GA Operators for Convex Continuous Parameter Spaces

Zbigniew Michalewicz* Thomas D. Logan†

Abstract

This paper discusses the application of genetic algorithms to constrained optimization problems for convex continuous parameter spaces. We present a set of six 'genetic' operators for GAs based on floating point representation. Various experiments indicate the importance of these operators at different stages of the evolution process.

*Department of Computer Science, University of North Carolina, Charlotte, NC, U.S.A.
†IBM, Charlotte, NC, U.S.A.

1 Introduction

The binary representation traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high-precision numerical problems. For example, for 100 variables with domains in the range $[-500, 500]$ where a precision of six digits after the decimal point is required, the length of the binary solution vector is 3000. This, in turn, generates a search space of about 10^{1000}. For such problems genetic algorithms based on binary string representation perform poorly. In this paper we describe the results of experiments with various genetic operators for floating point representation.

Many other researchers (e.g., [6], [23]) investigated GAs based on floating point representation. However, the optimization problems they considered were defined on a search space $\mathcal{D} \subseteq R^n$, where $\mathcal{D} = \prod_{k=1}^{n} \langle l_k, r_k \rangle$, i.e., each variable x_k was restricted to a given interval $\langle l_k, r_k \rangle$ $(1 \leq k \leq n)$. On the other hand it seems important to include other constraints into the considerations; as stated in [4]:

> "A little observation and reflection will reveal that all optimization problems of the real world are, in fact, constrained problems. Suppose one has an expression for the output of a chemical reaction vessel in which some set of chemical reactions are taking place and one wishes to maximize the output. It is also necessary to take into account material balance restrictions on the reactans and products, the laws governing flow of materials into and out of the reaction vessel, and other conditions. All of these are additional constraints on the variables of the function to be optimized."

In a constrained optimization problem, the geometric shape of the set of solutions in R^n is perhaps the most crucial characteristic of the problem, with respect to the degree of difficulty we are likely to encounter in attempting to solve the problem [4]. There is only one special type of set—a convex set—for which a significant amount of theory has been developed.

In this paper we are concerned with the following optimization problem:

optimize $f(x_1, \ldots, x_n) \in R$,

where $(x_1, \ldots, x_n) \in \mathcal{D} \subseteq R^n$ and $\mathcal{D}$ is a *convex* set.

The domain $\mathcal{D}$ is defined by ranges of variables ($l_k \leq x_k \leq r_k$ for $k = 1, \ldots, n$) and by a set of constraints $\mathcal{C}$. From the convexity of the set $\mathcal{D}$ it follows that for each point in the search space $(x_1, \ldots, x_n) \in \mathcal{D}$ there exists a feasible range $\langle left(k), right(k) \rangle$ of a variable x_k $(1 \leq k \leq n)$, where other variables x_i $(i = 1, \ldots, k-1, k+1, \ldots, n)$ remain fixed. In other words, for a given $(x_1, \ldots, x_k, \ldots, x_n) \in \mathcal{D}$:

$$y \in \langle left(k), right(k) \rangle \text{ iff } (x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \in \mathcal{D},$$

where all x_i's $(i = 1, \ldots, k-1, k+1, \ldots, n)$ remain constant. We assume also that the ranges $\langle left(k), right(k) \rangle$ can be efficiently computed.

For example, if $\mathcal{D} \subseteq R^2$ is defined as:

$-3 \leq x_1 \leq 3$,
$0 \leq x_2 \leq 8$,
and $x_1^2 \leq x_2 \leq x_1 + 4$,

then for a given point $(2, 5) \in \mathcal{D}$:

$left(1) = 1$, $right(1) = \sqrt{5}$,
$left(2) = 4$, $right(2) = 6$.

It means that the first component of the vector $(2, 5)$ can vary from 1 to $\sqrt{5}$ (while $x_2 = 5$ remains constant) and the second component of this vector can vary from 4 to 6 (while $x_1 = 2$ remains constant).

Of course, if the set of constraints $\mathcal{C}$ is empty, then the search space $\mathcal{D} = \prod_{k=1}^{n} \langle l_k, r_k \rangle$ is convex; additionally $left(k) = l_k$, $right(k) = r_k$ for $k = 1, \ldots, n$. It means that the proposed operators constitute a valid set regardless the presence of the constraints.

The paper is organized as follows. The next section discusses six operators used in a GA based on floating point representation. Section 3 presents a single test case and section 4 contains conclusions and some directions for future work.

2 Six operators

In this section we describe six 'genetic' operators for GAs based on floating point representation. The first three are unary operators (category of mutation), the other three are binary (various types of crossovers). We discuss them in turn.

2.1 Uniform mutation

This operator requires a single parent $\vec{x}$ and produces a single offspring $\vec{x}'$. The operator selects a random component $k \in (1, \ldots, n)$ of the vector $\vec{x} = (x_1, \ldots, x_k, \ldots, x_n)$ and produces $\vec{x}' = (x_1, \ldots, x_k', \ldots, x_n)$, where x_k' is a random value (uniform probability distribution) from the range $\langle left(k), right(k) \rangle$.

The operator plays an important role in the early phases of the evolution process as the solutions are allowed to move freely within the search space. In particular, the operator is essential in the case where the initial population consists of multiple copies of the same (feasible) point. Such situations may occur quite often in constrained optimization problems where users specify the starting point for the process. Moreover, such single starting point (apart from its drawbacks) has powerful advantage: it allows to develop an iterative process, where the next iteration starts from the best point of the previous iteration. This very technique were used in a development of a system to handle nonlinear constraints in not necessarily convex spaces [18].

Also, in the later phases of an evolution process the operator allows possible movement away from a local optimum in the search for a better point.

2.2 Boundary mutation

This operator requires also a single parent $\vec{x}$ and produces a single offspring $\vec{x}'$. The operator is a variation of the uniform mutation with x'_k being either $left(k)$ or $right(k)$, each with equal probability.

The operator aims at optimization problems where the optimal solution lies either on or near the boundary of the feasible search space. Consequently, if the set of constraints $\mathcal{C}$ is empty, and the bounds for variables are quite wide, the operator is a nuisance. However, in the presence of constraints it can prove extremely useful. A simple example demonstrates the usefulness of this operator. The example considers a linear programming problem; in such case we know that the global solution lies on the boundary of the search space.

Example 1.

The following test case was taken from [22]:

maximize $f(x_1, x_2) = 4x_1 + 3x_2$,

subject to the following constraints:

$$\begin{aligned}
&2x_1 + 3x_2 \leq 6,\\
&-3x_1 + 2x_2 \leq 3,\\
&2x_1 + x_2 \leq 4, \text{ and}\\
&0 \leq x_i \leq 2,\ i = 1, 2.
\end{aligned}$$

The known global optimum is $(x_1, x_2) = (1.5, 1.0)$, and $f(1.5, 1.0) = 9.0$.

To determine the usefulness of this operator in optimizing the above problem, ten experiments were run with all operators functioning and another ten experiments without boundary mutation. The system with boundary mutation found the global optimum easily in all runs, on average within 32 generation, whereas without the operator, even in 100 generations the *best* point found (in ten runs) was $\vec{x} = (1.501, 0.997)$ and $f(\vec{x}) = 8.996$ (the worst point was $\vec{x} = (1.576, 0.847)$ with $f(\vec{x}) = 8.803$).

2.3 Non-uniform mutation

This is the (unary) operator responsible for the fine tuning capabilities of the system. It is defined as follows. For a parent $\vec{x}$, if the element x_k was selected for this mutation, the result is $\vec{x}' = \langle x_1, \ldots, x'_k, \ldots, x_q \rangle$, where

$$x'_k = \begin{cases} x_k + \triangle(t, right(k) - x_k) & \text{if a random binary digit is 0} \\ x_k - \triangle(t, x_k - left(k)) & \text{if a random binary digit is 1} \end{cases}$$

The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as t increases (t is the generation number). This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. We have used the following function:

$$\triangle(t, y) = y \cdot r \cdot (1 - \frac{t}{T})^b,$$

where r is a random number from $[0..1]$, T is the maximal generation number, and b is a system parameter determining the degree of non–uniformity.

The operator has proven to be extremely useful in many test cases; one of them is given in the following example.

Example 2.

The following test case was taken from [21]:

$$\text{maximize } f(x_1, x_2, x_3, x_4, x_5) = (x_1-1.0)^2+(x_1-x_2^2)^2+(x_2-1.0)^2+(x_1-x_3^2)^2+ \\ +(x_3-1.0)^2+(x_1-x_4^2)^2+(x_4-1.0)^2+(x_1-x_5^2)^2+(x_5-1.0)^2,$$

subject to:

$$0 \le x_i \le 10, \; i = 1, \ldots, 5.$$

The known global optimum is $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 1, 1)$, and $f(1, 1, 1, 1, 1) = 0$.

Again, we executed several runs and compared the results. The progress of the both systems (with and without non-uniform mutation) is reported below:

Generation number	Best evaluation with the operator	Best evaluation without the operator
50	1.85312903	7.45429516
100	0.80920660	0.44312307
500	0.00698480	0.06966695
1000	0.00000000	0.01593169
5000	0.00000000	0.00155015

In all runs with non-uniform mutation the best point found by the system was

$$(1.00000000, 1.00000000, 1.00000000, 1.00000000, 1.00000000),$$

whereas without the operator the best point found was

$$(1.02938247, 1.01172984, 1.01243937, 1.01172805, 1.01172495).$$

2.4 Arithmetical crossover

This binary operator is defined as a linear combination of two vectors: if $\vec{x}_1$ and $\vec{x}_2$ are to be crossed, the resulting offspring are $\vec{x}_1' = a \cdot \vec{x}_1 + (1-a) \cdot \vec{x}_2$ and $\vec{x}_2' = a \cdot \vec{x}_2 + (1-a) \cdot \vec{x}_1$. This operator uses a random value $a \in [0..1]$, as it always guarantees closedness ($\vec{x}_1', \vec{x}_2' \in \mathcal{D}$). Such a crossover was called a *guaranteed average crossover* in [5] (when $a = 1/2$); *intermediate crossover* in [1]; *linear crossover* in [23]; and *arithmetical crossover* in [15], [16], and [17].

An importance of arithmetical crossover is presented by the following example.

Example 3.

Let us consider the following problem (taken from [9]):

$$\text{minimize } f(x_1, x_2, x_3, x_4, x_5) = -5sin(x_1)sin(x_2)sin(x_3)sin(x_4)sin(x_5) + \\ -sin(5x_1)sin(5x_2)sin(5x_3)sin(5x_4)sin(5x_5),$$

where

$$0 \leq x_i \leq \pi, \text{ for } 1 \leq i \leq 5.$$

The known global solution is $(x_1, x_2, x_3, x_4, x_5) = (\pi/2, \pi/2, \pi/2, \pi/2, \pi/2)$, and $f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$.

It seems that the system without arithmetical crossover has slower convergence. After 50 generations the average value of the best point (out of 10 runs) was -5.9814, and the average value of the best point after 100 generations was -5.9966. In the same time, these averages for the system with arithmetical crossover were -5.9930 and -5.9996, respectively.

Moreover, an interesting pattern that emerged showed that the system with arithmetical crossover was more stable, with much lower standard deviation of the best solutions (obtained in ten runs).

2.5 Simple crossover

This binary operator is defined as follows: if $\vec{x}_1 = (x_1, \ldots, x_n)$ and $\vec{x}_2 = (y_1, \ldots, y_n)$ are crossed after the k-th position, the resulting offspring are: $\vec{x}_1' = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_n)$ and $\vec{x}_2' = (y_1, \ldots, y_k, x_{k+1}, \ldots, x_n)$.

However, such operator may produce offspring outside the domain $\mathcal{D}$. To avoid this problem, we use the property of the convex spaces saying, that there exist $a \in [0, 1]$ such that

$$\vec{x}_1' = \langle x_1, \ldots, x_k, y_{k+1} \cdot a + x_{k+1} \cdot (1-a), \ldots, y_n \cdot a + x_n \cdot (1-a) \rangle$$

and

$$\vec{x}'_2 = \langle y_1, \ldots, y_k, x_{k+1} \cdot a + y_{k+1} \cdot (1-a), \ldots, x_n \cdot a + y_n \cdot (1-a) \rangle$$

are feasible.

The only question to be answered yet is how to find the largest a to obtain the greatest possible information exchange. The simplest method would start with $a = 1$ and, if at least one of the offspring does not belong to $\mathcal{D}$, decreases a by some constant $\frac{1}{q}$. After q attempts $a = 0$ and both offspring are in $\mathcal{D}$ since they are identical to their parents. The necessity for such maximal decrement is small in general and decreases rapidly over the life of the population.

It seems that the merits of simple crossover are the same as of arithmetical crossover (for experiments, we have used the problem from Example 3). The results showed that the system without simple crossover was even less stable than the system without arithmetical crossover; in this case the standard deviation of the best solutions obtained in ten runs was much higher. Also, the worst solution obtained in 100 generations had value of -5.9547—much worse then the worst solution obtained with all operators (-5.9982) or the worst solution obtained without arithmetical crossover (-5.9919).

2.6 Heuristic crossover

This operator was suggested in [23]. This is unique crossover for the following reasons: (1) it uses values of the objective function in determining the direction of the search, (2) it produces one offspring only, and (3) it may produce no offspring at all.

The operator generates a single offspring $\vec{x}_3$ from two parents, $\vec{x}_1$ and $\vec{x}_2$ according to the following rule:

$$\vec{x}_3 = r \cdot (\vec{x}_2 - \vec{x}_1) + \vec{x}_2,$$

where r is a random number between 0 and 1, and the parent $\vec{x}_2$ is not worse than $\vec{x}_1$, i.e., $f(\vec{x}_2) \geq f(\vec{x}_1)$ for maximization problems and $f(\vec{x}_2) \leq f(\vec{x}_1)$ for minimization problems.

It is possible for this operator to generate an offspring vector which is not feasible. In such a case another random value r is generated and another offspring created. If after w attempts no new solution meeting the constraints is found, the operator gives up and produces no offspring.

Example 4.

The following test case was taken from [3]:

$$\begin{aligned}\text{minimize } f(\overline{X}) = {} & 100(x_2 - x_1^2)^2 + (1-x_1)^2 + 90(x_4 - x_3^2)^2 + \\ & +(1-x_3)^2 + 10.1((x_2-1)^2 + (x_4-1)^2) + 19.8(x_2-1)(x_4-1),\end{aligned}$$

subject to:

$$-10.0 \leq x_i \leq 10.0, \; i = 1, 2, 3, 4.$$

The global solution is $\overline{X}^* = (1,1,1,1)$, and $f(\overline{X}^*) = 0$.

It seems that heuristic crossover contributes towards the precision of the solution found; it major responsibilitise are (1) fine local tuning, and (2) search in the promising direction.

The system without this operator, in 10,000 generations, produced solutions where each variable stayed within 10% from its optimal value; for example a typical output was

$$(x_1, x_2, x_3, x_4) = (0.95534354, 0.91261518, 1.04239142, 1.08687556),$$

with $f(x_1, x_2, x_3, x_4) = 0.00683880$. On the other hand, the system with heuristic crossover performed much better; its results are reported in the next section, where we compare the original system with its revised version.

3 Summary and a Test Case

All discussed operators were incorporated in the new version of the Genocop system and they differ from the ones present in the original version. The experiments and first results of the original Genocop [17] are fully discussed in [19] (the eight test cases include quadratic, nonlinear, and discontinuous functions with several linear constraints); here we concentrate on differences between these two versions and provide a single test case to demonstrate the superiority of the latter one.

The first change is connected with non-uniform mutation. The function $\triangle(t,y)$ was changed; instead of

$$\triangle(t,y) = y \cdot \left(1 - r^{(1-\frac{t}{T})^b}\right),$$

(used in the original Genocop), the following function was used:

$$\triangle(t,y) = y \cdot r \cdot (1 - \tfrac{t}{T})^b.$$

The second change was incorporated in the arithmetical crossover. The original Genocop uses a constant $a = 0.25$, whereas the new version generates a random number (between 0 and 1) every time the operator is called.

The third (most significant) change is introduction of the heuristic crossover, not present at all in the original system. The results of many test cases indicated the importance of this operator.

Both versions use an additional operator: an application of non-uniform mutation on all components of the selected parent. However, the original Genocop applied non-uniform mutation "from left to right", whereas the new version generates a random order of vector's components.[1]

[1] Additionally, the new version of the Genocop uses a different mechanism for parents' selection, and a new random number generator; however, we will not discuss these topics in this paper.

The significant difference in performance between the original and the new version of the Genocop system is demonstrated in the following test case (this is one of the test cases reported in [19]; the improvements were achieved in all cases; due to the limited space here we report on one case).

All runs of the system were performed on SUN SPARC station 2. We used the following parameters for all experiments:

> $pop\_size = 70$, $k = 28$ (number of parents in each generation), and $b = 6$ (coefficient for non-uniform mutation).

For each test case we run the system ten times. A single run of 10,000 generations on function of four variables (e.g., problem from Example 4) took 98 seconds to complete.

Example 5.

For the final example we have used the problem defined in the previous example (Example 4). The typical solution found by the original Genocop system in 1,000,000 generations (!) was

$$(x_1, x_2, x_3, x_4) = (0.983055, 0.966272, 1.016511, 1.033368),$$

with $f(x_1, x_2, x_3, x_4) = 0.001013$, whereas a typical solution returned by the revised system with six (revised) operators discussed in the paper in 10,000 generations only was

$$(x_1, x_2, x_3, x_4) = (1.000581, 1.001166, 0.999441, 0.998879),$$

with $f(x_1, x_2, x_3, x_4) = 0.0000012$.

This is an excellent improvement—the above (unconstrained) test function (a famous Colville function #7) proved difficult for many optimization methods. Even VFSR (Very Fast Simulated Reannealing, [14]) produced inferior result:

$$(x_1, x_2, x_3, x_4) = (1.1244, 1.2633, 0.8614, 0.7416),$$

with $f(x_1, x_2, x_3, x_4) = 0.062267$.

It seems that the proposed set of six operators performs its task in guiding the system towards the optimum.

4 Conclusions

We have presented six operators suitable for real coded genetic algorithms. The proposed operators (three mutations and three crossovers) emerged from many experiments with the Genocop system [19] and proved useful on many test cases. Since Genocop is in the public domain, it provides a convenient reference point for any further research on GAs with floating point representation.

Another aspect of our research provided us also some observations on the right choice of frequencies of the proposed operators. There is no doubt that the best arrangement would be a design of an adaptive mechanism for these frequencies (as it is done in evolution strategies [1]); this would be done in the next version of Genocop system together with the introduction of integer and Boolean variables.

The original Genocop system and its revised versions are available from anonymous ftp unccsun.uncc.edu, directory coe/evol, files genocop.tar.Z and genocop2.tar.Z, respectively.

References

[1] Bäck, T., Hoffmeister, F., and Schwefel, H.-P., *A Survey of Evolution Strategies*, in [2], pp.2–9.

[2] Belew, R. and Booker, L. (Editors), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1991.

[3] Colville, A.R., *A Comparative Study on Nonlinear Programming Codes*, IBM Scientific Center Report 320-2949, New York, 1968.

[4] Cooper, L., and Steinberg, D., *Introduction to Methods of Optimization*, W.B. Saunders, London, 1970.

[5] Davis, L., *Adapting Operator Probabilities in Genetic Algorithms*, in [20], pp.61–69.

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

[7] Eshelman, L.J., and Schaffer, J.D., *Real-Coded Genetic Algorithms and Interval-Schemata*, in Darrell Whitley (Ed.), Foundations of Genetic Algorithms II, Morgan Kaufmann, San Mateo, CA, 1992.

[8] Floudas, C.A. and Pardalos, P.M., *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, Lecture Notes in Computer Science, Vol.455, 1987.

[9] Floudas, C.A. and Pardalos, P.M., *Recent Advances in Global Optimization*, Princeton Series in Computer Science, Princeton University Press, Princeton, NJ, 1992.

[10] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.

[11] Goldberg, D.E., *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking*, University of Illinois at Urbana-Champaign, Technical Report No. 90001, September 1990.

[12] Hock, W. and Schittkowski K., *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, Vol.187, Springer-Verlag, 1981.

[13] Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[14] Ingberg, L., *Simulated Annealing: Practice and Theory*, to appear in Statistics and Computing, 1993.

[15] Michalewicz, Z. and Janikow, C., *Genetic Algorithms for Numerical Optimization*, Statistics and Computing, Vol.1, No.2, pp.75–91, 1991.

[16] Michalewicz, Z. and Janikow, C., *Handling Constraints in Genetic Algorithms*, in [2], pp.151–157.

[17] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, AI Series, New York, 1992.

[18] Michalewicz, Z., and Attia, N.F., *Genetic Algorithm + Simulated Annealing = Genocop II: A Tool for Nonlinear Programming*, submitted for publication, 1993.

[19] Michalewicz, Z. and Swaminathan, S., *GENOCOP: The First Experiments and Results*, submitted for publication, 1993.

[20] Schaffer, J., (Editor), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 1989.

[21] Schwefel, H.-P., *Numerical Optimization for Computer Models*, Wiley, Chichester, UK, 1981.

[22] Taha, H.A., *Operations Research: An Introduction*, 4th ed, Collier Macmillan, London, 1987.

[23] Wright, A.H., *Genetic Algorithms for Real Parameter Optimization*, in Rawlins, G. (Ed.), *Foundations of Genetic Algorithms*, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, Los Altos, CA, 1991, pp. 205–218.