DOES SIMPLE BINARY CROSSOVER HASTEN RCGA CONVERGENCE?

C. Lucas, Ph.D.
Center of Excellence on Control and Intelligent Processing, ECE Dept.,
University of Tehran, Iran; and SIS, IPM
e-mail: lucas@ipm.ir

A. Nayyeri, M.S.
Faculty of Electrical and Computer Engineering,
University of Tehran, I. R. of Iran
e-mail: a.nayyeri@ece.ut.ac.ir

M. Yabandeh, M.S.
Faculty of Electrical and Computer Engineering,
University of Tehran, I. R. of Iran
Corresponding Author
e-mail: m.yabandeh@ece.ut.ac.ir

Abstract - Real Coded Genetic Algorithm, RCGA, is the type of GA which operates on chromosomes with real valued parameters. Different mutation and crossover operations are defined for RCGA. One usable crossover for this kind of GA is to consider its chromosomes simply as bit strings and utilize the same operations as Binary Coded GA. In this paper, we attempt to show that this kind of crossover can not hasten the convergence process unless the break points fall at the boundaries of parameters in the chromosome.

Keywords: BCGA, Convergence Time, Crossover, Genetic Algorithm, RCGA, Real Numbers.

INTRODUCTION

Genetic Algorithm is a search methodology invented by Holland], which is inspired by the natural genetic theory. Today, GA is a very popular and effective method used to solve different problems. The most important reasons of this success are rooted in its simplicity and performance.

The main idea of this technique is generating diverse chromosomes and selecting the most appropriates to continue. In fact, in this method, we have an initial population of chromosomes, which are produced randomly or by a particular tactic. Then, iteratively, we try to generate new generations of population out of the previous ones using mutation, crossover and selection. Mutation is designed to generate a new chromosome out of an existing one by randomly changing it. In the crossover, two existing chromosomes are combined to generate new chromosomes. Finally, selection will help us to select the new population from the previous population and the new chromosomes which are generated with the help of mutation and crossover so that average fitness will tend to increase. In this paper, we wish to point out a very common [3,4,10] misuse of the algorithm when decision variables are continuous, but are represented via corresponding binary numbers.

In particular, we want to discuss the usual crossover operator.

Each chromosome in the population usually consists of different parameters concatenated to each other. In many theoretical and practical problems these parameters are real numbers. We call the GA working with real parameters in its chromosome, RCGA (Real Coded Genetic Algorithm). RCGAs have been used widely for optimization purposes by different people [1,2,4,6].

Crossover is a powerful technique to improve the speed of convergence of the GA. There are different techniques for RCGA crossover]. In the most popular one, two chromosomes are broken down into pieces and new chromosomes are configured by random replacement of these pieces. The place of breaking chromosomes plays an important role in the GA performance. As a matter of fact, random selection of this point can decrease the speed of convergence.

In this paper, we try to show that breaking the RCGA's chromosomes from any point other than the boundary of their chromosomes can ruin the performance of the GA. In other words, breaking a real parameter from its middle not only does not improve the convergence time of the algorithm, but also can result in an algorithm slower than the simple GA without crossover.

The theoretical foundations for the crossover operator in genetic algorithm will be discussed under the title, "Theory of Crossover." We point out why the theoretical justification for using the crossover operator does not necessarily apply to the RCGA with simple crossover in the binary representation. The section, "Practical Test," provides an empirical study where the inefficiency of the crossover operator, used with no regard to the significance of the binary digits and the border between the actual real variables, is shown via two benchmark examples. The last section is devoted to conclusions of the paper.

THEORY OF CROSSOVER

Crossover is the process of combination of two chromosomes with the hope of generating fitter offsprings. Crossover is not a necessary part of an evolutionary strategy; it just hastens the process. An intuition on this matter can be seen in the schema theorem [1]. When two chromosomes are mixed up, new genes with combined attributes are created. If the good part of each chromosome participates in this combination, then the new chromosome can be fitter than its parents. So, combining different attributes of the genes can cause the emergence of fitter individuals in a population.

There is no reason to believe that by breaking the concatenated strings of genes from an arbitrary point in the middle of the binary representation of a real number, useful combination of previously found building blocks could be obtained. Unless the crossover point happens to be in the boundary of the real numbers, it is more plausible to assume

that crossing over the binary digits will ruin what has already been found during the succession of previous generations. Factually, the second kind of breaking can cause a flaw in the process of searching, because it ruins one of the parameters which probably is already near its optimal value.

Holland [8] represents a theory for the GA stating that the number of fitter schemas is increasing during the process of the Genetic Algorithm. His lower bound for the expected value of the schema H in the generation t is:

$$E[m(\mathbf{H},t+1)] \geq \frac{m(\mathbf{H},t)\,f(\mathbf{H},t)}{f(t)} \left\{ 1 - p_c \frac{\delta(\mathbf{H})}{1-l}\, p_{diff}\,(\mathbf{H},t) - o\,(\mathbf{H}) p_m \right\}, \quad (1)$$

where $m(H,t)$ is the number of instances of schema H in generation t. The average fitness of these instances is given by f. Similarly, p_c and p_m are respectively the probabilities of crossover and mutation. $O(H)$ is the number of defined positions in H, and $\delta(H)$ is defined as the distance between the first and the last defined positions in it. Finally, p_{diff} represents the probability that the second parent does not match schema H.

It can clearly be seen that the values of δ, schema defining length, and o, schema order, play important roles in the formula. In fact, the roles of these attributes can be summarized to the building block hypothesis, which states that the GA works well when short, low-order, highly-fit schemas ("building blocks") recombine to form even more highly fit higher-order schemas [5,11]. In Goldberg's words, ". . . we construct better and better strings from the best partial solutions of past samplings"]. However, it should be considered that in an RCGA, the importance of these parameters is different due to their place in the sequence. For example, the order and length of the schemas 101*** and ***101 should be interpreted differently; although their length and order are equal, their values in the process of the solution are extremely different. To be clearer, imagine the value for which we are searching is 101101. Therefore, the worst value in the first schema is 101010, which is a much better estimation for the answer than 010101, the worst bit string in the second schema. It is also worth noting that, because GA usually finds a close estimation of the answer rather than the exact value of it, the arithmetic distance from the exact solution is more important than the number of bits in which our estimation is different from. This problem can be seen more clearly in some formats of storing real numbers, like floating point, where the bit strings of two arithmetically close values may be absolutely different. In fact, the concept of schema cannot be defined similarly in BCGA and RCGA.

In the next session, we show an experience that demonstrates this matter.

PRACTICAL TEST

The first example of the Goldberg's book] is finding the maximum of a function. Goldberg, in his book, used one parameter chromosome and the fitness value was generated by the

output of the function. He breaks the only gene in the middle simulating the behavior of the crossover operation. We have used similar examples to demonstrate the inefficiency of this still common method. It is worth noting that it is possible to apply some sophisticated and useful techniques to the experiment like breaking in more than one points and participating the best members of previous generation in the selection phase. But, we intentionally withdraw using those techniques to keep the genetic algorithm as simple as possible. This yields the differences in outcomes to depend only on the configurable parameter of the experiment, i.e. with or without the crossover.

For the first test, we chose the tangent function. Because, it has a deceptive nature, its maximum and minimum are near each other making the search process difficult. The function e^x is considered for another test in the (0,20) interval; it is a simple growing function and the optimum is obviously at the end of the interval. Both of the tests are done twice, with and without the crossover operator. The termination condition reaches the maximum point with a predefined error. The selection methodology for the algorithm is the proportionate soft selection. As parameters for comparison, we choose the number of generations and the required time to reach the answer. The time parameter indicates that with the use of the crossover, not only more generations are needed to reach the optimum point, but also the consumed time for crossover operation can even worsen the situation. Tests were done on a Pentium IV machine and, under the same condition, one thousand times each. The average results are shown in Tables 1 and 2.

Table 1: Results of the GA on the tan (x) in the (0.2π) interval.

	Average time (ms)	Average number of generation
With crossover	83.705	318.79
Without crossover	68.285	297.366

Table 2: Results of the GA on the e^x in the (0, 20) interval.

	Average time (ms)	Average number of generation
With crossover	4.343	21.614
Without crossover	3.102	20.491

CONCLUSIONS

The paper has criticized the binary coding of real numbers in GA, when the usual genetic

operators are used. Although mutations can also be argued to be wasteful, because right hand side binary digits are less significant than the left hand side ones, the paper aimed to show the wasteful nature of the crossover operator in particular. Our conjecture that crossover operations, which break down a gene in the middle, not only fail to improve the speed of the search process, but also slow it down, has been tested via two benchmark examples. This decrease can be the consequence of two notable matters. First, this kind of crossover has a destructive nature, which is explained in the previous session. Second, the consumed time by the crossover operation can be significant. The empirical result of this paper showed that both of these are important enough to be considered in running a GA.

Of course, two counter examples do not prove, nor could anyone ever expect to prove categorically. No matter how many counterexamples are provided, the deterioration of the algorithm's performance will prove to be the case in any other example as well. The use of crossover operator without paying any attention to the boundary between the real numbers will weaken the genetic motivation of the algorithm. Perhaps, one can always find pathological cases where genetic motivation does not yield efficiency. After all, the algorithm is stochastic and there may be cases where even random search yields better performance. However, it is believed that the arguments provided in this paper, which are further supported by two benchmark examples, will be enough to persuade users either to think of more proper use of crossover operators in RCGA, or use of mutation operator alone (in which case the algorithm, perhaps, should be assigned as real coded evolutionary algorithm).

REFERENCES

[1] Davis, L., *Handbook of Genetic Algorithms.* New York, Van Nostrand Reinhold, 1991.

[2] Eshelman, L. J. and Schaffer, J. D., "Real Coded Genetic Algorithms and Interval-Schemata." *Foundation of Genetic Algorithms 2,* Darrell Whitley, L. (Ed.), (Morgan Kaufmann Publishers, San Mateo), pp. 187-202, 1993.

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company Inc., 1989.

[4] Grewal, G., et al., "An EGA Approach to the Compile-Time Assignment of Data to Multiple Memories in Digital-Signal Processors." *ACM SIGARCH Computer Architecture News*, Vol. 31, No. 1, 2003.

[5] Forrest, S. and Mitchell, M., "Relative Building Block Fitness and the Building Block Hypothesis." *Foundations of Genetic Algorithms 2*, 1993.

[6] Herrera, F., et al., "Tuning Fuzzy Logic Controllers by Genetic Algorithms." *International Journal of Approximate Reasoning,* Vol. 12, pp. 299-315, 1995.

[7] Herrera, F., et al., "Tackling Real Coded Genetic Algorithm: Operations and Tools for Behavioral Analysis." *Artificial Intelligence Review,* Kluwer Academic Publisher, Netherlands. Vol. 12, pp. 265-319, 1998.

[8] Holland, J. H., *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[9] Michael, C., et al., "Generating Software Test Data by Evolution." *IEEE Trans. Software Testing*, Vol. 27, No. 12, 2001.

[10] Perez, C. A., et al., "Genetic Design of Biologically Inspired Receptive Fields for Neural Pattern Recognition." *IEEE Trans. Man and Cybernetics-Part B: Cybernetics,* Vol. 33, No. 2, 2003.

[11] Stephens, C. and WaelBroeck, H., "Schemata Evolution and Building Blocks." *Evolutionary Computation*, Vol. 7, No. 2, 1999.