μ_g is obtained from the following formula.

$$\mu_g = \mu_c + \rho \frac{\sigma_c}{\sigma_p}(p_g - \mu_p) \qquad (1)$$

$n/2$ offspring are generated, evaluated, and integrated with $n/2$ previously selected individuals. These n individuals will be the population of the next generation.

IV. Noisy OneMax

OneMax or a bit counting problem is a widely used problem for testing the performance of various genetic algorithms. The problem is defined as follows.

$$f(x) = \sum_{i=1}^{l} x_i \qquad (2)$$

where x is a chromosome, $x_i \in \{0,1\}$, and $l=|x|$

OneMax represents an optimization problem with independent variables. It can easily be solved by many genetic and hill climbing algorithms. For this problem, one type of EDA called cGA (compact genetic algorithm) scales as $\Theta(l \log l)$ while hill climbing algorithm scales as $\Theta(l)$ [2]. Another type of EDA called UMDA can solve the problem very fast [7].

Noisy OneMax is modified from OneMax. In order to make the problem harder than the original one, an exogenous noise was added after the chromosome is evaluated. The noise poses difficulty to genetic algorithms. Selection mechanism is likely be fooled by the noise. The problem is defined as follows.

$$f(x) = \sum_{i=1}^{l} x_i + G(0,\sigma^2_N) \qquad (3)$$

where G is a Gaussian noise with a mean 0 and a variance σ^2_N.

V. Solving One-Billion Bit Noisy OneMax

Sastry et. al. implemented parallel Compact Genetic Algorithm to solve one-billion-bit Noisy OneMax problem [2]. They conduct the experiment on 128- and 256- processor partitions of 1280-processor cluster.

We conduct the experiment on a one-billion-bit problem using a normal PC-class machine. The algorithm is implemented in Java language. It was compiled and run using JDK 1.6 on Pentium 4 HT 3GHz with 1 GByte of RAM. The program required a great deal of memory especially in solving the one-billion-bit problem. A one-billion-bit chromosome required 125,000,000 bytes. Actually, to solve OneMax problem using EDAAC, it is not necessary to allocate such large amount of memory. Arithmetic Coding Decompression can easily be modified to be a stream decompression. With a stream decompression (on-line one-bit-at-a-time), very small amount of memory can be used to evaluate a one-billion-bit OneMax chromosome. However, in this paper, we did not use the stream decompression. We allocate the amount of data that can store a one-billion-bit chromosome to get a better understanding of the execution time of large scale problems. In order to allocate such amount of memory, we run the Java program with the option –Xmx512m to set the maximum heap size to 512 MB.

We solved one-billion-bit Noisy OneMax problem using EDAAC. The noise was equals to the fitness variance. The fitness variance was set the maximum value ($0.25\times l$), which is the fitness variance of the population at the first generation.

EDAAC runs with the parameters shown in Table II. The population size is 3200 individuals. Please notice that this size is very small considering the size of the problems. The experiment is repeated for 10 times and the average figures of successful runs are reported.

TABLE II
Parameters of EDAAC

Parameter	Value
Population size	3200
Selection method	Tournament (size=4)
Number of best individual to keep	1
Maximum generations	100

Fig 4 shows the fitness curve of EDAAC in solving one-billion-bit Noisy OneMax and OneMax problem. The line with diamonds is the fitness of the best chromosome in solving OneMax problem. The line with squares is for Noisy OneMax problem. It shows the fitness of the best chromosome before the noise was added. For OneMax problem, all run were successful. For Noisy OneMax problems, 9 out of 10 runs were successful. The presence of noise equals to the fitness variance can make the problem more difficult. EDAAC can solve the problem with noise in about 33.64 hours. It can solve the OneMax with the equal size in 16.02 hours.

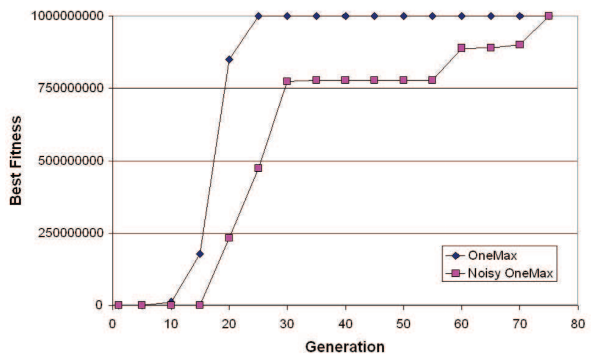

Fig 4. Fitness curve of EDAAC in solving one-billion-bit OneMax and Noisy OneMax problem.

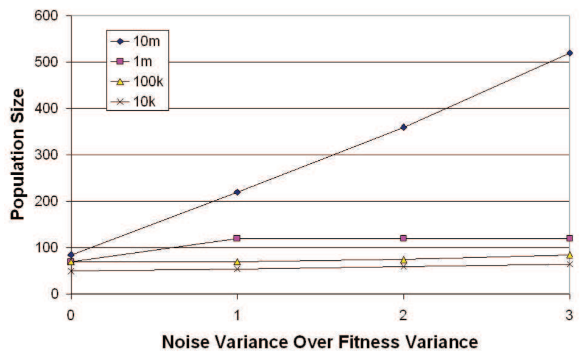

Fig 5. Effect of noise on population sizing.

VI. Effect of Noise on Population-Sizing

We conducted another experiment to see the effect of noise on population sizing. The noise was varied from 0 to 3. We used the bisection method to find a population size that can solve the problem within 100 generations with at least 90% success rate. We varied the size of the problem from 10k-bit to 10m-bit. Fig 5 shows that a larger population is required to solve the problem with higher level of noise.

VII. Conclusion and Future Work

EDAAC can solve one-billion-bit Noisy OneMax in reasonable amount of time using a normal PC-class computer. We believe that our result on computation time is a significant improvement over other methods to solve large scale problems. However, one might argue that such compressed encoding GA performed well on those problems because they have a high regularity solution. It would be interesting to investigate how EDAAC solves other large scale problems that the solution is not all 1's or all 0's. Moreover, the correlation measures the strength of the linear relationship between *p* and *c*. It would be interesting to invent an algorithm that allows non-linear relationship between those two variables. In addition, a new set of benchmark for large scale problems should be proposed.

References

[1] N. Kunasol, W. Suwannik, P. Chongstitvatana, "Solving One-Million-Bit Problems Using LZWGA," Proceedings of International Symposium on Communications and Information Technologies (ISCIT), October 18-20, 2006.

[2] K. Sastry, D. E. Goldberg, X. Llorà, "Towards billion bit optimization via efficient genetic algorithms," IlliGAL Report No. 2007007. University of Illinois at Urbana-Champaign, Urbana IL, 2007.

[3] S. Chen and S. Smith, "Improving Genetic Algorithms by Search Space Reduction (with Applications to Flow Shop Scheduling)," GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, 1999.

[4] F.G. Lobo, K. Deb, D.E. Goldberg, G. Harik, L. Wang, "Compressed Introns in a Linkage Learning Genetic Algorithm," Genetic Programming : Proceedings of the Third Annual Conference, Madison, Wisconsin, pages 551-558, 1998.

[5] W. Suwannik, N. Kunasol, P. Chongstitvatana, "Compressed Genetic Algorithm," Proceedings of Northeastern Computer Science and Engineering Conference, pages 203-211, March 31-April 1, 2005. (abstract in English)

[6] N. Kunasol, W. Suwannik, P. Chongstitvatana, "LZW-Encoding in Genetic Algorithm," Proceedings of Electrical Engineering Conference (EECON-28), pages 861-864, October 20-21, 2005. (abstract in English)

[7] T.K. Paul and H. Iba, "Linear and Combinatorial Optimizations by Estimation of Distribution Algorithms," 9th MPS Symposium on Evolutionary Computation, IPSJ, Japan, 2002.

[8] W. Suwannik, and P. Chongstitvatana, "Solving Large Scale Problems using Estimation Distribution Algorithm with Arithmetic Coding," Proceedings of International Symposium on Communications and Information Technologies (ISCIT), pages 358-363, October 16-19, 2007,

[9] J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," IBM J. Res. Develop., vol. 20, pp. 198-203, 1976.