Evolution Programming

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

Abstract

The paper presents the concept of evolution programming and discusses its applicability to various optimization problems.

Keywords: evolutionary computation, genetic algorithm, random algorithm, optimization technique.

1 Introduction

In this paper we describe the concept of *evolution programming* (EP), a technique based on genetic algorithms (GAs) but extended by a richer set of data structures together with an expanded set of 'genetic' operators (Michalewicz, 1992). Additionally, evolution programs incorporate ideas present in evolution strategies, scatter search, and simulated annealing. We discuss briefly similarities and differences between GAs and evolution programs; further in the paper the discussion is supported by a few test problems for which evolution programs were developed. It is interesting to observe non-string data structures used for chromosomal representation for the transportation problem; it is also interesting to look at a mixture of the ideas taken from other randomized techniques in developing two Genocop systems for mathematical programming problems. Evolution programs do not follow any fixed pattern for representing an individual in the population, nor they are limited to a small set of operators which alter the composition of the individuals. In other words, evolution (nature) is not regarded as a dogma which we have to copy, but rather as a source of inspirations (Schwefel, 1992).

The paper is organized as follows. In the next section we explain what evolution programs are and how they differ from genetic algorithms. Section 3 presents a classical example of an evolution program for the nonlinear transportation problem; the system we describe (Genetic-2n) uses matrices as chromosomal representation of individuals and non-standard genetic operators. Sections 4 and 5 discuss two other evolution programs, Genocop and Genocop II, respectively; both these systems incorporate features of evolution strategies, scatter search, and simulated annealing. The last section contains some concluding remarks.

The paper describes results of several experiments reported recently in Michalewicz, & Janikow, (1991a); Michalewicz, Vignaux, & Hobbs, (1991); Michalewicz, (1992); Michalewicz, & Swaminathan, (1993); Michalewicz, (1993); Michalewicz, & Attia, (1993), Michalewicz & Logan (1993).

2 Evolution Programs

The evolution program (see Figure 1) is a probabilistic algorithm which maintains a population of individuals, $P(t) = \{x_1^t, \ldots, x_n^t\}$ for iteration t. The size of the population, n, remains constant during the evolution process. Each individual of the population represents a potential solution to the problem at hand, and is implemented as some (possibly complex) data structure S. Each solution x_i^t is evaluated to give some measure of its "fitness". Then, a new population (iteration $t+1$) is formed by selecting the more fit individuals (select step). Some members of the new population undergo transformations (alter step) by means of "genetic" operators to form new solutions. There are unary transformations m_i (mutation type), which create new individuals by a small change in a single individual ($m_i : S \rightarrow S$), and higher order transformations c_j (crossover type), which create new individuals by combining parts from several (two or more) individuals ($c_j : S \times \ldots \times S \rightarrow S$). After some number of generations the program converges—the best individual hopefully represents a near-optimum (reasonable) solution.

procedure evolution program
begin
  $t \leftarrow 0$
  initialize $P(t)$
  evaluate $P(t)$
  while (**not** termination-condition) **do**
  begin
    $t \leftarrow t+1$
    select $P(t)$ from $P(t-1)$
    alter $P(t)$
    evaluate $P(t)$
  end
end

Figure 1: The structure of an evolution program

The concept of evolution programs is based entirely on the idea of genetic algorithms (Holland, 1975). The differences between EPs and GAs are hidden at the lower level: in EPs we simply consider a richer set of data structures together with an expanded set of genetic operators. In EPs chromosomes need not be represented by linear strings and the alternation process includes other "genetic" operators appropriate for the given structure and the given problem.

The binary alphabet offers the maximum number of schemata per bit of information (Goldberg, 1989) and consequently the bit string representation of solutions has dominated genetic algorithm research. The binary coding also facilitates theoretical analysis and allows elegant genetic operators: the strongest theoretical results for GAs have assumed binary alphabet for mathematical simplicity. However, it is clear that binary representations are not always appropriate for highly constrained problems and other representations are often more natural. The central question is how to take advantage of such representations in evolution programs. It should be also emphasized that Holland's work (Holland, 1975) describes genetic algorithms using finite alphabets, and that the 'implicit parallelism' result does not depend on using bit strings (Antonisse, & Keller, 1989). During the last ten years, various application–specific variations on the genetic algorithm were reported (Davis, 1987; Grefenstette, 1985; Grefenstette, 1987a; Groves, Michalewicz, Elia, & Janikow, 1990; Michalewicz, Vignaux, Hobbs, 1991; Smith, 1980; Smith, 1983; Vignaux, & Michalewicz, 1991). It seems that a "natural" representation of a potential solution for a given problem plus a family of applicable "genetic" operators might be quite useful in the approximation of solutions of many problems, and this nature-modeled approach (evolution programming) is a promising direction for problem solving in general.

The basic conceptual difference between binary-coded genetic algorithms and evolution programs is presented in Figures 2 and 3. Classical genetic algorithms, which operate on binary strings, require a modification of an original problem into appropriate (suitable for GA) form; this would include mapping between potential solutions and binary representation, taking care of decoders or repair algorithms, etc.

On the other hand, evolution programs would leave the problem unchanged, modifying a chromosome representation of a potential solution (using "natural" data structures), and applying appropriate "genetic" operators.

In other words, to solve a nontrivial problem using an evolution program, we can either transform the problem into a form appropriate for the genetic algorithm (Figure 2), or we can transform the genetic algorithm to suit the problem (Figure 3). Clearly, classical GAs take the former approach, and EPs the latter.

It is quite hard to draw a line between genetic algorithms and evolution programs. Obviously, any genetic algorithm constitutes an example of an evolution program. However, not every evolution program is a genetic algorithm. For example, it is not clear whether we can use the term "genetic algorithm" for a system with matrix representation and arithmetical crossover (Michalewicz, Vignaux, & Hobbs, 1991)? Note that such a system does not have a support of any schema theorem nor of building-block hypothesis. Additionally,

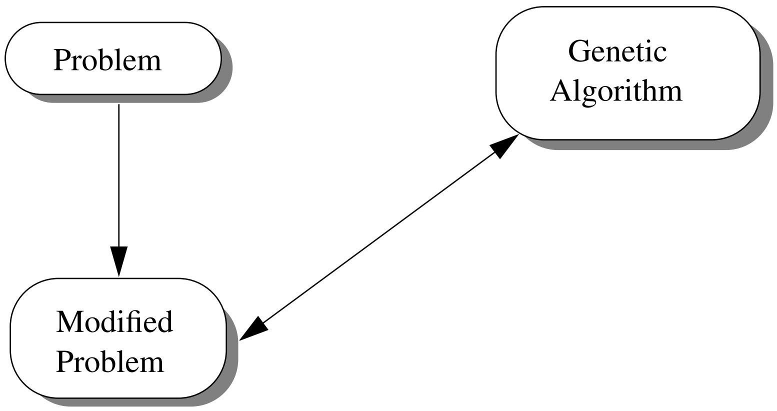

Figure 2: Genetic algorithm approach

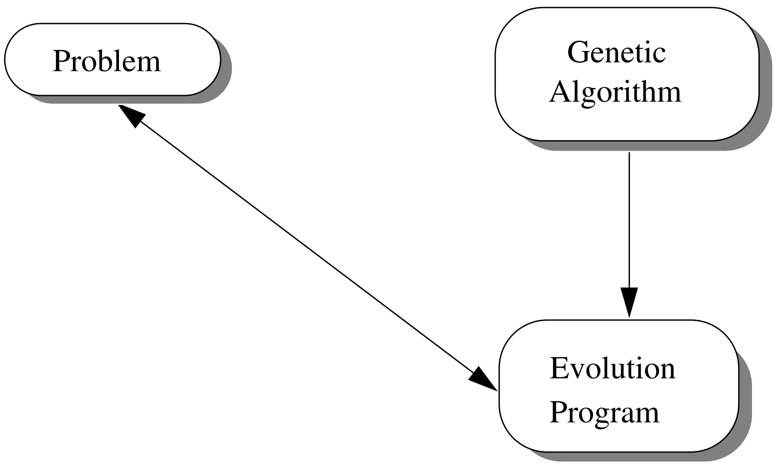

Figure 3: Evolution program approach

the system performs very well with a mutation as a single operator, i.e., where there is no 'recombination' operator at all!

However, we will not analyse this partition any further; instead we present illustrate the applicability of the evolution programming techniques to various optimization problems.

3 Evolution Program for the Transportation Problem

The classical examples of an evolution programming technique are Genetic-2 and Genetic-2n systems developed for linear and nonlinear transportation problems, respectively (Vignaux, & Michalewicz, 1991; Michalewicz, Vignaux, & Hobbs, 1991). To the best of the authors' knowledge it was the first GA-based system to use non-string chromosome structures; specialized 'genetic' operators were introduced to preserve feasibility of solutions. In the following subsections we describe the transportation problem and one of these systems: Genetic-2n, an evolution program for the nonlinear transportation problem.

3.1 The Transportation Problem

The transportation problem (Taha, 1987) is one of the simplest constrained optimization problems that have been studied. It seeks the determination of a minimum cost transportation plan for a single commodity from a number of sources to a number of destinations. A destination can receive its demand from one or more sources. The objective of the problem is to determine the amount to be shipped from each source to each destination such that the total transportation cost is minimized.

If the transportation cost on a given route is directly proportional to the number of units transported, we have a *linear transportation problem*. Otherwise, we have a *nonlinear transportation problem*.

Assume there are n sources and k destinations. The amount of supply at source i is $source(i)$ and the demand at destination j is $dest(j)$. The cost of transporting flow x_{ij} from source i to destination j is given as a function f_{ij} . Thus the total cost is a separable function of the individual flows rather than interactions between them. The transportation problem is given as:

$$\text{minimize } total = \sum_{i=1}^{n} \sum_{j=1}^{k} f_{ij}(x_{ij})$$

subject to

$$\begin{gathered}\sum_{j=1}^{k} x_{ij} \leq source(i), \text{ for } i = 1, 2, \ldots, n, \\ \sum_{i=1}^{n} x_{ij} \geq dest(j), \text{ for } j = 1, 2, \ldots, k, \\ x_{ij} \geq 0, \text{ for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, k.\end{gathered}$$

The first set of constraints stipulates that the sum of the shipments from a source cannot exceed its supply; the second set requires that the sum of the shipments to a destination must satisfy its demand.

The above problem implies that the total supply $\sum_{i=1}^{k} source(i)$ must at least equal total demand $\sum_{j=1}^{n} dest(j)$. When total supply equals total demand (total flow), the resulting formulation is called a *balanced* transportation problem. It differs from the above only in that all constraints are equations; that is,

$$\sum_{j=1}^{k} x_{ij} = source(i), \text{ for } i = 1, 2, \ldots, n,$$
$$\sum_{i=1}^{n} x_{ij} = dest(j), \text{ for } j = 1, 2, \ldots, k.$$

3.2 Genetic-2n

In Vignaux, & Michalewicz (1991) we described a few GA-based systems for the linear transportation problem. However, all systems based on a string representation gave disappointing results: either they returned nonfeasible solutions or they could not be generalized into nonlinear case (like system Genetic-1; Vignaux, & Michalewicz, 1991).

Perhaps the most natural representation of a solution for the transportation problem is a two dimensional structure. After all, this is how the problem is presented and solved by hand. In other words, a matrix $V = (x_{ij})$ $(1 \le i \le k,\ 1 \le j \le n)$ may represent a solution; each x_{ij} is a real number.

In this subsection we describe an evolution program, Genetic-2n (Michalewicz, Vignaux, & Hobbs, 1991), which is based on such two dimensional representation; we describe the most important components of this system in turn.

The *initialization* procedure creates an individual solution which satisfies all constraints. It will also turn out to be a fundamental component of one of the mutation operators. It will be called enough times at the start of a run to construct a starting population of solutions.

input: arrays $dest[k]$, $sour[n]$;
output: an array (x_{ij}) such that $x_{ij} \ge 0$ for all i and j, $\sum_{j=1}^{k} x_{ij} = sour[i]$ for $i = 1, 2, \ldots, n$, and $\sum_{i=1}^{n} x_{ij} = dest[j]$ for $j = 1, 2 \ldots, k$, i.e., all constraints are satisfied.

procedure initialization;
begin
  set all numbers from 1 to $k \cdot n$ as unvisited
  repeat
    select an unvisited random number q from 1 to $k \cdot n$ and set it as visited
    set (row) $i = \lfloor (q-1)/k + 1 \rfloor$
    set (column) $j = (q-1)\ mod\ k + 1$
    set $val = min(sour[i],\ dest[j])$
    set $x_{ij} = val$
    set $sour[i] = sour[i] - val$
    set $dest[j] = dest[j] - val$
  until all numbers are visited.
end

Procedure *initialization* creates a matrix of at most $k+n-1$ non-zero elements such that all constraints are satisfied. It is the method used for the linear case of the transportation problem (see Vignaux, & Michalewicz, 1991). Although other initialization procedures are feasible, this method will generate a solution that is at a vertex of the simplex which describes the convex boundary of the constrained solution space. The following example shows how it works:

Example 1. Let us consider a matrix with the following constraints:

$$sour[1] = 15.0,\ sour[2] = 25.0,\ \text{and}\ sour[3] = 5.0$$
$$dest[1] = 5.0,\ dest[2] = 15.0,\ dest[3] = 15.0,\ \text{and}\ dest[4] = 10.0.$$

There are altogether $3 \cdot 4 = 12$ numbers, all of them are unvisited at the beginning. Select the first random number, say, 10. This translates into row number $i = 3$ and column number $j = 2$. The $val = min(sour[3],\ dest[2]) = 5.0$, so $v_{32} = 5.0$. After the first iteration, $sour[3] = 0.0$ and $dest[2] = 10.0$.

We repeat these calculations with the next three random (unvisited) numbers, say 8, 5, and 3 (corresponding to row 2 and column 4, to row 2 and column 1, and to row 1 and column 3, respectively). The resulting matrix (v_{ij}) at this stage has the following contents:

	0.0	**10.0**	**0.0**	**0.0**
0.0			15.0	
10.0	5.0			10.0
0.0		5.0		

Note that the values of $sour[i]$ and $dest[j]$ are those given after 4 iterations.

If the further sequence of random numbers is 1, 11, 4, 12, 7, 6, 9, 2, the final matrix produced (with an assumed complete sequence of random numbers ⟨ 10, 8, 5, 3, 1, 11, 4, 12, 7, 6, 9, 2 ⟩) is:

	0.0	**0.0**	**0.0**	**0.0**
0.0	0.0	0.0	15.0	0.0
0.0	5.0	10.0	0.0	10.0
0.0	0.0	5.0	0.0	0.0

Obviously, after 12 iterations all $sour[i]$ and $dest[j]$ equal 0.0. Note also, that there are other sequences of numbers for which procedure *initialization* would produce the same solution. □

We defined three 'genetic' operators: two mutations and one crossover:

- **Mutation:**
  Two types of mutation operators are defined. The first mutation introduces as many zero entries into the matrix as possible. The second is modified to avoid choosing zero entries by selecting values from a range. They will be discussed in turn.

mutation-1 Assume that $\{i_1, i_2, \ldots, i_p\}$ is a subset of $\{1, 2, \ldots, k\}$, and $\{j_1, j_2, \ldots, j_q\}$ is a subset of $\{1, 2, \ldots, n\}$ such that $2 \le p \le k$, $2 \le q \le n$.

Denote a parent for mutation by the $(k \times n)$ matrix $V = (x_{ij})$. Then we can create a $(p \times q)$ submatrix $W = (w_{ij})$ from all elements of the matrix V in the following way: an element $x_{ij} \in V$ is in W if and only if $i \in \{i_1, i_2, \ldots, i_p\}$ and $j \in \{j_1, j_2, \ldots, j_q\}$ (if $i = i_r$ and $j = j_s$, then the element x_{ij} is placed in the r-th row and s-th column of the matrix W.

Now we can assign new values $sour_W[i]$ and $dest_W[j]$ ($1 \le i \le p$, $1 \le j \le q$) for matrix W:

$$\begin{aligned} sour_W[i] &= \textstyle\sum_{j \in \{j_1, j_2, \ldots, j_q\}} x_{ij},\ 1 \le i \le p, \\ dest_W[j] &= \textstyle\sum_{i \in \{i_1, i_2, \ldots, i_p\}} x_{ij},\ 1 \le j \le q. \end{aligned}$$

We can use procedure *initialization* to assign new values to the matrix W so that all constraints $sour_W[i]$ and $dest_W[j]$ are satisfied. Then we replace corresponding elements of matrix V by new elements from the matrix W. In this way all the global constraints ($sour[i]$ and $dest[j]$) are preserved.

The following example will illustrate the mutation operator.

Example 2. A transportation problem is defined with 4 sources and 5 destinations. The constraints are:

$$\begin{gathered} sour[1] = 8.0,\ sour[2] = 4.0,\ sour[3] = 12.0,\ sour[4] = 6.0, \\ dest[1] = 3.0,\ dest[2] = 5.0,\ dest[3] = 10.0,\ dest[4] = 7.0,\ dest[5] = 5.0. \end{gathered}$$

Assume that the following matrix V was selected as a parent for mutation:

0.0	0.0	5.0	0.0	3.0
0.0	4.0	0.0	0.0	0.0
0.0	0.0	5.0	7.0	0.0
3.0	1.0	0.0	0.0	2.0

Suppose that two rows $\{2, 4\}$ and three columns $\{2, 3, 5\}$ are selected. Then the corresponding submatrix W is:

4.0	0.0	0.0
1.0	0.0	2.0

Note, that $sour_W[1] = 4.0$, $sour_W[2] = 3.0$, $dest_W[1] = 5.0$, $dest_W[2] = 0.0$, $dest_W[3] = 2.0$. After the re-initialization of matrix W, it might have the following values:

2.0	0.0	2.0
3.0	0.0	0.0

So, finally, the child of matrix V after mutation, is:

0.0	0.0	5.0	0.0	3.0
0.0	2.0	0.0	0.0	2.0
0.0	0.0	5.0	7.0	0.0
3.0	3.0	0.0	0.0	0.0

□

mutation-2 This version is identical to mutation-1 except that in recalculating the contents of the chosen sub-matrix a modified version of the *initialization* routine is used.

It is changed from that described in section 3.2 in the following way:

The line

set $val = min(sour[i], dest[j])$

is replaced by:

set $val_1 = min(sour[i], dest[j])$
if (i is the last available row) **or**
(j is the last available column)
then $val = val_1$
else set val = random (real) number from $\langle 0, val_1 \rangle$

This change provides real numbers instead of integers and zeros but the procedure must be further modified as it currently produces a matrix which may violate the constraints.

For example, using the matrix from Example 1, suppose that the sequence of selected numbers is $\langle 3, 6, 12, 8, 10, 1, 2, 4, 9, 11, 7, 5 \rangle$ and that the first real number generated for number 3 (first row, third column) is 7.3 (which is within the range $\langle 0.0, min(sour[1], dest[3]) \rangle = \langle 0.0, 15.0 \rangle$). The second random real number for 6 (second row, second column) is 12.1, and the rest of the real numbers generated by the new initialization algorithm are: $3.3, 5.0, 1.0, 3.0, 1.9, 1.7, 0.4, 0.3, 7.4, 0.5$. The resulting matrix is:

	5.0	**15.0**	**15.0**	**10.0**
15.0	3.0	1.9	7.3	1.7
25.0	0.5	12.1	7.4	5.0
5.0	0.4	1.0	0.3	3.3

Only by adding 1.1 to the element x_{11} can we satisfy the constraints. So we need to add an additional (last) line to the *mutation-2* algorithm:

make necessary additions

This completes the modification of the it initialization procedure.

- **Arithmetical Crossover**

  Starting with two parents (matrices U and V) the arithmetical crossover operator will produce two children X and Y, where $X = c_1 \cdot U + c_2 \cdot V$ and $Y = c_1 \cdot V + c_2 \cdot U$ (where $c_1, c_2 \geq 0$ and $c_1 + c_2 = 1$). Since the constraint set is convex this operation ensures that both of children are feasible if both parents are. This is a significant simplification of the linear case where there was an additional requirement to maintain all components of the matrix as integers.

In addition to the set of control parameters used for the linear case (population size, mutation and crossover rates, random number starting seed, etc.) a few more are needed. These are the crossover proportions, c_1 and c_2, and m_1, a parameter to determine the proportion of mutation-1 in the mutations applied.

3.3 Testing Genetic-2n

In testing the Genetic-2 algorithm on the linear transportation problem (see Vignaux, & Michalewicz, 1991) we could have compared its solution with the known optimum found using the standard algorithm. Hence we could have determined how efficient the genetic algorithm was in absolute terms. Once we move to nonlinear objective functions, the optimum may not be known. Testing is reduced to comparing the results with those of other nonlinear solution methods that may themselves have converged to a local optimum.

We have chosen to compare the Genetic-2n algorithm method with the GAMS (General Algebraic Modeling System), a package for the construction and solution of mathematical programming models (Brooke, Kendrick, & Meeraus, 1988), with MINOS optimizer. GAMS represents a typical example of an industry-standard efficient method of solution. This system, being essentially a gradient-controlled method, found some of the problems we set up difficult or impossible to solve. A full comparison (on six categories of objective functions) is reported in Michalewicz, Vignaux, & Hobbs (1991) and Michalewicz (1992). A typical comparison of the optima between Genetic-2n and GAMS is shown below for a single problem.

Assume 3 sources and 4 destinations. The supply is:

$$source(1) = 10,\ source(2) = 15, \text{ and } source(3) = 20.$$

The demand is:

$$dest(1) = 3,\ dest(2) = 20,\ dest(3) = 5, \text{ and } dest(4) = 17.$$

The total flow in the above problem is 45. We have used the same function f for each flow f_{ij}; a cost-matrix was used to provide variation between flows. The matrix provides the c_{ij}'s which act to scale the basic function shape.

We adopted the following function f of the flows x_{ij}:

$$f(x_{ij}) = \begin{cases} 0 & if \;\; x_{ij} = 0, \\ d + c_{ij} \cdot \sqrt{x_{ij}} & otherwise, \end{cases}$$

for $i = 1, 2, 3$, $j = 1, 2, 3, 4$, where $d = 5.0$, and

$$\begin{array}{llll} c_{11} = 0.0 & c_{12} = 21.0 & c_{13} = 50.0 & c_{14} = 62.0 \\ c_{21} = 21.0 & c_{22} = 0.0 & c_{23} = 17.0 & c_{24} = 54.0 \\ c_{31} = 50.0 & c_{32} = 17.0 & c_{33} = 0.0 & c_{34} = 60.0. \end{array}$$

We solved the above problem using GAMS; the GAMS' best solution was 430.64, which was achieved for

$$\begin{array}{llll} x_{11} = 3.0 & x_{12} = 0.0 & x_{13} = 0.0 & x_{14} = 7.0 \\ x_{21} = 0.0 & x_{22} = 5.0 & x_{23} = 0.0 & x_{24} = 10.0 \\ x_{31} = 0.0 & x_{32} = 15.0 & x_{33} = 5.0 & x_{34} = 0.0. \end{array}$$

We have also performed 20 runs of Genetic-2n. The values of the total transportation cost varied from 397.02 (the worst case) for a solution (rounded to the second digit after the decimal point):

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 5.00 & x_{13} = 0.00 & x_{14} = 2.00 \\ x_{21} = 0.00 & x_{22} = 15.00 & x_{23} = 0.00 & x_{24} = 0.00 \\ x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 5.00 & x_{34} = 15.00, \end{array}$$

to the value of 356.98 (the best case) for a solution:

$$\begin{array}{llll} x_{11} = 3.00 & x_{12} = 7.00 & x_{13} = 0.00 & x_{14} = 0.00 \\ x_{21} = 0.00 & x_{22} = 13.00 & x_{23} = 2.00 & x_{24} = 0.00 \\ x_{31} = 0.00 & x_{32} = 0.00 & x_{33} = 3.00 & x_{34} = 17.00 \end{array}$$

201z200z The above example demonstrates the usefulness of the evolution programming approach for the nonlinear transportation problem. Results of further experiments on other test cases for Genetic-2n are reported in Michalewicz, Vignaux, & Hobbs, (1991); and Michalewicz (1992).

4 Handling Linear Constraints: Genocop

In this section we discuss another evolution program, Genocop, which optimizes a function with linear constraints. It is interesting to see how this GA-based system incorporates the ideas present in simulated annealing, scatter search, and evolutionary strategies. We can also claim that Genocop is a generalization of Genetic-2n, as Genetic-2n can handle only a particular set of linear constraints (equations present in the transportation problem), whereas Genocop can handle any set of linear constraints. Such hierarchical relationship between Genetic-2n, Genocop, and a few other evolution programs is fully discussed in (Michalewicz, 1993).

Let us start with the description of the problem. The general nonlinear programming problem $\mathcal{P}$ is to find $\overline{X}$ so as to

$$\text{optimize } f(\overline{X}),\ \overline{X} = (x_1, \ldots, x_q) \in R^q,$$

subject to $p \geq 0$ equations:

$$c_i(\overline{X}) = 0,\ i = 0, \ldots, p,$$

and $m - p \geq 0$ inequalities:

$$c_i(\overline{X}) \leq 0,\ i = p+1, \ldots, m.$$

In the area of constrained optimization, 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 (Cooper, 1970). There is only one special type of set—a convex set—for which a significant amount of theory has been developed.

In this section 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 $k = 1, \ldots, n$.

The assumption (important for the implementation of the Genocop system) that the bounds $\langle left(k), right(k) \rangle$ can be efficiently computed, turned our attention to linear constraints. For this reason, initially we consider a subclass $\mathcal{S} \subset \mathcal{P}$ of nonlinear programming problems, that can be formulated as follows:

optimize $f(\overline{X})$, $\overline{X} = (x_1, \ldots, x_q) \in R^q$,

subject to the following sets of linear constraints:

1. Domain constraints: $l_i \leq x_i \leq u_i$ for $i = 1, 2, \ldots, q$. We write $\overline{L} \leq \overline{X} \leq \overline{U}$, where $\overline{L} = \langle l_1, \ldots, l_q \rangle$, $\overline{U} = \langle u_1, \ldots, u_q \rangle$.

2. Equalities: $A\overline{X} = \overline{B}$, where $A = (a_{ij})$, $\overline{B} = \langle b_1, \ldots, b_p \rangle$, $1 \leq i \leq p$, and $1 \leq j \leq q$ (p is the number of equations).

3. Inequalities: $N\overline{X} \leq \overline{D}$, where $N = (n_{ij})$, $\overline{D} = \langle d_1, \ldots, d_{m-p} \rangle$, $1 \leq i \leq m-p$, and $1 \leq j \leq q$ ($m-p$ is the number of inequalities).

In this formulation, the search space of feasible solutions is convex and, of course, the bounds $\langle left(k), right(k) \rangle$ can be efficiently computed. We explore the properties of the convex search space in the design of the evolution program, Genocop. In the next section we return to the general nonlinear programming problem $\mathcal{P}$.

4.1 Genocop

Genocop borrows different ideas from several randomized methods. The structure of the Genocop is based on genetic algorithm. However, the system uses floating point representation for chromosomes (as evolutionary strategies do). One of the operators (arithmetical crossover) has its origin in scatter search, whereas the ideas of simulated annealing (smaller changes with lower temperature) were incorporated in other operator: non-uniform mutation.

As indicated earlier, Genocop is capable of optimizing a function with linear constraints (equations and inequalities). In some optimization techniques, such as linear programming, equality constraints are welcome since it is known that the optimum, if it exists, is situated at the surface of the convex set. Inequalities are converted to equalities by the addition of slack variables, and the solution method proceeds by moving from vertex to vertex, around the surface.

In contrast, for a method that generates solutions randomly, such equality constraints are a nuisance. In Genocop they are eliminated at the start, together with an equal number of problem variables; this action removes also part of the space to be searched. The remaining constraints, in the form of linear inequalities, form a convex set which must be searched for a solution. Its convexity ensures that linear combinations of solutions yield solutions without needing to check the constraints—a property used throughout this approach. The inequalities can be used to generate bounds for any given variable: such bounds are dynamic as they depend on the values of the other variables and can be efficiently computed. So the main idea behind this approach lies in (1) an elimination of the equalities present in the set of constraints, and (2) careful design of special "genetic" operators, which guarantee to keep all chromosomes within the constrained solution space. We discuss these two aspects of the algorithm in turn.

Suppose the equality constraint set is represented in matrix form:

$$A\overline{X} = \overline{B}.$$

We assume there are p independent equality equations (there are easy methods to verify this), i.e., there are p variables $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$ ($\{i_1, \ldots, i_p\} \subseteq \{1, 2, \ldots, q\}$) which can be determined in terms of the other variables. These can, therefore, be eliminated from the problem statement, as follows.

We can split the array A vertically into two arrays A_1 and A_2, such that the j-th column of the matrix A belong to A_1 iff $j \in \{i_1, \ldots, i_p\}$. Thus A_1^{-1} exists. Similarly, we split matrix N and vectors $\overline{X}, \overline{L}, \overline{U}$ (*ie.* $\overline{X}_1 = \langle x_{i_1}, \ldots, x_{i_p}\rangle$, $\overline{L}_1 = \langle l_{i_1}, \ldots, l_{i_p}\rangle$, and $\overline{U}_1 = \langle u_{i_1}, \ldots, u_{i_p}\rangle$).

Then

$$A_1\overline{X}_1 + A_2\overline{X}_2 = \overline{B}.$$

It is easily seen that

$$\overline{X}_1 = A_1^{-1}\overline{B} - A_1^{-1}A_2\overline{X}_2.$$

Using the above rule, we can eliminate the variables $x_{i_1}, \ldots, x_{i_p}$ replacing them by a linear combination of remaining variables. However, each variable x_{i_j} ($j = 1, 2, \ldots, p$) is constrained additionally by a domain constraint: $l_{i_j} \leq x_{i_j} \leq u_{i_j}$. Eliminating all variables x_{i_j} leads us to introduce a new set of inequalities:

$$\overline{L}_1 \leq A_1^{-1}\overline{B} - A_1^{-1}A_2\overline{X}_2 \leq \overline{U}_1,$$

which is added to the original set of inequalities.

The original set of inequalities,

$$N\overline{X} \leq \overline{D}$$

can be represented as

$$N_1\overline{X}_1 + N_2\overline{X}_2 \leq \overline{D}.$$

This can be transformed into

$$N_1(A_1^{-1}\overline{B} - A_1^{-1}A_2\overline{X}_2) + N_2\overline{X}_2 \leq \overline{D}.$$

So, after eliminating the p variables $x_{i_1}, \ldots, x_{i_p}$, the final set of constraints consists of the following inequalities only:

1. original domain constraints: $\overline{L}_2 \leq \overline{X}_2 \leq \overline{U}_2$,
2. new inequalities: $\overline{L}_1 \leq A_1^{-1}\overline{B} - A_1^{-1}A_2\overline{X}_2 \leq \overline{U}_1$,
3. original inequalities (after removal of $\overline{X}_1$ variables): $(N_2 - N_1A_1^{-1}A_2)\overline{X}_2 \leq \overline{D} - N_1A_1^{-1}\overline{B}$.

The Genocop system tries to locate an initial (feasible) solution by sampling the feasible region. If some predefined number of trials is unsuccessful, the system would prompt the user for a feasible initial point. The initial population consists of identical copies of such initial point (whether generated or provided by the user). The evaluation process is straightforward. The objective function f is given and each potential solution $\overline{X} = (x_1, \ldots, x_q)$ of the population yields a value of $f(x_1, \ldots, x_q)$.

There are six operators (three mutations and three crossovers) in the Genocop system which proved to be useful on many test problems. These are:

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 (Genocop II, described in the next section).

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.

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.

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.

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 (Davis, 1989) (when $a = 1/2$); *intermediate crossover* in (Bäck et al., 1991); *linear crossover* in (Wright, 1990); and *arithmetical crossover* in (Michalewicz & Janikow, 1991) and (Michalewicz, 1992).

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.

Heuristic crossover. This operator was suggested by Wright (1990). 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.

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

4.2 Testing Genocop

In order to evaluate the Genocop method, a set of test problems have been carefully selected to illustrate the performance of the algorithm and to indicate that it has been successful in practice. The eight test cases, which include quadratic, nonlinear, and discontinuous functions with several linear constraints, are fully discussed in Michalewicz, & Swaminathan (1993). Here we provide only two test cases.

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; classification step), $b = 2$ (coefficient for non-uniform mutation), $a = 0.25$ (parameter of arithmetical crossover).

For each test case we run the Genocop ten times. For all problems, the number of generations T was 500. Both test cases and the results of the Genocop system are reported in the following subsections.

Test Case #1.

The problem (taken from Hock, & Schittkowski, 1981) 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:

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

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 best known solution reported in Hock, & Schittkowski (1981) is

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

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

The Genocop system (in all ten runs) found points with better value than the one above:

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

for which the value of the objective function is equal to -47.760765. A single run of 500 iterations took 13 sec of CPU time.

Test Case #2

The following test case was taken from Colville (1968):

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

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

A typical solution returned by the Genocop in 10,000 generations was

$$\overline{X} = (1.000581, 1.001166, 0.999441, 0.998879),$$

with $f(\overline{X}) = 0.0000012$. A single run of 10,000 iterations took 105 sec of CPU time.

This is a very good result—the above (unconstrained) test function proved difficult for many optimization methods. Even VFSR (Very Fast Simulated Reannealing) produced inferior result:

$$\overline{X} = (1.1244, 1.2633, 0.8614, 0.7416),$$

with $f(\overline{X}) = 0.062267$.

5 Handling Nonlinear Constraints: Genocop II

Let us return now to the general nonlinear programming problem $\mathcal{P}$. It is clear that Genocop incorporates the knowledge on the convex search space in its operators and takes the advantage of quick transformations of linear constraints. Thus it is hard to generalize Genocop directly into a system capable of handling nonlinear constraints.

Here a new idea for construction of a new system capable of processing nonlinear constraints came from the traditional calculus based methods. Calculus based methods assume that the objective function $f(\overline{X})$ and all constraints are twice continuously differentiable functions of $\overline{X}$. The general approach in most methods is to transform the nonlinear problem $\mathcal{P}$ into a sequence of solvable subproblems. The amount of work involved in a subproblem varies considerably among methods. These methods require explicit (or implicit) second derivative calculations of the objective (or transformed) function which in some methods can be ill-conditioned and causes the algorithm to fail.

During the last 30 years there has been considerable research directed toward the nonlinear optimization problems and progress has been made in theory and practice (Floudas, & Pardalos, 1992). Several approaches have been developed in this area, among these are: the sequential quadratic penalty function (Broyden, & Attia, 1983; Attia, 1985), recursive quadratic programming method (Biggs, 1975), penalty trajectory methods (Murray, 1969), and the SOLVER method (Fletcher, 1981).

One of these approaches, the sequential quadratic penalty function method, seemed appropriate for coupling with Genocop; we describe the method briefly.

The method replaces a problem $\mathcal{P}$ by the problem $\mathcal{P}'$:

$$\text{optimize } F(\overline{X}, r) = f(\overline{X}) + \tfrac{1}{2r}\overline{C}^T\overline{C},$$

where $r > 0$ and $\overline{C}$ is a vector of all active constraints $c_1, \ldots, c_\ell$.

Fiacco and McCormick (1968) have shown that the solutions of $\mathcal{P}$ and $\mathcal{P}'$ were identical in the limit as $r \longrightarrow 0$. It was thought that $\mathcal{P}'$ could then be solved simply by minimizing $F(\overline{X}, r)$ for a sequence of decreasing positive values of r by Newton's method then being developed (Fiacco, & McCormick, 1968). This hope, however, was short-lived, because minimizing $F(\overline{X}, r)$ proved to be extremely inefficient for the smaller values of r; it was shown by Murray (1969) that this was due to the Hessian matrix of $F(\overline{X}, r)$ becoming increasingly ill-conditioned as $r \longrightarrow 0$. Since there seemed to be no obvious way of overcoming this problem, the method gradually fell into disuse. More recently, Broyden and Attia (1983) gave a method of overcoming the numerical difficulties associated with the simple quadratic penalty function. In this method the computation of the search direction does not require the solution of any system of linear equations, and can thus be expected to require less work than in some other algorithms. The method also provides an automatic technique for calculating the initial value for the parameter r and its successive values.

The above technique together with the existing system Genocop, was used in construction of a new system, Genocop II. The structure of the Genocop II is given in Figure 4. We discuss the steps of this algorithm in the remaining part of this section.

There are several steps of the algorithm in the first phase of its execution (before it enters the *while* loop). The parameter t (which counts the number of iterations of the algorithm, i.e., the number of times the algorithm Genocop is applied) is initialized to zero. The set of all constraints C is divided into three subsets: linear constraints L, nonlinear equations N_e and nonlinear inequalities N_i. A starting point $\overline{X}_s$ (which need not be feasible) for the following optimization process is selected (or a user is prompted for it). The set of active constraints A consists initially of elements of N_e and set $V \subseteq N_i$ of violated constraints from $N - i$. A constraint $c_j \in N_i$ is violated at point $\overline{X}$ iff $c_j(\overline{X}) > \delta$ $(j = p+1, \ldots, m)$, where δ is a parameter of the method. Finally, the initial temperature of the system τ is set to τ_0 (a parameter of the method).

In the main loop of the algorithm we apply Genocop to optimize a modified function

$$F(\overline{X}, \tau) = f(\overline{X}) + \tfrac{1}{2\tau}\overline{A}^T\overline{A}$$

with linear constraints L. Note that the initial population for Genocop consists of *pop_size* identical copies (of the initial point for the first iteration and of the best saved point for subsequent ones); several mutation operators introduce diversity in the population at early stages of the process. When Genocop converges, its best individual $\overline{X}^*$ is saved and used later as the starting point $\overline{X}_s$ for the next iteration. However, the next iteration is executed with a decreased value of the temperature parameter $(\tau \leftarrow g(\tau, t))$ and a new set of active constraints A:

$$A \leftarrow A - S \cup V,$$

where S and V are subsets of N_i satisfied and violated by $\overline{X}^*$, respectively.

procedure Genocop II
begin
  $t \longleftarrow 0$
  split the set of constraints C into
    $C = L \cup N_e \cup N_i$
  select a starting point $\overline{X}_s$
  set the set of active constraints, A to
    $A \longleftarrow N_e \cup V$
  set temperature $\tau \longleftarrow \tau_0$
  while (**not** termination-condition) **do**
  begin
    $t \longleftarrow t + 1$
    execute Genocop for the function
      $F(\overline{X}, \tau) = f(\overline{X}) + \frac{1}{2\tau}\overline{A}^T\overline{A}$
      with linear constraints L
      and the starting point $\overline{X}_s$
    save the best individual $\overline{X}^*$:
      $\overline{X}_s \longleftarrow \overline{X}^*$
    update A:
      $A \leftarrow A - S \cup V$,
    decrease temperature τ:
      $\tau \leftarrow g(\tau, t)$
  end
end

Figure 4: The structure of Genocop II

The mechanism of the algorithm is illustrated on the following example. The problem is to

$$\text{minimize } f(\overline{X}) = x_1 \cdot x_2^2,$$

subject to one nonlinear constraint:

$$c_1 \ : \ \ 2 - x_1^2 - x_2^2 \geq 0.$$

The known global solution is $\overline{X}^* = (-0.816497, -1.154701)$, and $f(\overline{X}^*) = -1.088662$. The starting feasible point is $\overline{X}_0 = (-0.99, -0.99)$.

After the first iteration of Genocop II (A is empty) the system converged to $\overline{X}_1 = (-1.5, -1.5)$, $f(\overline{X}_1) = -3.375$. The point $\overline{X}_1$ violates the constraint c_1, which becomes active. The point $\overline{X}_1$ is used as the starting point for the second iteration.

The second iteration ($\tau = 10^{-1}$, $A = \{c_1\}$) resulted in $\overline{X}_2 = (-0.831595, -1.179690)$, $f(\overline{X}_2) = -1.122678$. The point $\overline{X}_2$ is used as the starting point for the third iteration.

The third iteration ($\tau = 10^{-2}$, $A = \{c_1\}$) resulted in $\overline{X}_3 = (-0.815862, -1.158801)$, $f(\overline{X}_3) = -1.09985$.

The sequence of points $\overline{X}_t$ (where $t = 4, 5, \ldots$ is the iteration number of the algorithm) approaches the optimum.

5.1 Testing Genocop II

In order to evaluate the method of Genocop II, a set of test problems have been carefully selected to illustrate the performance of the algorithm and to indicate that it has been successful in practice. The five test cases, which include quadratic, nonlinear, and discontinuous functions with several nonlinear constraints, are discussed fully in Michalewicz, & Attia (1993). All runs of the system were performed on SUN SPARC station 2. We used the following parameters for Genocop 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), $\delta = 0.01$ (parameter which determines whether a constraint is active or not).

In most cases, the initial temperature τ_0 was set at 1 (i.e., $g(\tau, 0) = 1$); additionally, $g(\tau, t) = 10^{-1} \cdot g(\tau, t-1)$. For each test case we run the Genocop II ten times. For most problems, the number of generations necessary for Genocop to converge was 1000 (harder problems required larger number of iterations). We did not report the computational times for these test cases, since we do not have full implementation of Genocop II yet. The actions of the system were simulated by executing its the external loop in manual fashion: when Genocop converges, the best point is incorporated as the starting point for the next iteration, the constraints are checked for their activity status, and the evaluation function is adjusted accordingly.

Two characteristic test cases and the results of the Genocop II system are reported in the following subsections.

Test Case #1

The problem (taken from Floudas, & Pardalos, 1987) is

minimize $f(x, y) = -x - y$,

subject to nonlinear constraints:

$$\begin{aligned} c_1 &: \ y \le 2x^4 - 8x^3 + 8x^2 + 2, \\ c_2 &: \ y \le 4x^4 - 32x^3 + 88x^2 - 96x + 36, \end{aligned}$$

and bounds:

$0 \le x \le 3$ and $0 \le y \le 4$.

The known global solution is $\overline{X}^* = (2.3295, 3.1783)$, and $f(\overline{X}^*) = -5.5079$. The starting feasible point is $\overline{X}_0 = (0, 0)$. The feasible region is almost disconnected.

The Genocop II approached the optimum very closely at the 4th iteration. The progress of the system is reported in the table 1.

Iteration number	The best point	Active constraints
0	(0,0)	none
1	(3,4)	c_2
2	(2.06, 3.98)	c_1, c_2
3	(2.3298, 3.1839)	c_1, c_2
4	(2.3295, 3.1790)	c_1, c_2

Table 1: Progress of Genocop II on test case #1; for iteration 0 the best point is the starting point

Test Case #2

The problem (taken from Floudas, & Pardalos, 1987) is

minimize $f(\overline{X}) = (x_1 - 10)^3 + (x_2 - 20)^3$,

subject to nonlinear constraints:

$$\begin{aligned} c_1 &: \quad (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0, \\ c_2 &: \quad -(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0, \end{aligned}$$

and bounds:

$13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$.

The known global solution is $\overline{X}^* = (14.095, 0.84296)$, and $f(\overline{X}^*) = -6961.81381$ (see figure 5). The starting not feasible point is $\overline{X}_0 = (20.1, 5.84)$.

The Genocop II approached the optimum very closely at the 12th iteration. The progress of the system is reported in the table 2.

6 Conclusions

In the previous sections we examined a few evolution programs and demonstrated their usefulness on several test cases. In many cases evolution programs (perceived as hybrid

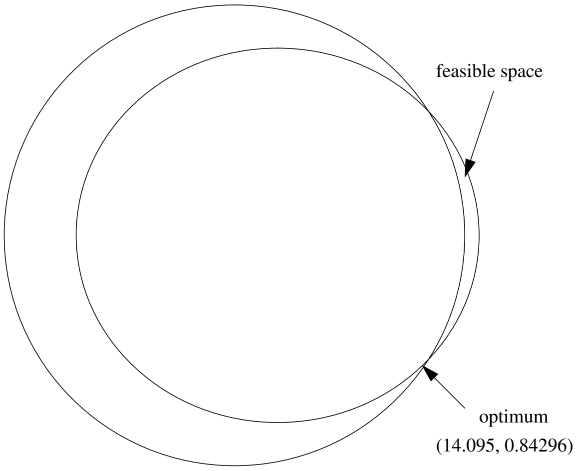

Figure 5: A feasible space for test case #2.

Iteration number	The best point	Active constraints
0	(20.1, 5.84)	c_1, c_2
1	(13.0, 0.0)	c_1, c_2
2	(13.63, 0.0)	c_1, c_2
3	(13.63, 0.0)	c_1, c_2
4	(13.73, 0.16)	c_1, c_2
5	(13.92, 0.50)	c_1, c_2
6	(14.05, 0.75)	c_1, c_2
7	(14.05, 0.76)	c_1, c_2
8	(14.05, 0.76)	c_1, c_2
9	(14.10, 0.87)	c_1, c_2
10	(14.10, 0.86)	c_1, c_2
11	(14.10, 0.85)	c_1, c_2
12	(14.098, 0.849)	c_1, c_2

Table 2: Progress of Genocop II on test case #2; for iteration 0 the best point is the starting point

systems, 'borrowing' ideas from scatter search, evolutionary strategies, simulated annealing, and, above all, genetic algorithms) perform much better than any of the 'classical' methods. For example, Genetic-2n gave excellent results for nonlinear transportation problem, for which a binary genetic algorithm failed (Vignaux & Michalewicz, 1991; Michalewicz, Vignaux & Hobbs, 1991); Genocop on some test cases (e.g., test case #2, Section 4) performed much better than Very Fast Simulated Reannealing. The general idea behind evolution programming lies within "natural" data structures and problem-sensitive "genetic operators"; this idea was explored 'to the limits' in all three presented systems.

However, it still might be difficult to build such a system (for a particular problem) from scratch. An experienced programmer would manage this task; however, the resulting program might be quite inefficient. Trade-offs between the effort of finding an effective representation for general-purpose evolution programs and the effort of developing more specialized systems are fully discussed in Michalewicz, (1993).

References

Antonisse, H.J. (1989). A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint. *Proceedings of the Third International Conference on Genetic Algorithms* (pp.86–91). George Mason University, VA: Morgan Kaufmann.

Antonisse, H.J., & Keller, K.S. (1987). Genetic Operators for High Level Knowledge Representation. *Proceedings of the Second International Conference on Genetic Algorithms* (pp.69–76). Cambridge, MA: Lawrence Erlbaum.

Attia, N.F., (1983). *New Methods of Constrained Optimization using Penalty Functions*, Ph.D. Thesis, Essex University, England.

Bäck, T., Hoffmeister, F., & Schwefel, H.-P. (1991). A Survey of Evolution Strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp.2–9). San Diego, CA: Morgan Kaufmann.

Belew, R., & Booker, L. (Eds.). (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, Los Altos, CA.

Brooke, A., Kendrick, D., & Meeraus, A. (1988). *GAMS: A User's Guide*. The Scientific Press, Redwood City, CA.

Biggs, M.C., (1975). Constrained Minimization using Recursive Quadratic Programming: Some Alternative Subproblem Formulations. In *Towards Global Optimization*, (Ed, L.C.W. Dixon and G.P. Szego), North-Holland.

Broyden, C.G. & Attia, N.F., (1983). A Smooth Sequential Penalty Function Method for Solving Nonlinear Programming Problem. Lecture Notes in Control and Information Science 59, System Modelling and Optimization, Proceedings of the 11th IFIP Conference, July 1983. Springer-Verlag, Berlin and New York.

Broyden, & Attia, N.F., (1988). Penalty Functions, Newton's Method, and Quadratic Programming. *Journal of Optimization Theory and Applications, 58.*

Colville, A.R., (1968). A Comparative Study on Nonlinear Programming Codes. IBM Scientific Center Report 320-2949, New York.

Coombs, S., & Davis, L. (1987). Genetic Algorithms and Communication Link Speed Design: Theoretical Considerations. *Proceedings of the Second International Conference on Genetic Algorithms* (pp.252–256). Cambridge, MA: Lawrence Erlbaum.

Cooper, L., & Steinberg, D., (1970). *Introduction to Methods of Optimization*, W.B. Saunders, London.

Davis, L., (Ed.). (1987). *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Davis, L., & Coombs, S. (1987). Genetic Algorithms and Communication Link Speed Design: Constraints and Operators. *Proceedings of the Second International Conference on Genetic Algorithms* (pp.257–260). Cambridge, MA: Lawrence Erlbaum.

Davis, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms* (pp.61–69). George Mason University, VA: Morgan Kaufmann.

Davis, L., (Ed.). (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York.

DeJong, K.A. (1985). Genetic Algorithms: A 10 Year Perspective. *Proceedings of the First International Conference on Genetic Algorithms* (pp.169–177). Pittsburgh, PA: Lawrence Erlbaum.

DeJong K.A., & Spears, W.M. (1989). Using Genetic Algorithms to Solve NP-Complete Problems. *Proceedings of the Third International Conference on Genetic Algorithms* (pp.124–132). George Mason University, VA: Morgan Kaufmann.

Dhar, V., & Ranganathan, N. (1990). Integer Programming vs. Expert Systems: An Experimental Comparison. *Communications of the ACM, 33*, 323–336.

Fiacco, A.V., & McCormick, G.P., (1968). *Nonlinear Programming*, John Wiley, New York.

Fletcher, R., (1981). *Practical Methods of Optimization*, Vol.2, of *Constrained Optimization*, John Wiley and Sons, Chichester and New York.

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

Floudas, C.A., & Pardalos, P.M. (1992). *Recent Advances in Global Optimization*, Princeton Series in Computer Science, Princeton University Press, Princeton, NJ.

Fogel, L.J., Owens, A.J., & Walsh, M.J. (1966). *Artificial Intelligence through Simulated Evolution.* Wiley, New York.

Forrest, S. (1985). Implementing Semantic Networks Structures using the Classifier System. *Proceedings of the First International Conference on Genetic Algorithms* (pp.24–44). Pittsburgh, PA: Lawrence Erlbaum.

Fox, B.R., & McMahon, M.B. (1990). Genetic Operators for Sequencing Problems. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann.

Glover, F., (1977). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences, 8*, pp.156–166.

Glover, F., (1989). Tabu Search — Part I. *ORSA Journal on Computing, 1*, pp.190–206.

Glover, F., (1990). Tabu Search — Part II. *ORSA Journal on Computing, 2*, pp.4–32.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA.

Goldberg, D.E. (1989a). Zen and the Art of Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms* (pp.80–85). George Mason University, VA: Morgan Kaufmann.

Grefenstette, J.J. (Ed.). (1985). *Proceedings of the First International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Grefenstette, J.J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, 16*, 122–128.

Grefenstette, J.J. (1987). Incorporating Problem Specific Knowledge into Genetic Algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann.

Grefenstette, J.J., (Ed.). (1987a). *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Groves, L., Michalewicz, Z., Elia, P. & Janikow, C. (1990). Genetic Algorithms for Drawing Directed Graphs. *Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems*, (pp.268–276). Knoxville, TN: North-Holland, Amsterdam.

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

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press.

Jones, D.R. & Beltramo, M.A. (1991). Solving Partitioning Problems with Genetic Algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp.442–449). San Diego, CA: Morgan Kaufmann.

Koza, J.R., (1990). Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. Report No. STAN–CS–90–1314, Stanford University.

Koza, J.R., (1992). *Genetic Programming.* MIT Press, Cambridge, MA.

McCormick, G.P., (1976). Computability of Global Solutions to Factorable Nonconvex Programs. Part I: Convex Underestimating Problems. *Mathematical Programming, 10*, pp.147–175.

Michalewicz, Z., & Janikow, C. (1991). Genetic Algorithms for Numerical Optimization. *Statistics and Computing, 1*, 75–91.

Michalewicz, Z., & Janikow, C. (1991a). Handling Constraints in Genetic Algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp.151–157). San Diego, CA: Morgan Kaufmann.

Michalewicz, Z. & Janikow, C. (1992). GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints. Accepted for publication, *Communications of the ACM.*

Michalewicz, Z., Janikow, C., & Krawczyk, J. (1992). A Modified Genetic Algorithm for Optimal Control Problems. *Computers & Mathematics with Applications, 23*, 83–94.

Michalewicz, Z., Vignaux, G.A., & Hobbs, M. (1991). A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem. *ORSA Journal on Computing, 3*, 307–316.

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

Michalewicz, Z. & Swaminathan, S., (1993). GENOCOP: First Experiments and Results. Submitted for publication.

Michalewicz, Z., (1993). A Hierarchy of Evolution Programs: An Experimental Study. *Evolutionary Computation, 1.*

Michalewicz, Z. & Attia, N.F., (1993). Genetic Algorithm + Simulated Annealing = GENOCOP II: A Tool for Nonlinear Programming. Submitted for publication.

Michalewicz, Z. & Logan, T.D., (1993). GA Operators for Convex Continuous Parameter Spaces. Submitted for publication.

Montana, D.J., & Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA.

Murray, W., (1969). An Algorithm for Constrained Minimization. *Optimization*, (Ed: R. Fletcher), pp.247–258, Academic Press, London and New York.

Rawlins, G. (1991). *Foundations of Genetic Algorithms.* First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, Los Altos, CA.

Rechenberg, I., (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog Verlag, Stuttgart, 1973.

Schaffer, J., (Ed.). (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, Los Altos, CA.

Schraudolph, N., & Belew, R. (1992). Dynamic Parameter Encoding for Genetic Algorithms, *Machine Learning, 9*, 9–21.

Schwefel, H.-P. (1981). *Numerical Optimization for Computer Models*, Wiley, Chichester, UK.

Schwefel, H.-P. (1992). Thesis presented at PPSN 92. *Genetic Algorithm Digest, 6.35*, October 1992.

Smith, S.F. (1980). A Learning System Based on Genetic Algorithms. PhD Dissertation, University of Pittsburgh.

Smith, S.F. (1983). Flexible Learning of Problem Solving Heuristics through Adaptive Search. *Proceedings of the Eighth International Conference on Artificial Intelligence*, Morgan Kaufman.

Starkweather, T., McDaniel, S., Mathias, K., Whitley, C., & Whitley, D. (1991). A Comparison of Genetic Sequencing Operators. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp.69–76). San Diego, CA: Morgan Kaufmann.

Taha, H.A. (1987). *Operations Research: An Introduction*, 4th ed, Collier Macmillan, London, UK.

Vignaux, G.A., & Michalewicz, Z. (1991). A Genetic Algorithm for the Linear Transportation Problem. *IEEE Transactions on Systems, Man, and Cybernetics, 21*, 445–452.

Whitley, D. (1989). The GENITOR Algorithm and Selection Pressure: Why Rank–Based Allocation of Reproductive Trials is Best. *Proceedings of the Third International Conference on Genetic Algorithms* (pp.116–121). George Mason University, VA: Morgan Kaufmann.

Wright, A.H. (1990). 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, pp. 205–218.