# A Survey of Constraint Handling Techniques in Evolutionary Computation Methods

**Zbigniew Michalewicz**

**Abstract**

One of the major components of any evolutionary system is the evaluation function. Evaluation functions are used to assign a quality measure for individuals in a population. Whereas evolutionary computation techniques assume the existence of an (efficient) evaluation function for feasible individuals, there is no uniform methodology for handling (i.e., evaluating) unfeasible ones. The simplest approach, incorporated by evolution strategies and a version of evolutionary programming (for numerical optimization problems), is to reject unfeasible solutions. But several other methods for handling unfeasible individuals have emerged recently. This paper reviews such methods (using a domain of nonlinear programming problems) and discusses their merits and drawbacks.

## 1 INTRODUCTION

Evolutionary computation techniques have received considerable attention regarding their potential as optimization techniques for complex functions. Many difficult functions have been examined; often they served as test-beds for different selection methods, various operators, different representations, and so forth. But evolutionary computation techniques have not developed any guideliness on how to deal with unfeasible solutions. For example, in the area of numerical optimization, evolution strategies (e.g., Bäck et al. 1991) and evolutionary programming techniques (modified to handle numerical optimization problems, e.g., Fogel and Stayton 1994) simply reject unfeasible individuals. Genetic algorithms (Holland 1975), on the other hand, penalize unfeasible individuals (e.g., Goldberg 1989), however, there is no general rules for designing penalty functions. A few hypothesis were formulated in Richardson et al. (1989), but they are rather

general (e.g., "penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints"). At the same time, "because these problems occur frequently, performing constrained optimization with GA's is a very important area of research" (Richardson et al. 1989).

In evolutionary computation methods the evaluation function serves as the only link between the problem and the algorithm. The evaluation function rates individuals in the population: better individuals have better chances for survival and reproduction. In many cases the process of selection of an evaluation function is straightforward. For example, if one searches for a minimum value of a function

$$F2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2,$$

where $-2.048 < x_i \leq 2.048$, (this is the Rosenbrock function from De Jong 1975), then every individual (chromosome) is easily evaluated; for example:

$$v_1 = \langle 0.231, -1.892 \rangle, \text{ and } F2(v_1) = 379.034.^1$$

It is important to note that every individual from the search space defined by inequalities

$$-2.048 < x_i \leq 2.048, (i = 1, 2),$$

is feasible. But in most optimization problems there is a significant distinction between the *search space* and the *feasible search space*. For example, if the above problem has an additional constraint:

$$x_1^2 \leq x_1 + \frac{1}{x_2},$$

the individual $v_1$ will violate the constraint and consequently will not be feasible.

Any evolutionary computation technique applied to a particular problem should address the issue of handling unfeasible individuals. In general, a search space $\mathcal{S}$ consists of two disjoint subsets of feasible and unfeasible subspaces, $\mathcal{F}$ and $\mathcal{U}$, respectively. We do not make any assumptions about these subspaces; in particular, they need not be convex and they need not be connected (e.g., as it is the case in the example in Figure 1 where feasible part $\mathcal{F}$ of the search space consists of two disjoined subsets). In solving optimization problems we search for a *feasible* optimum. During the search process we have to deal with various feasible and unfeasible individuals; for example (see Figure 1), at some stage of the evolution process, a population

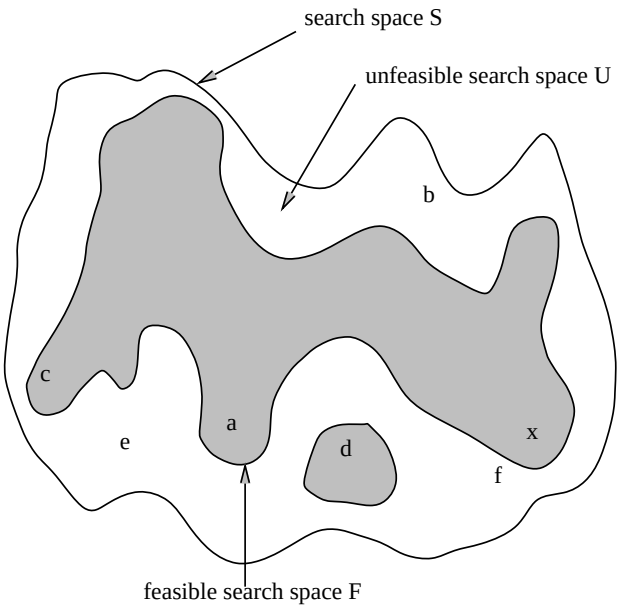


Figure 1: A search space and its feasible part

may contain some feasible (a, c, d) and unfeasible individuals (b, e, f), while the optimum solution is 'x'.

The problem of how to deal with unfeasible individuals is far from trivial. In general, we have to design two evaluations functions, $eval_f$ and $eval_u$, for feasible and unfeasible domains, respectively. There are many important questions to be addressed:

- how should two feasible individuals be compared, e.g., 'a' and 'c' from Figure 1? In other words, how should the function $eval_f$ be designed? (This is usually the easiest question: for most optimization problems, the evaluation function for feasible solutions is given, e.g., as function $F2$ in the earlier example).

- how should two unfeasible individuals be compared, e.g., 'b' and 'e'? In other words, how should the function $eval_u$ be designed?

- should we assume that $eval_f(s) \succ eval_u(p)$ for any $s \in \mathcal{F}$ and any $p \in \mathcal{U}$ (the symbol $\succ$ is interpreted as 'is better than', i.e., 'greater than' for maximization and 'smaller than' for minimization problems)? In other words, should we assume that *any* feasible solution is better than *any* unfeasible one? In particular (see Figure 1), which individual is better: feasible individual 'c' or unfeasible individual 'f' (note that the optimum is 'x')?

- should we consider unfeasible individuals harmful and eliminate them from the population? Or rather, should we consider them useful for helping the population to cross unfeasible regions and to arrive at the optimum point (e.g., from 'd' to 'x', Figure 1)?

- should we 'repair' unfeasible solutions by moving them into the closest point of the feasible space (e.g., the repaired version of 'f' might be optimum 'x', Figure 1)? In other words, should we assume that $eval_u(p) = eval_f(s)$, where 's' is a repaired version of 'p'? If so, should we replace 'p' by its repaired version 's' in the population or rather should we use a repair procedure for evaluation purpose only?

- since our aim is to find a feasible optimum solution, should we choose to penalize unfeasible individuals? In other words, should we extend the domain of function $eval_f$ and assume that $eval_u(p) = eval_f(p) + penalty(p)$? If so, how should such a penalty function $penalty(p)$ be designed?

Several trends for handling unfeasible solutions have emerged in the area of evolutionary computation. We discuss them in the following section using a domain of nonlinear programming problems. Section 3 provides additional comments and observations, and concludes the paper.

## 2 NUMERICAL OPTIMIZATION AND UNFEASIBLE SOLUTIONS

This section discusses several methods for handling unfeasible solutions for continuous numerical optimization problems. Richardson et al. (1989) claims: "Attempts to apply GA's with constrained optimization problems follow two different paradigms (1) modification of the genetic operators; and (2) penalizing strings which fail to satisfy all the constraints." This is not longer the case as a variety of methods have been proposed. Several of them are based on penalty functions, however, they differ in many important details on how the penalty function is designed and applied to unfeasible solutions. Other methods maintain the feasibility of the individuals in the population by means of specialized operators, impose a restriction that any feasible solution is 'better' than any unfeasible solution, consider constraints one at the time in a particular linear order, repair unfeasible solutions, use multiobjective optimization techniques, are based on cultural algorithms, or rate solutions using a particular co-evolutionary model. The following subsections define the nonlinear programming problem and discuss these methods in turn.

### Nonlinear programming problem

The general nonlinear programming problem for continuous variables is to find $\overline{X}$ so as to

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

where $\overline{X} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq R^n$ defines the search space and the set $\mathcal{F} \subseteq \mathcal{S}$ defines a *feasible* part of the search space. Usually, the search space $\mathcal{S}$ is defined as an $n$-dimensional rectangle in $R^n$ (domains of variables defined as lower and upper bounds):

$$left(i) \leq x_i \leq right(i), \quad 1 \leq i \leq n,$$

whereas the feasible set $\mathcal{F}$ is defined by the search space $\mathcal{S}$ and an additional set of constraints:

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

Most research on applications of evolutionary computation techniques to nonlinear programming problems have aimed at quite complex objective functions, however, the assumption was that $\mathcal{F} = \mathcal{S}$ (i.e., set of constraints is empty). Several test functions used by various researchers during the last 20 years considered only domains of $n$ variables; this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases, e.g., Wright (1991), Eshelman and Schaffer (1993), Fogel and Stayton (1994). In the following subsections we survey several techniques that have been developed for the case of $\mathcal{F} \subset \mathcal{S}$. All of these techniques use the objective function $f$ to evaluate a feasible individual, i.e.,

$$eval_f(\overline{X}) = f(\overline{X}), \text{ for } \overline{X} \in \mathcal{F}.$$

Most of these methods use also constraint violation measures $f_j$ (for the $j$-th constraint) for the construction of the $eval_u$; these functions are defined as

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

### The method of Homaifar, Lai, & Qi

Homaifar et al. (1994) assume that for every constraint we establish a family of intervals that determines appropriate penalty values. The method works as follows:

- for each constraint, create several ($\ell$) 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 (i.e., these individuals are feasible or unfeasible),
- evaluate individuals using the following formula

  $$eval(\overline{X}) = f(\overline{X}) + \sum_{j=1}^{m} R_{ij} f_j^2(\overline{X}),$$

  where $R_{ij}$ is a penalty coefficient for the $i$-th level of violation and the $j$-th constraint.

Note, that the function $eval$ is defined on $\mathcal{S}$, i.e., it serves both feasible and unfeasible solutions.

The weakness of the method is in the number of parameters: for $m$ constraints the method requires $m(2\ell + 1)$ parameters in total. In particular, for $m = 5$ constraints and $\ell = 4$ levels of violation, we need to set 45 parameters!

Recent experiments (Michalewicz 1995) indicate that the quality of solutions heavily depends on the values of these parameters. If the penalty coefficients $R_{ij}$ are moderate, the algorithm may converge to an unfeasible solution; this may happen if the value of the objective function (together with all penalties) for such unfeasible solution is still more attractive than values of the objective function for feasible solutions (see Michalewicz 1995 for an example of such a case). On the other hand, if the penalty coefficients $R_{ij}$ are "too large," the method is equivalent to rejecting unfeasible solutions.

It is quite likely that for a given problem there exists an optimal set of parameters for which the system would return a feasible near-optimum solution, however, it might be quite difficult to find it. It seems that the above method should be extended by an additional algorithm that determines all levels of violations and all penalty coefficients on the basis of several components; these include (1) the type of the objective function, (2) the number of variables, (3) number of constraints, (4) types of constraints, and (5) the ratio between the sizes of the feasible search space and the whole search space $|\mathcal{F}|/|\mathcal{S}|$.

### The method of Joines & Houck

Joines and Houck (1994) assume dynamic penalties; individuals are evaluated (at the iteration $t$) by the following formula:

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

where $C$, $\alpha$ and $\beta$ are constants. As in Homaifar et al. (1994), the function *eval* evaluates both feasible and unfeasible solutions.

The method is quite similar to Homaifar et al. (1994), but it requires many fewer parameters ($C$, $\alpha$ and $\beta$), and this is independent of the total number of constraints. Also, the penalty component is not constant but changes with the generation number. Instead of defining several levels of violation, the pressure on unfeasible 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 $t$), this component assumes large values.

The results of experiments (see Joines and Houck 1994) indicated that the quality of of the solution was very sensitive to changes in values of these three parameters. Also, additional experiments (e.g., Michalewicz 1995) with one particular setting ($C = 0.5$, $\alpha = \beta = 2$) resulted sometimes in early convergence of the algorithm to either an unfeasible solution (which was more attractive than feasible ones), or to a feasible solution which was far away from the global optimum. It seems that the penalty components (being constantly increased through the growing value of the generation number) change the objective function in a significant way. Once the population is trapped in a feasible (or unfeasible) local optimum, it may stay there forever. It is interesting to note that in most experiments (Michalewicz 1995) the algorithm converged in early generations.

### The method of Michalewicz & Janikow

Michalewicz and Janikow (1991) assume linear constraints only and a feasible starting point (or feasible initial population). A closed set of operators maintains feasibility of solutions. For example, when a particular component $x_i$ of a solution vector $\overline{X}$ 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 $\overline{X}$) 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

$$a\overline{X} + (1 - a)\overline{Y}$$

of two feasible solution vectors $\overline{X}$ and $\overline{Y}$ 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 $\mathcal{F}$). Consequently, there is no need to define the function $eval_u$; the function $eval_f$ is (as usual) the objective function $f$.

The method can be generalized to handle nonlinear constraints provided that the resulting feasible search space $\mathcal{F}$ is convex. The method does not require any special parameters apart from standard parameters for any evolutionary system (like population size, probabilities of operators, etc.) It gave surprisingly good performance on many test functions (see, for example, Michalewicz et al. 1994). But the weakness of the method lies in its inability to deal with nonconvex search spaces (i.e, to deal with nonlinear constraints in general).

### The method of Michalewicz & Attia

Michalewicz and Attia (1994) take advantage of the previous method: linear and nonlinear constraints are processed separately. The method 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 all linear constraints,
- create a set of active constraints $A$; include there all nonlinear equations and all violated nonlinear inequalities.
- set the initial temperature $\tau = \tau_0$,
- evolve the population using the following formula:

  $$eval(\overline{X}, \tau) = f(\overline{X}) + \frac{1}{2\tau} \sum_{j \in A} f_j^2(\overline{X}),$$

  (only active constraints are considered),
- if $\tau < \tau_f$, stop, otherwise
  - decrease temperature $\tau$,
  - the best solution serves as a starting point of the next iteration,
  - update the set of active constraints $A$,
  - repeat the previous step of the main part.

This is the only method described here which distinguishes between linear and nonlinear constraints. As in the previous method, the algorithm maintains feasibility of all linear constraints using a set

of closed operators. At every iteration the algorithm considers active constraints only, the pressure on unfeasible solutions is increased due to the decreasing values of temperature $\tau$.

The method has an additional unique feature: it starts from a single point (this feature, however, is not essential. The only important requirement is that the next population contains the best individual from the previous population). Consequently, it is relatively easy to compare this method with other classical optimization methods whose performance is tested (for a given problem) from some starting point.

The method requires 'starting' and 'freezing' temperatures, $\tau_0$ and $\tau_f$, respectively, and the cooling scheme to decrease temperature $\tau$. Several experiments provided good results for many test functions, e.g., Michalewicz and Attia (1994), Michalewicz (1995), however, the method is quite sensitive to values of its parameters. Some experiments (Michalewicz and Attia 1994) indicated that the system may converge to a near-optimum solution just in one iteration (i.e., for one temperature $\tau = \tau_0$), in a few iterations, or many iterations (even for problems with two variables only). Other experiments gave different results for different cooling schemes. The question of how to settle these parameters for a particular optimization problem remains open.

### The method of Powell & Skolnick

Powell and Skolnick (1993) incorporate a heuristic rule (suggested earlier by Richardson et al. 1989) for processing unfeasible solutions: "every feasible solution is better than every unfeasible solution." This rule is implemented in the following way: evaluations of feasible solutions are mapped into the interval $(-\infty, 1)$ and unfeasible solutions — into the interval $(1, \infty)$ (for minimization problems). This is equivalent (for ranking and tournament selection methods) to the following evaluation procedure:

$$eval_f(\overline{X}) = f(\overline{X}),$$
$$eval_u(\overline{X}) = f(\overline{X}) + r \sum_{j=1}^{m} f_j(\overline{X}),$$

where $r$ is a constant, and

$$eval(\overline{X}) = \begin{cases} eval_f(\overline{X}), & if\ \overline{X} \in \mathcal{F} \\ eval_u(\overline{X}) + \rho(\overline{X}, t), & if\ \overline{X} \in \mathcal{S} - \mathcal{F}. \end{cases}$$

The function $\rho(\overline{X}, t)$ influences unfeasible solutions only; it is defined as

$$\rho(\overline{X}, t) = \max\{0, \max_{\overline{X} \in \mathcal{F}}\{eval_f(\overline{X})\} - \min_{\overline{X} \in \mathcal{S}-\mathcal{F}}\{eval_u(\overline{X})\}.$$

In other words, unfeasible individuals have increased penalties: they may not be better than the worst ($\max_{\overline{X} \in \mathcal{F}} \{eval_f(\overline{X})\}$) feasible individual.

The method requires just one parameter $r$. But the key concept behind this method is the assumption of superiority of feasible solutions over unfeasible ones. The usefulness of this assumption can constitute an interesting point for discussion and various experiments. There is no doubt that for many optimization problems the assumption works very well (see, for example, Powell and Skolnick (1993) for experimental results from a numerical optimization domain, and Michalewicz and Xiao (1995) for experimental results from a mobile robot domain), however, the topology of the feasible search space might be an important factor here. Several recent experiments (Michalewicz 1995) indicate that for problems with a small ratio $|\mathcal{F}|/|\mathcal{S}|$ the algorithm is often trapped into an unfeasible solution. The method should require at least one feasible individual to be placed in the initial population (this would be similar to providing a starting feasible point for the optimization process) or feasible initial population (in that case, however, unfeasible individuals are practically removed from the population due to the selection process). The question on the influence of a single feasible individual of the initial population on the quality of the final result remains open.

### The method of Schoenauer & Xanthakis

Another approach is based on the idea of handling constraints in a particular order; Schoenauer and Xanthakis (1993) called this method a "behavioural memory" approach.

The initial steps of the method are devoted to sampling the feasible region; only in the final step the objective function $f$ is optimized.

- start with a random population of individuals (i.e., these individuals are feasible or unfeasible),
- set $j = 1$ ($j$ is constraint counter),
- evolve this population to minimize the violation of the $j$-th constraint, until a given percentage of the population (so-called flip threshold $\phi$) is feasible for this constraint. In this case

  $$eval(\overline{X}) = g_1(\overline{X}).$$

- set $j = j + 1$,
- the current population is the starting point for the next phase of the evolution, minimizing the violation of the $j$-th constraint:

$$eval(\overline{X}) = g_j(\overline{X}).^2$$

During this phase, points that do not satisfy at least one of the 1st, 2nd, ... ,$(j-1)$-th constraints are eliminated from the population. The halting criterion is again the satisfaction of the $j$-th constraint by the flip threshold percentage $\phi$ of the population.

- if $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the objective function $f$ rejecting unfeasible individuals.

The method requires that there is a linear order of all constraints; these constraints are processed in turn. It is unclear what is the influence of the order of constraints on the results of the algorithm; experiments (Michalewicz 1995) indicated that different orders provide different results (different in the sense of the total running time and precision). Also, the authors recommended a sharing scheme (to maintain diversity of the population). In total, the method requires 3 parameters: the sharing factor $\sigma$, the flip threshold $\phi$, and a particular permutation of constraints, which determine their order.

The method has a few merits. One of them is that in the final step of the algorithm the objective function $f$ is optimized (as opposed to its modified form). But for larger feasible spaces the method just provides additional computational overhead, and for very small feasible search spaces it is essential to maintain a diversity in the population. As Schoenauer and Xanthakis (1993) wrote: "We do not claim to outperform all other methods for constraints handling using GAs. In particular when feasible region is large, using penalty function may be a cheaper strategy. [...] But in many problems, like in engineering optimization for instance, the feasible region is small and quite sparse in the whole search space [...]." Recent experiments (Michalewicz 1995) indicate that the method provides a reasonable performance except when the feasible search space is "too small": in such cases the method is likely to fail (due to computational effort to generate feasible solutions).

### Rejection of unfeasible individuals

This "death penalty" method is a popular option in many evolutionary techniques like evolution strategies or evolutionary programming. The method of eliminating unfeasible solutions from a population may work reasonably well when the feasible search space $\mathcal{F}$ is convex and it constitutes a reasonable part of the whole search space (e.g., evolution strategies do not allow equality constraints since with such

constraints the ratio between the sizes of $\mathcal{F}$ and $\mathcal{S}$ is zero). Otherwise such an approach has serious limitations. For example, for problems where the ratio between the sizes of $\mathcal{F}$ and $\mathcal{S}$ is small and an initial population consists of unfeasible individuals only, it might be essential to improve them (as oppose to 'reject' them). Moreover, quite often the system can reach the optimum solution easier if it is possible to "cross" an unfeasible region (especially in non-convex feasible search spaces).

The method of rejection of unfeasible individuals was recently tested (Michalewicz 1995) for several numerical optimization problems, where the ratio of $|\mathcal{F}|/|\mathcal{S}|$ was between 0% and 0.5% for all (five) test cases. As expected, the method performed worse than other methods discussed earlier, despite its additional advantage of starting from a feasible initial population. It seems that limiting the search to feasible part of the search space does not always enhance the search. Similar observation was made in connection with the method of Michalewicz and Attia (1994); in this method the search was limited to a feasible search space with respect to linear constraints. Surprisingly, for one test case (which consisted of linear and nonlinear inequalities) the results were worse than results of methods which considered the whole search space and just penalized unfeasible solutions without any distinction between linear and nonlinear constraints. This result confirms an observation by Richardson et al. (1989): "Many seem to believe that penalty functions should be harsh, so that the GA will avoid the forbidden spaces. The foundations of GA theory, however, say that GA's optimize by combining partial information from all the population. Therefore, the unfeasible solutions should provide information and not just be thrown away."

### Repair methods

Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) it is relatively easy to 'repair' an unfeasible individual. Such repaired version can be used either for evaluation only, i.e.,

$$eval_u(\overline{X}) = eval_f(\overline{Y}),$$

where $\overline{Y}$ is a repaired (i.e., feasible) version of $\overline{X}$, or it can also replace the original individual in the population (with some probability). Recently (see Orvosh and Davis 1993) a so-called 5%-rule was reported: 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 unfeasible originals.

However, the author is not aware of any evolutionary computation techniques for numerical optimization problems which repair unfeasible individuals (whether for evaluation or a replacement). Clearly, there are some possibilities here. One can incorporate some constraint satisfaction methods and/or classical optimization techniques to determine a feasible (not necessarily closest in the Euclidean sense) point $\overline{Y}$ of a given unfeasible solution $\overline{X}$. It would be interesting to experiment with different repair algorithms and different replacements ratios for numerical problems with various characteristics (e.g., number of variables, types of constraints, relative size of the feasible search space, etc.)

### Multi-objective optimization methods

One possible constraint handling technique may utilize multi-objective optimization methods, where the objective function $f$ and constraint violation measures $f_j$ constitute a $(m+1)$-dimensional vector $\vec{v}$:

$$\vec{v} = (f, f_1, \ldots, f_m).$$

Using some multi-objective optimization method, we can attempt to minimize its components: an ideal solution $\overline{X}$ would have $f_i(\overline{X}) = 0$ for $1 \leq i \leq m$ and $f(\overline{X}) \leq f(\overline{Y})$ for all $\overline{Y} \in \mathcal{F}$.

The classical methods for multiobjective optimization include a method of objective weighting, where multiple objective functions $f_j$ are combined into one overall objective function $eval$:

$$eval(\overline{X}) = \sum_{j=0}^{m} w_j f_j(\overline{X}),$$

where $f_0 \equiv f$, the weights $w_j \in [0..1]$ and $\sum_{j=0}^{m} w_j = 1$. Different weight vectors provide different Pareto-optimal solutions. Another method (method of distance functions) combines multiple objective functions into one on the basis of demand-level vector $\overline{Y}$:

$$eval(\overline{X}) = (\sum_{j=0}^{m} |f_i(\overline{X}) - y_i|^r)^{\frac{1}{r}},$$

where (usually) $r = 2$ (Euclidean metric). But these classical methods applied to constrained optimization problems are equivalent to penalty approaches.

It is also possible to experiment with evolutionary techniques for multi-objective optimization, e.g., with Schaffer's VEGA (Vector Evaluated Genetic Algorithm) system for multi-objective optimization (Schaffer 1984). The main idea behind the VEGA system was a division of the population into (equal sized) subpopulations; each subpopulation was "responsible" for a single objective. The selection procedure was performed independently for each objective, but crossover was performed across subpopulation boundaries. Additional heuristics were developed (e.g., wealth redistribution scheme, crossbreeding plan) and studied to decrease a tendency of the system to converge towards individuals which were not the best with respect to any objective. But analysis of VEGA shows (Richardson et al. 1989) that the effect is the same as if fitness were a linear combination of $f_i$'s.

Recently, Srinivas and Deb (1993) proposed a technique, NSGA, (Nondominated Sorting Genetic Algorithm), which is based on several layers of classifications of the individuals. Before the selection is performed, the population is ranked on the following basis: all nondominated individuals are classify into one category (with a dummy fitness value, which is proportional to the population size, to provide an equal reproductive potential for these individuals). To maintain the diversity of the population, these classified individuals are shared with with their dummy fitness values (see previous subsection). Then this group of classified individuals are ignored and another layer of nondominated individuals is considered. The process continues until all individuals in the population are classified.

It might be interesting to experiment with such techniques in the context of constrained numerical optimization; the autor is not aware of any results in this area.

### The method of Paredis

An interesting approach was recently 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. It means, that individuals from the population of solutions are considered from the whole search space $\mathcal{S}$, and that there is no distinction between feasible and unfeasible individuals (i.e., there is only one evaluation function $eval$ without any split into $eval_f$ or $eval_u$). The value of $eval$ is determined on the basis of constraint violations measures $f_j$'s;[3] however, better $f_j$'s (e.g., active constraints) would contribute more towards the value of $eval$.

It would be interesting to adopt this approach to constrained numerical optimization problems and compare it with the other methods. But the major difficulty to be resolved in such adaptation seems very much the same as in many other methods: how to balance the pressure of feasibility of a solution with the pressure to minimize the objective function.

### Use of cultural algorithms

The research on cultural algorithms (Reynolds 1994) was triggered by observations that culture might be another kind of inheritance system. But it is not clear what the appropriate structures and units to represent the adaptation and transmission of cultural information are. Neither it is clear how to describe the interaction between natural evolution and culture. Reynolds developed a few models to investigate the properties of cultural algorithms; in these models, the belief space is used to constrain the combination of traits that individuals can assume. Changes in the belief space represent macroevolutionary change and changes in the population of individuals represent microevolutionary change. Both changes are moderated by the communication link.

The general intuition behind belief spaces is to preserve those beliefs associated with "acceptable" behavior at the trait level (and, consequently, to prune away unacceptable beliefs). The acceptable beliefs serve as constraints that direct the population of traits. It seems that the cultural algorithms may serve as a very interesting tool for numerical optimization problems, where constraints influence the search in a direct way (consequently, the search in constrained spaces may be more efficient than in unconstrained ones!). Very recently Reynolds et al. (1995) investigated a possibility of applying cultural algorithms for constrained numerical optimization. The first experiments indicate a great potential behind this approach.

## 3 FURTHER DISCUSSION

The previous section surveyed several constraint handling methods for numerical optimization problems. A few of these methods share some similarities, however, the majority of these methods are based on different methodologies; they can be classified into several categories:

- methods based on penalty functions,
- methods based on rejection of unfeasible individuals,

- methods based on specialized operators,
- methods based on the assumption of the superiority of feasible solutions over unfeasible solutions,
- methods based on behavioral memory,
- methods based on repair algorithms,
- methods based on multi-objective optimization techniques,
- methods based on co-evolutionary models, and
- methods based on cultural algorithms.

It seems that the majority of methods proposed for constraint handling for the continuous numerical optimization problems are based on penalty functions. In general, we can classify penalty functions into two classes: (1) static penalties, where penalties are functions of the degree of violation of constraints (e.g., the method of Homaifar et al.); and (2) dynamic penalties, where penalties are functions of the degree of violation of constraints as well as the generation number $t$ (e.g., Joines & Houck 1994 and Michalewicz & Attia 1994). In addition, a promising direction for applying penalty functions is the use of adaptive penalties: penalty factors can be incorporated in the chromosome structures in a similar way as some control parameters are represented in the structures of evolution strategies and evolutionary programming. All the above penalties are based on the degree of constraint violation, however, this need not be always the case. It might be worthwhile to experiment with methods where penalties are based rather on the distance between a point and the feasible search space: $penalty(\overline{X}) = dist(\overline{X}, \mathcal{F})$—such methods provide better results in many combinatorial optimization problems (Richardson et al. 1989). The appropriate choice of the penalty function may depend on (1) the ratio between sizes of the feasible and the whole search space, (2) the topological properties of the feasible search space, (3) the type of the objective function, (4) the number of variables, (5) number of constraints, (6) types of constraints, and (7) number of active constraints at the optimum. Thus the use of penalty functions is not trivial and only some partial analysis of their properties (e.g., Richardson et al. 1989, Siedlecki and Sklanski 1989) is available.

The rejection methods (death penalty methods) do not belong to the category of penalty-based methods, since they do not construct $eval_u$ at all. These methods are not concerned with the basic problem of all penalty approaches: how to design an evaluator $eval_u$

that balances the preservation of information with the pressure for feasibility.

The penalty approaches define $eval_u$ on the basis of $eval_f$. It might be worthwhile to experiment with an *independent* evaluation function $eval_u$ for unfeasible individuals. The function may take several parameters into account, e.g., number of violated constraints, the amount of violation, the distance from the feasible region (assuming that some metric is introduced), and so forth. There are two main issues to be resolved here. First, it is necessary to build an evaluation function that would distinguish between two unfeasible individuals in a meaningful way. Secondly, we should be able to compare a feasible and unfeasible solutions, since different functions would apply for their evaluations ($eval_f$ and $eval_u$, respectively). We discuss briefly these two issues in turn.

It is difficult to compare two unfeasible solutions; this is generally true for most optimization problems (e.g., scheduling, traveling salesman problem, path evaluation, numerical optimization). There are a few possibilities for constructing evaluation functions for unfeasible individuals; an evaluation function $eval_u$ may (1) count the number of violations for a given solution, (2) consider the 'amount' of unfeasibility in terms of constraint violation measures $f_j$'s, or (3) compute the effort of 'repairing' the individual. It seems that in the area of numerical optimization, so far only the second approach has been examined.

As mentioned earlier, it is important to address the issue of comparing feasible and unfeasible solutions. This issue requires the answer for the following question: "Is it possible that some unfeasible solution is 'better' than some feasible one?" It seems that the (somewhat risky) answer 'no' would help us in such comparisons, e.g., we can increase the value of $eval_u$ (i.e., making it less attractive) for any unfeasible individual $\overline{X}$ by a constant (within a given generation of the evolutionary process); this constant represents the difference in values between the best unfeasible and the worst feasible individuals. This was precisely the approach of the method by Powell & Skolnick. The answer 'yes' for the above question (i.e., allowing some unfeasible solutions be better than some feasible ones) may lead to complex calculations. Moreover, judging from difficulties in comparing two unfeasible solutions, any proposed method would have its drawbacks.

The methods which incorporate specialized operators usually explore some regularities of the feasible search space. For example, the method of Michalewicz & Janikow takes advantage of the properties of convex feasible spaces $\mathcal{F}$: (1) for any internal point $\overline{X}$ and any

line $v$ such that $\overline{X} \in v$, the line line $v$ intersects $\mathcal{F}$ in precisely two points (which determine the left and right boundaries of a domain for a variable — component of vector $\overline{X}$ — being mutated); and (2) for any $\overline{X} \in \mathcal{F}$ and $\overline{Y} \in \mathcal{F}$, their linear combination $a\overline{X} + (1-a)\overline{Y} \in \mathcal{F}$ for $0 \leq a \leq 1$. The methods that use specialized operators are quite effective, but it is quite difficult to generalize them for arbitrary feasible search spaces.

It seems that many other methods have an interesting potential for constrained numerical optimization problems. The methods that repair unfeasible individuals (whether for evaluation only or for both, evaluation and replacement) deserve much greater attention. As discussed previously, it would be interesting to experiment with different repair algorithms and different replacements ratios for numerical problems with various characteristics. It is also worthwhile to experiment further with methods based on multi-objective optimization models, co-evolution models, and cultural algorithms.

### Footnotes

[1] In genetic algorithms, the individual $v_1$ is represented as a binary string; in this particular case each variable is represented as a string of 12 bits, where the strings (000000000000) and (111111111111) correspond to the left and right boundaries of the domain, i.e., $-2.047$ and 2.048, respectively. In this representation,

$x_1 = 0.231 = (1000011100110), x_2 = -1.892 = (000010011011)$

and consequently $v_1 = (100001110011000000100110 11)$.

[2] To simplify notation, we do not distinguish between inequality constraints $g_j$ and equations $h_j$; all $m$ constraints are denoted by $g_j$.

[3] In the original approach, the author proposes so-called life-time fitness evaluation, where the score of an individual is defined as the sum of the payoffs it received during some number of last evaluations.

### Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant IRI-9322400. The author wishes to thank David Fogel for his constructive comments.

### References

Bäck, T., F. Hoffmeister and H.-P. Schwefel (1991). A Survey of Evolution Strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 2–9.

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

De Jong, K.A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).

Eshelman, L.J. and J.D. Schaffer (1993). Real-Coded Genetic Algorithms and Interval Schemata. In *Foundations of Genetic Algorithms – 2*, ed. D. Whitley, Los Altos, CA, Morgan Kaufmann, 187–202.

Fogel, D.B. and L.C. Stayton (1994). On the Effectiveness of Crossover in Simulated Evolutionary Optimization. *BioSystems*, **32**: 171–182.

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

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

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

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

Homaifar, A., S. H.-Y. Lai and X. Qi (1994). Constrained Optimization via Genetic Algorithms. *Simulation*, **62**: 242–254.

Joines, J.A. and C.R. Houck (1994). On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs. In *Proceedings of the Evolutionary Computation Conference—Poster Sessions*, part of the IEEE World Congress on Computational Intelligence, Orlando, 26–29 June 1994, 579–584.

Michalewicz, Z., (1995). Genetic Algorithms, Numerical Optimization, and Constraints. Submitted for publication, 1995.

Michalewicz, Z. and N. Attia (1994). In Evolutionary Optimization of Constrained Problems. *Proceedings of the 3rd Annual Conference on*

*Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, River Edge, NJ, World Scientific Publishing, 98–108.

Michalewicz, Z. and C. Janikow (1991). Handling Constraints in Genetic Algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 151–157.

Michalewicz, Z., T.D. Logan and S. Swaminathan (1994). Evolutionary Operators for Continuous Convex Parameter Spaces. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, River Edge, NJ, World Scientific Publishing, 84–97.

Michalewicz, Z. and J. Xiao (1995). Evaluation of Paths in Evolutionary Planner/Navigator. Submitted for publication.

Orvosh, D. and L. Davis (1993). Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 650.

Paredis, J. (1994). Co-evolutionary Constraint Satisfaction. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, New York, Springer-Verlag, 46–55,

Powell, D. and M.M. Skolnick (1993). Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 424–430.

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

Reynolds, R.G. (1994). An Introduction to Cultural Algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, River Edge, NJ, World Scientific, 131–139.

Reynolds, R.G., Z. Michalewicz and M. Cavaretta (1995). Using Cultural Algorithms for Constraint Handling in Genocop. *Proceedings of the 4th Annual Conference on Evolutionary Programming*, San Diego, CA, March 1–4, (this volume).

Richardson, J.T., M.R. Palmer, G. Liepins and M. Hilliard (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 191–197.

Schaffer, J.D. (1984). Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms. Doctoral dissertation, Vanderbilt University.

Schoenauer, M., and S. Xanthakis (1993). Constrained GA Optimization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 573–580.

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

Siedlecki, W. and J. Sklanski (1989). Constrained Genetic Optimization via Dynamic Reward–Penalty Balancing and Its Use in Pattern Recognition. In *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, Morgan Kaufmann Publishers, 141–150.

Srinivas, N. and K. Deb (1993). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India.

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