EVOLUTIONARY OPERATORS FOR CONTINUOUS CONVEX PARAMETER SPACES

Zbigniew Michalewicz
Department of Computer Science, University of North Carolina
Charlotte, NC 28223, USA

and

Thomas D. Logan
IBM, Charlotte, NC 28257, USA

and

Swarnalatha Swaminathan
Department of Computer Science, University of North Carolina
Charlotte, NC 28223, USA

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.

1. Introduction

The binary representation traditionally used in genetic algorithms has some drawbacks when applied to multidimensional, high-precision numerical optimization 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. This paper describes the results of experiments with various genetic operators for floating point representation.

Many other researchers[5,20] investigated GAs based on floating point representation. But 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 $\langle l_k, r_k \rangle$ $(1 \leq k \leq n)$. Yet it seems important to include other constraints into the considerations; as stated in[3]:

> "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 that is likely to be encountered in attempting to solve the problem[3]. 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:

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

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:

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

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 recently developed Genocop system[16] provides a method of handling constraints that is both general and problem independent. Genocop does not use the concept of penalty functions nor does it use a technique of eliminating offspring

generated outside the feasible space (e.g., evolution strategies). The main idea lies in (1) an elimination of the equalities present in the set of constraints, and (2) careful design of special operators, which guarantee to keep all offspring within the constrained solution space. This can be done very efficiently for linear constraints.

The first version of this technique was implemented recently. The system was tested on several linearly constrained problems; the experience based on these results helped in the development of the improved version of the system.

The paper is organized as follows. The next section presents the experimental results of the original Genocop system on several test cases. Section 3 discusses six operators used in a GA based on floating point representation in the next version of the system. Section 4 presents a single test case and section 5 contains conclusions and some directions for future work.

2. Test Cases

In order to evaluate the method, a set of test problems has been carefully selected to illustrate the performance of the algorithm and to indicate its degree of success. The eight test cases include quadratic, nonlinear, and discontinuous functions with several linear constraints. 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), $b = 2$ (coefficient for non-uniform mutation), $a = 0.25$ (parameter of arithmetical crossover).

Genocop was executed ten times for each test case. For most problems, the maximum number of generations T was either 500 or 1000 (harder problems required larger number of iterations). All test cases and the results of the Genocop system are reported in the following subsections.

2.1. Test Case #1

The problem[8] is

> minimize $f(\overline{X}, y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$,

subject to:

$$6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leq 6.5, \quad 10x_1 + 10x_3 + y \leq 20,$$
$$0 \leq x_i \leq 1, \qquad\qquad 0 \leq y.$$

The global solution is $(\overline{X}^*, y^*) = (0, 1, 0, 1, 1, 20)$, and $f(\overline{X}^*, y^*) = -213$.

Genocop found solutions that were very close to the optimum in all ten runs; a typical discovered optimum point was:

$$(0.000055, 0.999998, 0.000041, 0.999989, 1.000000, 19.999033),$$

for which the value of the objective function is equal to -212.990997. A single run of 1000 iterations took 29 sec of CPU time.

2.2. Test Case #2

The problem[11] is

$$\text{minimize } f(\overline{X}) = \sum_{j=1}^{10} x_j(c_j + \ln \frac{x_j}{x_1+\ldots+x_{10}}),$$

subject to:

$$\begin{aligned} &x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2, \quad x_4 + 2x_5 + x_6 + x_7 = 1, \\ &x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1, \quad x_i \geq 0.000001, \ (i = 1, \ldots, 10), \end{aligned}$$

where

$$\begin{aligned} &c_1 = -6.089; \ c_2 = -17.164; \ c_3 = -34.054; \ c_4 = -5.914; \ c_5 = -24.721; \\ &c_6 = -14.986; \ c_7 = -24.100; \ c_8 = -10.708; \ c_9 = -26.662; \ c_{10} = -22.179; \end{aligned}$$

The previously best known solution[11] was

$$\begin{aligned} \overline{X}^* = (&.01773548, .08200180, .8825646, .0007233256, .4907851, \\ &.0004335469, .01727298, .007765639, .01984929, .05269826), \end{aligned}$$

and $f(\overline{X}^*) = -47.707579$.

Genocop found points with better value than the one above in all ten runs:

$$\begin{aligned} \overline{X}^* = (&.04034785, .15386976, .77497089, .00167479, .48468539, \\ &.00068965, .02826479, .01849179, .03849563, .10128126), \end{aligned}$$

for which the value of the objective function is equal to -47.760765. A single run of 500 iterations took 11 sec of CPU time.

2.3. Test Case #3

The problem[8] is

$$\text{minimize } f(\overline{X}, \overline{Y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i,$$

subject to:

$$\begin{aligned} &2x_1 + 2x_2 + y_6 + y_7 \leq 10, && 2x_1 + 2x_3 + y_6 + y_8 \leq 10, \\ &2x_2 + 2x_3 + y_7 + y_8 \leq 10, && -8x_1 + y_6 \leq 0, \\ &-8x_2 + y_7 \leq 0, && -8x_3 + y_8 \leq 0, \\ &-2x_4 - y_1 + y_6 \leq 0, && -2y_2 - y_3 + y_7 \leq 0, \\ &-2y_4 - y_5 + y_8 \leq 0, && 0 \leq x_i \leq 1, \ i = 1, 2, 3, 4, \\ &0 \leq y_i \leq 1, \ i = 1, 2, 3, 4, 5, 9, && 0 \leq y_i, \ i = 6, 7, 8. \end{aligned}$$

The global solution is $(\overline{X}^*, \overline{Y}^*) = (1,1,1,1,1,1,1,1,1,3,3,3,1)$, and $f(\overline{X}^*, \overline{Y}^*) = -15$.

Genocop found the optimum in all ten runs; a typical optimum point found was:

(1.000000, 1.000000, 1.000000, 1.000000, 0.999995, 1.000000, 0.999999,
1.000000, 1.000000, 2.999984, 2.999995, 2.999995, 0.999999),

for which the value of the objective function is equal to -14.999965. A single run of 1000 iterations took 55 sec of CPU time.

2.4. Test Case #4

The problem[7] is

$$\text{maximize } f(\overline{X}) = \frac{3x_1+x_2-2x_3+0.8}{2x_1-x_2+x_3} + \frac{4x_1-2x_2+x_3}{7x_1+3x_2-x_3},$$

subject to:

$$\begin{array}{ll} x_1 + x_2 - x_3 \le 1, & -x_1 + x_2 - x_3 \le -1, \\ 12x_1 + 5x_2 + 12x_3 \le 34.8, & 12x_1 + 12x_2 + 7x_3 \le 29.1, \\ -6x_1 + x_2 + x_3 \le -4.1, & 0 \le x_i,\ i = 1,2,3. \end{array}$$

The global solution is $\overline{X}^* = (1,0,0)$, and $f(\overline{X}^*) = 2.471428$.

Genocop found the optimum in all ten runs; a single run of 500 iterations took 9 sec of CPU time.

2.5. Test Case #5

The problem[8] is

$$\text{minimize } f(\overline{X}) = x_1^{0.6} + x_2^{0.6} - 6x_1 - 4x_3 + 3x_4,$$

subject to:

$$\begin{array}{ll} -3x_1 + x_2 - 3x_3 = 0, & x_1 + 2x_3 \le 4, \\ x_2 + 2x_4 \le 4, & x_1 \le 3, \\ x_4 \le 1, & 0 \le x_i,\ i = 1,2,3,4. \end{array}$$

The best known global solution is $\overline{X}^* = (\frac{4}{3}, 4, 0, 0)$, and $f(\overline{X}^*) = -4.5142$.

Genocop found this point in all ten runs; a single run of 500 iterations took 9 sec of CPU time.

2.6. Test Case #6

The problem[2] is

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

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 $f(\overline{X}^*) = 0$.

Genocop approached the optimum quite closely in all ten runs; a typical optimum point found was:

(0.953835, 0.909714, 1.043783, 1.089695),

for which the value of the objective function is equal to 0.007296, and

(1.018352, 1.037137, 0.980476, 0.961101),

for which the value of the objective function is equal to 0.001333. However, the number of runs was set at 500000, which took 103 min of CPU time.

2.7. Test Case #7

The problem[8] is

$$\text{minimize } f(x, \overline{Y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5,$$

subject to:

$$\begin{aligned}
&x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leq 16,\\
&-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leq -1,\\
&2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 \leq 24,\\
&0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 \leq 12,\\
&-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leq 3,\\
&y_3 \leq 1,\ y_4 \leq 1, \text{ and } y_5 \leq 2,\\
&x \geq 0,\ y_i \geq 0, \text{ for } 1 \leq i \leq 5.
\end{aligned}$$

The global solution is $(x, \overline{Y}^*) = (0, 6, 0, 1, 1, 0)$, and $f(x, \overline{Y}^*) = -11.005$.

Genocop approached the optimum quite closely in all ten runs; a typical optimum point found was:

(0.000000 5.976089, 0.005978, 0.999999, 1.000000, 0.000000),

for which the value of the objective function is equal to -10.988042. A single run of 1000 iterations took 29 sec of CPU time.

2.8. Test Case #8

The problem was constructed from three separate problems[11] in the following way:

$$\text{minimize } f(\overline{X}) = \begin{cases} f_1 = x_2 + 10^{-5}(x_2 - x_1)^2 - 1.0 & if \;\; 0 \le x_1 < 2 \\ f_2 = \frac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3 & if \;\; 2 \le x_1 < 4 \\ f_3 = \frac{1}{3}(x_1 - 2)^3 + x_2 - \frac{11}{3} & if \;\; 4 \le x_1 \le 6 \end{cases}$$

subject to:

$$\begin{array}{l} x_1/\sqrt{3} - x_2 \ge 0, \\ -x_1 - \sqrt{3}x_2 + 6 \ge 0, \\ 0 \le x_1 \le 6, \text{ and } x_2 \ge 0. \end{array}$$

The function f has three global solutions:

$$\overline{X}_1^* = (0,0),\ \overline{X}_2^* = (3,\sqrt{3}), \text{ and } \overline{X}_3^* = (4,0),$$

in all cases $f(\overline{X}_i^*) = -1$ $(i = 1,2,3)$.

We made three separate experiments. In experiment k $(k = 1,2,3)$ all functions f_i except f_k were increased by 0.5. As a result, the global solution for the first experiment was $\overline{X}_1^* = (0,0)$, the global solution for the second experiment was $\overline{X}_2^* = (3,\sqrt{3})$, and the global solution for the third experiment was $\overline{X}_3^* = (4,0)$.

Genocop found global optima in all runs in all three cases; a single run of 500 iterations took 9 sec of CPU time.

3. Six operators

In this section we describe six genetic operators based on floating point representation, which were used in modified version of the Genocop system. The first three are unary operators (category of mutation), the other three are binary (various types of crossovers). We discuss them in turn.

3.1. 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 a single starting point (apart from its drawbacks) has powerful advantage: it allows for developing an iterative process, where the next iteration starts from the best point of the previous iteration. This very technique was used in a development of a system to handle nonlinear constraints in spaces that were not necessarily convex[17].

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.

3.2. 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 is constructed for 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. But it can prove extremely useful in the presence of constraints. A simple example demonstrates the utility of this operator. The example is a linear programming problem; in such case we know that the global solution lies on the boundary of the search space.

Example 1.

Let us consider the following test case[19]:

$$\text{maximize } f(x_1, x_2) = 4x_1 + 3x_2,$$

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 utility 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 generations, 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 $f(\vec{x}) = 8.803$).

3.3. 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[16].

3.4. 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*[4] (when $a = 1/2$); *intermediate crossover*[1]; *linear crossover*[20]; and *arithmetical crossover*[14,15,16].

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

Example 2.

Let us consider the following problem[8]:

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

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 appears 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).

3.5. 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 $\vec{x}_2' = (y_1, \ldots, y_k, x_{k+1}, \ldots, x_n)$. Such operator may produce offspring outside the domain $\mathcal{D}$. To avoid this, we use the property of the convex spaces stating, that there exists $a \in [0, 1]$ 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 remaining question to be answered 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).

3.6. Heuristic crossover

This operator[20] is a unique crossover for the following reasons: (1) it uses values of the objective function in determining the direction of the search, (2) it produces only one offspring, 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.

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

Example 3. For a test case, we used the test case #6 from Section 2.

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.

4. Summary and a Test Case

All of the operators discussed were incorporated in the new version of the Genocop system and they differ from those of the original version.

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:

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

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.*

The significant difference in performance between the original and the new version of the Genocop system is demonstrated in the following test case.

All runs of the system were again 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).

*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.

For each test case we run the system ten times.

Example 4.

This example is case #6 (Section 2). 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 only 10,000 generations 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[13]) produced an 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.

5. 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 Genocop 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 some observations on the appropriate 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. Also, it seems worthwhile to investigate the interdependence of the proposed operators (i.e., the importance of one operator may depend on the existence of another).

All presented test-cases are relatively small (up to 13 variables). It is interesting to test the performance of Genocop on larger scale problems (more than 100 variables). A careful analysis should provide some hints on the behavior of the system as a function of variables; we plan to run all necessary tests as soon as the next version of the system is ready.

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.

Acknowledgements:

The author wishes to thank David Fogel for his constructive comments.

References:

1. Bäck, T., Hoffmeister, F., and Schwefel, H.-P., *A Survey of Evolution Strategies*, Proceedings of the Fourth International Conference on Genetic Algorithms, R.K. Belew and L. Booker (Eds.), Morgan Kaufmann Publishers, Los Altos, CA, 1991, pp.2–9.
2. Colville, A.R., *A Comparative Study on Nonlinear Programming Codes*, IBM Scientific Center Report 320-2949, New York, 1968.
3. Cooper, L., and Steinberg, D., *Introduction to Methods of Optimization*, W.B. Saunders, London, 1970.
4. Davis, L., *Adapting Operator Probabilities in Genetic Algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, J.D. Schaffer (Ed.), Morgan Kaufmann Publishers, Los Altos, CA, 1989, pp.61–69.
5. Davis, L., (Ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
6. Eshelman, L.J., and Schaffer, J.D., *Real-Coded Genetic Algorithms and Interval-Schemata*, in D. Whitley (Ed.), Foundations of Genetic Algorithms II, Morgan Kaufmann, San Mateo, CA, 1992.
7. 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.
8. Floudas, C.A. and Pardalos, P.M., *Recent Advances in Global Optimization*, Princeton Series in Computer Science, Princeton University Press, Princeton, NJ, 1992.
9. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.
10. Goldberg, D.E., *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking*, University of Illinois at Urbana-Champaign, Technical Report No. 90001, September 1990.
11. Hock, W. and Schittkowski K., *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, Vol.187, Springer-Verlag, 1981.
12. Holland, J., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
13. Ingber, L., *Simulated Annealing: Practice and Theory*, Technical Paper, 1993.
14. Michalewicz, Z. and Janikow, C., *Genetic Algorithms for Numerical Optimization*, Statistics and Computing, Vol.1, No.2, pp.75–91, 1991.
15. Michalewicz, Z. and Janikow, C., *Handling Constraints in Genetic Algorithms*, Proceedings of the Fourth International Conference on Genetic Algorithms,

R.K. Belew and L. Booker (Eds.), Morgan Kaufmann Publishers, Los Altos, CA, 1991, pp.151–157.
16. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, AI Series, New York, 1992.
17. Michalewicz, Z., and Attia, N.F., *Evolutionary Optimization of Constrained Problems*, in this volume.
18. Schwefel, H.-P., *Numerical Optimization for Computer Models*, Wiley, Chichester, UK, 1981.
19. Taha, H.A., *Operations Research: An Introduction*, 4th ed, Collier Macmillan, London, 1987.
20. Wright, A.H., *Genetic Algorithms for Real Parameter Optimization*, in G. Rawlins (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.