Evolutionary Algorithms for Constrained Parameter Optimization Problems

Zbigniew Michalewicz* and Marc Schoenauer†

Abstract

Evolutionary computation techniques have received a lot of attention regarding their potential as optimization techniques for complex numerical functions. However, they have not produced a significant breakthrough in the area of nonlinear programming due to the fact that they have not addressed the issue of constraints in a systematic way. Only recently several methods have been proposed for handling nonlinear constraints by evolutionary algorithms for numerical optimization problems; however, these methods have several drawbacks and the experimental results on many test cases have been disappointing.

In this paper we (1) discuss difficulties connected with solving the general nonlinear programming problem, (2) survey several approaches which have emerged in the evolutionary computation community, and (3) provide a set of eleven interesting test cases, which may serve as a handy reference for future methods.

1 Introduction

The general nonlinear programming (NLP) problem is to find $\vec{x}$ so as to

optimize $f(\vec{x})$, $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$,

where $\vec{x} \in \mathcal{F} \subseteq \mathcal{S}$. The *objective function* f is defined on the *search space* $\mathcal{S} \subseteq \mathbb{R}^n$ and the set $\mathcal{F} \subseteq \mathcal{S}$ defines the *feasible region*. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds):

*Department of Computer Science, University of North Carolina, Charlotte, NC 28223, USA; e-mail: *zbyszek@uncc.edu* and Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland; e-mail: *zbyszek@ipipan.waw.pl.*

†CMAP – URA CNRS 756, Ecole Polytechnique, Palaiseau 91128, France, e-mail: *marc.schoenauer@polytechnique.fr.*

$$l(i) \leq x_i \leq u(i), \quad 1 \leq i \leq n,$$

whereas the feasible region $\mathcal{F} \subseteq \mathcal{S}$ is defined by a set of m additional constraints ($m \geq 0$):

$$g_j(\vec{x}) \leq 0, \text{ for } j = 1, \ldots, q, \text{ and } h_j(\vec{x}) = 0, \text{ for } j = q + 1, \ldots, m.$$

At any point $\vec{x} \in \mathcal{F}$, the constraints g_k that satisfy $g_k(\vec{x}) = 0$ are called the *active* constraints at $\vec{x}$. By extension, equality constraints h_j are also called active at all points of $\mathcal{S}$.

The NLP problem, in general, is intractable. If the objective function f and functions g_j's and h_j's expressing constraints are arbitrary, then there is little choice apart from methods based on exhaustive search. This is the reason for identifying several special cases of the NLP. One of the studied cases is the one where all functions g_j's and h_j's are linear; such problems are called *linearly constrained optimization* problems. If, additionally, the objective function f is polynomial at most quadratic, the problem is called *quadratic programming* problem. The best known special case of quadratic programming is where the objective function f is linear as well; this problem is called *linear programming* problem. There is also an important special case, called *unconstrained optimization*, where there are no constraints at all, i.e., $m = 0$ and $\mathcal{F} = \mathcal{S}$.

One of the main difficulties in solving NLP is the problem of local optima; a feasible point $\vec{x_0} \in \mathcal{F}$ is a local optimum for the objective function f iff there exist $\epsilon > 0$ such that for all $\vec{x}$ in the ϵ-neighborhood of $\vec{x_0}$ in $\mathcal{F}$, $f(\vec{x}) > f(\vec{x_0})$.[1] Local optima just satisfy the mathematical requirements on the derivatives of the functions and many optimization techniques based on gradient methods aim at local optimization only.

In general, it is impossible to develop a deterministic method for the NLP in the global optimization category, which would be better than the exhaustive search. As stated by Gregory (1995):

> "It's unrealistic to expect to find one general NLP code that's going to work for every kind of nonlinear model. Instead, you should try to select a code that fits the problem you are solving. If your problem doesn't fit in any category except 'general', or if you insist on a globally optimal solution (except when there is no chance of encountering multiple local optima), you should be prepared to have to use a method that boils down to exhaustive search, i.e., you have an intractable problem."

Evolutionary algorithms are global methods, which aim at complex objective functions (e.g., non differentiable or discontinuous). However, most research on applications of evolutionary computation techniques to nonlinear programming problems has been concerned with complex objective functions but no constraints ($\mathcal{F} = \mathcal{S}$). Many of the test functions used by various researchers during the last 20 years did not include any constraints (apart from specified domains

[1] For minimization problems.

of variables); this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases proposed since then (Eshelman and Schaffer, 1993; Fogel and Stayton, 1994; Wright, 1991). Only recently several approaches have emerged which aim at general nonlinear programming problems.

It seems worthwhile to extend evolutionary techniques by some constraint-handling methods. It is also important to gain some experimental feedback on merits and drawbacks of these methods. The ultimate goal is to create a useful evolutionary toolbox for the general NLP problem!

The paper is organized as follows. The following section surveys briefly several operators which has been developed for evolutionary algorithms with floating-point representation. Section 3 presents several constraint-handling techniques for numerical optimization problems which have emerged in evolutionary computation techniques over the years. Most methods are enlightened by an example of how they behave on a specific NLP problem. Section 4 summarizes all methods and indicates some directions for future research.

2 Numerical Optimization and Operators

Most evolutionary algorithms for numerical optimization problems use vectors of floating point numbers for their chromosomal representations. For such representations, many operators have been proposed during the last 30 years. Several constraint-handling methods are discussed in the following section; some of these methods (e.g., Genocop, section 3.1.1) require a set of specific operators. For that reason, we discuss briefly these operators in turn.

2.1 Mutation operators

The most popular mutation operator is *Gaussian mutation*, which modified all components of the solution vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ by adding a random noise:

$$\vec{x}^{t+1} = \vec{x}^t + N(0, \vec{\sigma}),$$

where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviations $\vec{\sigma}$. Such a mutation is used in evolution strategies (Bäck et al., 1991) and evolutionary programming (Fogel, 1995). (One of the historical differences between these techniques lies in adjusting vector of standard deviations $\vec{\sigma}$).

Other types of mutations include *non-uniform mutation*, where

$$x_k^{t+1} = \begin{cases} x_k^t + \triangle(t, r(k) - x_k) & \text{if a random binary digit is 0} \\ x_k^t - \triangle(t, x_k - l(k)) & \text{if a random binary digit is 1} \end{cases}$$

for $k = 1, \ldots, n$. 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. In experiments reported by Michalewicz et al. (1994), the following function was used:

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

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.

It is also possible to experiment with a *uniform mutation*, which changes a single component of the solution vector; e.g., if $\vec{x}^t = (x_1, \ldots, x_k, \ldots, x_n)$, then $\vec{x}^{t+1} = (x_1, \ldots, x'_k, \ldots, x_n)$, where x'_k is a random value (uniform probability distribution) from the domain of variable x_k. A special case of uniform mutation is *boundary mutation*, where x'_k is either left of right boundary of the domain of x_k.

2.2 Crossover operators

There are several interesting types of crossover operators. The first family is directly designed from the well-known bitstring crossover operators (known also as "discrete recombination" operators): 1-point, 2-point and uniform crossover, except that the "genes" exchanged between the two parents are real-valued coordinates. For instance, *uniform crossover*, works as follows. Two parents, $\vec{x} = \langle x_1, \ldots, x_n \rangle$ and $\vec{y} = \langle y_1, \ldots, y_n \rangle$, produce an offspring, $\vec{z} = \langle z_1, \ldots, z_n \rangle$, where $z_i = x_i$ or $z_i = y_i$ with equal probability for all $i = 1, \ldots, n$ (the second offspring is created by reversing decisions for all components).

Some multi-parents versions of these operators have also been designed. Mühlenbein and Voigt (1995) investigated the properties of a recombination operator called *gene pool recombination* (default recombination mechanism in evolution strategies (Schwefel, 1981), where the genes are randomly picked from the gene pool defined by the selected parents. A similar multi-parent crossover was also investigated also by Eiben et al. (1994) in the context of combinatorial optimization.

Another possibility is *arithmetical crossover*. Here two vectors, $\vec{x}$ and $\vec{y}$ produce two offspring, $\vec{p}$ and $\vec{q}$, which are linear combinations of their parents, i.e.,

$$\vec{p} = a \cdot \vec{x} + (1 - a) \cdot \vec{y} \quad \text{and} \quad \vec{q} = (1 - a) \cdot \vec{x} + a \cdot \vec{y}.$$

This operator has also been called *guaranteed average crossover* (Davis, 1989) (when $a = 1/2$), and *intermediate crossover* (Schwefel, 1981).

It is possible to generalize arithmetical crossover into multi-parent operator; parents $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_r}$ may produce a family of offspring:

$$\vec{y} = a_1\vec{x_1} + a_2\vec{x_2} + \ldots + a_r\vec{x_r},$$

for any choice of $(a_i) \in [0,1]^r$ such that $a_1 + a_2 + \ldots + a_r = 1$.

The *simplex crossover* operator proposed by Renders and Bersini (1994) for numerical optimization problems can also be viewed as another generalization of arithmetical crossover; this crossover operator involves computing the centroid of group of parents and moving from the worst individual beyond the centroid point. More precisely, the operator selects $k > 2$ parents (set J), determines the best and the worst individual within the selected group ($\vec{b}$ and $\vec{w}$, respectively), computes the centroid $\vec{c}$ of the selected group with removed worst individual:

$$\vec{c} = \sum_{\vec{x_i} \in J - \vec{w}} \vec{x_i}/(k-1),$$

and computes the 'reflected point' $\vec{y}$ (i.e., the offspring) obtained from the worst one:

$$\vec{y} = \vec{c} + (\vec{c} - \vec{w}).$$

(This operator includes also a few extra options, which are based on the outcome of comparisons between the values of $f(\vec{y})$, $f(\vec{w})$, and $f(\vec{b})$; for details, see Renders and Bersini, 1994).

Some of these ideas were embodied earlier in the evolutionary procedure called *scatter search* (Glover, 1977). The process generates initial populations by screening good solutions produced by heuristics. The points used as parents are then joined by linear combinations with context-dependent weights, where such combinations may apply to multiple parents simultaneously. The linear combination operators are further modified by adaptive rounding processes to handle components required to take discrete values. (The vectors operated on may contain both real and integer components, as opposed to strictly binary components.) Finally, preferred outcomes are selected and again subjected to heuristics, whereupon the process repeats. The approach has been found useful for mixed integer and combinatorial optimization problems. (For background and recent developments see, e.g., (Glover, 1994)).

An interesting variation along this line is the *heuristic crossover* operator proposed by Wright (1991); this crossover uses values of the objective function in determining the direction of the search, and it produces only one offspring. The operator generates a single offspring $\vec{z}$ from two parents, $\vec{x}$ and $\vec{y}$ according to the following rule:

$$\vec{z} = r \cdot (\vec{x} - \vec{y}) + \vec{x},$$

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

Recently, some results of experiments with a *geometrical crossover* operator were reported (Michalewicz et al., 1996). Geometrical crossover can only be applied to problems where each variable takes nonnegative values only, i.e., $0 \leq l(i) \leq x_i \leq u(i)$, so its use is quite restricted in comparison with other types of crossovers (e.g., arithmetical crossover). However, for many engineering problems, all problem variables are positive; moreover, it is always possible to replace variable $x_i \in \langle l(i), u(i) \rangle$ which can take negative values (i.e., where $l(i) < 0$) with a new variable $y_i = x_i - l(i)$.

The geometrical crossover operator takes two parents and produces a single offspring; for parents $\vec{x}$ and $\vec{y}$ the offspring is $\vec{z} = \langle \sqrt{x_1 y_1}, \ldots, \sqrt{x_n y_n} \rangle$. Also it is possible to generalize this operator to include several parents:

$$\vec{y} = \langle (x_{1,1})^{\alpha_1} (x_{1,2})^{\alpha_2} \ldots (x_{1,r})^{\alpha_r}, \ldots, (x_{n,1})^{\alpha_1} (x_{n,2})^{\alpha_2} \ldots (x_{n,r})^{\alpha_r} \rangle,$$

where $\alpha_1 + \ldots + \alpha_r = 1$ and $0 \leq \alpha_i \leq 1$ for all $1 \leq i \leq r$.

3 Constraint-handling methods

During the last few years several methods were proposed for handling constraints by genetic algorithms for parameter optimization problems; this section discusses them in turn. It is useful to group these methods into four categories:

1. methods based on preserving feasibility of solutions,
2. methods based on penalty functions,
3. methods which make a clear distinction between feasible and infeasible solutions, and
4. other hybrid methods.

The subsequent four subsections 3.1–3.4 describe methods of these categories, respectively. Also, for most methods we provide with a test case and the results of the method on this test case. We do not differentiate between minimization and maximization problems (we present test cases of both categories); note that

min $\{f(\vec{x})\}$ is equivalent to max $\{-f(\vec{x})\}$, for $\vec{x} \in \mathcal{F}$.

3.1 Methods based on preserving feasibility of solutions

There are two methods which fall into this category; we discuss them in turn.

3.1.1 Use of specialized operators

The Genocop (for GEnetic algorithm for Numerical Optimization of COnstrained Problems) system was developed by Michalewicz and Janikow (1991). The idea behind the system is based on specialized operators which transform feasible individuals into feasible individuals, i.e., operators, which are closed on the feasible part $\mathcal{F}$ of the search space. The method assumes linear constraints only and a feasible starting point (or feasible initial population). Linear equations are used to eliminate some variables; they are replaced as a linear combination of remaining variables. Linear inequalities are updated accordingly. A closed set of operators maintains feasibility of solutions. For example, when a particular component x_i of a solution vector $\vec{x}$ is mutated, the system determines its current domain $dom(x_i)$ (which is a function of linear constraints and remaining values of the solution vector $\vec{x}$) and the new value of x_i is taken from this domain (either with flat probability distribution for uniform mutation, or other probability distributions for non-uniform and boundary mutations). In any case the offspring solution vector is always feasible. Similarly, arithmetic crossover, $a\vec{x} + (1-a)\vec{y}$, of two feasible solution vectors $\vec{x}$ and $\vec{y}$ yields always a feasible solution (for $0 \leq a \leq 1$) in convex search spaces (the system assumes linear constraints only which imply convexity of the feasible search space $\mathcal{F}$). The heuristic crossover described in section 2.2 generates a single offspring $\vec{z}$ (note that an additional feasibility check is required in such a case) from two parents.

Genocop proved its usefulness for linearly constrained optimization problems; it gave surprisingly good performance on many test functions (Michalewicz et al., 1994; Michalewicz, 1996). For example, let us consider the following problem (Floudas and Pardalos, 1987) to minimize a function:

$$G1(\vec{x}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i,$$

subject to the following constraints:

$$\begin{array}{lll} 2x_1 + 2x_2 + x_{10} + x_{11} \leq 10, & 2x_1 + 2x_3 + x_{10} + x_{12} \leq 10, & 2x_2 + 2x_3 + x_{11} + x_{12} \leq 10, \\ -8x_1 + x_{10} \leq 0, & -8x_2 + x_{11} \leq 0, & -8x_3 + x_{12} \leq 0, \\ -2x_4 - x_5 + x_{10} \leq 0, & -2x_6 - x_7 + x_{11} \leq 0, & -2x_8 - x_9 + x_{12} \leq 0, \end{array}$$

and bounds $0 \leq x_i \leq 1$, $i = 1, \ldots, 9$, $0 \leq x_i \leq 100$, $i = 10, 11, 12$, $0 \leq x_{13} \leq 1$.

The problem has 13 variables and 9 linear constraints; the function $G1$ is quadratic with its global minimum at

$$\vec{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1),$$

where $G1(\vec{x}^*) = -15$. Six (out of nine) constraints are active at the global optimum (all except the following three: $-8x_1 + x_{10} \leq 0$, $-8x_2 + x_{11} \leq 0$, $-8x_3 + x_{12} \leq 0$).

Genocop requires less than 1,000 generations to arrive at the global solution.

Note that the linear constraints imply convexity of the feasible search space, which usually require smaller effort of the search process than non convex ones, where the feasible parts of the search space can be disjoint and irregular. Note also, that the method can be generalized to handle nonlinear constraints provided that the resulting feasible search space $\mathcal{F}$ is convex. But the weakness of the method lies in its inability to deal with non convex search spaces (i.e, to deal with nonlinear constraints in general).

3.1.2 Searching the boundary of feasible region

One of the main reasons that lie behind difficulties in locating the global solution is the inability of evolutionary systems to search precisely the boundary area between feasible and infeasible regions of the search space. This ability can be quite important in the case of optimization problems with nonlinear equality constraints or with active nonlinear constraints at the target optimum.

Some other heuristic methods recognized the need for searching areas close to the boundary of the feasible region. For example, one of the most recently developed approach for constrained optimization is strategic oscillation. Strategic oscillation was originally proposed in accompaniment with the strategy of scatter search (Glover, 1977), and more recently has been applied to a variety of problem settings in combinatorial and nonlinear optimization (see, for example, the review of Glover and Kochenberger (1995)). The approach is based on identifying a critical level, which represents a boundary between feasibility and infeasibility. The basic strategy is to approach and cross the feasibility boundary, by a design that is implemented either by adaptive penalties and inducements (which are progressively relaxed or tightened according to whether the current direction of search is to move deeper into a particular region or to move back toward the boundary) or by simply employing modified gradients or sub-gradients to progress in the desired direction.

Evolutionary computation techniques have a huge potential in incorporating specialized operators which search the boundary of feasible and infeasible regions in an efficient way. This might be quite significant: it is a common situation for many constrained optimization problems that some constraints are active at the target global optimum. This optimum thus lies on the boundary of the feasible space. One the other hand, it is commonly acknowledged that restricting the size of the search space in evolutionary algorithms (as in most search algorithms) is generally beneficial. Hence, it seems natural in the context of constrained optimization to restrict the search of the solution to the boundary of the feasible part of the space. We provide two examples of a such approach; for more details the reader is referred to (Schoenauer and Michalewicz, 1996).

An interesting constrained numerical optimization test case emerged recently; the problem (Keane, 1994) is to maximize a function:

$$G2(\vec{x}) = \left|\frac{\sum_{i=1}^n cos^4(x_i) - 2\prod_{i=1}^n cos^2(x_i)}{\sqrt{\sum_{i=1}^n ix_i^2}}\right|,$$

subject to

$$\prod_{i=1}^n x_i \geq 0.75 \ , \sum_{i=1}^n x_i \leq 7.5n \ , \text{and bounds } \ 0 \leq x_i \leq 10 \ \text{ for } \ 1 \leq i \leq n.$$

Function $G2$ is nonlinear and its global maximum is unknown, lying somewhere near the origin. The problem has one nonlinear constraint and one linear constraint; the latter one is inactive around the origin and will be forgotten in the following.

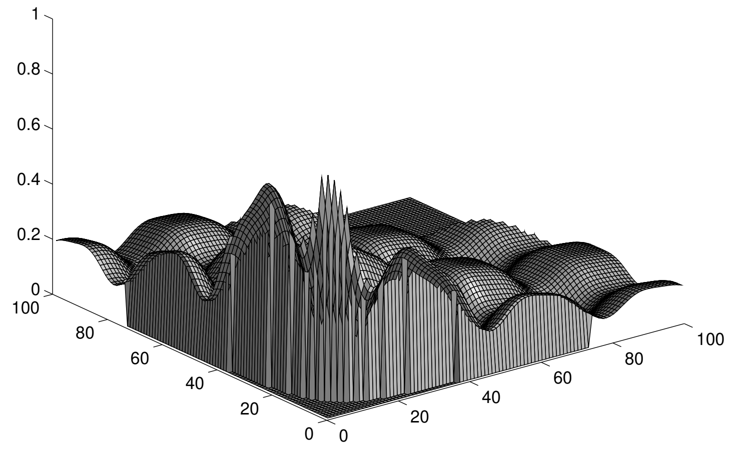

Figure 1: The graph of function $G2$ for $n = 2$. Infeasible solutions were assigned value zero

Some potential difficulties of solving this test case are illustrated in figure 1, where infeasible points were assigned a value of zero. The interesting boundary between feasible and infeasible regions is defined by the equation $\Pi x_i = 0.75$. It is a difficult problem, on which no standard method (be it deterministic or evolutionary) gave satisfactory results. As Keane (1994) noted:

> "I am currently using a parallel GA with 12 bit binary encoding, crossover, inversion, mutation, niche forming and a modified Fiacco-McCormick constraint penalty function to tackle this. For $n = 20$ I get values like 0.76 after 20,000 evaluations."

This test case was the first one on which the idea of searching only the boundary was used (Michalewicz et al., 1996): due to the simple analytical formulation of the constraint, *ad hoc* specific initialization procedure and operators could be designed:

- **Initialization**: Randomly choose a positive variable for x_i, and use its inverse as a variable for x_{i+1}. The last variable is either 0.75 (when n is odd), or is multiplied by 0.75 (if n is even), so that the point lies on the boundary surface.
- **Crossover**: The *geometrical crossover* is defined by

  $$(x_i)(y_i) \rightarrow (x_i^{\alpha} y_i^{1-\alpha}), \text{ with } \alpha \text{ randomly chosen in } [0, 1]$$

  Figure 2 illustrates the possible offspring from two parents for all values of α.
- **Mutation**: Pick two variables randomly, multiply one by a random factor q and the other by $\frac{1}{q}$ (restrict q to respect the bounds on the variables).

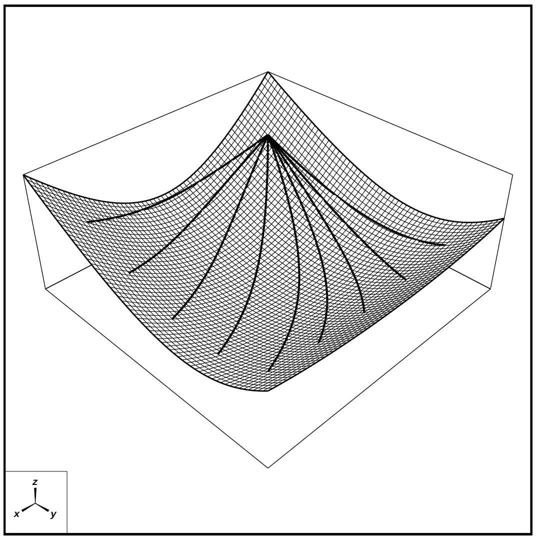

Figure 2: The geometrical crossover on the hyperboloid around its center: when parameter α goes from 0 to 1, the offspring of two parents defines the line joining them on the plot.

The simple evolutionary algorithm described above gave outstanding results. For the case $n = 20$ the system reached the value of 0.80 in less than 4,000 generations (with population size of 30, probability of crossover $p_c = 1.0$, and probability of mutation $p_m = 0.06$) in all runs. The best value found (namely 0.803553) was better than the best values of any method discussed earlier, whereas the worst value found was 0.802964. Similarly, for $n = 50$, all results (in 30,000 generations) were better than 0.83 (with the best of 0.8331937).

It was interesting to note the importance of geometrical crossover. With fixed population size (kept constant at 30), the higher values of probability of crossover p_c, the better results of the system were observed. Similarly, the best mutation rates were relatively low ($p_m \approx 0.06$).

Since evolutionary methods for searching boundaries of feasible region constitute a new development in constrained optimization, let us illustrate this approach on another test case. The test problem was constructed for the case of sphere (Michalewicz et al., 1996; Schoenauer and Michalewicz, 1996); the task is to maximize

$$G3(\vec{x}) = (\sqrt{n})^n \cdot \prod_{i=1}^{n} x_i,$$

where

$$\sum_{i=1}^{n} x_i^2 = 1 \quad \text{and} \quad 0 \le x_i \le 1 \quad \text{for} \quad 1 \le i \le n.$$

The function $G3$ has a global solution at $(x_1, \ldots, x_n) = (\frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$ and the value of the function in this point is 1. The evolutionary algorithm uses the following components:

- **Initialization**: Randomly generate n variables y_i, calculate $s = \sum_{i=1}^{n} y_i^2$, and initialize an individual (x_i) by $x = y_i/s$ for $i \in [1, n]$.

- **Crossover**: The *sphere crossover* produces one offspring (z_i) from two parents (x_i) and (y_i) by:

$$z_i = \sqrt{\alpha x_i^2 + (1-\alpha) y_i^2} \;\; i \in [1, n], \text{ with } \alpha \text{ randomly chosen in } [0, 1].$$

- **Mutation**: Similarly, the problem-specific mutation transforms (x_i) by selecting two indices $i \neq j$ and a random number p in $\langle 0, 1 \rangle$, and setting:

$$x_i \rightarrow p \cdot x_i \text{ and } x_j \rightarrow q \cdot x_j, \text{where } q = \sqrt{(\frac{x_i}{x_j})^2 (1 - p^2) + 1}.$$

The simple evolutionary algorithm described above gave very good results. For the case $n = 20$ the system reached the value of 0.99 in less than 6,000 generations (with population size of 30, probability of crossover $p_c = 1.0$, and probability of mutation $p_m = 0.06$) in all runs. The best value found in 10,000 generations was 0.999866.

3.2 Methods based on penalty functions

In Evolutionary Computation as in many other fields of optimization, most of the constraint-handling methods are based on the concept of (exterior) penalty functions, which penalize infeasible solutions, i.e. try to solve an unconstrained problem (on $\mathcal{F}$) using the modified fitness function:

$$eval(\vec{x}) = \begin{cases} f(\vec{x}), & if\ \vec{x} \in \mathcal{F} \\ f(\vec{x}) + penalty(\vec{x}), & otherwise, \end{cases}$$

where $penalty(\vec{x})$ is zero, if no violation occurs, and is positive, otherwise. Usually, the *penalty* function is based on the distance of a solution from the feasible region $\mathcal{F}$, or on the effort to "repair" the solution, i.e., to force it into $\mathcal{F}$. The former case is the most popular one; in many methods a set of functions f_j $(1 \leq j \leq m)$ is used to construct the penalty, where the function f_j measures the violation of the j-th constraint in the following way:

$$f_j(\vec{x}) = \begin{cases} \max\{0, g_j(\vec{x})\}, & if\ 1 \leq j \leq q \\ |h_j(\vec{x})|, & if\ q+1 \leq j \leq m. \end{cases}$$

However, these methods differ in many important details, how the penalty function is designed and applied to infeasible solutions. In the following subsections we discuss all known methods in turn.

3.2.1 Method of static penalties

The method was proposed by Homaifar, Lai, and Qi (1994); it assumes that for every constraint we establish a family of intervals which determine appropriate penalty coefficient. It works as follows:

- for each constraint, create several (ℓ) levels of violation,
- for each level of violation and for each constraint, create a penalty coefficient R_{ij} ($i = 1, 2, \ldots, \ell$, $j = 1, 2, \ldots, m$); higher levels of violation require larger values of this coefficient.
- start with a random population of individuals (feasible or infeasible),
- evolve the population; evaluate individuals using the following formula

  $eval(\vec{x}) = f(\vec{x}) + \sum_{j=1}^{m} R_{ij} f_j^2(\vec{x})$.

The weakness of the method is in the number of parameters. For m constraints the method requires $m(2\ell + 1)$ parameters in total: m parameters to establish number of intervals for each constraint, ℓ parameters for each constraint, defining the boundaries of the intervals (levels of violation), and ℓ parameters for each constraint representing the penalty coefficients R_{ij}. In particular, for $m = 5$ constraints and $\ell = 4$ levels of violation, we need to set 45 parameters! Clearly, the results are parameter dependent. It is quite likely that for a given problem there exist one optimal set of parameters for which the system returns feasible near-optimum solution, however, it might be quite hard to find it.

A limited set of experiments reported by Michalewicz (1995a) indicates that the method can provide good results if violation levels and penalty coefficients R_{ij} are tuned to the problem. For example, Homaifar et al. (1994) experimented with the following problem (Himmelblau, 1992). Minimize a function of 5 variables:

$$G4(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141,$$

subject to three double inequalities:

$$\begin{array}{l} 0 \leq 85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5 \leq 92 \\ 90 \leq 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \leq 110 \\ 20 \leq 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 25, \end{array}$$

and bounds:

$$78 \leq x_1 \leq 102,\ 33 \leq x_2 \leq 45,\ 27 \leq x_i \leq 45 \text{ for } i = 3, 4, 5.$$

The best solution obtained in 10 runs (Homaifar et al., 1994) was

$$\vec{x} = (80.49, 35.07, 32.05, 40.33, 33.34)$$

with $G4(\vec{x}) = -30005.7$, whereas the optimum solution (Himmelblau, 1992) is

$$\vec{x}^* = (78.0, 33.0, 29.995, 45.0, 36.776),$$

with $G4(\vec{x}^*) = -30665.5$. Two constraints (upper bound of the first inequality and the lower bound of the third inequality) are active at the optimum.

However, the importance of proper values for penalty coefficients should be emphasized. For example, Homaifar et al. (1994) reported a value 50 for such penalty coefficient for the violation of the lower boundary of the first constraint and the value of 2.5 for the violation of the upper boundary of the first constraints, without any comments on the source for these values.

3.2.2 Method of dynamic penalties

The method was proposed by Joines and Houck (1994). As opposed to the previous method, the authors assumed dynamic penalties. Individuals are evaluated (at the iteration t) by the following formula:

$$eval(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\vec{x}),$$

where C, α and β are constants. A reasonable choice for these parameters, reported by Joines and Houck (1994), is $C = 0.5$, $\alpha = \beta = 2$. The method requires much smaller number (independent of the number of constraints) of parameters than the first method. Also, instead of defining several levels of violation, the pressure on infeasible solutions is increased due to the $(C \times t)^{\alpha}$ component of the penalty term: towards the end of the process (for high values of the generation number t), this component assumes large values.

One of the problems (Hock and Schittkowski, 1981) experimented by Joines and Houck (1994) was to minimize a function

$$G5(\vec{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + 0.000002/3x_2^3$$

subject to

$$\begin{array}{l}
x_4 - x_3 + 0.55 \geq 0,\ x_3 - x_4 + 0.55 \geq 0, \\
1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \\
1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \\
1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0 \\
0 \leq x_i \leq 1200,\ i = 1, 2, \text{ and } -0.55 \leq x_i \leq 0.55,\ i = 3, 4.
\end{array}$$

The best known solution is

$$\vec{x}^* = (679.9453, 1026.067, 0.1188764, -0.3962336),$$

and $G5(\vec{x}^*) = 5126.4981$.

The best result of experiments reported by Joines and Houck (1994) (many runs, various values of α and β) gave a value of 5126.6653 of the objective function. No solution was fully feasible due to the three equality constraints, but the sum of the violated constraints was quite small (10^{-4}). On the other hand, the method seems to provide too strong penalties: often the factor $(C \times t)^\alpha$ grows too fast to be useful. The system has little chances to escape from local optima: in most experiments reported by Michalewicz (1995a) the best individual was found in early generations. The method gave very good results for test cases where the objective functions were quadratic.

3.2.3 Method of annealing penalties

This method, called Genocop II, is also based on dynamic penalties and was described by Michalewicz and Attia (1994) and Michalewicz (1996). Its modified version works as follows:

- divide all constraints into four subsets: linear equations, linear inequalities, nonlinear equations, and nonlinear inequalities,
- select a random single point as a starting point (the initial population consists of copies of this single individual). This initial point satisfies linear constraints,
- set the initial temperature $\tau = \tau_0$,
- evolve the population using the following formula:

  $$eval(\vec{x}, \tau) = f(\vec{x}) + \frac{1}{2\tau} \sum_{j=1}^{m} f_j^2(\vec{x}),$$

- if $\tau < \tau_f$, stop, otherwise
  - decrease temperature τ,
  - the best solution serves as a starting point of the next iteration,
  - repeat the previous step of the algorithm.

This is the only method we are aware of (except Genocop III, section 3.3.3) which distinguishes between linear and nonlinear constraints. The algorithm maintains feasibility of all linear constraints using a set of closed operators, which convert a feasible solution (feasible in terms of linear constraints only) into another feasible solution. At every iteration the algorithm considers active constraints only, the pressure on infeasible solutions is increased due to the decreasing values of temperature τ.

The method has an additional unique feature: it starts from a single point.[2] Consequently, it is relatively easy to compare this method with other classical optimization methods whose performance are tested (for a given problem) from some starting point.

The method requires a starting and 'freezing' temperatures, τ_0 and τ_f, respectively, and the cooling scheme to decrease temperature τ. Standard values (reported by Michalewicz and Attia (1994)) are $\tau_0 = 1$, $\tau_{i+1} = 0.1 \cdot \tau_i$, with $\tau_f = 0.000001$.

One of the test problems (Floudas and Pardalos, 1987) experimented by Michalewicz and Attia (1994) was:

$$\text{minimize } G6(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3,$$

subject to nonlinear constraints:

$$\begin{array}{ll} c1: & (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0, \\ c2: & -(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0, \end{array}$$

and bounds:

$$13 \leq x_1 \leq 100 \text{ and } 0 \leq x_2 \leq 100.$$

The known global solution is $\vec{x}^* = (14.095, 0.84296)$, and $G6(\vec{x}^*) = -6961.81381$ (see figure 3). Clearly, both constraints are active at the optimum. The starting point, which is not feasible, is $\vec{x}_0 = (20.1, 5.84)$.

GENOCOP II approached the optimum very closely at the 12th iteration. The progress of the system is reported in the table 1.

In (Michalewicz, 1995a) some experimental evidence is presented to the effect that linear constraints of a problem may prevent the system to move closer to the optimum. This is an interesting example of damaging effect of limiting population to the feasible (with respect to linear constraints) region only. Additional experiments indicated that the method is very sensitive to the cooling scheme.

3.2.4 Methods of adaptive penalties

In this section we discuss two methods which are based on adaptive penalty functions.

The first method was developed by Bean and Hadj-Alouane (1992), and Hadj-Alouane and Bean (1992). As the previous method, it uses a penalty function, however, one component of the penalty function takes a feedback from the search process. Each individual is evaluated by the formula:

[2] This feature, however, is not essential. The only important requirement is that the next population contains the best individual from the previous population.

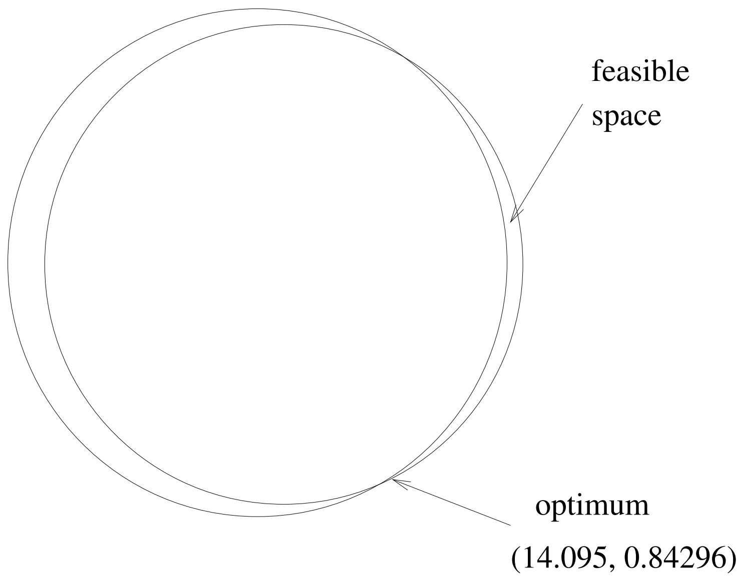

Figure 3: A feasible space for test case of $G6$

$eval(\vec{x}) = f(\vec{x}) + \lambda(t)\sum_{j=1}^{m} f_j^2(\vec{x})$,

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\cdot\lambda(t), & if\ \vec{b}^i \in \mathcal{F}\ for\ all\ t-k+1 \le i \le t \\ \beta_2\cdot\lambda(t), & if\ \vec{b}^i \in \mathcal{S}-\mathcal{F}\ for\ all\ t-k+1 \le i \le t \\ \lambda(t), & otherwise, \end{cases}$$

where $\vec{b}^i$ denotes the best individual, in terms of function $eval$, in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method (1) decreases the penalty component $\lambda(t+1)$ for the generation $t+1$, if all best individuals in the last k generations were feasible, and (2) increases penalties, if all best individuals in the last k generations were infeasible. If there are some feasible and infeasible individuals as best individuals in the last k generations, $\lambda(t+1)$ remains without change.

The method introduces three additional parameters, β_1, β_2, and the time horizon, k. It seems that there is an interesting analogy between this method and the method used in early evolution strategies to optimize the convergence rate, where a "1/5 success rule" was proposed (Rechenberg, 1973; Bäck et al., 1991):

> The ratio φ of successful mutations to all mutations should be 1/5. Increase the variance of the mutation operator, if φ is greater than 1/5; otherwise, decrease it.

The 1/5 success rule emerged as a conclusion of the process of optimizing convergence rates of two functions (the so-called corridor model and sphere model). The rule was applied every k

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 1: Progress of GENOCOP II on test case of function $G6$; for iteration 0 the best point is the starting point.

generations:

$$\vec{\sigma}^{t+1} = \begin{cases} c_d \cdot \vec{\sigma}^t, & \text{if } \varphi(k) < 1/5, \\ c_i \cdot \vec{\sigma}^t, & \text{if } \varphi(k) > 1/5, \\ \vec{\sigma}^t, & \text{if } p_s(k) = 1/5, \end{cases}$$

where $\varphi(k)$ is the success ratio of the mutation operator during the last k generations, and $c_i > 1$, $c_d < 1$ regulate the increase and decrease rates for the variance of the mutation. Schwefel in his experiments (Schwefel, 1981) used the following values: $c_d = 0.82$, $c_i = 1.22 = 1/0.82$.

The intuitive reason behind adaptation of penalties in Bean's and Hadj-Alouane's method is similar to the reason behind the 1/5 success rule: the increased efficiency of the search. In evolution strategies, if successful, the search would continue in "larger" steps; if not, the steps would be shorter. In Bean's and Hadj-Alouane's method, if constraints do not pose a problem, the search would continue with decreased penalties, if not, the penalties would be increased. The presence of both feasible and infeasible individuals in the set of best individuals in the last k generations means that the current value of penalty component $\lambda(t)$ is set right.

The adaptive penalty function used by Smith and Tate (1993) incorporates both the search length and constraint severity feedback. It involves the estimation of a near-feasible threshold q_j for each constraint $1 \leq j \leq m$); such thresholds indicate distances from the feasible region $\mathcal{F}$ which are "reasonable" (or, in other words, which determine "interesting" infeasible solutions, i.e., solutions relatively close to the feasible region). Thus the evaluation function is defined as

$$eval(\vec{x},t) = f(\vec{x}) + F_{feas}(t) - F_{all}(t)\sum_{j=1}^{m}(f_j(\vec{x})/q_j(t))^k,$$

where $F_{all}(t)$ denotes the unpenalized value of the best solution yet found (up to generation t), $F_{feas}(t)$ denotes the value of the best feasible solution yet found (up to generation t), and k is a constant. Note, that the near-feasible thresholds $q_j(t)$ are dynamic, i.e., they are adjusted during the search (for example, it is possible to define $q_j(t) = q_j(0)/(1+\beta_j t)$ thus resulting in increasing the penalty component over time).

To the best of our knowledge, neither of the adaptive methods described in this subsection has been applied to continuous nonlinear programming problems.

3.2.5 Death penalty method

This method just rejects infeasible individuals (death penalty); the method has been used by evolution strategies (Bäck et al., 1991) and simulated annealing. For some problems, this simple method provides quality results.

For example, one of the test cases considered recently (Michalewicz, 1995a) included the following problem (Hock and Schittkowski, 1981). Minimize a function:

$$\begin{aligned} G7(\vec{x}) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 + 2(x_6-1)^2 + \\ & 5x_7^2 + 7(x_8-11)^2 + 2(x_9-10)^2 + (x_{10}-7)^2 + 45, \end{aligned}$$

subject to the following constraints:

$$\begin{array}{ll} 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0, & -3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0, \\ -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0, & -x_1^2 - 2(x_2-2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0, \\ 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0, & -5x_1^2 - 8x_2 - (x_3-6)^2 + 2x_4 + 40 \geq 0, \\ 3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \geq 0, & -0.5(x_1-8)^2 - 2(x_2-4)^2 - 3x_5^2 + x_6 + 30 \geq 0, \end{array}$$

and bounds

$$-10.0 \leq x_i \leq 10.0, \quad i = 1, \ldots, 10.$$

The problem has 3 linear and 5 nonlinear constraints; the function $G7$ is quadratic and has its global minimum at

$$\begin{aligned} \vec{x}^* = (& 2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, \\ & 1.430574, 1.321644, 9.828726, 8.280092, 8.375927), \end{aligned}$$

where $G7(\vec{x}^*) = 24.3062091$. Six (out of eight) constraints are active at the global optimum (all except the last two).

The "death penalty" method provided with respectable solutions the best of which had the value of 25.653. To evaluate this method it is necessary to initialize a population by feasible solutions. This different initialization scheme makes the comparison of all the methods even harder. However, in experiments reported in (Michalewicz, 1995a), an interesting pattern emerged: the method generally gave a quite poor performance. Also, the method was not as stable as other methods; the standard deviation of returned solutions was relatively high.

3.2.6 Segregated genetic algorithm

The method of so-called segregated genetic algorithm was proposed by Le Riche et al. (1995) as yet another way to handle the problem of the robustness of the penalty level (see section 3.2). The tuning of the penalty level encounters the following dilemma: a too small penalty level leads to solutions which are infeasible (some penalized points still exhibit higher penalized fitness than the best feasible point); a too high penalty level restricts the search inside the feasible region, forbidding any shot-cut across the infeasible region, and thus eventually failing to converge to the optimal solution.

The idea is to design two different penalized fitness functions with static penalty terms p_1 and p_2 (see section 3.2): penalty p_1 is purposely too small, while penalty p_2 is hopefully too high. All individuals of the current population undergo crossover and mutation. The values of the two fitness functions $f_i(\vec{x}) = f(\vec{x}) + p_i(\vec{x})$, $i = 1, 2$, are computed for each resulting offspring (at no extra cost in term of objective function evaluation), and two ranked lists are created according to the values of the fitnesses of all individuals (parents plus offspring) for each one of the fitnesses. The selection of the parents of next generation is then achieved by picking up alternatively the best individual from each list (and removing this individual from both lists afterwards). The main idea is that such a selection scheme will result roughly in maintaining two subpopulations: the individuals selected on the basis of f_1 will more likely lie in the infeasible region while the ones selected on the basis of f_2 will probably stay in the feasible region; the overall process is thus allowed to reach the feasible optimum from both sides of the boundary of the feasible region.

This method gave excellent results in the domain of Laminated Design Optimization (Le Riche et al., 1995), but has not yet, to the best of our knowledge, been applied to continuous nonlinear programming problems.

3.3 Methods based on a search for feasible solutions

There are a few methods which emphasize the distinction between feasible and infeasible solutions in the search space $\mathcal{S}$. One method considers the problem constraints in a sequence; a switch from one constraint to another is made upon arrival of a sufficient number of feasible individuals in the population. The second method is based on an assumption, that any feasible solution is better than any infeasible one. The third method repairs infeasible individuals. We discuss these methods in turn.

3.3.1 Behavioral memory method

The method was proposed by Schoenauer and Xanthakis (1993); the authors called this method a "behavioral memory" approach. It works as follows:

- start with a random population of individuals (feasible or infeasible),
- set $j = 1$ (j is a constraint counter),
- evolve this population with $eval(\vec{x}) = f_j(\vec{x})$, until a given percentage of the population (so-called flip threshold ϕ) is feasible for this constraint,[3]
- set $j = j + 1$,
- the current population is the starting point for the next phase of the evolution, where $eval(\vec{x}) = f_j(\vec{x})$ (defined in the section 3.2). During this phase, points that do not satisfy one of the 1st, 2nd, ..., or $(j-1)$-th constraint are eliminated from the population. The stop criterion is again the satisfaction of the j-th constraint by the flip threshold percentage ϕ of the population.
- if $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the objective function, i.e., $eval(\vec{x}) = f(\vec{x})$, rejecting infeasible individuals.

The method requires a linear order of all constraints which are processed in turn. It is unclear what is the influence of the order of constraints on the results of the algorithm; our experiments indicated that different orders provide different results (different in the sense of the total running time and precision).

In total, the method requires three parameters: the sharing factor σ, the flip threshold ϕ, and a particular order of constraints. The method is very different to the other methods, and, in general, is different than other penalty approaches, since it considers only one constraint at the time. Also, in the last step of the algorithm the method optimizes the objective function f itself with a death penalty.

The method was evaluated by Schoenauer and Xanthakis (1993) on the following problem:

Maximize the function

$$G8(\vec{x}) = \frac{\sin^3(2\pi x_1)\cdot\sin(2\pi x_2)}{x^3\cdot(x_1+x_2)},$$

subject to the following constraints:

[3]The method suggests the use of a sharing scheme to maintain diversity of the population.

$$c1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0,$$
$$c2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

and bounds:

$$0 \leq x_1 \leq 10 \text{ and } 0 \leq x_2 \leq 10.$$

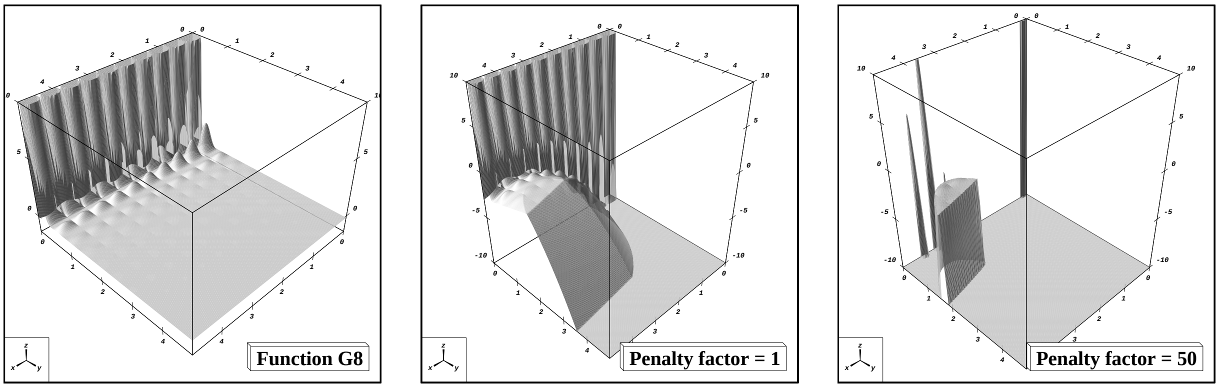

Figure 4: Function $G8$ together with two fitness landscapes corresponding to two different penalization parameters α: a small value of α leaves the highest peaks outside the feasible region, while a large value of α makes the feasible region look like a needle in a haystack (all plots are truncated between -10 and 10 for the sake of visibility)

Function $G8$ has many local optima, the highest peaks are located along the x axis (e.g., $G8(0.00015, 0.0225) > 1540$). In the feasible region, however, $G8$ has two maxima of almost equal fitness of value of 0.1. Figures 4 show the penalized fitness landscape for different penalty parameters (i.e., plots of the function $G8(\vec{x}) - \alpha(c1(\vec{x})^+ + c1(\vec{x})^+)$ for different values of α). All values of the penalized function are truncated at -10 and 10 to allow values in the feasible region (around 0) to be distinguishable.

This situation makes any attempt to use penalty parameters a difficult task: small penalty parameters leave the feasible region hidden among much higher peaks of the penalized fitness (Figure 4-b) while penalty parameters large enough to allow the feasible region to really outstand in the penalized-fitness landscape imply vertical slopes of that region (Figure 4-c): the partial credit principle, allowing the evolutionary algorithm to gently climb such slopes is not valid, and the discovery of the global maximum of the penalized fitness thus relies on a lucky move from the huge and flat low-fitness region.

On the other hand, the behavioral memory method has no difficulty in localizing 80% of the population first into the feasible region for constraint $c1$ (step 1), then into the feasible region for both constraints (step 2), as can be seen on Figure 5. The optimization of $G8$ itself in that region is then straightforward (see the smooth bimodal fitness landscape in that

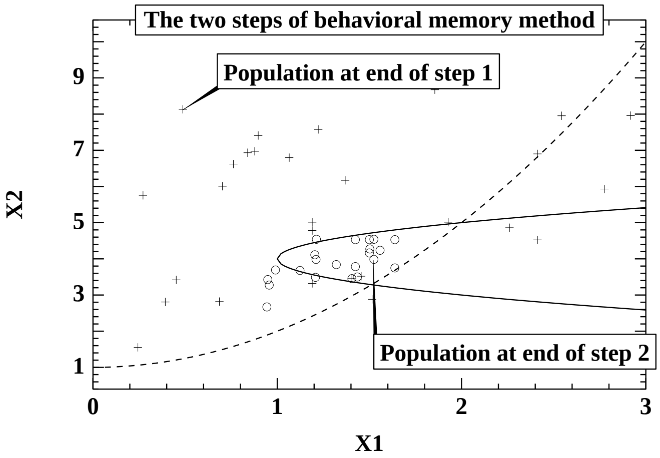

Figure 5: Illustration of a sample run of behavioral memory method on $G8$: zoom around the feasible region. The dashed curve is constraint $c1$ and the continuous curve is constraint $c2$. The population at end of first step samples the feasible region for $c1$ while that of the end of step 2 samples the feasible region for both $c1$ and $c2$

region in Figure 4). Throughout all steps, however, the diversity in the population had to be preserved: the resulting population is to be used as the starting point for another evolution, and hence must sample as uniformly as possible the target feasible region in the two first steps; the objective function is multi-modal in that feasible region, hence care must be taken to locate both peaks. This requirement, achieved using the sharing scheme, is even more mandatory when the feasible region is made of several distinct connected components, as shown by Schoenauer and Xanthakis (1993) on a similar problem (with a slightly different constraint $c2$).

3.3.2 Method of superiority of feasible points

The method was developed by Powell and Skolnick (1993). The method is based on a classical penalty approach with one notable exception. Each individual is evaluated by the formula:

$$eval(\vec{x}) = f(\vec{x}) + r \sum_{j=1}^{m} f_j(\vec{x}) + \theta(t, \vec{x}),$$

where r is a constant; however, the original component $\theta(t, \vec{x})$ is an additional iteration dependent function which influences the evaluations of infeasible solutions. The point is that the method distinguishes between feasible and infeasible individuals by adopting an additional heuristic rule (suggested earlier by Richardson et al., 1989): for any feasible individual $\vec{x}$ and any infeasible individual $\vec{y}$: $eval(\vec{x}) < eval(\vec{y})$, i.e., any feasible solution is better than any infeasible one. This can be achieved in many ways; one possibility is to set

$$\theta(t, \vec{x}) = \begin{cases} 0, & if\ \vec{x} \in \mathcal{F} \\ \max\{0, \max_{\vec{x} \in \mathcal{F}}\{f(\vec{x})\} - \min_{\vec{x} \in \mathcal{S}-\mathcal{F}}\{f(\vec{x}) + r \sum_{j=1}^{m} f_j(\vec{x})\}\}, & otherwise. \end{cases}$$

In other words, infeasible individuals have increased penalties: their values cannot be better than the value of the worst feasible individual (i.e., $\max_{\vec{x}\in\mathcal{F}}\{f(\vec{x})\}$).[4]

For example, the above method was experimented with by Michalewicz (1995a) on one of the problems described by Hock and Schittkowski (1981): minimize the function

$$G9(\vec{x}) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,$$

subject to the following constraint:

$$\begin{array}{ll} 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \ge 0, & 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \ge 0, \\ 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \ge 0, & -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \ge 0 \end{array}$$

and bounds:

$$-10.0 \le x_i \le 10.0,\ i = 1,\ldots,7.$$

The problem has 4 nonlinear constraints; the function $G9$ is nonlinear and has its global minimum at

$$\vec{x}^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227),$$

where $G9(\vec{x}^*) = 680.6300573$. Two (out of four) constraints are active at the global optimum (the first and the last one).

The method of Powell and Skolnick gave a reasonable performance; the best solution found was 680.934. However, for some test problems (Michalewicz, 1995a) the method may have some difficulties in locating a feasible solution.

3.3.3 Repairing infeasible individuals

This recently developed method, called Genocop III (Michalewicz and Nazhiyath, 1995) is based on the idea of repairing infeasible solutions and also incorporates some concepts of co-evolution.

As in the original Genocop (section 3.1.1), linear equations are eliminated, number of variables is reduced, and linear inequalities are modified accordingly. All points included in the initial population satisfy linear constraints; specialized operators maintain their feasibility (in the sense of linear constraints) from one generation to the next. We denote the set of points which satisfy the linear constraints by $\mathcal{F}_l \subseteq \mathcal{S}$.

Nonlinear equations require an additional parameter (γ) to define the precision of the system. All nonlinear equations $h_j(\vec{x}) = 0$ (for $j = q+1,\ldots,m$) are replaced by a pair of inequalities:

[4] Powell and Skolnick (1993) achieved the same result by mapping evaluations of feasible solutions into the interval $(-\infty, 1)$ and infeasible solutions into the interval $(1, \infty)$. For ranking and tournament selections this implementational difference is not important.

$$-\gamma \leq h_j(\vec{x}) \leq \gamma,$$

so we deal only with nonlinear inequalities. These nonlinear inequalities restrict further the set $\mathcal{F}_l$: they define the fully feasible part $\mathcal{F} \subseteq \mathcal{F}_l$ of the search space $\mathcal{S}$.

Genocop III incorporates the original Genocop system, but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points from $\mathcal{F}_l$ which satisfy linear constraints of the problem. As mentioned earlier, the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators (see section 3.1.1). The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, i.e., they satisfy *all* constraints.[5] Reference points $\vec{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e., $eval(\vec{r}) = f(\vec{r})$). On the other hand, search points from P_s are "repaired" for evaluation and the repair process works as follows. Assume, there is a search point $\vec{s} \in P_s$. If $\vec{s} \in \mathcal{F}$, then $eval(\vec{s}) = f(\vec{s})$, since $\vec{s}$ is fully feasible. Otherwise (i.e., $\vec{s} \notin \mathcal{F}$), the system selects[6] one of the reference points, say $\vec{r}$ from P_r and creates a sequence of random points $\vec{z}$ from a segment between $\vec{s}$ and $\vec{r}$ by generating random numbers a from the range $\langle 0, 1\rangle$: $\vec{z} = a\vec{s} + (1-a)\vec{r}$.[7] Once a fully feasible $\vec{z}$ is found, $eval(\vec{s}) = eval(\vec{z}) = f(\vec{z})$.[8]

Additionally, if $f(\vec{z})$ is better than $f(\vec{r})$, then the point $\vec{z}$ replaces $\vec{r}$ as a new reference point in the population of reference points P_r. Also, $\vec{z}$ replaces $\vec{s}$ in population of search points P_s with some probability of replacement p_r.

The structure of Genocop III is shown in figure 6 and the procedure for evaluating (not necessarily fully feasible) search points from population P_s is given in figure 7. Note that there is some asymmetry between processing population of search points P_s and population of reference points P_r: while we apply selection procedure and operators to P_s every generation, population P_r is modified every k (parameter of the method) generations (however, some additional changes in P_r are possible during the evaluation of search points, see figure 7). The main reason behind this arrangement is efficiency of the system: search within feasible part of the search space $\mathcal{F}$, as much less effective for NLP problems, is treated as a background event. Note also, that the "selection" and "alternation" steps are reversed in the evolution loop for the P_r: due to a low probability of generating feasible offspring, first parent individual reproduce and later the best feasible individuals (from both parents and offspring) are selected for the next population (this is similar to the $(\mu + \lambda)$ selection of evolution strategies (Bäck et al., 1991).

[5] If Genocop III has difficulties in locating such a reference point for the purpose of initialization, the user is prompted for it. In cases, where the ratio $|\mathcal{F}|/|\mathcal{S}|$ of feasible points in the search space is very small, it may happen that the initial set of reference points consists of a multiple copies of a single feasible point.

[6] Better reference points have better chances to be selected; a selection method based on nonlinear ranking was used.

[7] Note that all such generated points $\vec{z}$ belong to $\mathcal{F}_l$.

[8] Clearly, in different generations the same search point $\mathcal{S}$ can evaluate to different values due to the random nature of the repair process.

```
procedure Genocop III
begin
   t ← 0
   initialize P_s(t)
   initialize P_r(t)
   evaluate P_s(t)
   evaluate P_r(t)
   while (not termination-condition) do
   begin
      t ← t + 1
      select P_s(t) from P_s(t − 1)
      alter P_s(t)
      evaluate P_s(t)
      if t mod k = 0 then
      begin
         alter P_r(t)
         select P_r(t) from P_r(t − 1)
         evaluate P_r(t)
      end
   end
end
```

Figure 6: The structure of Genocop III

Genocop III uses the objective function for evaluation of fully feasible individuals only, so the evaluation function is not distorted as in methods based on penalty functions. It introduces only few additional parameters (the population size of reference points, probability of replacement, frequency of application of operators to the population of reference points, precision γ). It always returns a feasible solution. A feasible search space $\mathcal{F}$ is searched (population P_r) by making references from the search points and by application of operators (every some number of generations, k in figure 6). The neighborhoods of better reference points are explored more often. Some fully feasible points are moved into the population of search-points P_s (replacement process), where they undergo additional transformation by specialized operators.

One of the most interesting parameters of the developed system is the probability of replacement p_r (replacement of $\vec{s}$ by $\vec{z}$ in population of search points P_s; see figure 7). Recently, Orvosh and Davis (1993) reported a so-called 5%-rule: this heuristic rule states that in many combinatorial optimization problems, an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. However, neither Davis (1995) nor the authors are aware of any experiments connected with the probability of replacement for repair algorithms applied for problems in numerical domains. Experiments reported by Michalewicz and Nazhiyath (1995) were done for $p_r = 0.20$; this value

```
procedure evaluate P_s(t)
begin
   for each s ∈ P_s(t) do
      if s ∈ F
      then evaluate s (as f(s)) else
      begin
         select r ∈ P_r(t)
         generate z ∈ F
         evaluate s (as f(z))
         if f(r) > f(z) then replace r by z in P_r
         replace s by z in P_s with probability p_r
      end
end
```

Figure 7: Evaluation of population P_s

provided with the best results on a selected test suite.

The results of experiments were quite good. For example, the following problem (Hock and Schittkowski, 1981) proved very difficult to all methods described earlier. The problem is to minimize

$$G10(\vec{x}) = x_1 + x_2 + x_3,$$

subject to the following constraints:

$$\begin{array}{l} 1 - 0.0025(x_4 + x_6) \geq 0, \quad 1 - 0.0025(x_5 + x_7 - x_4) \geq 0, \\ 1 - 0.01(x_8 - x_5) \geq 0, \quad x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0, \\ x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0, \quad x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 \geq 0, \end{array}$$

and bounds

$$100 \leq x_1 \leq 10000, \quad 1000 \leq x_i \leq 10000,\ i = 2, 3, \quad 10 \leq x_i \leq 1000,\ i = 4, \ldots, 8.$$

The problem has 3 linear and 3 nonlinear constraints; the function $G10$ is linear and has its global minimum at

$$\vec{x}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979),$$

where $G10(\vec{x}^*) = 7049.330923$. All six constraints are active at the global optimum.

For the above problem the best result of Genocop III was 7286.650: much better than the best result of the best system from those discussed in (Michalewicz, 1995a). Similar performance was observed on two other problems, $G9$ (with 680.640) and $G7$ (with 25.883). Additional interesting observation was connected with stability of the system. Genocop III had a very low standard deviation of results. For example, for problem $G9$, all results were between 680.640 and 680.889; on the other hand, other systems produced variety of results (between 680.642 and 689.660, see Michalewicz (1995a)).

Of course, all resulting points $\vec{x}$ were feasible, which was not the case with other systems (e.g., Genocop II produced a value of 18.917 for the problem $G7$, the systems based on the methods of Homaifar, Lai, and Qi (section 3.2.1) and Powell and Skolnick (section 3.3.2) gave results of 2282.723 and 2101.367, respectively, for the problem $G10$, and these solutions were not feasible).

3.4 Hybrid methods

It is relatively easy to develop hybrid methods which combine evolutionary computation techniques with deterministic procedures for numerical optimization problems. Waagen et al. (1992) combined evolutionary programming technique (with floating point representation, Fogel, 1995) with the direction set method of Hooke-Jeeves; this hybrid method was tested on three (unconstrained) test functions. Myung et al. (1995) considered a similar approach, but they experimented with constrained problems. Again, they combined floating-point evolutionary programming technique with some other method—developed by Maa and Shanblatt (1992). However, while the method of Waagen et al. (1992) incorporated the direction set algorithm as a problem-specific operator of his evolutionary technique, Myung et al. (1995) divided the whole optimization process into two separate phases. During the first phase, evolutionary algorithm optimizes the function

$$eval(\vec{x}) = f(\vec{x}) + \frac{s}{2}\left(\sum_{j=1}^{m} f_j^2(\vec{x})\right),$$

where s is a constant. After the termination of this phase, the second phase of the optimization algorithm of Maa and Shanblatt (1992) was applied to the best solution found during the first phase; this phase iterates until the system

$$\vec{x}' = -\bigtriangledown f(\vec{x}) - \left[\sum_{j=1}^{m} \bigtriangledown f_j(\vec{x})(s f_j(\vec{x}) + \lambda_j)\right],$$

is in equilibrium, where the Lagrange multipliers are updated as $\lambda_j' = \epsilon s f_j$ for a small positive constant ϵ.

This hybrid method was successfully applied to a few test cases; for example, one of the test cases was to minimize

$G11(\vec{x}) = x_1^2 + (x_2 - 1)^2,$

subject to a nonlinear constraint:

$$x_2 - x_1^2 = 0,$$

and bounds:

$$-1 \le x_i \le 1,\ i = 1, 2.$$

The known global solutions are $\vec{x}^* = (\pm 0.70711, 0.5)$, and $G11(\vec{x}^*) = 0.75000455$.

For this problem, the first phase of the evolutionary process converges quickly in 100 generations to $(\pm 0.70711, 0.5)$, and the second phase drives the system to the global solution. However, the method was not tested for problems of higher dimensions.

Several other constraint handling methods deserve also some attention. For example, some methods use of the values of objective function f and penalties f_j ($j = 1, \ldots, m$) as elements of a vector and apply multi-objective techniques to minimize all components of the vector. For example, Schaffer's (1985) Vector Evaluated Genetic Algorithm (VEGA) selects $1/(m+1)$ of the population based on each of the objectives. Such an approach was incorporated by Parmee and Purchase (1994) in the development of techniques for constrained design spaces. On the other hand, in the approach by Surry et al. (1995), all members of the population are ranked on the basis of constraint violation. Such rank r, together with the value of the objective function f, leads to the two-objective optimization problem. This approach gave a good performance on optimization of gas supply networks (Surry et al., 1995).

Also, an interesting approach was reported by Paredis (1994). The method (described in the context of constraint satisfaction problems) is based on a co-evolutionary model, where a population of potential solutions co-evolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions. There is some development connected with generalizing the concept of "ant colonies" (Colorni et al., 1991) (which were originally proposed for order-based problems) to numerical domains (Bilchev and Parmee, 1995); first experiments on some test problems (from Table 3) gave very good results (Bilchev, 1995). It is also possible to incorporate the knowledge of the constraints of the problem into the belief space of cultural algorithms (Reynolds, 1994); such algorithms provide a possibility of conducting an efficient search of the feasible search space (Reynolds et al., 1995). Recently, Kelly and Laguna (1996) developed a new procedure for difficult constrained optimization problems, which

> "...has succeeded in generating optimal and near-optimal solutions in minutes to complex nonlinear problems that otherwise have required days of computer time in order to verify optimal solutions. These problems include nonconvex, nondifferentiable and mixed discrete functions over constrained regions. The feasible space explicitly incorporates linear and integer programming constraints and implicitly incorporates other nonlinear constraints by penalty functions. The objective functions

> to be optimized may have no 'mathematically specifiable' representation, and can even require simulations to generate. Due to this flexibility, the approach applies also to optimization under uncertainty (which increases its relevance to practical applications)."

The method combines evolutionary processes with (1) scatter search, (2) with procedures of mixed integer optimization, and (3) with adaptive memory strategies of tabu search.

However, all the above methods are on early stages of their development, or they have not been applied yet to the general NLP, so we do not discuss them any further.

4 Conclusions and Future Work

It is difficult to compare the constraint-handling methods which were discussed in the previous section of this paper. The main reason is that the original formulations of these methods are based on different representations (binary or float), and consequently they use different operators. For example, the method of static penalties (section 3.2.1) was proposed for a standard GA with binary representation, standard crossover and mutation. Genocop (section 3.1.1) uses floating point representation with seven specialized operators. The method of superiority of feasible points (section 3.3.2) was also developed for floating point representation. However, their operators were different to those of Genocop. Bean and Hadj-Alouane adapted penalties (section 3.2.4) for integer programming problems and used binary representation to code integers. Death penalty methods (section 3.2.5) were mainly used by evolution strategies (floating point representation with Gaussian mutation and adaptation of control parameters), and the segregated GAs (section 3.2.6) has not been applied yet to the NLP. For these reasons we do not provide numerical results of the above methods on all test cases defined in the previous section. The only general observation we can make at this stage is that for constrained numerical optimization problems, the floating point representation provides with much better results (some of experiments on comparisons between binary and floating point representations for unconstrained problems were reported in Michalewicz, 1996, chapter 5).

Some partial comparisons of several constraint-handling methods were reported (Michalewicz, 1995a), where a modified version of Genocop system was used for all experiments; the code was updated accordingly for each considered method. The results of these comparisons are given in Table 2.

It is not clear what characteristics of a constrained problem make it difficult for an evolutionary technique. It seems that no single parameter (number of linear, nonlinear, active constraints, the ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ of feasible points in $\mathcal{S}$, type of the function, number of variables) proved to be significant as a major measure of difficulty of the problem. For example, many methods approached the optimum quite closely for the test cases $G1$ and $G7$ (with $\rho = 0.0111\%$ and $\rho = 0.0003\%$, respectively), whereas most of the methods experienced difficulties for the test case $G10$ (with $\rho = 0.0010\%$). Two quadratic functions (the test cases $G1$

Test Case	Exact opt.		Method of section 3.2.1	Method of section 3.2.2	Method of section 3.3.1	Method of section 3.2.3	Method of section 3.3.2	Method of section 3.2.5	Method of section 3.2.5(f)
		b	−15.002	−15.000	−15.000	−15.000	−15.000		−15.000
$G1$	−15.000	m	−15.002	−15.000	−15.000	−15.000	−15.000	—	−14.999
		w	−15.001	−14.999	−14.998	−15.000	−14.999		−13.616
		c	0, 0, 4	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0		0, 0, 0
		b	2282.723	3117.242	7485.667	7377.976	2101.367		7872.948
$G10$	7049.331	m	2449.798	4213.497	8271.292	8206.151	2101.411	—	8559.423
		w	2756.679	6056.211	8752.412	9652.901	2101.551		8668.648
		c	0, 3, 0	0, 3, 0	0, 0, 0	0, 0, 0	1, 2, 0		0, 0, 0
		b	680.771	680.787	680.836	680.642	680.805	680.934	680.847
$G9$	680.630	m	681.262	681.111	681.175	680.718	682.682	681.771	681.826
		w	689.660	682.798	685.640	680.955	685.738	689.442	689.417
		c	0, 0, 1	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
		b	24.690	25.486		18.917	17.388		25.653
$G7$	24.306	m	29.258	26.905	—	24.418	22.932	—	27.116
		w	36.060	42.358		44.302	48.866		32.477
		c	0, 1, 1	0, 0, 0		0, 1, 0	1, 0, 0		0, 0, 0

Table 2: Experimental results. For some methods, the best (b), median (m), and the worst (w) result (out of 10 independent runs) are reported together with the number (c) of violated constraints at the median solution: the sequence of three numbers indicate the number of violations by more than 1.0, more than 0.1, and more than 0.001, respectively. The symbol '—' stands for 'the results were not meaningful'. The results for method of section 3.2.5 report also the results of experiments (case 3.2.5(f)) where the initial population was feasible.

and $G7$) with a similar number of constraints (9 and 8, respectively) and an identical number (6) of active constraints at the optimum, gave a different challenge to most of these methods. Also, several methods were quite sensitive to the presence of a feasible solution in the initial population (e.g., method of section 3.2.5; see table 2).

All test cases are summarized in table 3; for each test case we list number n of variables, type of the function f, the relative size of the feasible region in the search space given by the ratio ρ, the number of constraints of each category (linear inequalities LI, nonlinear equations NE and inequalities NI), and the number a of active constraints at the optimum (including equality constraints).

Despite some obvious weaknesses of the method of static penalties (section 3.2.1), this basic method, which requires fixed user-supplied penalty parameters, remains the most popular one to handle constraints for evolutionary algorithms and many other optimization frameworks. The main reason for that popularity is that it certainly is the simplest technique to implement: it requires only a straightforward modification of the objective function. Moreover, it is fair to say that it gives quite reasonable results in many cases. Nevertheless, as emphasized in some of

Function	n	Type of f	ρ	*LI*	*NE*	*NI*	a
$G1$	13	quadratic	0.0111%	9	0	0	6
$G2$	k	nonlinear	99.8474%	0	0	2	1
$G3$	k	polynomial	0.0000%	0	1	0	1
$G4$	5	quadratic	52.1230%	0	0	6	2
$G5$	4	cubic	0.0000%	2	3	0	3
$G6$	2	cubic	0.0066%	0	0	2	2
$G7$	10	quadratic	0.0003%	3	0	5	6
$G8$	2	nonlinear	0.8560%	0	0	2	0
$G9$	7	polynomial	0.5121%	0	0	4	2
$G10$	8	linear	0.0010%	3	0	3	6
$G11$	2	quadratic	0.0000%	0	1	0	1

Table 3: Summary of eleven test cases. The ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ was determined experimentally by generating 1,000,000 random points from $\mathcal{S}$ and checking whether they belong to $\mathcal{F}$ (for $G2$ and $G3$ we assumed $k = 50$). *LI, NE*, and *NI* represent the number of linear inequalities, and nonlinear equations and inequalities, respectively

the research cited in this paper, there are also many other cases for which the penalty method fails. This raises two questions: (1) what features of the problem can guide the choice of the constraint-handling technique to be incorporated into an evolutionary algorithm (either among those surveyed here, or based on a new paradigm), and (2) is it possible to detect *a priori* such problems, which are resistant to the penalty approach? Let us provide a few comments addressing these two questions.

The experimental results of this paper suggest that making an appropriate *a priori* choice of an evolutionary method for a nonlinear optimization problem remains an open question. It seems that more complex properties of the problem (e.g., the characteristic of the objective function together with the topology of the feasible region) may constitute quite significant measures of the difficulty of the problem. Also, some additional measures of the problem characteristics due to the constraints might be helpful. However, this kind of information is not generally available *a priori*. One possibility would be to measure how fast the feasible region grows when the constraints are relaxed. We discuss briefly these concepts in the following paragraphs.

Consider the general NLP problem of section 1. Suppose that each constraint $g_j, j \in [1, q]$ and $h_j, j \in [q+1, m]$ is upper-bounded on $\mathcal{S}$ by $M_j \geq 0$. Then, for every value of $\omega \in [0, 1]$, we can define $\mathcal{F}_\omega$ as

$$\mathcal{F}_\omega = \{\vec{x} \in \mathcal{S} \; : \; \forall j \in [1, q] \; g_j(\vec{x}) \leq \omega M_j \; and \; \forall j \in [q+1, m] \; |h_j(\vec{x})| \leq \omega M_j\}.$$

It is interesting to observe that $\mathcal{F}_0 = \mathcal{F}$ and $\mathcal{F}_1 = \mathcal{S}$. The way $\rho_\omega = |\mathcal{F}_\omega|/|\mathcal{S}|$ behaves around $\omega = 0$ — measured, for instance, by the value

$$\sigma_\omega = \frac{d\rho_\omega}{d\omega}|_{\omega=0},$$

can serve as an indicator of "how fast" the feasible region expands around its boundary.

Of course, the difficulty of the NLP also heavily depends on the objective function f itself. So one possible measure of difficulty of the constrained problem is an evaluation of the optimum of the objective function f with relaxed constraints. The following formula,[9]

$$\varphi_\omega = \frac{\max_{\vec{x}\in\mathcal{F}_\omega} f(\vec{x})}{\max_{\vec{x}\in\mathcal{F}} f(\vec{x})},$$

can serve as a such measure. Unfortunately, this ratio can only be estimated *a posteriori* by solving a large number of NLPs, all as difficult as the original one.

In general, there are three possible sources of difficulties of the NLP. These difficulties can arise from the objective function itself, from the definition of the feasible region (i.e., from the constraints alone), or from the coupling between the objective function and the constraints. Hence the following features should be taken into account when trying to characterize different NLPs:

- The ruggedness of the unconstrained fitness landscape certainly is the first important characteristic influencing the overall problem difficulty; for instance, linear or convex objective functions will always result in easier constrained problems than almost-chaotic functions, for the same set of constraints; however, this characteristic can hardly be estimated *a priori*.
- The sparseness of the feasible region indeed is a crucial factor of difficulty: in some problems, finding a single feasible point is the major difficulty. Moreover, a high slope of the constraints on the border of the feasible region is another way by which a constrained problem can resist the penalty method; some hints have been given above (the factors ρ and σ_ω can be numerically estimated *a priori*).
- A high ratio between the highest global optima of the objective function (on the whole domain where it is defined) and of the optima of the constrained function (e.g., the local optima of the objective function in the feasible region), as well as a small distance between these global optima and the feasible region can make the constrained optimization problem almost intractable for penalty methods; unfortunately, it is almost impossible to estimate this factor *a priori*. However, some useful *a posteriori* informations about the behavior of different methods might be obtained at the cost of heavy computations by comparing their respective performances on gradually relaxed hard NLPs.

[9] For maximization problems. The objective function f takes non-negative values.

- The number of active constraints at the optimum is of course of importance: the more constraints that are active at the optimum, the more likely to succeed are algorithms searching close to the boundary of the feasible region; the extreme case being that described in section 3.1.2. But here again it is very difficult to predict such a number *a priori*.

It seems that the most promising approach at this stage of research is experimental, involving the design of a scalable test suite of constrained optimization problems, in which many of these features could be easily tuned. Then it should be possible to test new methods with respect to the corpus of all available methods.

Such experimental suite should contain:

- Convex problems (for validation only, as any gradient-bassed method will outperform any stochastic algorithm by several orders of magnitude on such problems);
- Convex objective function and non-convex constraints (from easy to difficult in slope, number of connected components, etc.);
- Non-convex objective function (with multiple optima) with convex constraints;
- Non-convex objective function (with numerous multiple optima) and highly non-convex steep constraints (with $\mathcal{F}$ consisting of possibly disjoint regions).

Such an experimental test suite is currently under construction.

Acknowledgments:

This material is based upon work supported by the National Science Foundation under Grant IRI-9322400. The authors would to thank Ken De Jong for his suggestions which made the article more readable.

References

Bäck, T., F. Hoffmeister, and H.-P. Schwefel (1991). A survey of evolution strategies. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, pp. 2–9. Morgan Kaufmann.

Bean, J. C. and A. B. Hadj-Alouane (1992). A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

Bilchev, G. (1995). Private communication.

Bilchev, G. and I. Parmee (1995). Ant colony search vs. genetic algorithms. Technical report, Plymouth Engineering Design Centre, University of Plymouth.

Colorni, A., M. Dorigo, and V. Maniezzo (1991). Distributed optimization by ant colonies. In *Proceedings of the First European Conference on Artificial Life*, Paris. MIT Press/Bradford Book.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the 3^{rd} International Conference on Genetic Algorithms*, pp. 61–69. Morgan Kaufmann.

Davis, L. (1995). Private communication.

DeJong, K. (1975). *The Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph. D. thesis, University of Michigan, Ann Harbor. *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).

Eiben, A., P.-E. Raue, and Z. Ruttkay (1994). Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, Number 866 in LNCS, pp. 78–87. Springer Verlag.

Eshelman, L. and J. D. Schaffer (1993). Real-coded genetic algorithms and interval-schemata. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*, Los Altos, CA, pp. 187–202. Morgan Kaufmann.

Floudas, C. and P. Pardalos (1987). *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Volume 455 of *LNCS*. Berlin: Springer Verlag.

Fogel, D. and L. Stayton (1994). On the effectiveness of crossover in simulated evolutionary optimization. *BioSystems 32*, 171–182.

Fogel, D. B. (1995). *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences 8*(1), 156–166.

Glover, F. (1994). Genetic algorithms and scatter search: Unsuspected potentials. *Statistics, and Computing 4*, 131–140.

Glover, F. and G. Kochenberger (1995). Critical event tabu search for multidimensional knapsack problems. In *Proceedings of the International Conference on Metaheuristics for Optimization*, pp. 113–133. Kluwer Publishing.

Gregory, J. (1995). Nonlinear Programming FAQ, Usenet sci.answers. Available at ftp://rtfm.mit.edu/pub/usenet/sci.answers/nonlinear-programming-faq.

Hadj-Alouane, A. B. and J. C. Bean (1992). A genetic algorithm for the multiple-choice integer program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan.

Himmelblau, D. (1992). *Applied Nonlinear Programming*. McGraw-Hill.

Hock, W. and K. Schittkowski (1981). *Test Examples for Nonlinear Programming Codes*, Volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag.

Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), 242–254.

Joines, J. and C. Houck (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 579–584. IEEE Press.

Keane, A. (1994, May 19). Genetic Algorithms Digest. V8n16.

Kelly, J. and M. Laguna (1996, April 8). Genetic Algorithms Digest. V10n16.

Leriche, R. G., C. Knopf-Lenoir, and R. T. Haftka (1995). A segragated genetic algorithm for constrained structural optimization. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pp. 558–565.

Maa, C. and M. Shanblatt (1992). A two-phase optimization neural network. *IEEE Transactions on Neural Networks 3*(6), 1003–1009.

Michalewicz, Z. (1995a). Genetic algorithms, numerical optimization and constraints. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms*, pp. 151–158. Morgan Kaufmann.

Michalewicz, Z. (1995b). Heuristic methods for evolutionary computation techniques. *Journal of Heuristics 1*(2), 177–206.

Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs*. New-York: Springer Verlag. 3rd edition.

Michalewicz, Z. and N. Attia (1994). Evolutionary optimization of constrained problems. In *Proceedings of the* 3^{rd} *Annual Conference on Evolutionary Programming*, pp. 98–108. World Scientific.

Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the* 4^{th} *International Conference on Genetic Algorithms*, pp. 151–157. Morgan Kaufmann.

Michalewicz, Z., T. Logan, and S. Swaminathan (1994). Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the* 3^{rd} *Annual Conference on Evolutionary Programming*, pp. 84–97. World Scientific.

Michalewicz, Z. and G. Nazhiyath (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In D. B. Fogel (Ed.), *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pp. 647–651. IEEE Press.

Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In P. J. Angeline and T. Bäck (Eds.), *Proceedings of the 5^{th} Annual Conference on Evolutionary Programming*. To appear.

Mühlenbein, H. and H.-M. Voigt (1995). Gene poool recombination for the breeder genetic algorithm. In *Proceedings of the Metaheuristics International Conference*, pp. 19–25.

Myung, H., J.-H. Kim, and D. Fogel (1995). Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the 4^{th} Annual Conference on Evolutionary Programming*, pp. 449–463. MIT Press.

Orvosh, D. and L. Davis (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest (Ed.), *Proceedings of the 5^{th} International Conference on Genetic Algorithms*, pp. 650. Morgan Kaufmann.

Paredis, J. (1994). Coevolutionary constraint satisfaction. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the 3^{rd} Conference on Parallel Problems Solving from Nature*, pp. 46–55. Springer Verlag.

Parmee, I. and G. Purchase (1994). The development of directed genetic search technique for heavily constrained design spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, pp. 97–102. University of Plymouth.

Powell, D. and M. M. Skolnick (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest (Ed.), *Proceedings of the 5^{th} International Conference on Genetic Algorithms*, pp. 424–430. Morgan Kaufmann.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag.

Renders, J.-M. and H. Bersini (1994). Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pp. 312–317. IEEE Press.

Reynolds, R. (1994). An introduction to cultural algorithms. In *Proceedings of the 3^{rd} Annual Conference on Evolutionary Programming*, pp. 131–139. World Scientific.

Reynolds, R., Z. Michalewicz, and M. Cavaretta (1995). Using cultural algorithms for constraint handling in Genocop. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the 4^{th} Annual Conference on Evolutionary Programming*, pp. 298–305. MIT Press.

Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Proceedings of the 3^{rd} International Conference on Genetic Algorithms*, pp. 191–197. Morgan Kaufmann.

Schaffer, D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the* 1^{st} *International Conference on Genetic Algorithms*. Laurence Erlbaum Associates.

Schoenauer, M. and Z. Michalewicz (1996). Evolutionary computation at the edge of feasibility. W. Ebeling, and H.-M. Voigt (Eds.), *Proceedings of the* 4^{th} *Conference on Parallel Problems Solving from Nature*, Springer Verlag.

Schoenauer, M. and S. Xanthakis (1993). Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 573–580. Morgan Kaufmann.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. New-York: John Wiley & Sons. 1995 – 2^{nd} edition.

Smith, A. and D. Tate (1993). Genetic optimization using a penalty function. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms*, pp. 499–503. Morgan Kaufmann.

Surry, P., N. Radcliffe, and I. Boyd (1995). A multi-objective approach to constrained optimization of gas supply networks. In T. Fogarty (Ed.), *Proceedings of the AISB-95 Workshop on Evolutionary Computing*, Volume 993, pp. 166–180. Springer Verlag.

Waagen, D., P. Diercks, and J. McDonnell (1992). The stochastic direction set algorithm: A hybrid technique for finding function extrema. In D. B. Fogel and W. Atmar (Eds.), *Proceedings of the* 1^{st} *Annual Conference on Evolutionary Programming*, pp. 35–42. Evolutionary Programming Society.

Wright, A. (1991). Genetic algorithms for real parameter optimization. In J. G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 205–218. Morgan Kaufmann.