2009 Ninth International Conference on Intelligent Systems Design and Applications

Speeding up the Genetic Algorithm Convergence Using Sequential Mutation and Circular Gene Methods

Mehdi Baradaran Nia
Electrical and Computer Engineering Department
University of Tabriz
Tabriz, Iran
e-mail: mbaradaran@tabrizu.ac.ir

Yousef Alipouri
Electrical and Computer Engineering Department
University of Tabriz
Tabriz, Iran
e-mail: alipouri.yousef@gmail.com

***Abstract*—Genetic Algorithms (GAs) are intelligent computational tools which their simplicity, accuracy and adaptable topology cause them to be used in globally minimum or maximum finding problems. Developing the GAs to increase their speed in finding the global minimum or maximum of a cost function has been a big challenge until now and many variants of GA has been evolved to accomplish this goal. This paper presents two new Sequential Mutation Method and Circular Gene Method to increase the speed of the GA. These methods attain a better final answer accompanied by lesser use of cost function evaluations in comparison with the original GA and some other known complementary methods. In addition, it speeds up reaching the minimum or maximum point regarding the number of generations. A number of common test functions with known minimum values and points are tested and the results are compared with some other algorithms such as original GA, Bacterial Evolutionary Algorithm, Jumping Gene and PSO. Simulation results show that the presented methods in this paper can reach the global minimum point through lesser generations and evaluations of the cost function in comparison with the traditional methods.**

Keywords: Genetic algorithm; speeding up the convergence; sequential mutation method; circular gene method

I. INTRODUCTION

There are many methods to locate minimum or maximum of a function. The classical way to find the extrema of a function is to take the gradient of the function and set it equal to zero. But this method is usually complicated especially for the multivariable functions and it is difficult to deal with and is not usable when the function is discrete or no derivation can be taken out of it. Although it works well when the minimum is nearby, it cannot deal well with cliffs or boundaries, where the gradient cannot be calculated. These difficulties necessitate searching for other methods to find the minimum and maximum of the functions. Genetic algorithm is a useful method in this aspect. GA is an optimization and searching technique based on the biological genetic development principles which was firstly developed by John Holland [1,2].

The GA begins, like any other numerical optimization algorithm, by setting the parameters, defining the optimization variables and the cost function. It ends like other optimization algorithms too, by testing the convergence. Some variants of GA have been introduced to speed up its convergence.

Some of these variants are Bacterial Evolutionary Algorithm (BEA) and Jumping Gene (JG) methods. Also some other minimum finding algorithm such as Particle Swarm Optimization (PSO) can be considered in this regard.

The BEA incorporates an operation analog to the direct transfer of strands of genes from host cells to other cells, which is called gene transfer operation. The gene transfer operation substitutes the crossover operation, allowing the recombination of information between different individuals, in the expectation that this will lead to the creation of better individuals. In BEA algorithm, first chromosome in the current population is copied m times and $m-1$ clones (genes) is mutated. The mutation operation is applied to the randomly chosen (i)th part. The m clones are then evaluated using the cost function and the fittest clone survives. This procedure is repeated for all parts of the chromosome and for all chromosomes [3,4,5]. This method needs to produce less generations but its drawback is to recall the cost function much more times during the algorithm running in comparison with the other methods.

The phenomenon of jumping genes, also known as transposons, was first discovered by McClintock from her work on corn plants [6,7]. Further experimental observation also indicated that there were two ways in which the JGs could move around the genome. The first one was called cut-and-paste, which means a piece of DNA is cut and pasted somewhere else. The second one was known as copy-and-paste. This means that the genes remain at the same location while the message in the DNA is copied into RNA and then copied back into DNA at another place in the genome [8]. JG method is actually a mutation operator which was developed by considering these biological explanations to be used as an evolutionary algorithm in minimum or maximum finding problems.

The PSO operates on "particles", driving them toward promising regions discovered adaptively by the whole swarm. The idea behind the algorithm was inspired by social behavior of animals, such as bird flocking or fish schooling and PSO was formulated by Edward and Kennedy

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.140

in 1995 [1,9]. PSO is similar to GA respecting that both of them begin with a random population but the original PSO does not have evolutionary operators such as crossover and mutation and is not binary encoded. In PSO, each particle moves about the cost function surface with a velocity. Velocities and positions get updated with regard to the position of the best particle [1,10].

Although GA's variants have been able to speed up the original algorithm in finding the extremum point in comparison with the original algorithm, but they still suffer from being slow especially when the number of the variables of the cost function is high. The basic idea of the method proposed in this paper is to change the mutation operator in order to speed up the GA. In this paper we introduce two new operators and compare their performance with the methods introduced previously. Then, these two operators are applied to the GA simultaneously to increase the efficiency of its performance. Simulation results show that when these two operators are applied to the GA, its performance gets enhanced with respect to the speed of convergence and the number of cost function evaluations. Also its efficiency is revealed when the number of the variables of the cost functions is high.

The organization of this paper is as follow: Section 2 gives a brief explanation to genetic algorithms. Section 3 and 4 introduce the SMM, CGM and the basic ideas beyond them, respectively. Section 5 explains the reason of merging two proposed methods. In section 6 simulation results for proposed methods are presented and the new operators are compared with some previously known algorithms and finally section 7 presents the conclusion and summarizes the simulation results.

II. GENETIC ALGORITHM

The original GA begins by defining a chromosome or an array of variable values. Then each chromosome gets a cost found by evaluating the cost function and the costs and associated chromosomes are ranked from the lowest cost to the highest. Only the best ones are selected to continue, while the rest is deleted. Two chromosomes are selected from the pool of chromosomes to produce two new offspring. Pairing takes place in the mating population until offspring are born to replace the discarded chromosomes. Mating is the creation of one or more offspring from the parents selected in the pairing process. The genetic makeup of the population is limited by the current members of the population. The most common form of mating involves two parents that produce two offspring. A crossover point is randomly selected between the first and last bits of the parents' chromosomes. First parent passes its binary codes to the left of the crossover point of the first offspring. Similarly, second parent passes its binary codes to the left of the same crossover point of the second offspring. Then, the binary codes to the right of the crossover point are changed in similar way. So each offspring contains portions of the binary codes of both parents. Some functions have many local minima. If we do nothing to solve the tendency to converge quickly, we could end up in a local rather than a global minimum. To avoid this problem of overly fast convergence, we force the routine to explore other areas of the cost surface by randomly introducing changes, called mutation, in some of the variables. For the binary GA, random numbers are chosen to select the row and columns of the variables to be mutated [1] and mutation is done by simply changing a bit from 0 to 1 and vice versa.

III. SEQUENTIAL MUTATION METHOD

One of the most important operators in the GA is the mutation operator. Mutation makes new attributes in genes and makes it possible to release the GA from the local minimums and go toward the global minimum. Also, it makes available to seek all parts of the searching space to find the global minimum. In this method the place of applying mutation is changed.

In the original GA, the mutation operates on some randomly chosen bit(s) of chromosomes. But in sequential mutation method (SMM), the mutation is applied only on one gene in each chromosome. In other words, only one gene of a chromosome is changed during the mutation and the other genes survive. The SMM has a very important property and it is speeding up finding the global minimum with less recalling the cost function which will be discussed widely in the results section.

In SMM, in each iteration, only one gene is selected from each chromosome and the mutation is applied on only the bits of this gene. These bits are selected randomly and this procedure is done for all the chromosomes in the current generation. For example, if the mutation rate of the algorithm is μ, the number of genes of each chromosome is n, the number of chromosomes in each population is N_{pop}, and the number of bits of each gene is N_{gene}, so the total number of bits in each population is $N_{bits} = n \times N_{pop} \times N_{gene}$ and the number mutations is $N_{mute} = round(\mu \times N_{bits})$. In this case, regarding the SMM, $round(N_{mute}/N_{pop})$ bits of the selected gene in each chromosome are mutated. The mutation places are chosen randomly. The selection of the gene is in a sequential order and the place of the selected gene goes one step further by increasing the number of generations. Figure 1 illustrates the place of applying the sequential mutation operator on one chromosome. In this figure, *n* is the number of genes in each chromosome or the number of variables in the cost function. After running the algorithm for *n* iterations, in (*n+1*)th iteration, the SMM is applied on the first genes of all chromosomes and this procedure is continued.

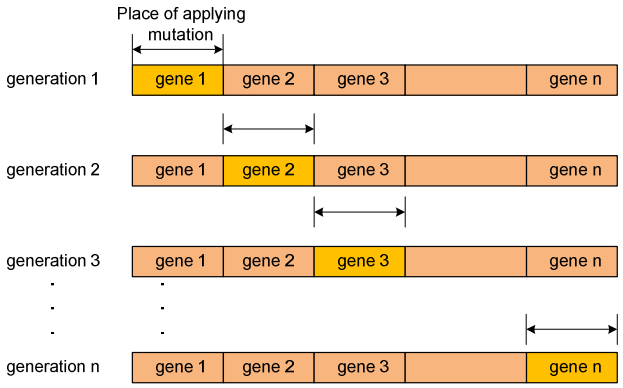

Figure 1. Place of applying the mutation on the chromosome in SMM.

IV. CIRCULAR GENE METHOD

The basic idea behind the CGM is derived from the biological behavior of the bacteria and some other organisms. Dulbecco and Vogt, and Weil and Vinograd in 1963 discovered that double stranded DNA of the polyoma virus (minute infectious agent which is normally present in extremely small amounts in the wild mice) exists in a closed circular form. At present it is generally acknowledged that this form, called circular DNA, is found in bacteria, cytoplasm (gelatinous or liquid material within a cell) in animals and archaea (microorganisms which are similar to bacteria in size and simplicity of structure) as well as in eukaryotic cells (organisms whose cells are organized into complex). Furthermore, giant DNA molecules in higher organisms form loop structures held together by protein fasteners in which each loop is largely analogous to the closed circular DNA. In plants and animals DNA is organized into linear chromosomes and is found in the nucleus of cells. DNA in bacteria and some other mentioned organisms is organized circularly. Bacteria have one large circular DNA molecule carrying most of their inherited information and mutation on them is done by changing the sequence of amino acids in a protein of DNAs [11,12,13].

By considering these biological explanations we can see the chromosomes in circular form due to the fact that the chromosomes of the multivariable cost functions have many bits and by deeming a large-bit-chromosome as a bacterium chromosome, we can apply the mutation on it. As considered above, the mutation is done on some specific biological chromosomes by changing the place of the amino acids in DNAs. So we can implement this idea with rotating some bits (or genes) of the chromosomes. We call this technique as circular gene method (CGM). The CGM is applied after the crossover and before the original mutation.

By applying the CGM, the 1st gene gives its value to the 2nd gene and takes its value from the *n*th gene; the 2nd gene gives its value to the 3rd gene and takes its value from the 1st one, and the (*k*)th gene gives its value to the (*k+1*)th gene and takes its value from the (*k-1*)th gene. In fact, in CGM, the places of the genes in each iteration are shifted one step to the left by considering the chromosome to be circular. Like SMM, CGM is applied on all chromosomes in each iteration. Figure 2 illustrates the procedure of applying the CGM on one chromosome in n iterations.

V. SMM AND CGM

Simulation results show that applying SMM can speed up finding the global minimum of the cost functions but individual using of CGM does not have expectable results in comparison with other methods. But merging these two methods (apply SMM and CGM together) has the advantages of both SMM and CGM and has the best result among methods introduced in this paper. Its effects get clear when these methods are used for the cost functions which have many variables. The overall flowchart of the algorithm is shown in figure 3.

VI. SIMULATION RESULTS

The cost function that is going to be minimized or maximized may have many variables or the number of arithmetical operations used in the cost function may be high. Also it is possible that for evaluation of a cost function using given independent variables, some experimental tasks by either simulation or real running of the case, which may take a long time to be done, is needed. So the number of times which a cost function is recalled to be calculated can be considered as a criterion to measure the speed of the algorithms.

In evolutionary algorithms, the main operators (such as selection, cross over and mutation for GA and position and velocity updating for PSO) are applied in a repeating manner to produce new generations with the aim of approaching the global extremum of the cost function. It is clear that the number of production of the new generations, which is also considered as the number of iterations, can be used as another criterion to measure the speed of the algorithms.

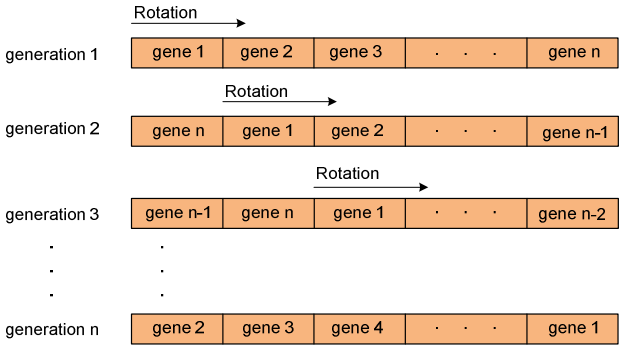

Figure 2. Circular gene method; the chromosome rotates and the place of the genes is shifted to the left.

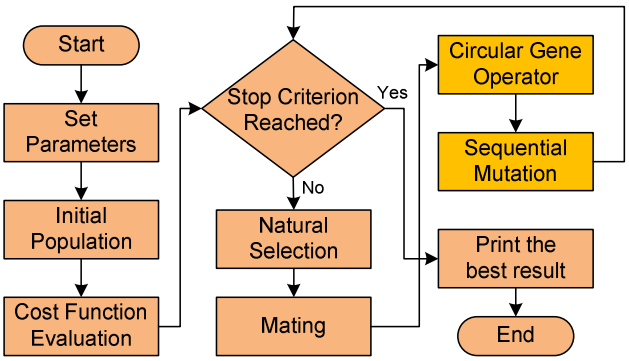

Figure 3. Flowchart of the algorithm and the place of applying SMM and CGM.

In this paper, two mentioned criteria, the number of cost function evaluations and the number of iterations are used to compare the performance of different heuristic algorithms.

Introduced methods are tested on 12 standard cost functions (which are available in [1]) but due to the limitation of presenting all the results in this paper, the results of only two of them are considered here. These functions have some local and one global minimums. Results of the proposed methods are compared with the original GA, Jumping Gene, BEA and PSO Algorithms.

For each cost function, drawn figures show the mean results of 50-time-running the algorithms. Because the random development of the algorithms causes them to have different answers in different runnings, this is done for reducing the effect of the chance and increasing the reliability in the results.

To show the capability of the proposed methods, two figures are drawn for each cost function. First figure shows the cost of the elite gene (or best particle) versus the number of iterations. This curve gives us a measurement of how fast the corresponding algorithm converges and reaches the global minimum. Also, it is a suitable way to compare different algorithms in terms of the final value they reach for a specified number of iterations. Second figure is a bar chart. This figure has 6 corresponding bars for each method. Each bar shows how many times the cost function is recalled until the algorithm reaches y percent of the global minimum. Relevant percents of y for each bar are noted on the top side of the figures. In bar charts, horizontal axis shows that which method belongs to each bar and vertical axis represents the number of the recalled cost functions.

In this paper the number of chromosomes (or particles) in each population is equaled to 8, the mutation factor (the fraction of the bits which gets mutated) is 0.1, each gene is coded with 16 bits, inertia factor and acceleration coefficients for PSO is 1 and m for BEA is equal with the number of genes.

It can be seen from convergence figures 4, 6 and 8 that using SMM and CGM simultaneously has the best result among other algorithms.

BEA shows a very fast fall in convergence figures in the first generations. This phenomenon is due to the fact that BEA uses lots of mutations in each generation. Indeed, it reaches its cost function evaluation limit in the first generations. The figures 5, 7 and 9 confirm this explanation. The simulation codes are written in a way that when the number of recalled cost function reaches its maximum number, the program stops the algorithm. This is why the curve convergence of BEA in figures 4, 6 and 8 are incomplete.

In bar charts, each method has a maximum limitation in the number of recalled cost functions. When one method reaches to this limitation, the program stops. The maximum limitation is determined with respect to the maximum number of permitted cost function evaluation by corresponding method. In some figures, BEA violates this limitation due to the fact that BEA can recall the cost function more times in comparison with other methods in each iteration. Because the limitation is checked in the program at the beginning of each iteration, so BEA has the opportunity to exceed this limitation.

PSO have better result than BEA in recalling the cost function. Also PSO has an acceptable convergence plot when the number of variables in cost function increases but its performance is still worse than SMM and CGM.

JG-copy and JG-cut are two methods that have better result than previous ones when the number of variables of the cost function is low. Although these two methods show a proper behavior in convergence figures, their drawback is the high number of recalled cost functions.

The results of the all tested cost functions show that using SMM and CGM simultaneously has the best result in both convergence curves and the number of recalled cost function charts. These advantages get highlighted when the number of variables in the cost function increases.

SMM and CGM are capable to deal with the multivariable cost functions and the results get better by increasing the number of the variables in the cost function. To show this phenomenon, table 1 is mentioned. This table shows the values for the average of five-time-running of the algorithms for cost function $1+\sum_1^N \frac{x(n)}{4000}-\prod_1^N Cos\big(x(n)\big)$ for different *N*s. The value of global minimum of this cost function is 0 which occurs at $\underline{X}=0$. All of the algorithms are run until a pre-specified number of recalled cost functions are reached. This number is shown in the last row of the table. The components of table 1 show the final values of each algorithm. This table reveals the capability of SMM and CGM when the number of variables in the cost function increases.

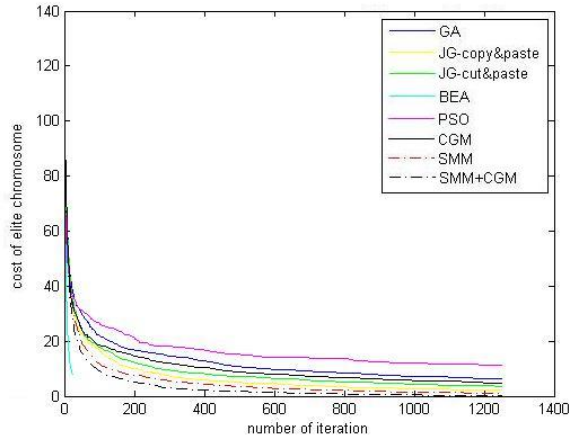

Figure 4. Plot of the minimum cost as a function of the generation for the cost function $10N + \sum_1^N [x(n)^2 - 10Cos(2\pi x(n))]$ and N=5; Global minimum at $\underline{X} = 0$ with value 0.

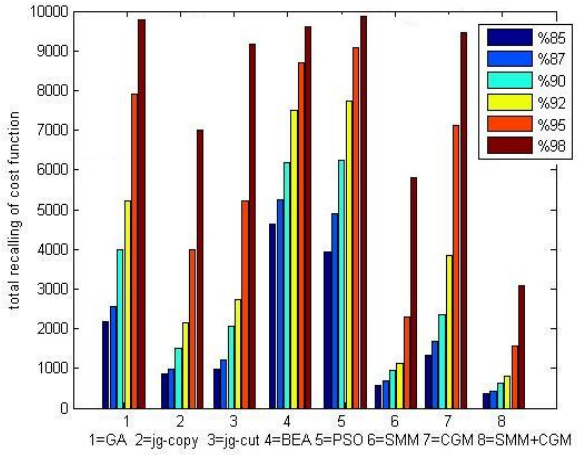

Figure 5. Plot of the number of recalled cost functions to reach y percent of the global minimum for the cost function $10N + \sum_1^N [x(n)^2 - 10Cos(2\pi x(n))]$ and N=5.

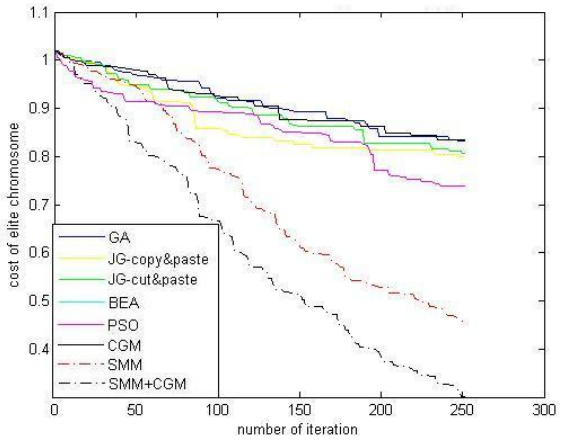

Figure 6. Plot of the minimum cost as a function of the generation for the cost function $1 + \sum_1^N \frac{x(n)}{4000} - \prod_1^N Cos(x(n))$ and N=15; Global minimum at $\underline{X} = 0$ with value 0.

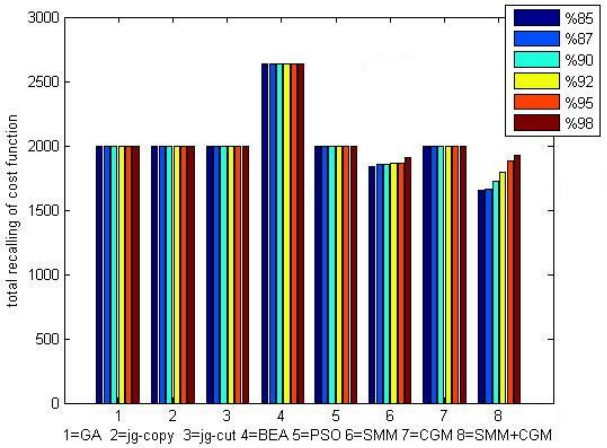

Figure 7. Plot of the number of recalled cost functions to reach y percent of the global minimum for the cost function $\mathbf{1} + \sum_1^N \frac{x(n)}{4000} - \prod_1^N Cos(x(n))$ and N=15.

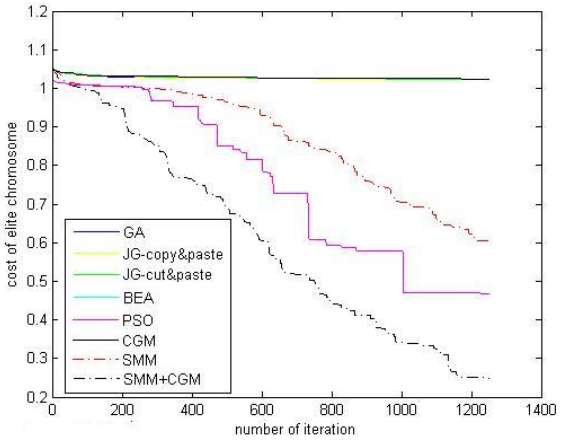

Figure 8. Plot of the minimum cost as a function of the generation for the cost function $1 + \sum_1^N \frac{x(n)}{4000} - \prod_1^N \mathrm{Cos}(x(n))$ and N=50; Global minimum at $\underline{X} = 0$ with value 0.

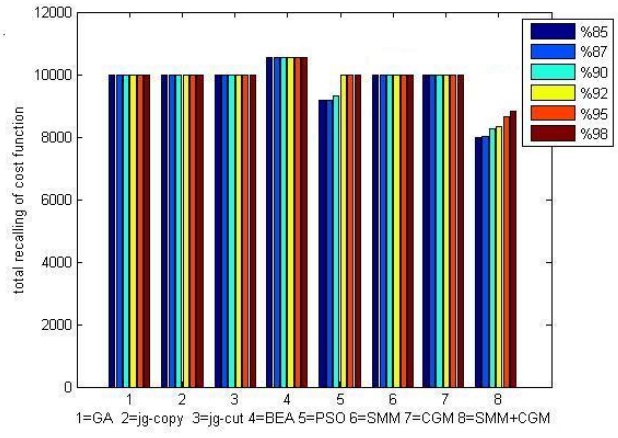

Figure 9. Plot of the number of recalled cost functions to reach y percent of the global minimum for the cost function $1 + \sum_1^N \frac{x(n)}{4000} - \prod_1^N Cos(x(n))$ and N=50.

TABLE I. FINAL VALUE OF ALGORITHMS AFTER A SPECIFIED NUMBER OF COST FUNCTION $1 + \sum_1^N \frac{x(n)}{4000} - \prod_1^N Cos(x(n))$ IS RECALLED FOR DIFFERENT *N*S

Number of variables (N)	3	5	10	15	20	40	50	100
GA	0.0045	0.0163	0.3100	0.5855	0.8261	1.0222	1.0336	1.0926
JG-copy	0.0005	0.0152	0.3573	0.5501	0.8356	1.0224	1.0316	1.0943
JG-paste	0.0051	0.0206	0.3266	0.5710	0.8640	1.0225	1.0328	1.1028
BEA	0.0036	0.0632	0.6590	0.8733	0.9917	1.0424	1.0543	1.1246
PSO	0.0087	0.0374	0.3607	0.3640	0.4208	0.4661	0.4573	0.6473
SMM	0.0039	0.0106	0.0570	0.0705	0.0682	0.6039	0.6113	0.7893
CGM	0.0051	0.0200	0.2312	0.6286	0.8730	1.0241	1.0349	1.0942
SMM+CGM	0.0014	0.0070	0.0435	0.0520	0.0423	0.2496	0.3995	0.3054
Number of recalled cost functions	3000	4000	4000	10000	15000	10000	15000	30000

VII. CONCLUTION

Two new methods, SMM and CGM, have been presented. These methods are applied to the original GA instead of regular mutation. Simulation results show that using SMM and CGM simultaneously can speed up the convergence of the GA accompanied by less recalling the cost function in comparison with other algorithms. The future prospect of the approaches proposed here is to theoretical analysis of the new methods and utilize them in continuous minimum finding algorithms.

REFERENCES

[1] A. Randy, L. Haupt, and B. S. E. Haupt, Practical Algorithm Genetics, 2nd ed., John Wiley & Sons, 2004.

[2] J. H. Holland, Adaptation in Natural and Artificial System, University of Michigan Press, 1975.

[3] A. H. S. Kim, and B. P.N. Roschke, "Design of fuzzy logic controller for smart base isolation system using genetic algorithm," Engineering Structures, Elsevier, Vol. 28, pp. 84-96, 2006.

[4] N. E. Nawa and T. Furuhashi, "Fuzzy system parameters discovery by bacterial evolutionary algorithm," IEEE Trans. on Fuzzy Systems, Vol. 7, No. 5, pp. 608 – 616, October 1999.

[5] N. Kubota, T. Fukuda, T. Arakawa, and K. Shimojima, "Evolutionary transition on virus-evolutionary genetic algorithm," IEEE Int. Conf. on Evolutionary Computation, Indianapolis, pp. 291–296, April 1997.

[6] W. K. S. Tang, S. T. W. Kwong, and K. F. Man, "A jumping genes paradigm: theory, verification, and applications," IEEE Circuits and Systems mag., Vol. 8, Issue. 4, pp. 18-38, 2008.

[7] K. S. N. Ripon, S. Kwong, and K. F. Man, "A real-coding jumping gene genetic algorithm for multiobjective optimization," Journal of Inf. Sciences, Vol. 177, No. 2, pp. 632–654, 2007.

[8] T. M. Chan, K. F. Man, S. Kwong, and K. S. Tang, "A Jumping Gene Paradigm for Evolutionary Multiobjective Optimization," IEEE Trans. on Evolutionary Computation, Vol. 12, No. 2, pp. 143-159, April 2008.

[9] J. Kennedy and R. C. Eberhart. "Particle swarm optimization," IEEE Int. Conf. on Neural Networks. Piscataway, NJ, pp. 1942–1948, 1995.

[10] S. N. Sivanandam and S. N. Deepa, Introduction to Genetic Algorithms, Springer, 2008.

[11] A. V. Vologodskii, Biophysical Chemistry Textbook, 5th ed., online book, Bloomfield, 1999.

[12] http://en.wikipedia.org/wiki/Circular_DNA

[13] http://encyclopedia.farlex.com/Deoxyribose+nucleic+ acid