Evolutionary Computation Techniques for Nonlinear Programming Problems

Zbigniew Michalewicz
Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA
zbyszek@mosaic.uncc.edu

Abstract

The paper presents several evolutionary computation techniques and discusses their applicability to nonlinear programming problems. On the basis of this presentation we discuss also a construction of a new hybrid optimization system, Genocop II, and present its experimental results on a few test cases (nonlinear programming problems).

Keywords: *evolutionary computation, genetic algorithm, random algorithm, optimization technique, nonlinear programming, constrained optimization.*

INTRODUCTION

The general nonlinear programming problem $\mathcal{NP}$ is to find $\overline{X}$ so as to

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:

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

There is no known method of determining the global maximum (or minimum) to the general nonlinear programming problem. Only if the objective function f and the constraints c_i satisfy certain properties, the global optimum can sometimes be found. Several algorithms were developed for unconstrained problems (e.g., direct search method, gradient method) and constrained problems (these algorithms usually are classified as indirect and direct methods. An indirect method attacts the problem by extracting one or more linear problems from the original one, whereas a direct method tries to determine successive search points. This is usually done by converting the original problem into unconstrained one for which gradient methods are applied with some modifications; Taha, 1987). Despite the active research and progress in global optimization in recent years (Floudas, & Pardalos, 1992), it is probably fair to say that no efficient solution procedure is in sight for the general nonlinear problems $\mathcal{NP}$.

There are many other problems connected with traditional optimization techniques. For example, most proposed methods are local in scope, they depend on the existence of derivatives, and they are insufficiently robust in discontinuous, vast multimodal, or noisy search spaces (Goldberg, 1989). It is important then to investigate other (heuristic) methods, which, for many real world problems, may prove very useful.

In this paper we survey several evolutionary computation techniques and present one particular method which approaches nonlinear optimization problems. The concept of the presented system (Genocop II) is based on the ideas taken from the recent developments in area of optimization combined together with components of genetic algorithms, simulated annealing, evolution strategies, and scatter search. The prototype of the new system was run on several test-cases; the results of experiments are presented and compared with known optimas.

The paper is organized as follows. The next section provides a brief discussion on genetic algorithms, scatter search, simulated annealing, and evolution strategies. This is followed by a description of the system (Genocop) capable of optimizing a function (which need not be continuous) with linear constraints. The discussion is supported by a few experimental results; in one test case the system found the point better than the best known optimum. Then we discuss briefly one particular (gradient-based) traditional method to solve nonlinear programming problems and indicate, how the idea of the method can be combined with existing Genocop system to yield Genocop II: a general nonlinear programming tool. Again, the discussion is illustrated by a few experimental results of the system on nonlinear test cases. The last section contains conclusions and some directions for future work.

EVOLUTIONARY COMPUTATION TECHNIQUES

In this section we provide a brief discussion on genetic algorithms, scatter search, simulated annealing, and evolutionary strategies, and indicate the way they approach the problems in the class $\mathcal{NP}$.

Genetic Algorithms

Genetic Algorithms (GAs) (Holland, 1975) are stochastic algorithms whose search methods model a natural phenomena: genetic inheritance and Darwinian strife for survival. As stated in (Davis, 1987):

> "... the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The 'knowledge' that each species has gained is embodied in the makeup of the chromosomes of its members."

Genetic algorithms (GA) start with a population of randomly generated candidates and 'evolve' towards better solutions by applying genetic operators, modeled on the genetic processes occurring in nature. This population undergoes evolution in a form of natural selection. In each generation, relatively 'good' solutions reproduce to give offspring that replace the relatively 'bad' solutions which die. An evaluation or objective function plays the role of the environment to distinguish between good and bad solutions. The structure of a simple genetic algorithm is shown in Figure 1.

```
procedure genetic algorithm
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
   begin
      t = t + 1
      select P(t) from P(t − 1)
      recombine P(t)
      evaluate P(t)
   end
end
```

Figure 1: Genetic algorithm

During iteration t each solution, $\overline{X}_i$, from the population (the population size pop_size remains fixed through the evolution process) is evaluated by computing $f(\overline{X}_i)$, a measure of its fitness. A new population, $P(t+1)$, is then formed: we select solutions to reproduce on the basis of their relative fitness, and the selected solutions are recombined using genetic operators to form the new population.

The mutation operator arbitrarily alters one or more components of a selected structure so as to increase the structural variability of the population. Each position of each

solution vector in the population undergoes a random change with a probability defined by the mutation rate.

The crossover operator combines the features of two parent structures to form two similar offspring. It operates by swapping corresponding segments of a string representing the parent solutions.

For problems in the class $\mathcal{NP}$, genetic algorithms usually use penalty function approach to handle constraints. The original problem $\mathcal{NP}$ is transformed into:

$$\text{optimize } f(x_1, x_2, \ldots, x_q) + \epsilon \cdot \delta \sum_{i=1}^{m} W_i,$$

where m is the number of all constraints (p and $m - p$ are, respectively, the numbers of equations and inequalities), δ is a penalty coefficient, ϵ is -1 for maximalization and $+1$ minimalization problems, and W_i is a penalty related to the i-th constraint ($i = 1, \ldots, m$). It is also possible to reject nonfeasible offspring, however, equality constraints make this approach useless (the probability of generating a feasible offspring in the presence of equations is less then slim).

There is a large empirical evidence of the usefulness of GAs for a variety of unconstrained problems optimization (Grefenstette, 1985; Grefenstette, 1987; Schaffer, 1989; Belew, & Booker, 1991). However, the proposed penalty approach did not work well for heavily constrained problems (Michalewicz, & Janikow, 1992).

Scatter Search

Similarly to genetic algorithms, the scatter search (SS) techniques (Glover, 1977) maintain a population of potential solutions (vectors $\overline{X}_i$ are called reference points). This strategy unites preferred subsets of reference points to generate trial points (offspring) by weighted linear combinations, and selects the best members to become the source of new reference points (new population). A new twist here is the use of multicrossover (called weighted combination), where several (more than two) parents contribute in producing an offspring. In (Glover, 1989; Glover, 1990) the idea of the scatter search was extended by combining it with a tabu search—a technique which restricts the selection of new offspring (it requires memory where a historical set of individuals is kept). The structure of a scatter/tabu search algorithm is shown in Figure 2.

After initialization and evaluation, scatter/tabu search algorithm classifies (classify $P(t)$ step) population of solutions $\overline{X}_1, \ldots, \overline{X}_{pop\_size}$ into several sets. These include (1) a set of elite historical generators V consisting of some (fixed) number of best solutions through the whole process, (2) a set of tabu generators $T \subseteq V$ consisting of solutions currently excluded from considerations, (3) a set of selected historical generators V^* consisting of the best elements of $V - T$, and (4) a set of selected current generators S^* consisting of the best elements of S. The classification step (classify $P(t)$) is repeated later in the iteration phase of the algorithm.

```
procedure scatter/tabu search
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   classify P(t)
   while (not termination-condition) do
   begin
      t = t + 1
      create R(t)
      evaluate R(t)
      select P(t) from P(t − 1) and R(t)
      classify P(t)
   end
end
```

Figure 2: Scatter/tabu search

During each iteration a set $R(t)$ of trial points is created. The trial points correspond to offspring of the population $P(t)$; they are evaluated and (some of them) are incorporated into the new population (select $P(t)$ from $P(t-1)$ and $R(t)$).

For problems in the class $\mathcal{NP}$, scatter/tabu search technique takes a different approach than genetic algorithms. Instead of using penalty functions, these methods start with a feasible population of potential solutions. The set of trial points, generated during each iteration, is also feasible: a nonfeasible point is rejected at once and is not considered any further.

Simulated Annealing

The simulated annealing (SA) technique (Kirkpatrick et al., 1983) was derived from statistical mechanics for finding near globally-minimum-cost solutions to large optimization problems. It generalizes the hillclimbing methods and eliminates their main disadvantage: dependence of the solution on the starting point, and statistically promises to deliver an optimal solution. This is achieved by introducing a probability ρ of acceptance (i.e., replacement of the current point by a new point): $\rho = 1$, if the new point provides a better value of the objective function; however, $\rho > 0$, otherwise. In the latter case, the probability of acceptance ρ is a function of the values of objective function for the current point and the new point, and an additional control parameter, "temperature", T. In general, the lower temperature T is, the smaller the chances for the acceptance of a new point are. During execution of the algorithm, the temperature of the system, T, is lowered in steps. The algorithm terminates for some small value of T, for which virtually no changes are accepted anymore. The structure of the simulated annealing procedure is given in Figure 3.

```
procedure simulated annealing
begin
   t ← 0
   initialize temperature T
   select a current string X_c at random
   evaluate X_c
   repeat
      repeat
         select a new string X_n
            in the neighborhood of X_c
            by flipping a single bit of X_c
         if f(X_c) ≤ f(X_n)
            then X_c ← X_n
            else if random[0,1) < exp{(f(X_n) − f(X_c))/T}
                 then X_c ← X_n
      until (termination-condition)
      T ← g(T, t)
      t ← t + 1
   until (stop-criterion)
end
```

Figure 3: Simulated annealing

The function $random[0,1)$ returns a random number from the range $[0,1)$. The (termination-condition) checks whether 'thermal equilibrium' is reached, i.e., whether the probability distribution of the selected new strings approaches the Boltzmann distribution (Aarts, & Korst, 1989). However, in some implementations (Ackley, 1987), this repeat loop is executed just k times (k is an additional parameter of the method).

The temperature T is lowered in steps ($g(T,t) < T$ for all t). The algorithm terminates for some small value of T: the (stop-criterion) checks whether the system is 'frozen', i.e., virtually no changes are accepted anymore.

Simulated annealing technique can incorporate penalties for violated constraints or reject nonfeasible string $\overline{X}_n$. For example, the code for VFSR (very fast simulated reannealing, Ingber, 1989) rejects nonfeasible points. Also quite often the simulated annealing procedure is modified to permit adaptive changes in the ranges of the parameters to take into account new information that might make it more efficient to cut down the size of the search space.

Evolutionary Strategies

Evolution strategies (ES) are techniques developed for parameter optimization problems (Bäck et al., 1991; Schwefel, 1981). Early evolution strategies used a floating point number

representation, with mutation being the only recombination operator. They have been applied to various optimization problems with continuously changeable parameters.

The multimembered evolution strategies evolved further (Schwefel, 1981) to mature as

$(\mu + \lambda)$–ESs and (μ, λ)–ESs;

the main idea behind these strategies was to allow control parameters (like mutation variance) to self-adapt rather than changing their values by some deterministic algorithm. The structure of an evolutionary strategy algorithm is given in Figure 4.

```
procedure evolutionary strategy
begin
   t = 0
   initialize P(t)
   evaluate P(t)
   while (not termination-condition) do
   begin
      t = t + 1
      select P(t) from P(t − 1)
      recombine P(t)
      evaluate P(t)
   end
end
```

Figure 4: Evolutionary strategy

Note that this structure is the same as the one for a genetic algorithm (Figure 1) except that the order of steps

```
select P(t) from P(t − 1)
recombine P(t)
```

is reversed. The algorithm first enters the reproduction phase and later selects a new population. Of course, there are other differences hidden on the lower levels of the structure.

In the $(\mu+\lambda)$–ES, μ individuals produce λ offspring. The new (temporary) population of $(\mu+\lambda)$ individuals is reduced by a selection process again to μ individuals. On the other hand, in the (μ, λ)–ES, the μ individuals produce λ offspring ($\lambda > \mu$) and the selection process selects a new population of μ individuals from the set of λ offspring only. By doing this, the life of each individual is limited to one generation. This allows the (μ, λ)–ES to perform better on problems with an optimum moving over time, or on problems where the objective function is noisy.

The operators used in the $(\mu + \lambda)$–ESs and (μ, λ)–ESs incorporate two-level learning: their control parameter $\overline{\sigma}$ is no longer constant, nor it is changed by some deterministic

algorithm, but it is incorporated in the structure of the individuals and undergoes the evolution process. To produce an offspring, the system acts in two stages:

- select two individuals,

  $(\overline{X}^1, \overline{\sigma}^1) = ((x_1^1, \ldots, x_n^1), (\sigma_1^1, \ldots, \sigma_n^1))$ and
  $(\overline{X}^2, \overline{\sigma}^2) = ((x_1^2, \ldots, x_n^2), (\sigma_1^2, \ldots, \sigma_n^2))$,

  and apply a recombination (crossover) operator. There are two types of crossovers:

  - discrete, where the new offspring is

    $$(\overline{X}, \overline{\sigma}) = ((x_1^{q_1}, \ldots, x_n^{q_n}), (\sigma_1^{q_1}, \ldots, \sigma_n^{q_n})),$$

    where $q_i = 1$ or $q_i = 2$ (so each component comes from the first or second preselected parent),
  - intermediate, where the new offspring is

    $$(\overline{X}, \overline{\sigma}) = (((x_1^1+x_1^2)/2, \ldots, (x_n^1+x_n^2)/2), ((\sigma_1^1+\sigma_1^2)/2, \ldots, (\sigma_n^1+\sigma_n^2)/2)).$$

  Each of these operators can be applied also in a global mode, where the new pair of parents is selected for *each* component of the offspring vector.

- apply mutation to the offspring $(\overline{X}, \overline{\sigma})$ obtained; the resulting new offspring is $(\overline{X}', \overline{\sigma}')$, where

  $\overline{\sigma}' = \overline{\sigma} \cdot e^{N(0, \Delta\overline{\sigma})}$ and
  $\overline{X}' = \overline{X} + N(0, \overline{\sigma}')$,

  where $\Delta\overline{\sigma}$ is a parameter of the method.

Evolution strategies deal with constraints in a direct way. They assume a set of $m \geq 0$ inequalities,

$$h_1(\overline{X}) \geq 0, \ldots, h_m(\overline{X}) \geq 0,$$

as part of the optimization problem. If, during some iteration, an offspring does not satisfy all of these constraints, then the offspring is disqualified, i.e., it is not placed in a new population. If the rate of occurrence of such illegal offspring is high, the ESs adjust their control parameters, e.g., by decreasing the components of the vector $\overline{\sigma}$. Note that all equalities $c_i = 0$ must be replaced by a pair of inequalities $c_i \leq 0$ and $c_i \geq 0$ which may lead for some inefficiencies. One such particular case is fully discussed in Michalewicz, (1993), where we compare several systems on single test problem (balanced nonlinear transportation problem, i.e., problem with equality constraints).

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) only. The reason is that in the area of constrained optimization, the geometric shape of the set of solutions in R^q 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. For this reason, we considered first a subclass of $\mathcal{NP}$ of nonlinear programming problems, where all constraints c_i are linear (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. The convexity of the search space 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. The full description of the Genocop system is given in (Michalewicz, 1992); here we discuss briefly these two aspects of the algorithm in turn.

Let us assume we wish to optimize a function of six variables:

$$f(x_1, x_2, x_3, x_4, x_5, x_6),$$

subject to the following constraints:

$$\begin{aligned}
&2x_1 + x_2 + x_3 = 6,\\
&x_3 + x_5 - 3x_6 = 10,\\
&x_1 + 4x_4 = 3,\\
&x_2 + x_5 \leq 120,
\end{aligned}$$

$-40 \leq x_1 \leq 20$, $50 \leq x_2 \leq 75$,
$0 \leq x_3 \leq 10$, $5 \leq x_4 \leq 15$,
$0 \leq x_5 \leq 20$, $-5 \leq x_6 \leq 5$.

We can take an advantage from the presence of three independent equations and express four variables as functions of the remaining three:

$x_1 = 3 - 4x_4$,
$x_2 = -10 + 8x_4 + x_5 - 3x_6$,
$x_3 = 10 - x_5 + 3x_6$.

Thus we have reduced the original problem to the optimization problem of a function of three variables x_4, x_5, and x_6:

$$g(x_4, x_5, x_6) = f((3 - 4x_4), (-10 + 8x_4 + x_5 - 3x_6), (10 - x_5 + 3x_6), x_4, x_5, x_6),$$

subject to the following constraints (inequalities only):

$-10 + 8x_4 + 2x_5 - 3x_6 \leq 120$, (original $x_2 + x_5 \leq 120$),
$50 \leq 3 - 4x_4 \leq 20$, (original $50 \leq x_1 \leq 20$),
$-20 \leq -10 + 8x_4 + x_5 - 3x_6 \leq 75$, (original $-20 \leq x_2 \leq 75$),
$0 \leq 10 - x_5 + 3x_6 \leq 10$, (original $0 \leq x_3 \leq 10$),
$5 \leq x_4 \leq 15$,
$0 \leq x_5 \leq 20$, and
$-5 \leq x_6 \leq 5$.

These can be reduced further; for example the second and fifth inequalities can be replaced by a single one:

$5 \leq x_4 \leq 10.75$.

Such transformation completes the first step of our algorithm: elimination of equalities. The resulting search space is, of course, convex. From the convexity of the search space it follows that for each feasible point $(x_1, \ldots, x_n)$ there exists a feasible range $\langle left(k), right(k) \rangle$ of a variable x_k $(1 \leq k \leq n)$, where remaining variables x_i $(i = 1, \ldots, k-1, k+1, \ldots, n)$ are fixed. In other words, for a given $(x_1, \ldots, x_k, \ldots, x_n)$:

$y \in \langle left(k), right(k) \rangle$ iff $(x_1, \ldots, y, \ldots, x_n)$ is feasible,

where all x_i's $(i = 1, \ldots, k-1, k+1, \ldots, n)$ remain constant. Since all inequalities are linear, the ranges $\langle left(k), right(k) \rangle$ can be efficiently computed.

For example, for the feasible space defined in the previous example and a given feasible point $(x_4, x_5, x_6) = (10, 8, 2)$:

$left(1) = 7.25$, $right(1) = 10.375$,
$left(2) = 6$, $right(2) = 11$,
$left(3) = 1$, $right(3) = 2.666$,

($left(1)$ and $right(1)$ are ranges of the first component of the vector $(10, 8, 2)$, i.e., of the variable x_4, etc.). It means that the first component of the vector $(10, 8, 2)$ can vary from 7.25 to 10.375 (while $x_5 = 8$ and $x_6 = 2$ remain constant), the second component of this vector can vary from 6 to 11 (while $x_4 = 10$ and $x_6 = 2$ remain constant), and the third component of this vector can vary from 1 to 2.666 (while $x_4 = 10$ and $x_5 = 8$ remain constant).

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).

There are several operators in the Genocop system which proved to be useful on many test problems. We discuss them in turn.

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

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. Since this operator aims specifically at this condition, it can prove extremely useful in such instances.

Non-uniform mutation

This (unary) operator 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

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

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. 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 is responsible for the fine tuning capabilities of the system. The significance of this operator is visible in the late stages of the evolution process.

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).

The operator explores points in the search space which belong to line connecting its parents. Like non-uniform mutation, the operator plays a significant role in fine local tuning.

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.

The operator is responsible for sampling the search space; two parents may produce offspring in a 'new' part of the search space.

Heuristic crossover

This operator was suggested in 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 operator explores a point in the search space by moving outward from the better parent along the line connecting the two parents. Again, the operator gains its importance at the later stages of the evolution process.

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 three 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), and $b = 2$ (coefficient for non-uniform mutation).

For each test case we run the Genocop ten times. For all problems, the number of generations T was 500 or 1000. Three 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:

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

where

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

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

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

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

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

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

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.

It is interesting to note that while the best point found improves the best known value for the objective function by less than 0.06, the point is located in a very different area of the search space than the best point reported in Hock, & Schittkowski (1981).

Test Case #2

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

$$2x_1 + 2x_2 + y_6 + y_7 \leq 10,$$
$$2x_1 + 2x_3 + y_6 + y_8 \leq 10,$$
$$2x_2 + 2x_3 + y_7 + y_8 \leq 10,$$
$$-8x_1 + y_6 \leq 0,$$
$$-8x_2 + y_7 \leq 0,$$

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

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

The Genocop system 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 24 sec of CPU time.

Test Case #3

The third example demonstrates the power of the Genocop—we consider a test case where the objective function is discontinuous. The problem was constructed from three separate problems taken from Hock, & Schittkowski (1981) in the following way:

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

subject to:

$$
\begin{aligned}
&x_1/\sqrt{3} - x_2 \geq 0,\\
&-x_1 - \sqrt{3}x_2 + 6 \geq 0,\\
&0 \leq x_1 \leq 6, \text{ and } x_2 \geq 0.
\end{aligned}
$$

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 $\overline{X}_3^* = (4,0)$.

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

Summary

As discussed in the previous section, Genocop worked very well for many experimental test problems with linear constraints; these included discontinuous functions. However, it is not clear how we can generalize Genocop to handle nonlinear constraints (i.e., problems in $\mathcal{NP}$ class). Some sets of nonlinear constraints can still yield a convex search spaces—the property important for many operators (all mutations, arithmetical crossover). However, even in this case the process of finding the ranges $left(k)$ and $right(k)$ might be hard computationally. Another possibility would be to cover the search space by (not necessarily disjoint) family of convex subspaces and run Genocop on each of these. Again, the method would still have many computational problems.

For the above reasons, we looked at traditional optimization methods for further inspiration. One particular method (described in the next section) was selected for 'mating' with the Genocop system. The result was Genocop II, which is discussed later in the paper; the first results of the system are more than encouraging.

A TRADITIONAL CALCULUS BASED METHOD

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 $\mathcal{NP}$ 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).

In this section, we discuss briefly one of these approaches, the sequential quadratic penalty function method.

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

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

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 $\mathcal{NP}$ and $\mathcal{NP}'$ were identical in the limit as $r \longrightarrow 0$. It was thought that $\mathcal{NP}'$ 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. 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 (1988), Attia (1985) 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 (Attia, 1985).

GENOCOP II

The technique discussed in the previous section, 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 5. 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:

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 5: The structure of Genocop II

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

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

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.

TEST CASES

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 include quadratic, nonlinear, and discontinuous functions with several nonlinear constraints.

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), $\delta = 0.01$ (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.

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

Test Case #1

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

$$\text{minimize } f(\overline{X}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

subject to nonlinear constraints:

$$\begin{aligned} c_1 &: \; x_1 + x_2^2 \geq 0, \\ c_2 &: \; x_1^2 + x_2 \geq 0, \end{aligned}$$

and bounds:

$$-0.5 \leq x_1 \leq 0.5, \text{ and } x_2 \leq 1.0.$$

The known global solution is $\overline{X}^* = (0.5, 0.25)$, and $f(\overline{X}^*) = 0.25$. The starting feasible point is $\overline{X}_0 = (0, 0)$.

The Genocop II found the exact optimum in one iteration, since none of the nonlinear constraints are active at optimum.

Test Case #2

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

$$\text{minimize } f(x, y) = -x - y,$$

subject to nonlinear constraints:

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

and bounds:

$$0 \leq x \leq 3 \text{ and } 0 \leq y \leq 4.$$

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 #2; for iteration 0 the best point is the starting point

Test Case #3

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

Figure 6: A feasible space for test case #3.

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

Test Case #4

The problem (taken from Betts, 1977) is

$$\text{minimize } f(x_1, x_2) = 0.01x_1^2 + x_2^2,$$

subject to nonlinear constraints:

$$\begin{aligned} c_1 &: \quad x_1 x_2 - 25 \geq 0, \\ c_2 &: \quad x_1^2 + x_2^2 - 25 \geq 0, \end{aligned}$$

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 #3; for iteration 0 the best point is the starting point

and bounds:

$$2 \leq x_1 \leq 50 \text{ and } 0 \leq x_2 \leq 50.$$

The global solution is $\overline{X}^* = (\sqrt{250}, \sqrt{2.5}) = (15.811388, 1.581139,$ and $f(\overline{X}^*) = 5.0$. The starting (not feasible) point is $\overline{X}_0 = (2, 2)$.

It is interesting to note that the standard cooling scheme (i.e., $g(\tau, t) = 10^{-1} \cdot g(\tau, t-1)$) did not produce good results, however, when the cooling process was slowed down (i.e., $g(\tau, 0) = 5$ and $g(\tau, t) = 2^{-1} \cdot g(\tau, t-1)$), the system approached optimum easily (table 3).

Iteration number	The best point	Active constraints
0	(2,2)	c_1, c_2
1	(3.884181, 3.854748)	c_1
2	(15.805878, 1.581057)	c_1
3	(15.811537, 1.580996)	c_1

Table 3: Progress of Genocop II on test case #4; for iteration 0 the best point is the starting point

Test Case #5

The final test problem (taken from Hock, & Schittkowski, 1981) is

$$\text{minimize } f(\overline{X}) = (x_1 - 2)^2 + (x_2 - 1)^2,$$

subject to a nonlinear constraint:

$$c_1 \ : \ -x_1^2 + x_2 \geq 0,$$

and a linear constraint:

$$x_1 + x_2 \leq 2.$$

The global solution is $\overline{X}^* = (1, 1)$ and $f(\overline{X}^*) = 1$. The starting (feasible) point is $\overline{X}_0 = (0, 0)$.

The Genocop II approached the optimum very closely at the 6th iteration. The progress of the system is reported in the table 4.

Iteration number	The best point	Active constraints
0	(0, 0)	c_1
1	(1.496072, 0.503928)	c_1
2	(1.020873, 0.979127)	c_1
3	(1.013524, 0.986476)	c_1
4	(1.002243, 0.997757)	c_1
5	(1.000217, 0.999442)	c_1
6	(1.000029, 0.999971)	c_1

Table 4: Progress of Genocop II on test case #5; for iteration 0 the best point is the starting point

CONCLUSIONS

There are several interesting points connected with the method of Genocop II. First of all, as any other GA-based method, it does not require any implicit (or explicit) calculations of any gradient or Hessian matrix of the objective function and constraints. Consequently, the method does not suffer from the ill-conditioned Hessian problem usually associated with some calculus based methods.

It should be noted that any genetic algorithm can be used in place of Genocop for the inner loop for Genocop II. In such a case all constraints (linear and nonlinear) should be considered for placement in the set of active constraints A (the elements of L should be distributed between N_e and N_i). However, such method is much slower and less effective: for efficiency reasons, it is much better to process linear constraints separately (as it is done in Genocop).

Further research includes several activities. We plan to experiment with Genocop, which is the key element in success of Genocop II by (1) checking additional operators; (2) modifying the initialization procedure; (3) introducing integer and Boolean variables; and (4) introducing adaptive frequencies for operators to enhance the system even further. It is clear that different operators have different significance at different stages of the evolution process. Some researchers already looked at this possibility (Schaffer et al., 1989; Fogarty, 1989; Shaefer, 1987); also, adaptive parameters were incorporated very early in evolutionary strategies (Bäck et al., 1991).

For Genocop II, we plan to complete its full implementation (the test cases reported in this paper were run 'manually' executing Genocop several times with different temperatures, i.e., different pressures on active constraints). Then we plan to experiment further with problems of high dimensionality, different cooling schemes ($g(\tau, t)$), and different values of a parameter δ, which decides whether a constraint is active or not.

The Genocop system is in the public domain and is available through ftp from uncc-sun.uncc.edu (or ftp 152.15.10.88), anonymous, directory coe/evol, file genocop.tar.Z.

References

Aarts, E.H.L. and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*, Wiley, Chichester, UK.

Ackley, D.H. (1987). An Empirical Study of Bit Vector Function Optimization. in L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Los Altos, CA, pp.170–204.

Attia, N.F. (1985). *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. In R. Belew and L. Booker (Eds.) *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.

Betts, J.T. (1977). An Accelerated Multiplier Method for Nonlinear Programming. *Journal of Optimization Theory and Applications, 21*, pp.137–174.

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

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. 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.*

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. (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.

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.

Fogarty, T.C., (1989). Varying the Probability of Mutation in the Genetic Algorithm. In J. Schaffer (Ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, (pp.104–109). Morgan Kaufmann Publishers, Los Altos, CA.

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

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.

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

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.

Ingber, L. (1989). Very Fast Simulated Re-annealing. *Mathematical Computer Modelling, 12*, pp.967–973.

Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science, 220*, pp.671–680.

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*, pp.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*, pp.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.

Murray, W. (1969). An Algorithm for Constrained Minimization. In R. Fletcher (Ed.) *Optimization*, 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., Caruana, R., Eshelman, L., and Das, R. (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In J. Schaffer (Ed.) *Proceedings of the Third International Conference on Genetic Algorithms*, (pp.51–60). Morgan Kaufmann Publishers, Los Altos, CA.

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

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

Shaefer, C.G. (1987). The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique. In J.J. Grefenstette (Ed.) *Proceedings of the Second International Conference on Genetic Algorithms*, (pp.50–55). Lawrence Erlbaum Associates, Hillsdale, NJ.

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, 1991, pp.205–218.