An adaptive penalty scheme in genetic algorithms for constrained optimization problems

Helio J.C. Barbosa
LNCC/MCT
Rua Getulio Vargas 333
25651 070 Petropolis RJ, BRAZIL
hcbm@lncc.br

Afonso C.C. Lemonge
Departamento de Estruturas
Faculdade de Engenharia
Universidade Federal de Juiz de Fora
36036 330 Juiz de Fora MG, BRAZIL
lemonge@numec.ufjf.br

Abstract

A parameter-less adaptive penalty scheme for genetic algorithms applied to constrained optimization problems is proposed. Besides being simple, adaptive and not requiring any parameter, it provides a different penalty parameter for each constraint. The performance of this new scheme is examined using several test-problems from the evolutionary computation literature.

1 INTRODUCTION

The design and implementation of a robust genetic algorithm (GA) for constrained problems is not an easy task. Several possibilities have been explored by the evolutionary computation community and they can be roughly grouped as: 1) penalty techniques, 2) repair methods, 3) special decoders, 4) special operators, 5) selection techniques, 6) hybrid methods and 7) other methods.

Repair methods use domain knowledge in order to move infeasible offspring into the feasible set. However there are situations when it is very expensive, or even impossible, to construct such a repair operator, drastically reducing the range of applicability of repair methods. Like repair methods, the design of special decoders[1] that always extract a feasible phenotype from a given genotype is not trivial in general and cannot always be done. In special situations genetic operators can be constructed so that a feasible offspring is always generated from feasible parents[2]. Hybrid methods, combining mathematical programming and evolutionary techniques have also been developed[3, 4]. Modified selection schemes used alone [5] or in conjunction with additional features [6] are also proposed. Under the heading "other methods" one usually collects those which cannot be clearly classified in one of the previous groups such as the use of co-evolution (see[7, 8]). For other methods found in the evolutionary computation literature see[2, 9, 10, 4] and references therein.

Penalty techniques range from simple naive schemes (like the "death penalty": any infeasible offspring is just discarded with no consideration for its potential information content) to penalty schemes involving from one to several parameters. Those parameters can remain constant (the most common case) or be dynamically varied along the evolutionary process according to an exogenous schedule[11, 12] or an adaptive procedure[13]. Penalty methods, though quite general, require considerable domain knowledge and experimentation in each particular application in order to be effective.

In contrast with previous approaches where a single penalty parameter is used for all constraints in a given problem, an adaptive scheme is proposed here which automatically sizes the penalty parameter corresponding to *each* constraint along the evolutionary process.

In the next section the main penalty techniques found in the literature are summarized. Section 3 presents the adaptive scheme proposed and Section 4 discusses several numerical experiments with test-problems. The paper ends with conclusions in Section 5.

2 THE CONSTRAINED OPTIMIZATION PROBLEM

A standard constrained optimization problem in R^n can be thought of as the minimization of a given objective function $f(x)$, where $x \in R^n$ is the vector of design/decision variables, subject to inequality constraints $g_p(x) \geq 0$, $p = 1, 2, \ldots, \bar{p}$ as well as equality constraints $h_q(x) = 0$, $q = 1, 2, \ldots, \bar{q}$. Additionally, the variables may be subject to bounds $x_i^L \leq x_i \leq x_i^U$ but this type of constraint is trivially enforced in a GA and need not be considered here.

2.1 Penalty functions

Although a few papers [14] do consider a *multiplicative* penalty function, in the widely used additive penalty formulation one is led to the unconstrained minimization of a modified objective function

$$F(x) = f(x) + penal(x) \tag{1}$$

where the penalty function $penal(x)$ is zero if x is a feasible solution and greater than zero otherwise. It is useful to define the amount of violation of the j-th constraint by the solution $x \in R^n$ as

$$v_j(x) = \begin{cases} |h_j(x)| & \text{equality case} \\ \max\{0, -g_j(x)\} & \text{inequality case} \end{cases} \tag{2}$$

It is also common to design penalty functions that grow with the vector of violations $v(x) \in R^m$ where $m = \bar{p} + \bar{q}$ is the number of constraints to be penalized. The most popular penalty function is given by

$$penal(x) = k \sum_{j=1}^{m} (v_j(x))^2 \tag{3}$$

where k is the penalty parameter. It can be shown that as $k \to \infty$, the corresponding solution x_k^* of the penalized problem tends to the solution x^* of the original problem. Although it is easy to obtain the unconstrained problem, the definition of a good penalty parameter k is usually a very time-consuming trial-and-error process.

Among the many penalty techniques found in the literature[15, 3, 16] some of them – more closely related to the work presented here – will be briefly discussed below.

2.1.1 Some methods in the literature

Homaifar *et al.* [17] write the fitness function as in (1) – (3) but allow the user to define several levels of constraint violation in such a way that the penalty coefficients grow as higher levels of violation are reached. As a result, the method requires a penalty parameter k_j^l for the l-th level of violation of the j-th constraint. This is an attractive strategy because, at least in principle, it allows for a good control of the penalization process. The weakness of this method is the large number of parameters that must be set by the user for each problem.

Joines & Houck [11] introduce dynamic penalty parameters and the fitness function $F(x)$ can be written as in (1) – (3) with the penalty parameter, given by $k = (C \times t)^\alpha$, increasing with the generation number t.

Bean & Alouane [18] use (1) – (3) but with the penalty parameter $k = \lambda(t)$ adapted at each generation by the following rules:

$$\lambda(t+1) = \begin{cases} (\frac{1}{\beta_1})\lambda(t) & \text{if } b^i \in \mathcal{F} \text{ for all } t-g+1 \le i \le t \\ \beta_2\lambda(t) & \text{if } b^i \notin \mathcal{F} \text{ for all } t-g+1 \le i \le t \\ \lambda(t) & \text{otherwise} \end{cases}$$

where b^i is the best element at generation t, $\mathcal{F}$ is the feasible region, $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$.

Coit *et al.* [13], use the fitness function:

$$F(x) = f(x) + (F_{feas}(t) - F_{all}(t)) \sum_{j=1}^{m} (v_j(x)/v_j(t))^\alpha$$

where $F_{all}(t)$ corresponds to the best solution until the generation t (without penalty), F_{feas} corresponds to the best feasible solution and α is a constant.

Schoenauer & Xhantakis [19] handle constrained problems in stages: (i) first evolve a randomly generated population considering only the first constraint until a certain percentage of the population is feasible with respect to that constraint; (ii) the final population of the first stage of the process is used in order to optimize with respect to the second constraint. During this stage, the elements that had violated the previous constraint are removed from the population, (iii) the process is repeated until all the constraints are processed. This strategy becomes less attractive as the number of constraints grows and is potentially dependent on the order in which the constraints are processed.

Le Riche *et al.* [20] create two sets of candidate solutions where one of them is evaluated with a penalty parameter k_1 and the other with a parameter k_2. With $k_1 \gg k_2$ there are two different levels of penalization and there is a higher chance of maintaining feasible as well as infeasible individuals in the population and to get offsprings near the boundary between the feasible and infeasible regions.

Powell & Skolnick [21] propose a method of superiority of feasible points where each candidate solution is evaluated by the following expression:

$$F(x) = f(x) + r \sum_{j=1}^{m} v_j(x) + \theta(t, x)$$

where r is a constant. The main assumption is that any feasible solution is better than any infeasible solution. This assumption is enforced by a convenient definition of the function $\theta(t, x)$. Deb [6] modified this scheme by introducing tournament selection coupled with the fitness function:

$$F(x) = \begin{cases} f(x) & \text{if } x \text{ is feasible} \\ f_{max} + \sum_{j=1}^{m} v_j(x) & \text{otherwise} \end{cases} \tag{4}$$

where f_{max} – the function value of the worst feasible solution in the population – replaces $f(x)$, leading to a higher

(potentially excessive) penalty. However, Deb's[6] constraint handling scheme (which also involves niching and a controlled mutation) works very well for his real-coded GA, but not for the binary-coded one.

Hamida & Schoenauer [22] propose an adaptive algorithm accounting for the proportion of feasible individuals in the population, using a seduction/selection strategy to mate feasible and infeasible individuals and a selection scheme to favor a fraction of all feasible individuals.

Runarsson & Yao [5] propose a novel approach to balance objective and penalty function values by a stochastic ranking. Using a real coding, very good results are obtained for continuous optimization problems.

Recently, Wright & Farmani [23] proposed a method that requires no parameters and represents the constraint violation by a single infeasibility measure.

3 THE PROPOSED METHOD

In this work a method without any type of user defined penalty parameter is proposed. An adaptive scheme is developed which uses information from the population such as the average of the objective function and the level of violation of each constraint during the evolution.

The fitness function proposed is written as:

$$F(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible,} \\ \overline{f}(x) + \sum_{j=1}^{m} k_j v_j(x) & \text{otherwise} \end{cases}$$

where

$$\overline{f}(x) = \begin{cases} f(x), & \text{if } f(x) > \langle f(x) \rangle, \\ \langle f(x) \rangle & \text{otherwise} \end{cases} \tag{5}$$

and $\langle f(x) \rangle$ is the average of the objective function values in the current population. In the Figure 1 feasible as well as infeasible solutions are shown. Among the 6 infeasible solutions, the individuals #3, #4, #5 and #6 have their objective function values (represented by opened circles), less than the average objective function and, according to the proposed method, have $\overline{f}$ given by $\langle f(x) \rangle$. The solutions #1 and #2 have objective function values which are worst than the population average and thus have $\overline{f}(x) = f(x)$.

The penalty parameter is defined at each generation by:

$$k_j = |\langle f(x) \rangle| \frac{\langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \tag{6}$$

and $\langle v_l(x) \rangle$ is the violation of the l-th constraint averaged over the current population. Denoting by *pop* the population size, one could also write

$$k_j = \frac{|\sum_{i=1}^{pop} f(x^i)|}{\sum_{l=1}^{m} [\sum_{i=1}^{pop} v_l(x^i)]^2} \sum_{i=1}^{pop} v_j(x^i) \tag{7}$$

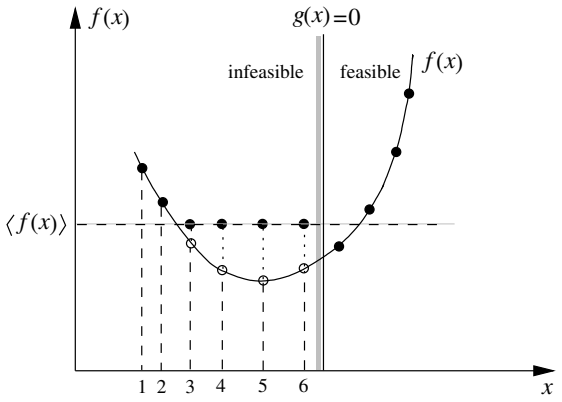

Figure 1: The definition of the function $\overline{f}$.

With the proposed definition one can prove the following:

Property – An individual whose j-th violation equals the average of the j-th violation in the current population for all j, has its fitness value given by:

$$F(\tilde{x}) = \begin{cases} f(\tilde{x}) + |\langle f(x) \rangle| & \text{if } f(\tilde{x}) > \langle f(x) \rangle \\ \langle f(x) \rangle + |\langle f(x) \rangle| & \text{otherwise} \end{cases}$$

Proof – In fact, let $\tilde{x}$ be such an element. By definition,

$$F(\tilde{x}) = \overline{f}(\tilde{x}) + \sum_{j=1}^{m} \frac{|\langle f(x) \rangle| \langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} v_j(\tilde{x}).$$

But, by hypothesis, $v_j(\tilde{x}) = \langle v_j(x) \rangle$ for all j leading to

$$\begin{aligned} F(\tilde{x}) &= \overline{f}(\tilde{x}) + \frac{|\langle f(x) \rangle|}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \sum_{j=1}^{m} [\langle v_j(x) \rangle]^2 \\ &= \overline{f}(\tilde{x}) + |\langle f(x) \rangle| \end{aligned}$$

and the results follow from eq. (5).

In the next section several examples from the literature are considered in order to test the robustness of the proposed adaptive parameter-less scheme. It should be emphasized that the accuracy of the final results of the search depends also on other components of the algorithm not considered here – such as coding, operators and selection scheme – besides the penalization procedure itself. Good results reported in the literature (such as in [5]) are obtained with real coding, more sophisticated selection schemes and more effective genetic operators. The need for better operators in order to obtain good results in constrained continuous optimization has already been pointed out by Hamida & Petrowski [24].

4 NUMERICAL EXPERIMENTS

In order to investigate the robustness of the proposed penalty procedure, several optimization problems from the literature are solved using a simple generational GA with Gray code, rank-based selection and elitism. The operation of recombination was applied with probability $p_r = 0.8$. Standard one-point, two-point and uniform crossover operators were applied each one with its respective relative probability (in this work, $p_c^1 = 0.2, p_c^2 = 0.4$ and $p_c^u = 0.4$). Mutation was applied bit-wise to the offsprings with rate $p_m = 0.03$. The equality constraints were converted into one inequality constraint bounding the absolute value of the degree of violation by 0.0001 (That is, $|h(x)| \leq 0.0001$).

4.1 Test-problem 1

In the first example to be investigated, from [6], the function to be minimized and the constraints are given respectively by

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2,$$
$$4.84 - (x_1 - 0.05)^2 - (x_2 - 2.5)^2 \geq 0,$$
$$x_1^2 + (x_2 - 2.5)^2 - 4.84 \geq 0$$

The search space is bounded by $0 \leq x_i \leq 6, i = 1, 2$ and the optimum solution is $(x_1, x_2) = (2.246826, 2.381865)$. with a function value $f(x) = 13.59085$. The number of generations allowed was 50 and the best solution found was $(2.2468493, 2.3823017)$ corresponding to $f(x) = 13.59085$, the worst one was $(1.5048537, 4.1170674)$ with $f(x) = 152.54840$ and the average value was $f(x) = 30.7488$. In this Test-problem 50 independent runs were performed with a population size equal to 50 and each variable coded with 20 bits. Table 1 presents a comparison of results for this test-problem.

	This study	Ref.[6]
best	13.59085	13.59658
worst	152.54840	244.11616

Table 1: Comparison of results on Test-problem1. The optimum value is 13.59085.

One can note that the results found for the Test-problem 1 are better than those found in the reference [6] using the same number of function evaluations.

4.2 Test-problem 2

In this test problem, from Michalewicz[25], the function

$$f(x, \mathbf{y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5$$

is to be minimized over the set $y_3 \leq 1$, $y_4 \leq 1$, $y_5 \leq 2$, $x \geq 0$ and $y_i \geq 0$ for $1 \leq i \leq 5$. subject to

$$x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leq 16$$
$$-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leq -1$$
$$2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 \leq 24$$
$$0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 \leq 12$$
$$-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leq 3$$

The global solution is $(x, \mathbf{y}^*) = (0, 6, 0, 1, 1, 0)$ with $f(x, \mathbf{y}^*) = -11$. Using a population size of 70, three sets of 10 independent runs allowing for 1000, 5000 and 20000 generations were performed. Each variable was coded with 25 bits and the results are shown in the Table 2. Michalewicz [25], using GENECOP, was able to find the optimum solution in 1000 generations for all runs. However, it must be noted that GENECOP is designed to maintain feasibility for that particular constraint set (*linear* inequalities).

	maxgen		
	1000	5000	20000
x	0.000001907	0.000000715	0.000000000
y_1	6.000138938	5.998155534	5.999983966
y_2	0.002714396	0.000119925	0.000008345
y_3	0.999999881	1.000000000	1.000000000
y_4	0.989082306	0.999896646	0.999980778
y_5	0.002151549	0.000607729	0.000004590
best	-10.98587	-10.99879	-10.99997
average	-10.96019	-10.99447	-10.99965
worst	-10.90891	-10.98327	-10.99887

Table 2: Comparison of results in the Test-problem 2.

4.3 Test-problem 3

The previous test-problems presented only continuous variables. The present one, from [26], involves continuous as well as discrete variables. The function to be maximized is

$$f(x_1, x_2, x_3, y_1, y_2) = -5.357854x_1^2 -$$
$$0.835689y_1x_3 - 37.29329y_1 + 40792.141$$

subject to

$$a_1 + a_2y_2x_3 + a_3y_1x_2 - a_4x_1x_3 \leq 92$$
$$a_5 + a_6y_2x_3 + a_7y_1y_2 - a_8x_1^2 - 90 \leq 20$$
$$a_9 + a_{10}x_1x_3 + a_{11}y_1x_1 - a_{12}x_1x_2 - 20 \leq 5$$

The range of the continuous variables x_1, x_2 and x_3 is $[27, 45]$ and the ranges of the integer variables y_1 and y_2 are $[78, 102]$ and $[33, 45]$, respectively. The coefficients a_1 to a_{12} are given in the Table 3.

$a_1 = 85.334407$	$a_5 = 80.51249$	$a_9 = 9.3009610$
$a_2 = 0.0056858$	$a_6 = 0.0071317$	$a_{10} = 0.0047026$
$a_3 = 0.0006262$	$a_7 = 0.0029955$	$a_{11} = 0.0012547$
$a_4 = 0.0022053$	$a_8 = 0.0021813$	$a_{12} = 0.0019085$

Table 3: Coefficients for Test-problem 3.

The global solution is, for any value of x_2 and y_2, $(27, x_2, 27, 78, y_2)$. The number of bits per variable was 25, except for the variables y_1 and y_2 which were coded with 8 bits each. The number of generations allowed was 100 and the population size was set to 250. All 10 runs produced the optimal solution with $f = 32217.4$. In Table 4 a comparison is made between some of the results obtained by Costa & Oliveira [26] for this problem. The first column corresponds to a standard penalty scheme – eq. (1) – (3) – with $k = 10^3$, the second one to Deb's penalty scheme (eq. 4) and the last one to the present study. The search process was terminated by Costa & Oliveira [26] when: (i) 1000 generations were performed or (ii) no improvement was observed after 50 generations or (iii) all the population converged to a single solution. The Table 4 shows the average number of function evaluations (avgnf) and the percentage of runs finding the optimal solution (% success).

	standard ($k = 10^3$)	Deb's	This study
avgnf	37167	35255	25000
% success	100	100	100

Table 4: Comparison of results on Test-problem 3.

4.4 Test-problem 4

This test corresponds to a mechanical design minimization problem studied by Deb [6]. The design variables are $\{h, l, t, b\}$, with bounds $0.125 \leq h \leq 10$ and $0.1 \leq l, t, b \leq 10$, and the objective function is

$$f_w = 1.10471h^2l + 0.04811tb(14.0 + l)$$

subject to

$$13,600 - \tau(x) \geq 0, \quad 30,000 - \sigma(x) \geq 0,$$
$$b - h \geq 0,\ P_c(x) - 6,000 \geq 0,\ 0.25 - \delta(x) \geq 0$$

The expressions for $\tau(x)$, $\sigma(x)$, $P_c(x)$ and $\delta(x)$ are:

$$\tau(x) = \sqrt{\left(\tau^{'}(x)\right)^2 + \left(\tau^{''}(x)\right)^2 + l\tau^{'}(x)\tau^{''}(x)/c}$$
$$c = \sqrt{0.25\left(l^2 + (h+t)^2\right)}, \quad \sigma(x) = \frac{504000}{t^2b}$$
$$P_c(x) = 64746.022\left(1 - 0.0282346t\right)tb^3$$
$$\delta(x) = \frac{2.1952}{t^3b}, \quad \tau^{'}(x) = \frac{6000}{\sqrt{2}hl}$$
$$\tau^{''}(x) = \frac{6000\left(14 + 0.5l\right)c}{2\left\{0.707hl\left(l^2/12 + 0.25\left(h+t\right)^2\right)\right\}}$$

In [6] Deb investigates two situations in order to compare the results for this example. The first one corresponds to a binary coded GA with standard penalty scheme – eq. (1) - (3) – and four constant penalty coefficients ($k = 10^0$, $k = 10^1$, $k = 10^3$ and $k = 10^6$). The population size was set to 80, the maximum number of generations was 500 and 50 independent runs were performed. Each variable was coded with 10 bits and the total length of the chromosome was 40. The Table 5 shows the comparison of results and it is clear that in this study better values were found.

	best	worst
$k = 10^0$	2.41324	483.50177
$k = 10^1$	3.14206	7.45453
$k = 10^3$	3.38277	10.65891
$k = 10^6$	3.72929	9.42353
This study	2.39623	3.39956

Table 5: Comparison of results on Test-problem 4 using standard penalty and the proposed technique.

The second situation (referred to as the "welded beam revisited" by Deb) corresponds to a real coded GA proposed by Deb where, among other features, niching was used. The population size was set to 80. Two series of 50 runs were performed allowing for 500 and 4000 generations respectively. Each variable was coded with 30 bits generating a chromosome 120 bits long. Results for this test are shown in Table 6.

	maxgen	This study	Ref. [6]	
			w/o niching	with niching
best	500	2.38352	2.44271	2.38119
	4000	2.38159	—	2.38119
worst	500	3.73060	7.44425	2.64583
	4000	2.95533	—	2.38355

Table 6: Comparison of results on Test-problem 4 using Deb's real coded GA and the proposed technique.

It is clear that Deb's results only improve when niching is added to his penalty scheme.

4.5 The G-Suite

In this section the 11 well known G1-G11 test-problems presented by Koziel & Michalewicz [1] are considered. The G-Suite is made up of distinct kinds of functions and

involves constraints given by linear inequalities LI, nonlinear equalities NE and nonlinear inequalities NI. The summary of the Test-cases is presented in the Table 7 where the column "n" shows the number of parameters, "a" indicates the number of active constraints at the optimum solution and f^* denotes the known optimum value of f. More details of each of these problems can be found in [1] and [5]. An extended discussion involving each one of these problems and different techniques from the evolutionary computation literature can be found in [27].

Three set of experiments were performed using a population size of 70. For the first experiment 20 independent runs were performed with the maximum number of generations set to 5000. In the second experiment the maximum number of generations is set to 20000. Finally, the third experiment consists of 10 independent runs (with 5000 generations) where the best solution obtained in the first experiment is introduced in the initial population.

Name	n	Type of $f(x)$	LI	NE	NI	a	f^*
G1	13	quadratic	9	0	0	6	-15.0
G2	20	nonlinear	9	0	6	1	0.803553
G3	10	polynomial	0	1	0	1	1.0
G4	5	quadratic	0	0	6	2	-30655.5
G5	4	cubic	2	3	0	3	5126.4981
G6	2	cubic	0	0	2	2	-6961.8
G7	10	quadratic	3	0	5	6	24.306
G8	2	nonlinear	0	0	2	0	0.0958250
G9	7	polynomial	0	0	2	0	680.63
G10	8	linear	3	0	3	6	7049.33
G11	2	quadratic	0	1	0	1	0.75

Table 7: Summary of 11 Test Cases.

A comparison of the results obtained with the simple adaptive procedure proposed here with those obtained by Koziel & Michalewicz [1] is made in the Tables 8, 9, 10, 11, 12 and 13 where the best results are indicated in **boldface**.

It can be noted that the proposed scheme provides better results than those obtained using the more complex homomorphous mapping technique of Koziel & Michalewicz [1].

The same observation applies to the results recently obtained by Wright & Farmani [23] using a binary code.

5 CONCLUSIONS

A new simple adaptive parameter-less penalty scheme for the solution of constrained optimization problems via genetic algorithms has been proposed. Its main feature, besides being adaptive and not requiring any parameter, is to automatically define a different penalty parameter for each constraint.

In all problems tested so far (including tests not shown here) the procedure has produced good results when compared to the literature available. It must also be emphasized that such good results were obtained in spite of the use of an otherwise very naive binary coded genetic algorithm with standard operators. Good results reported in the literature are often obtained with real coding, more sophisticated selection schemes, more aggressive genetic operators and by exploring the particular structure of the constraint set.

The procedure proposed here is simpler, can be implemented for binary coded genetic algorithms leading to good results for mixed continuous-discrete optimization problems. Additional tests in larger real-world applications are being conducted with good results so far.

Acknowledgements

The authors acknowledge the support received from CNPq (301233/86-1 and 475398/2001-7) and FAPEMIG (TEC-692/99). The authors would also like to thank the reviewers for the corrections and suggestions which helped improve the quality of the paper.

References

[1] S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[2] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In H-M. Voigt; W. Ebeling; I. Rechenberg and H-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN IV*, volume 1141, pages 245–254, Berlin, 1996. Springer-Verlag. LNCS.

[3] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[4] J.-H. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 2(1):129–140, 1997.

[5] T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[6] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, June 2000.

[7] P. Husbands. Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In T. Fogarty, editor, *Evolutionary Computing*, volume 865, pages 150–165. Springer-Verlag, 1994. Lecture Notes in Computer Science.

[8] H.J.C. Barbosa. A coevolutionary genetic algorithm for constrained optimization problems. In *Proc. of the Congress on Evolutionary Computation*, pages 1605–1611, Washington, DC, USA, 1999.

[9] R. Hinterding and Z. Michalewicz. Your brains and my beauty: Parent matching for constrained optimization. In *Proc. of the Fifty Int. Conf. on Evolutionary Computation*, pages 810–815, Alaska, May 4-9 1998.

[10] S. Koziel and Z. Michalewicz. A decoder-based evolutionary algorithm for constrained optimization problems. In T. Bäck; A.E. Eiben; M. Schoenauer and H.-P. Schwefel, editors, *Proc. of the Fifth Parallel Problem Solving from Nature*, Amsterdam, September 27-30 1998. Springer-Verlag. Lecture Notes in Computer Science.

[11] J.A Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In Z. Michalewicz; J.D. Schaffer; H.-P. Schwefel; D.B. Fogel and H. Kitano, editors, *Proc. of the First IEEE Int. Conf. on Evolutionary Computation*, pages 579–584, June 19-23 1994.

[12] Z. Michalewicz and N. Attia. Evolutionary optimization of constrained problems. In *Proc. of the Third Annual Conference on Evolutionary Programming*, pages 98–108, River Edge, 1994. World Scientific Publishing.

[13] D.W. Coit, A.E. Smith, and D.M. Tate. Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing*, 6(2):173–182, 1996.

[14] S.E. Carlson and R. Shonkwiler. Annealing a genetic algorithm over constraints. In *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, pages 3931–3936, 1998.

[15] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation. In *Proc. of the 4th Int. Conf. on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press.

[16] Z. Michalewicz, D. Dasgupta, R.G. Le Riche, and M. Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(2):851–870, 1996.

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

[18] J.C. Bean and A.B. Alouane. A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Departament of Industrial and Operations Engineering, The University of Michigan, 1992.

[19] M. Schoenauer and S. Xanthakis. Constrained GA optimization. In S. Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580, Los Altos, CA, 1993. Morgan Kaufmann Publishers.

[20] R.G. Le Riche, C. Knopf-Lenoir, and R.T. Haftka. A segregated genetic algorithm for constrained structural optimization. In L.J. Eshelman, editor, *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 558–565, Pittsburgh, PA., July 1995.

[21] D. Powell and M.M. Skolnick. Using genetic algorithms in engineering design optimization with nonlinear constraints. In S. Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 424–430, San Mateo, CA, 1993. Morgan Kaufmann.

[22] S.B. Hamida and M. Schoenauer. An adaptive algorithm for constrained optimization problems. In *Parallel Problem Solving from Nature – PPSN VI*, volume 1917, pages 529–538, Berlim, 2000. Springer-Verlag. Lectures Notes in Computer Science.

[23] J.A. Wright and R. Farmani. Genetic algorithms: A fitness formulation for constrained minimization. In *Proc. of the Genetic and Evolutionary Computation Conference – GECCO 2001*, pages 725–732, San Francisco, CA., 2001. Morgan Kaufmann.

[24] S. B. Hamida and A. Petrowski. The need for improving the exploration operators for constrained optimization problems. In *2000 Congress on Evolutionary Computation*, pages 1176–1183, San Diego, CA, USA, July 2000. IEEE Service Center.

[25] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[26] L. Costa and P. Oliveira. Evolutionary algorithm approach to the solution of mixed integer non-linear programming problems. *Computers and Chemical Engineering*, 25:257–266, 2001.

[27] Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 1999.

Function	Experiment 1		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.6002978	0.7724640	0.7031971
G3	0.9391756	0.9939078	0.9757277
G4	**−30660.76**	**−30665.24**	**−30663.40**
G5	**6040.595**	**5126.571**	**5389.364**
G6	**−6961.779**	**−6961.796**	**−6961.789**
G7	42.01618	**24.86371**	29.86465
G8	**0.0725015**	**0.0958250**	**0.0926157**
G9	**682.1562**	**680.7590**	**681.4076**
G10	**10002.93**	**7086.404**	**8161.997**
G11	0.7579745	**0.75**	0.7503349

Table 8: Experiment 1.

Function	Experiment 1		
	worst	best	average
G1	−14.0566	−14.7207	−14.4609
G2	**0.78427**	**0.79506**	**0.79176**
G3	**0.9917**	**0.9983**	**0.9965**
G4	−30617.0	−30662.5	−30643.8
G5	—	—	—
G6	−4236.7	−6901.5	−6191.2
G7	**38.682**	25.132	**26.619**
G8	0.0291434	**0.095825**	0.0871551
G9	682.88	681.43	682.18
G10	11894.5	7215.8	9141.7
G11	**0.75**	**0.75**	**0.75**

Table 9: Experiment 1 (Koziel & Michalewicz [1]).

Function	Experiment 2		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.6499022	0.7918570	0.7514353
G3	**0.9983935**	1.000307	**0.9997680**
G4	**−30664.91**	**−30665.51**	**−30665.29**
G5	**6040.595**	**5126.571**	**5389.347**
G6	**−6961.796**	**−6961.796**	**−6961.796**
G7	33.07581	24.85224	27.90973
G8	**0.0795763**	**0.0958250**	**0.0942582**
G9	**681.6396**	**680.6678**	**680.9640**
G10	**9977.767**	**7080.107**	**8018.938**
G11	**0.75**	**0.75**	**0.75**

Table 10: Experiment 2.

Function	Experiment 2		
	worst	best	average
G1	−14.6154	−14.7864	−14.7082
G2	**0.79119**	**0.79953**	**0.79671**
G3	0.9978	**0.9997**	0.9989
G4	−30643.8	−30645.9	−30655.3
G5	—	—	—
G6	−5473.9	−6952.1	−6342.6
G7	**25.069**	**24.62**	**24.826**
G8	0.0291438	**0.095825**	0.0891568
G9	683.18	680.91	681.16
G10	9659.3	7147.9	8163.6
G11	**0.75**	**0.75**	**0.75**

Table 11: Experiment 2 (Koziel & Michalewicz [1]).

Function	Experiment 3		
	worst	best	average
G1	**−15.00**	**−15.00**	**−15.00**
G2	0.7729277	0.7780233	0.7741776
G3	**0.9973952**	**0.9997135**	**0.9987124**
G4	**−30665.25**	**−30665.51**	**−30665.39**
G5	**5126.571**	**5126.571**	**5126.571**
G6	**−6961.796**	**−6961.796**	**−6961.796**
G7	**24.86371**	**24.86130**	**24.86346**
G8	0.0918033	**0.0958250**	0.0940601
G9	**680.7590**	**680.7222**	**680.7461**
G10	**7082.667**	**7080.328**	**7081.146**
G11	**0.75**	**0.75**	**0.75**

Table 12: Experiment 3.

Function	Experiment 3		
	worst	best	average
G1	−14.5732	−14.7184	−14.6478
G2	**0.78279**	**0.79486**	**0.78722**
G3	0.996	0.9978	0.997
G4	−30645.6	−30661.5	−30653.1
G5	—	—	—
G6	−6390.6	−6944.4	−6720.4
G7	26.182	25.09	25.545
G8	**0.0958246**	**0.0958250**	**0.0958248**
G9	683.58	681.72	682.56
G10	7685.8	7321.2	7498.6
G11	**0.75**	**0.75**	**0.75**

Table 13: Experiment 3 (Koziel & Michalewicz [1]).