Journal of Global Optimization **27:** 427–446, 2003.

Numerical Comparison of Some Penalty-Based Constraint Handling Techniques in Genetic Algorithms

KAISA MIETTINEN, MARKO M. MÄKELÄ and JARI TOIVANEN
Department of Mathematical Information Technology, P.O. Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland (e-mail: miettine@mit.jyu.fi, makela@mit.jyu.fi, tene@mit.jyu.fi)

(Received 14 August 2000; accepted in revised form 22 April 2003)

Abstract. We study five penalty function-based constraint handling techniques to be used with genetic algorithms in global optimization. Three of them, the method of superiority of feasible points, the method of parameter free penalties and the method of adaptive penalties have already been considered in the literature. In addition, we introduce two new modifications of these methods. We compare all the five methods numerically in 33 test problems and report and analyze the results obtained in terms of accuracy, efficiency and reliability. The method of adaptive penalties turned out to be most efficient while the method of parameter free penalties was the most reliable.

Key words: constrained optimization, genetic algorithms, global optimization, penalty functions

1. Introduction

Genetic algorithms are well-known stochastic methods of global optimization based on the evolution theory of Darwin (see, for example, [10]). They have successfully been applied in different real-world applications, for example, in aeronautics, electrical engineering, scheduling and signal processing (see [26, 27], among others).

In their basic form, genetic algorithms are not capable of handling constraint functions limiting the set of feasible solutions. Therefore, some additional methods are needed to keep the solutions in the feasible region. The constraint handling methods can roughly be divided into two classes depending on whether penalty functions are utilized or not. For more detailed classifications, see [5, 9, 18, 19, 23] and references therein.

Many methods not dealing with penalty functions are mainly problem-dependent or they are restricted to certain types of functions. For a collection of such methods, see for example, [23] and references therein. One type of such methods are so-called repair operators. Infeasible solutions can, for example, be projected into the feasible region in linear problems or the consideration can be restricted to the boundary of the feasible region (for the latter, see [30]).

As a rule, in constraint handling methods based on penalty functions, a penalty term is added to the objective function penalizing the function values outside the feasible region. Summaries of these methods are given, for example, in [3, 9, 18, 23]. This type of methods are popular because they are easy to implement on top of an underlying optimizer (like genetic algorithms). Different penalty-based methods have been suggested in [3, 6, 9, 12, 14, 20, 32], among others. An alternative to traditional penalty methods is the segregated genetic algorithm in [16], which uses two penalty parameters and splits the population into two. Let us also mention a behavioural memory-based method in several phases suggested in [31]. In this method, the feasible region is first sampled by minimizing some constraint violation and then the resulting population is used to optimize the objective function.

Other related methods include approaches utilizing augmented Lagrangian functions [1] and multiobjective optimization. Multiobjective optimization methods can treat the constraint functions as additional objective functions. This approach can utilize penalty functions to form a second function to be minimized. Alternatively, all the constraint functions can be treated separately (see [4], among others). One more constraint handling approach is stochastic ranking [29]. In this method, both over- and underpenalization are avoided by adjusting the penalization stochastically.

We decided to restrict our consideration to penalty function-based methods where the existing implementation of genetic algorithms could be utilized by modifying the fitness function only. Besides the easiness of implementation, utilizing the same underlying genetic algorithm facilitated the comparability of the methods. In this paper, we numerically compare three methods, the method of superiority of feasible points [23, 28], the method of parameter free penalties [7] and the method of adaptive penalties [11, 23]. We also introduce two modifications of the adaptive methods and propose a new formulation to the method of parameter free penalties.

An important criterion in selecting the constraint handling methods to be considered here was the generality of the approach. In other words, the method should be as problem-independent as possible to be applicable to a wide range of optimization problems. The selected methods were numerically compared by solving a variety of 33 test problems from the literature.

The results of this work are to be used in selecting a suitable constraint handling technique for global optimization with genetic algorithms. This technique is needed in a multiobjective optimization system as an underlying solver. The system, called WWW-NIMBUS (see, for example, [24, 25]), is based on interactive co-operation between a human decision maker and an optimizer. The interaction is a way of searching for the best compromise between several conflicting goals and a global optimizer is an essential part in the success of the search. In NIMBUS, a part of the human input is modelled as additional constraints. This explains the need of a reliable constraint handling technique.

This paper is organized as follows. In Section 2 we consider the optimization problem and genetic algorithms in general. Five constraint handling techniques are described in Section 3. Test problems and results are introduced in Section 4, and the paper is concluded with a discussion in Section 5.

2. Elements of Genetic Algorithms

We consider an optimization problem of the form

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad x \in S, \end{aligned} \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function* and $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ is a vector of *variables*. The *feasible region* $S \neq \emptyset$ is bounded by box constraints and m inequality constraints, that is,

$$S = \{x \in \mathbb{R}^n \mid x^l \leqslant x \leqslant x^u \text{ and } g_j(x) \leqslant 0 \text{ for } j = 1, \ldots, m\}, \tag{2.2}$$

where $x^l, x^u \in \mathbb{R}^n$ and $g_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \ldots, m$.

We are looking for the *global minimum* (assuming it exists) of problem (2.1), in other words, the point $x^* \in S$ such that

$$f(x^*) \leqslant f(x) \text{ for all } x \in S.$$

In nonconvex problems, both locally and globally optimal solutions may exist. Then, special methods of global optimization are needed in order to find the global ones. Global optimization methods can be divided into deterministic and stochastic ones. Deterministic methods are usually based on some special assumptions on the problem to be solved (see, for example, [13]), whereas stochastic methods utilize randomness. Because of their general nature, stochastic methods work even with discontinuous functions. Genetic algorithms represent this type of methods. In what follows, we present them in the form implemented for our testing purposes (to be reported in Section 4).

The basic idea behind genetic algorithms is to artificially imitate the evolution process of nature. The algorithms are based on the evaluation of a set of solutions, called a *population*. The population is treated with genetic operators.

At the iteration (in other words, generation) i the population X^i consists of a number of N *individuals* x^j, that is, solutions, where N is called a *population size*. Typically, the population is initialized by randomly generated individuals.

When individuals are encoded using real numbers the corresponding methods are called *real-coded* genetic algorithms. Each individual $x^j = (x_1^j, \ldots, x_n^j)^T$ is a vector of variables where each variable is a real number. The suitability of an individual is determined by the value of a so-called *fitness function* based on the objective function. Note that in this paper we are minimizing the fitness function.

The next generation is created by the genetic operators *selection*, *crossover* and *mutation*. Parents are chosen by selection and new offsprings are produced with crossover and mutation. All these operators include randomness. *Elitism* guarantees that the current best solutions are not lost by copying the best individuals of the old population as such to the new population. All the above-mentioned operators can be realized in many different ways. Here we introduce only those that have been used in our tests.

Let us in this section assume that the only constraints in problem (2.1) are lower and upper bounds, that is, $m = 0$ in (2.2).

2.1. SELECTION

In the selection process, the best individuals are chosen as parents for the crossover operator. In the so-called *tournament selection*, the best of a number of randomly selected individuals is to be chosen as one parent. A parameter called *tournament size* determines how many individuals are compared when selecting a parent. Among the other possibilities for selecting parents is, for example, roulette-wheel selection, see [10].

2.2. CROSSOVER

In crossover, the parents chosen in the selection phase are cross-bred in order to create offsprings forming the next generation. This is realized by exchanging information between the selected parents whenever a randomly generated number is less than a pre-specified parameter, called *crossover rate*.

Crossover can be done in many different ways. A brief description can be found, for example, in [22]. In the *single-point crossover*, information is exchanged starting from a randomly generated crossover point. As the real variables form a string, the crossover point is located between two variables. Notice that no crossover takes place in this method if the problem has only one variable. In this case, mutation is the only genetic operator that can modify the individuals.

In *uniform crossover*, variables from the parents are selected with equal probability to form offsprings whereas linear combinations of the parents are generated in *arithmetic crossover*. *Heuristic crossover* produces only one offspring y (or none at all) from the parents x^1 and x^2 with the formula

$$y = r(x^2 - x^1) + x^2,$$

where r is a random number between 0 and 1 and the parent x^2 is not worse than the parent x^1. If the new offspring does not satisfy the box constraints, a new random number is generated. If required, this process is repeated up to four times (the number of tries can naturally be changed).

In the experiments performed, heuristic crossover produced best results (when the four crossovers described above where compared) and was, thus, used in the final experiments.

2.3. MUTATION

Mutation means that the new offsprings are modified with some probability determined by a parameter, called a *mutation rate*. The real numbers in the real-coded string are gone through one by one and a random number is generated for each of them. If the random number is lower than the mutation rate, the mutation takes place.

In real-coding, the following algorithm is performed (see [17]). The corresponding real number is denoted by x_i and the mutated counterpart by $\hat{x}_i$. To start with, a new random number $s \in [0, 1]$ is generated.

1. Set $t = (x_i - x_i^l)/(x_i^u - x_i^l)$.
2. Compute $\hat{t}$, where

$$\hat{t} = \begin{cases} t - t\left(\frac{t-s}{t}\right)^p & \text{if } s < t \\ t & \text{if } s = t \\ t + (1-t)\left(\frac{s-t}{1-t}\right)^p & \text{if } s > t. \end{cases}$$

3. Set $\hat{x}_i = (1 - \hat{t}\,)x_i^l + \hat{t}x_i^u$.

The parameter p, called a *mutation exponent*, defines the distribution of the mutation. The probability of small mutations increases when $p > 1$.

Note that the mutation rate must not be too high. Otherwise, the mutation might turn the process into a pure random search. Other possibilities to carry out mutation include Gaussian and uniform mutation (see, for example [23]).

2.4. ELITISM

Elitism guarantees that the objective function values do not increase from one generation to another. This is realized by copying the best individuals to the next generation. *Elitism size* is the parameter for the number of the best individuals to be copied.

2.5. GENETIC ALGORITHM

The basic genetic algorithm can be presented in the form

1. Set population size, tournament size, crossover rate, mutation rate, mutation exponent and elitism size. Set the parameters of the stopping criterion.
2. Initialize the population with random numbers.

3. Compute the fitness function values. Perform selection, crossover, mutation and elitism in order to create a new population.
4. If the stopping criterion is not satisfied, return to step 3. Otherwise, choose the best individual found as the final solution.

As a stopping criterion, one can fix the maximum number of iterations, that is, generations. In addition, one can set a tolerance to the difference between the best results of a fixed number of iterations. Besides the maximum number of iterations, in our numerical experiments, the solution process was stopped if the difference between the best fitness function values of the last hundred iterations was less than 0.01.

3. Handling Constraints

In their basic form, genetic algorithms can only handle box constraints. Naturally, it is always possible to reject those randomly generated points that violate the given constraints (see for example, [33]). However, this may be a very ineffective way, in particular, for problems where feasible solutions are difficult to be found.

Constraint functions can be taken into consideration by using penalty functions. This means that the constraints are added to the objective function in one way or another. Leaving the feasible region is penalized by increasing the fitness function value with a penalty term multiplied by a so-called penalty coefficient.

In this paper, we test some penalty-based methods. Let us first consider their basic idea. In *penalty function methods*, the function to be minimized is of the form

$$f(x) + r_i\Big(\sum_{j=1}^{m} \max\big[0, g_j(x)\big]^q\Big), \tag{3.1}$$

where $r_i > 0$ is an iteration-dependent *penalty coefficient*, i is the iteration number and $q \geqslant 1$ (see for example, [2]). If $q = 1$, the method is called an *exact penalty function method* and a *quadratic* one if $q = 2$.

The main difficulty of the penalty function methods lies in selecting the initial value and updating strategy for the penalty coefficient. If the value is too small, the unconstrained problem (3.1) may produce a solution outside the feasible region. On the other hand, if the value is too large, approaching the boundary outside the feasible region is difficult and the boundary may remain unexamined. However, at least one of the constraints is usually active in the optimal point and, for this reason, it is important to search for the solution in the whole feasible region including boundaries.

Many penalty function methods, like [6, 12, 20], involve problem-dependent parameters. Furthermore, often, the performance of such methods depends on the successful selection of the parameter values. These two features are not desirable for general-purpose solvers. On the other hand, some other constraint handling

methods, like [3, 16, 29], that have performed well in the comparisons available in the literature, involve modifications on the genetic operators. Because we chose to restrict ourselves to methods not requiring such modifications, these approaches are not studied in this paper.

The idea of adaptive penalties is that the penalization is adjusted during the solution process. This means that the method is not necessarily so sensitive to the initial values of the parameters. In what follows, we introduce three adaptive penalty function methods and two related variants for handling constraints.

3.1. METHOD OF SUPERIORITY OF FEASIBLE POINTS

The idea of the *method of superiority of feasible points* (SFP method) is presented in [28]. An example of the realization of the idea is given in [23]. In the SFP method, an additional function is added to the penalized objective function. This iteration-dependent function assures that infeasible solutions always have worse fitness function values than feasible solutions.

The problem to be solved with a new fitness function is of the form

$$\begin{aligned} &\text{minimize} \quad \hat{f}(x) = f(x) + r\Big(\sum_{j=1}^{m} \max\big[0, g_j(x)\big]\Big) + \theta_i(x) \\ &\text{subject to} \quad x^l \leqslant x \leqslant x^u, \end{aligned} \tag{3.2}$$

where

$$\theta_i(x) = \begin{cases} 0 & \text{if } X^i \cap S = \emptyset \text{ or } x \in S \\ \alpha & \text{otherwise} \end{cases} \tag{3.3}$$

and

$$\alpha = \max\Big[0, \max_{y \in X^i \cap S} f(y) - \min_{z \in X^i \setminus S} \Big[f(z) + r\Big(\sum_{j=1}^{m} \max\big[0, g_j(z)\big]\Big)\Big]\Big]. \tag{3.4}$$

In other words, the function θ_i in (3.2) modifies the individuals of the population in such a way that they cannot have better fitness function values outside the feasible region than what they have inside. Note that the penalty coefficient r is here constant because the function θ_i should discriminate between feasible and infeasible individuals.

It is a straightforward task to incorporate the SFP method to the basic genetic algorithm. The main difference is the fitness function used.

SFP algorithm

1. Set r, the population size N and the other genetic parameters as well as the parameters of the stopping criterion. Set $i = 1$.

2. Generate a random initial population X^1. Set $f_{best} = \infty$.
3. If the best individual in X^i according to $\hat{f}$ is feasible and it gives the best fitness function value so far, update f_{best} and save that individual to x_{best}. If the stopping criterion is satisfied, stop.
4. Set $X^{i+1} = \emptyset$. Carry out elitism. Repeat selection, crossover and mutation until X^{i+1} has N individuals.
5. Set $i = i + 1$. Goto step 3.

If $f_{best} < \infty$, the final solution is x_{best}. Otherwise, the algorithm could not find any feasible solution.

As a modification to the SFP method, a quadratic penalty function was tested instead of the exact penalty function in (3.2). However, in general, the results obtained were worse and, thus, they are not considered here.

3.2. METHOD OF PARAMETER FREE PENALTIES

The *method of parameter free penalties* (PFP method) is introduced in [7]. Here we present this method as a modification of the SFP method. In other words, the PFP method can be formulated in such a way that an additional function is added to the penalized objective function. Again, this iteration-dependent function assures that infeasible solutions always have worse fitness function values than feasible solutions. Furthermore, actually, the fitness function values of infeasible solutions do not depend on objective function values.

The problem to be solved with a new fitness function $\hat{f}$ is of the form

$$\begin{aligned} &\text{minimize} \quad \hat{f}(x) = f(x) + \sum_{j=1}^{m} \max\left[0, g_j(x)\right] + \hat{\theta}_i(x) \\ &\text{subject to} \quad x^l \leqslant x \leqslant x^u, \end{aligned} \tag{3.5}$$

where

$$\hat{\theta}_i(x) = \begin{cases} 0 & \text{if } x \in S \\ -f(x) & \text{if } X^i \cap S = \emptyset \\ -f(x) + \max\limits_{y \in X^i \cap S} f(y) & \text{otherwise.} \end{cases} \tag{3.6}$$

The PFP algorithm is the same as for the SFP method except that the fitness function is different and there is no penalty coefficient r.

The function $\hat{\theta}_i$ in (3.5) ensures that the infeasible solutions are always directed towards the feasible region S. Due to this, the PFP method is more likely to find feasible solutions. On the other hand, the convergence might deteriorate because the objective function is completely neglected in the case of infeasible solutions.

A quadratic penalty function was tested in the PFP method instead of the exact function in (3.5). In general, the results obtained were quite similar and, thus, they are not considered here.

3.3. METHOD OF ADAPTIVE PENALTIES

The *method of adaptive penalties* (AP method) tries to avoid infeasible solutions by adjusting the penalty coefficient. The method was originally proposed for multiple-choice integer programs in a report, later published in [11]. Anyhow, it can be generalized for other problems as, for example, in [23].

A new parameter h is used in the AP method. This parameter is the number of iterations whose best individuals are examined. If all the best individuals of the past h iterations are feasible, the penalty coefficient is decreased by dividing it with a parameter $c_1 > 1$. Otherwise, if all the best individuals of the past h iterations are infeasible, the penalty coefficient is increased by multiplying it with a parameter $c_2 > 1$. If some of the best individuals are feasible and some infeasible, we continue with the current penalty coefficient. Thus, the penalty coefficient is updated if there is a possibility that the boundary of the feasible region is not covered or the search concentrates on infeasible solutions.

The problem to be solved with a new fitness function $\hat{f}$ is of the form

$$\begin{aligned} &\text{minimize} \quad \hat{f}(x) = f(x) + r_i\Big(\sum_{j=1}^{m} \max\big[0, g_j(x)\big]^2\Big) \\ &\text{subject to} \quad x^l \leqslant x \leqslant x^u, \end{aligned} \tag{3.7}$$

where the value of the penalty coefficient r_i is checked at each iteration i after the first h iterations. Let us denote the best individual of the iteration j by y^j. The coefficient is updated according to the formula

$$r_{i+1} = \begin{cases} \frac{r_i}{c_1} & \text{if } i \geqslant h \text{ and } y^j \in S \text{ for all } i-h+1 \leqslant j \leqslant i \\ c_2 r_i & \text{if } i \geqslant h \text{ and } y^j \notin S \text{ for all } i-h+1 \leqslant j \leqslant i \\ r_i & \text{otherwise,} \end{cases} \tag{3.8}$$

where $c_1, c_2 > 1$ and $c_1 \neq c_2$ (in order to avoid cycling).

AP algorithm

1. Set $c_1, c_2 > 1$ and r_1. Set h, the population size N and the other genetic parameters as well as the parameters of the stopping criterion. Set $i = 1$.
2. Generate a random initial population X^1. Set $f_{best} = \infty$.
3. Save the best individual of X^i according to $\hat{f}$ as y^i. If this individual is feasible and it gives the best fitness function value so far, update f_{best} and save that individual to x_{best}. If the stopping criterion is satisfied, stop.
4. Set $X^{i+1} = \emptyset$. Carry out elitism. Repeat selection, crossover and mutation until X^{i+1} has N individuals.
5. Calculate r_{i+1} according to (3.8). Set $i = i + 1$ and goto step 3.

As in the SFP method, the final solution is x_{best} unless $f_{best} = \infty$. In the latter case, the algorithm did not manage to find any feasible solution.

The magnitude of the constants c_1 and c_2 is examined in [11]. It is shown that the method is rather robust with respect to these values. Some formulas are given in [11] also for the determination of h and r_1. However, they are specific to the multiple-choice integer program.

An exact penalty function was tested in the AP method instead of the quadratic function in (3.7). However, the results obtained were mainly worse and, thus, they are not presented here.

3.4. MODIFICATIONS

The idea of the AP method is to update the penalty coefficient based on the feasibility of the best individuals of h iterations. In this approach, the feasibility of the whole population is not considered. On the other hand, if the best solution of a population is infeasible whereas the second best solution is feasible, the best (feasible) solution found is not updated. This is the case even if the second best solution would be better than the current x_{best}. This is in principle a drawback of the method.

For these reasons, the first trial to be made was to modify the AP method so that the best feasible solution found is recorded and the penalty coefficient is updated if the whole population lies inside or outside the feasible region. In the tests, it was checked whether it is necessary to carry out the process at every iteration. It turned out that frequent checking improved the results. Note that the parameter h was not needed in this approach. In all the other respects, the modification corresponded to the AP algorithm. However, the results obtained were not better than those of the AP method and, thus, they are not reported here.

As far as the SFP method is concerned, it may happen that no feasible solutions are found. If the initial population does not contain any feasible solutions, the fixed penalty coefficient may not be large enough to direct the population towards the feasible region. That is why it is interesting to combine the above-mentioned modified adaptive penalty coefficient technique and the SFP method. This combination will be called MASF. This algorithm corresponds to the modified AP algorithm with exact penalties and the additional function θ_i.

A direct combination of the original AP and the SFP methods is not sensible because the update process checks only the best individuals and they are always feasible in the SFP method (assuming they exist). Nevertheless, the AP and SFP methods can be combined by not using the θ-function when determining the need to update the penalty coefficient. This modification will be called SFPAP. For this algorithm, the following function is defined

$$\hat{f}_\theta(x) = f(x) + r_i\Big(\sum_{j=1}^{m} \max\big[0, g_j(x)\big]^2\Big) + \theta_i(x), \tag{3.9}$$

where r_i is defined by (3.8) and $\theta_i(x)$ is given by (3.3) and (3.4).

SFPAP algorithm

1. Set $c_1, c_2 > 1$ and r_1. Set h, the population size N and the other genetic parameters as well as the parameters of the stopping criterion. Set $i = 1$.
2. Generate a random initial population X^1. Set $f_{best} = \infty$.
3. Save the best individual of X^i according to $\hat{f}$ in (3.7) as y^i. If the best individual of X^i according to $\hat{f}_\theta$ is feasible and it gives the best fitness function value so far, update f_{best} and save that individual to x_{best}. If the stopping criterion is satisfied, stop.
4. Set $X^{i+1} = \emptyset$. Carry out elitism. Repeat selection, crossover and mutation until X^{i+1} has N individuals.
5. Calculate r_{i+1} according to (3.8). Set $i = i + 1$ and goto step 3.

4. Numerical Experiments

Numerical experiments were carried out to test the performance of the five constraint handling methods described. A number of 33 test problems of different types were selected from the literature.

The set of 33 test problems used is briefly introduced in Table 1. In the table, after the number of the problem (np), the number of variables is denoted by n and the objective functions are classified into linear (lin), quadratic (quad) and nonlinear (nonl) functions. The ratio of the area of the feasible region to the box constrained area is denoted by ρ expressed in percentages. The values of ρ were calculated by shooting 100 million random points in the box-constrained area and counting the feasible ones. The constraints of different types are denoted by LE (linear equations), LI (linear inequalities), NE (nonlinear equations) and NI (nonlinear inequalities). Finally, the reference and the best known objective function value are given.

All the methods were implemented in Fortran 77 and the test runs were performed on an HP9000/C160, 160MHz computer. Random numbers were generated by using the routine G05CAF from the NAG subroutine library.

In these experiments, each equation constraint was transformed into two inequality constraints with a tolerance 0.01. Alternatively, equation constraints could have been taken into consideration as such by modifying the penalty functions.

Because the idea was to compare different constraint handling techniques, the input parameters of the genetic algorithm were constant in all the experiments. The population size N was 101, the maximum number of iterations 500, the crossover rate 0.8, elitism size 1, tournament size 3 and the mutation rate was 0.1. In addition, the mutation exponent p was 4. The solution process was stopped if the difference between the best fitness function values of 100 last iterations was less than 0.01.

The parameters of each constraint handling method were selected so that the performance was good on the average for a subset of the problems tested. Then the number of problems was increased and the performance of the different methods

Table 1. Text problems

np	n	obj.f.	ρ	LE	LI	NE	NI	from	best known
1	8	lin	.000578	0	6	0	0	[8], pr. 3.1	7049.25
2	5	quad	26.960078	0	0	0	6	[8], pr. 3.2	−30665.5387
3	6	quad	11.312849	0	4	0	2	[8], pr. 3.3	−310.0
4	4	nonl	.043394	1	2	0	0	[8], pr. 4.3	−4.5142
5	4	nonl	.013552	1	2	0	0	[8], pr. 4.4	−2.07
6	6	nonl	.000000	3	3	0	0	[8], pr. 4.5	−11.96
7	2	lin	44.200537	0	0	0	2	[8], pr. 4.6	−5.5079
8	2	quad	.332226	0	0	1	0	[8], pr. 4.7	−16.78
9	4	nonl	.000000	0	2	3	0	[23], G5	5126.4981
10	50	nonl	100.000000	0	0	0	2	[23], G2	−0.8331937
11	5	nonl	.000001	0	0	3	0	[15], pr. 6	0.0539498
12	2	nonl	24.99898	0	2	0	0	[21], test #8	−1.0
13	2	nonl	.861168	0	0	0	2	[23], G8	−0.095825
14	23	nonl	.000000	0	0	1	0	[23], G3	−1.0
15	10	nonl	.000000	3	0	0	0	[21], test #2	−47.760765
16	2	nonl	7.329000	0	0	0	2	[15], pr. 1	0.25
17	2	quad	96.644521	0	0	0	2	[15], pr. 4	5.0
18	7	nonl	.524944	0	0	0	4	[23], G9	680.6300573
19	13	quad	.000244	0	9	0	0	[23], G1	−15.0
20	2	nonl	.006711	0	0	0	2	[23], G6	−6961.81381
21	10	quad	.000110	0	3	0	5	[23], G7	24.3062091
22	2	quad	37.492715	0	1	0	1	[15], pr. 5	1.0
23	5	quad	95.256165	0	1	0	0	[8], pr. 2.1	−17.0
24	6	quad	23.404995	0	2	0	0	[8], pr. 2.2	−213.0
25	13	quad	.237391	0	9	0	0	[8], pr. 2.3	−15.0
26	6	quad	1.827590	0	5	0	0	[8], pr. 2.4	−11.005
27	10	quad	.004728	0	11	0	0	[8], pr. 2.5	−268.01
28	10	quad	.007350	0	5	0	0	[8], pr. 2.6	−39.0
29	20	quad	.000000	0	10	0	0	[8], pr. 2.7/1	−394.7506
30	20	quad	.000000	0	10	0	0	[8], pr. 2.7/2	−884.75058
31	20	quad	.000000	0	10	0	0	[8], pr. 2.7/3	−8695.01193
32	30	nonl	99.999947	0	0	0	2	[23], G2	−0.75
33	70	nonl	100.000000	0	0	0	2	[23], G2	−0.57

was compared. This practice was adopted to simulate cases where the methods should perform well without tuning the parameters separately for each problem to be solved.

In the SFP method, the penalty coefficient r was set as 10000.0 and in all the other methods (except PFP) the initial penalty coefficient r_1 was 1.0. In the AP method, the number of iterations whose best individuals were examined h was 10, the decrement multiplier c_1 was 3.0 and the increment multiplier c_2 was 4.0. In the MASF method, the decrement coefficient did not seem to have a significant role. In the tests, the values $c_1 = 20.0$ and $c_2 = 30.0$ were used. In the SFPAP method, the parameters h, c_1 and c_2 were the same as in the AP method.

Because of the stochastic nature of genetic algorithms, a hundred test runs were performed for each of the 33 test problems. We recorded the best objective function values for each test run. In what follows, we list for each test problem (np) the means of the best values (mean), the best values of the 100 runs (min) as well as the standard deviations (dev). The number of iterations was also recorded and we show their means (iter). Finally, we present the percentage of the runs where feasible solutions could be found (fea). Tables 2, 3 and 4 contain the above-mentioned information obtained with the SFP, PFP, AP, MASF and SFPAP methods, correspondingly. The best values of mean, min and iter are underlined for each test problem. Note that for some problems, the obtained min values are smaller than the best known objective function values. This is explained by the fact that equation constraints were transformed into two inequality constraints with tolerances.

5. Discussion and Conclusions

Judging the superiority of constraint handling techniques on the basis of comparisons published in the literature is difficult because of the different implementations, genetic operators and different parameter values used, etc. Furthermore, the comparisons often involve a small number of test problems and a surprisingly small number of independent runs. These observations and drawbacks motivated our research. That is why a number of 33 test problems were used in a uniform computer environment with fixed genetic operators and using 100 independent runs.

The methods are not easy to compare due to their stochastic nature. Here we decided to compare the average performances. One should also keep in mind that different stopping criteria as well as different parameter setups could have produced different results.

We analyze the results obtained based on three criteria: accuracy, efficiency and reliability. Here, accuracy reflects how close the mean objective function value was to the best known one. By efficiency, we refer to the average number of iterations used. Finally, how well the methods managed to find feasible solutions can be understood as their reliability.

It is natural that all the versatile information contained in Tables 2–4 cannot easily be compressed without losing valuable data. Still, we present a very

Table 2. Results with the SFP and PFP methods.

	SFP					**PFP**				
np	mean	min	dev	iter	fea	mean	min	dev	iter	fea
1	7893.74	7116.64	1285.08	416.5	68	8464.55	7292.10	1294.91	101.0	100
2	−30665.53	−30665.54	0.06	296.3	100	−30665.53	−30665.54	0.01	296.3	100
3	−309.84	−310.00	1.60	217.6	100	−308.58	−310.00	12.69	224.5	100
4	−4.52	−4.53	0.15	145.8	100	−4.41	−4.53	0.40	148.4	100
5	−3.13	−3.14	0.02	128.9	100	−3.14	−3.14	0.00	129.7	100
6	−13.32	−13.41	0.26	298.8	100	−13.38	−13.41	0.15	300.0	100
7	−5.51	−5.51	0.00	110.6	100	−5.51	−5.51	0.00	110.9	100
8	−16.78	−16.78	0.00	115.5	100	−16.78	−16.78	0.00	116.5	100
9	4239.21	4221.83	62.14	404.6	100	4755.32	4221.83	531.20	102.7	100
10	−0.56	−0.64	0.03	394.7	100	−0.56	−0.64	0.03	394.7	100
11	0.38	0.05	0.29	304.5	100	0.58	0.05	0.35	108.2	100
12	−1.00	−1.00	0.00	102.8	100	−1.00	−1.00	0.00	102.7	100
13	−0.10	−0.10	0.01	106.4	100	−0.10	−0.10	0.01	106.7	100
14					0	−0.78	−1.01	0.09	461.4	100
15	−47.01	−48.11	0.77	467.2	100	−47.03	−47.97	0.77	446.7	100
16	0.25	0.25	0.00	106.0	100	0.25	0.25	0.00	106.0	100
17	5.00	5.00	0.00	110.5	100	5.00	5.00	0.00	110.0	100
18	681.56	680.68	0.76	377.4	100	682.75	680.75	2.17	225.3	100
19	−14.94	−15.00	0.34	323.9	100	−14.98	−15.00	0.20	327.1	100
20	−6961.81	−6961.81	0.00	143.7	100	−6961.81	−6961.81	0.00	143.4	100
21	26.87	24.77	1.37	459.4	100	32.63	25.76	5.64	107.6	100
22	1.00	1.00	0.00	106.7	100	1.00	1.00	0.00	106.6	100
23	−15.98	−17.00	1.16	154.4	100	−15.81	−17.00	1.54	155.1	100
24	−212.98	−213.00	0.08	213.3	100	−212.98	−213.00	0.09	216.6	100
25	−15.00	−15.00	0.00	313.0	100	−15.00	−15.00	0.00	313.4	100
26	−11.00	−11.00	0.00	204.3	100	−10.99	−11.00	0.05	212.7	100
27	−265.81	−268.01	3.21	469.2	100	−265.06	−268.00	3.42	468.6	100
28	−36.66	−39.00	5.19	259.4	100	−37.05	−39.00	4.73	251.2	100
29	−135.08	−221.11	38.63	494.3	100	−132.02	−247.72	40.54	498.2	100
30	−593.81	−696.76	34.34	463.6	100	−586.45	−698.08	29.27	458.1	100
31	−3043.37	−5374.88	682.90	486.6	100	−3106.12	−5424.69	695.63	490.1	100
32	−0.66	−0.74	0.04	350.1	100	−0.66	−0.74	0.04	350.1	100
33	−0.50	−0.57	0.03	408.3	100	−0.50	−0.57	0.03	408.3	100

Table 3. Results with the AP method.

	AP				
np	mean	min	dev	iter	fea
1	9050.59	7329.72	1464.15	101.9	100
2	−30662.00	−30665.53	7.95	101.0	100
3	−309.70	−310.00	1.60	101.0	100
4	−4.53	−4.53	0.00	106.8	100
5	−3.14	−3.14	0.00	106.0	100
6	−13.23	−13.41	0.60	240.7	100
7	−5.51	−5.51	0.00	102.1	100
8	−16.78	−16.78	0.00	101.0	100
9	4323.04	4221.83	315.88	303.9	91
10	−0.48	−0.57	0.04	212.4	100
11	0.31	0.05	0.24	257.0	100
12	−1.00	−1.00	0.00	103.9	100
13	−0.10	−0.10	0.01	102.2	100
14	−0.01	−0.07	0.02	101.0	100
15	−44.48	−47.94	1.64	101.0	100
16	0.25	0.25	0.00	106.3	100
17	5.00	5.00	0.00	107.7	100
18	680.83	680.64	0.20	193.5	100
19	−14.80	−15.00	0.45	215.6	100
20	−6961.81	−6961.81	0.00	103.6	100
21	27.71	24.62	2.20	201.6	100
22	1.00	1.00	0.00	104.7	100
23	−15.69	−17.00	1.93	121.7	100
24	−213.00	−213.00	0.02	187.3	100
25	−14.97	−15.00	0.07	245.1	100
26	−10.96	−11.00	0.25	184.5	100
27	−264.90	−268.01	3.66	423.2	100
28	−35.89	−38.95	5.55	169.9	100
29	−100.07	−192.60	35.78	224.0	78
30	−575.36	−673.38	32.54	249.1	78
31	−1761.01	−2690.87	296.32	103.3	76
32	−0.58	−0.73	0.06	157.6	100
33	−0.46	−0.54	0.03	371.4	100

Table 4. Results with the MASF and SFPAP methods.

	MASF					**SFPAP**				
np	mean	min	dev	iter	fea	mean	min	dev	iter	fea
1	8208.74	7103.78	985.27	206.6	100	8995.57	7369.27	1682.95	101.8	100
2	−30665.39	−30665.54	0.24	309.5	100	−30665.36	−30665.54	0.34	302.9	100
3	−308.74	−310.00	12.60	228.2	100	−310.00	−310.00	0.00	226.1	100
4	−4.41	−4.53	0.40	102.7	100	−4.52	−4.53	0.15	128.3	100
5	−3.14	−3.14	0.00	129.0	100	−3.13	−3.14	0.02	119.5	100
6	−13.32	−13.41	0.27	281.2	100	−13.17	−13.41	0.93	271.5	100
7	−5.51	−5.51	0.00	110.7	100	−5.51	−5.51	0.00	111.1	100
8	−16.78	−16.78	0.00	110.4	100	−16.78	−16.78	0.00	105.3	100
9	4656.19	4221.83	490.06	101.0	91	4382.28	4221.83	375.80	332.8	93
10	−0.57	−0.63	0.03	399.9	100	−0.57	−0.63	0.03	399.9	100
11	0.33	0.05	0.27	321.0	100	0.30	0.05	0.23	296.4	100
12	−1.00	−1.00	0.00	102.6	100	−1.00	−1.00	0.00	102.6	100
13	−0.10	−0.10	0.01	104.5	100	−0.10	−0.10	0.00	104.0	100
14	−0.06	−0.28	0.05	101.0	100	−0.01	−0.07	0.02	101.0	100
15	−44.76	−47.46	1.62	101.0	100	−44.45	−47.56	1.52	101.0	100
16	0.25	0.25	0.00	105.7	100	0.25	0.25	0.00	105.8	100
17	5.00	5.00	0.00	110.3	100	5.00	5.00	0.00	110.0	100
18	681.34	680.72	0.50	368.0	100	681.86	680.75	0.73	378.2	100
19	−14.98	−15.00	0.20	330.0	100	−14.99	−15.00	0.02	365.8	100
20	−6961.81	−6961.81	0.00	138.5	100	−6961.81	−6961.81	0.00	125.1	100
21	27.06	24.95	1.49	457.2	100	27.70	24.88	1.90	459.6	100
22	1.00	1.00	0.00	107.0	100	1.00	1.00	0.00	106.7	100
23	−16.06	−17.00	1.19	154.5	100	−15.81	−17.00	1.43	157.9	100
24	−212.99	−213.00	0.04	224.6	100	−212.98	−213.00	0.05	217.5	100
25	−14.98	−15.00	0.20	313.8	100	−14.98	−15.00	0.01	345.7	100
26	−11.00	−11.00	0.01	200.8	100	−11.00	−11.00	0.01	199.3	100
27	−263.52	−268.00	7.30	412.6	100	−265.62	−268.00	4.11	441.3	100
28	−36.61	−39.00	5.38	253.5	100	−37.12	−39.00	4.17	255.5	100
29	−48.33	−76.46	12.11	101.0	19	−118.09	−225.81	45.86	312.8	86
30	−528.72	−540.90	9.51	101.0	21	−611.67	−692.98	33.97	473.8	72
31	−1673.70	−3240.98	397.84	101.0	23	−1761.01	−2690.87	296.32	103.3	76
32	−0.67	−0.75	0.04	372.1	100	−0.67	−0.75	0.04	372.1	100
33	−0.50	−0.57	0.03	428.3	100	−0.50	−0.57	0.03	428.3	100

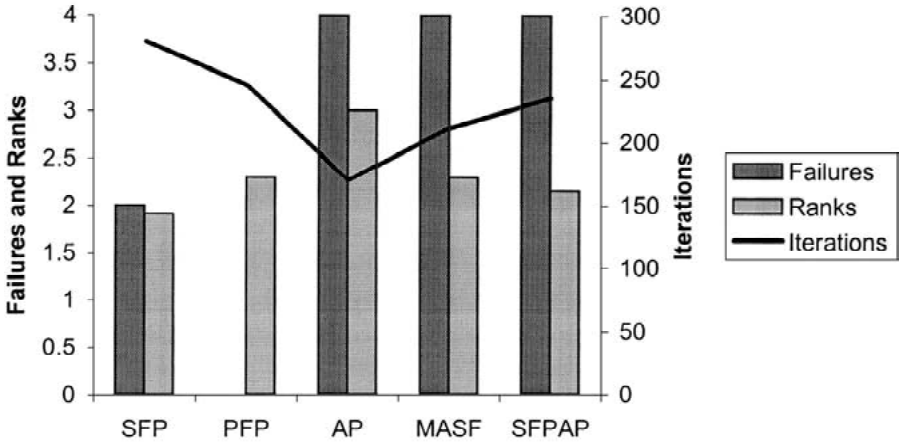

Figure 1. Rough overview of the three criteria.

rough overview of the three evaluation criteria used in Figure 1. In any case, we emphasize the importance of the details in the tables.

As far as accuracy is concerned, we have ranked the means of the solutions related to each problem and present the averages of these ranks for each method. The efficiency is measured by the average number of iterations used by each method. In the figure, this is referred to as iterations. Finally, the word failures indicates the number of problems where the method failed in finding feasible solutions. In this way, we determine the reliability.

The success of the SFP method in finding feasible solutions turned out to depend highly on the value of the penalty coefficient. In the tested problems, the penalty coefficient was set to a relatively high level which explains the reasonable reliability of the method. One can conclude from the numerical results that if the SFP method was able to find feasible solutions, it resulted in a high average accuracy by paying the price of low efficiency. However, it is noteworthy that for one test problem SFP did fail completely in finding feasible solutions. The AP method was the most efficient in terms of iterations required. On the other hand, the results obtained with AP did not meet the accuracy of the others.

The performances of MASF and SFPAP were in between AP and SFP. On the average, SFPAP required more iterations than MASF and they both needed more iterations than AP but less than SFP. They did not fail completely in finding feasible solution in any of the problems because of the penalty coefficient updating. The SFPAP method had a similar reliability to AP and MASF was slightly less reliable in finding feasible solutions.

Based on the computational experiences we can say that all the methods involving constraint handling parameters were rather sensitive to the choice of their values. Thus, the fact that no parameters have to be specified, is a significant benefit of PFP. Furthermore, it was the only method that was always able to find feasible solutions. The results obtained were also satisfactory in terms of accuracy. The

only drawback of this methods was the large number of iterations required. In this respect, PFP was comparable to SFPAP.

To conclude, one can say that PFP outperformed SFP in efficiency and reliability. Besides, the difference in accuracy was rather insignificant. The strength of PFP can be explained by the similar nature of the two methods; the main difference being that PFP could better direct the search towards the feasible region, if required. The other three methods were good compromises when balancing between accuracy, efficiency and reliability. The selection between them depends on the preferences of the user.

For our purposes in the WWW-NIMBUS system, a general-purpose solver was required. Thus, the reliability of PFP is valuable. Another important advantage of PFP is the fact that no parameters are required in handling constraints. Because the solutions obtained were also satisfactory, we decided to select the PFP method for the WWW-NIMBUS system. Furthermore, since AP was so efficient and needed on the average less iterations than the others, we decided to include it as well to the WWW-NIMBUS system as an alternative solver.

Acknowledgements

This research was supported by the Academy of Finland, grants #65760, #8583 and #66407. The authors thank Mr. Janne Mäkinen for preliminary results, Mr. Jari Huikari for assistance with the test problems and Mr. Markku Könkköla for Figure 1. The authors are also grateful to the reviewers for their constructive comments which helped to improve this paper.

References

1. Adeli H. and Cheng N.-T. (1994), Augmented Lagrangian genetic algorithm for structural optimization, *Journal of Aerospace Engineering* 7(1) 104–118.
2. Bazaraa M. S., Sherali H. D. and Shetty C. M. (1993). *Nonlinear Programming: Theory and Algorithms*. 2nd edition, John Wiley & Sons.
3. Ben Hamida S. and Schoenauer M. (2000). An adaptive algorithm for constrained optimization problems. In: Schoenauer M., Deb K., Rudolph G., Yao X., Lutton E., Merelo J. J. and H.-P. Schwefel (eds.) *Proceedings of the 6th Conference on Parallel Problems Solving from Nature*. Springer, Berlin, 529–539.
4. Camponogara E. and Talukdar S. N. (1997). A Genetic Algorithm for Constrained and Multiobjective Optimization. In: Alander J. (ed.), *Proceedings of the 3rd Nordic Workshop on Genetic Algorithms and Their Applications*. University of Vaasa, 49–61.
5. Coello Coello C. A. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art, *Computer Methods in Applied Mechanics and Engineering* 191 (11–12), 1245–1287.
6. Coit D. W. and Smith A. E. (1996). Penalty guided genetic search for reliability design optimization, *International Journal of Computers and Industrial Engineering* 30 (4), 895–904.
7. Deb K. (2000). An efficient constraint handling method for genetic algorithms, *Computer Methods in Applied Mechanics and Engineering*, 186, 311–338.

8. Floudas C. A. and Pardalos P. M. (1987). A collection of test problems for constrained global optimization algorithms. Springer, Berlin.
9. Gen M. and Cheng R. (1996). A survey of penalty techniques in genetic algorithms. In: Fukuda, T. and Furuhashi, T. (eds.), *Proceedings of the 1996 International Conference on Evolutionary Computation*. IEEE, 804–809.
10. Goldberg D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Publishing Company, Inc., Reading, MA.
11. Hadj-Alouane A. B. and Bean J. C. (1997). A genetic algorithm for the multiple-choice integer program, *Operations Research*, 45 (1), 92–101.
12. Homaifar A., Qi C. X. and Lai S. H. (1994). Constrained optimization via genetic algorithms, *Simulation*, 62 (4), 242–254.
13. Horst R. and Pardalos P. M. (1995). Handbook of Global Optimization. Kluwer Academic Publishers, Dordrecht.
14. Kazarlis S. and Petridis V. (1998). Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In: Eiben A. E., Bäck T., Schoenauer M. and Schwefel, H.-P. (eds.), *Proceedings of the parallel problem solving from nature*. Springer, Berlin, 211–220.
15. Kim J.-H. and Myung H. (1997). Evolutionary programming techniques for constrained optimization problems, *IEEE Transactions on Evolutionary Computation*, 1 (2), 129–140.
16. Le Riche R. G., Knopf-Lenoir C. and Haftka R. T. (1995). A segregated genetic algorithm for constrained structural optimization. In: Eshelman L.J. (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 558–565.
17. Mäkinen R. A. E., Périaux J. and Toivanen J. (1999). Multidisciplinary shape optimization in aerodynamics and electromagnetics using genetic algorithms, *International Journal for Numerical Methods in Fluids*. 30 (2), 149–159.
18. Michalewicz Z. (1995). Genetic algorithms, numerical optimization, and constraints. In: Eshelman L. J. (ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 151–158.
19. Michalewicz Z. (1995). A survey of constraint handling techniques in evolutionary computation methods. In: McDonnell J. R., Reynolds R. G. and Fogel D. B. (eds.), *Proceedings of the 4th Annual Conference on Evolutionary Programming*. MIT Press, 135–155.
20. Michalewicz Z. and Attia N. F. (1994). Evolutionary optimization of constrained problems. In: Sebald A.V. and Fogel L.J. (eds.) *Proceedings of the 3rd Annual Conference on Evolutionary Programming*. World Scientific, 98–108.
21. Michalewicz Z., Logan T. D. and Swaminathan S. (1994). Evolutionary operators for continuous convex parameter spaces. In: Sebald A. V. and Fogel L. J. (eds.), *Proceedings of the 3rd Annual Conference on Evolutionary Programming*. World Scientific, 84–97.
22. Michalewicz Z., Nazhiyath G. and Michalewicz M. (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In: Angeline P. J. and Bäck T. (eds.), *Proceedings of the 5th Annual Conference on Evolutionary Programming*. MIT Press, Cambridge, MA, 305–312.
23. Michalewicz Z. and Schoenauer M. (1996). Evolutionary algorithms for constrained parameter optimization problems, *Evolutionary Computation*, 4 (1), 1–32.
24. Miettinen K. and Mäkelä M. M. (1995). Interactive bundle-based method for nondifferentiable multiobjective optimization: NIMBUS, *Optimization*, 34 (3), 231–246.
25. Miettinen K. and Mäkelä M. M. (2000). Interactive multiobjective optimization system WWW-NIMBUS on the Internet, *Computers & Operations Research*, 27 (7–8), 709–723.
26. Miettinen K., Mäkelä M. M., Neittaanmäki P. and Périaux J. 1999 (eds). Evolutionary algorithms in engineering and computer science. John Wiley & Sons, New York.
27. Miettinen K., Mäkelä M. M. and Toivanen J. (1999) (eds). Proceedings of EUROGEN99 – short course on evolutionary algorithms in engineering and computer science, Reports of the

Department of Mathematical Information Technology, Series A. Collections, No. A 2/1999. University of Jyväskylä.
28. Powell D. and Skolnick M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In: Forrest S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 424–431.
29. Runarsson T. P. and Yao X. (2000). Stochastic ranking for constrained evolutionary optimization, *IEEE Transactions on Evolutionary Computation*, 4 (3), 284–294.
30. Schoenauer M. and Michalewicz Z. (1997). Boundary operators for constrained parameter optimization problems. In: Bäck T. (ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 322–329.
31. Schoenauer M. and Xanthakis S. (1993). Constrained GA optimization. In: Forrest S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 573–580.
32. Smith A. E. and Tate D. M. (1993). Genetic optimization using a penalty function. In: Forrest S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 499–505.
33. Smith R. L. (1984). Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions, *Operations Research*, 32 (6), 1296–1308.