An Adaptive Penalty Scheme for Steady-State Genetic Algorithms

Helio J.C. Barbosa[1] and Afonso C.C. Lemonge[2]

[1] LNCC/MCT
Rua Getulio Vargas 333
25651 070 Petropolis RJ, BRAZIL
hcbm@lncc.br
[2] Depto. 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 steady-state genetic algorithms applied to constrained optimization problems is proposed. For each constraint, a penalty parameter is adaptively computed along the run according to information extracted from the current population such as the existence of feasible individuals and the level of violation of each constraint. Using real coding, rank-based selection, and operators available in the literature, very good results are obtained.

1 Introduction

Evolutionary algorithms (EAs) are weak search algorithms which can be directly applied to unconstrained optimization problems where one seeks for an element x belonging to the search space S, which minimizes (or maximizes) the real function f. Such EAs usually employ a fitness function closely related to f.

The straightforward application of EAs to constrained optimization problems (COPs) is not possible due to the additional requirement that a set of constraints must be satisfied. Several difficulties may arise: (i)the objective function may be undefined for some or all infeasible elements, (ii)the check for feasibility can be more expensive than the computation of the objective function value, and (iii)an informative measure of the degree of infeasibility of a given candidate solution is not easily defined. It is easy to see that even if both the objective function $f(x)$ and a measure of constraint violation $v(x)$ are defined for all $x \in S$ it is not possible to know in general which of two given infeasible solutions is closer to the optimum and thus should be operated upon or kept in the population. For minimization problems, for instance, one can have $f(x_1) > f(x_2)$ and $v(x_1) = v(x_2)$ or $f(x_1) = f(x_2)$ and $v(x_1) > v(x_2)$ and still have x_1 closer to the optimum.

It is also important to note that –for convenience and easier reproducibility– most comparisons between EAs in the literature have been conducted in problems with constraints which can be written as $g_i(x) \leq 0$, where each $g_i(x)$ is a given *explicit* function of the independent(design) variable $x \in \mathbb{R}^n$. Although

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 718–729, 2003.

the available test problems attempt to represent different types of difficulties one is expected to encounter when dealing with practical situations, very often the constraints cannot be put explicitly in the form $g_i(x) \leq 0$. For instance, in structural engineering design most constraints (such as stress and deformation) are only known as implicit functions of the design variables. In order to check if a constraint has been violated, a whole computational simulation (carried out by a specific code expending considerable computational resources) is required.

The techniques for handling constraints within EAs can be classified either as *direct* (feasible or interior), when only feasible elements in S are considered or as *indirect* (exterior), when both feasible and infeasible elements are used during the search process. Direct techniques comprise the use of: a) *closed* genetic operators (in the sense that when applied to feasible parents they produce feasible offspring) which can be designed provided enough domain knowledge is available [1], b) special decoders [2] (which always generate feasible individuals from any given genotype) although no applications considering implicit constraints have been published, c) repair techniques [3,4] which use domain knowledge in order to move an infeasible offspring into the feasible set (a challenge when implicit constraints are present), and d) "the death penalty", when any infeasible element is simply discarded irrespective of its potential information content.

Summarizing, direct techniques are problem dependent (with the exception of the "death penalty") and actually of extremely reduced practical applicability.

Indirect techniques comprise the use of: a) Lagrange multipliers [5], which may also lead to a min-max problem defined for the associated Lagrangean $\mathcal{L}(x, \lambda)$ where the primal variables x and the multipliers λ are approximated by two different populations in a coevolutionary GA [6], b) fitness as well as constraint violation values in a multi-objective optimization setting [7], c) special selection techniques [8], and d) "lethalization": any infeasible offspring is just assigned a given, very low, fitness value [9]. For other methods proposed in the evolutionary computation literature see [1,10,11,12,13] and references therein.

Methods to tackle COPs which require the knowledge of constraints in explicit form have thus limited practical applicability. This fact, together with simplicity of implementation are perhaps the main reasons why penalty techniques, in spite of their shortcomings, are the most popular ones.

In a previous paper [14] a penalty scheme which does not require the knowledge of the explicit form of the constraints as a function of the decision/design variables and is free of parameters to be set by the user was developed. In contrast with previous approaches where a single penalty parameter is used for all constraints, an adaptive scheme automatically sizes the penalty parameter corresponding to *each* constraint along the evolutionary process. However, the method was conceived for a *generational* genetic algorithm (GA), where the fitness of the whole population is computed at each generation.

In this paper, the procedure proposed in [14] is extended to the case of a steady-state GA where, in each "generation", usually only one or two (in general just a few) new individuals are introduced in the population. Substantial

modifications were necessary in order to finally obtain a robust procedure capable of reaching very good results in a standard test-problem suite.

In the next section the penalty method and some of its implementations within EAs are presented. In Section 3 the proposed adaptive scheme for steady-state GAs is discussed, Section 4 presents numerical experiments with several test-problems from the literature and the paper closes with some conclusions.

2 Penalty Methods

A standard COP 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.

Penalty techniques can be classified as *multiplicative* or *additive*. In the multiplicative case [15], a positive penalty factor $p(v(x), T)$ is introduced in order to amplify the value of the fitness function of an infeasible individual in a minimization problem. One would have $p(v(x), T) = 1$ for a feasible candidate solution x and $p(v(x), T) > 1$ otherwise. Also, $p(v(x), T)$ increases with the "temperature" T and with constraint violation. An initial value for the temperature is required as well as the definition of a function such that T grows with the generation number. This type of penalty has received much less attention in the evolutionary computation (EC) community than the additive type. In the additive case, a penalty functional is added to the objective function in order to define the fitness value of an infeasible element. They can be further divided into: (a)*interior* techniques[1] and (b)*exterior* techniques, where a penalty functional is introduced

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

such that $P(x) = 0$ if x is feasible and $P(x) > 0$ otherwise (for minimization problems). In both cases, as $k \to \infty$, the sequence of minimizers of the unconstrained problem converges to the solution of the original constrained one.

Defining the amount of violation of the j-th constraint by the candidate solution $x \in R^n$ as

$$v_j(x) = \begin{cases} |h_j(x)|, & \text{for an equality constraint,} \\ \max\{0, -g_j(x)\} & \text{otherwise} \end{cases}$$

it is 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

$$P(x) = \sum_{j=1}^{m} (v_j(x))^{\beta} \tag{2}$$

[1] When a barrier functional, which grows rapidly as x approaches the boundary of the feasible domain, is added to the objective function.

where $\beta = 2$. Although it is easy to obtain the unconstrained problem, the definition of a good penalty parameter k is usually a time-consuming trial-and-error process.

Powell & Skolnick[16] proposed a method enforcing the superiority of any feasible solution over any infeasible one defining the fitness as

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

where $\theta(t, x)$ is conveniently defined and r is a constant.

A variant, (see Deb[17]) uses the fitness function:

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

where f_{max} is the objective function value of the worst feasible solution.

Besides the widely used case of a single constant penalty parameter k, several other proposals are available [18,10,19] and some of them, more closely related to the work presented here, will be briefly discussed in the following.

2.1 Related Methods in the Literature

Two-level Penalties. Le Riche et al.[20] present a GA where two fixed penalty parameters k_1 and k_2 are used independently in two different populations. The idea is to create two sets of candidate solutions where one of them is evaluated with the parameter k_1 and the other with the 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 offspring near the boundary between the feasible and infeasible regions.

Multiple Coefficients. Homaifar et al.[21] proposed different penalty coefficients for different levels of violation of each constraint. The fitness function is written as

$$F(x) = f(x) + \sum_{j=1}^{m} k_{ij} (v_j(x))^2$$

where i denotes one of the l levels of violation defined for 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, $m(2l+1)$, of parameters that must be set by the user for each problem.

Dynamic Coefficients. Joines & Houck[22] proposed that the penalty parameters should vary dynamically along the search according to an exogenous schedule. The fitness function $F(x)$ was written as in (1) and (2) with the penalty parameter, given by $k = (C \times t)^{\alpha}$, increasing with the generation number t.

Adaptive Penalties. A procedure where the penalty parameters change according to information gathered during the evolution process was proposed by Bean & Hadj-Alouane[23]. The fitness function is again given by (1) and (2) but with the penalty parameter $k = \lambda(t)$ adapted at each generation by the rules:

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

where b^i is the best element at generation i, $\mathcal{F}$ is the feasible set, $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$. In this method the penalty parameter of the next generation $\lambda(t+1)$ decreases when all best elements in the last g generations were feasible, increases if all best elements were infeasible and otherwise remains without change.

The method proposed by Coit et al.[24], uses 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 & Xanthakis[25] presented a strategy that handles constrained problems in stages: (i) initially, a randomly generated population is evolved 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.

Recently, Hamida & Schoenauer[26] proposed an adaptive scheme using a niching technique with adaptive radius to handle multimodal functions.

Other Techniques. Runarsson & Yao[8] presented a novel approach where a good balance between the objective and the penalty function values is sought by means of a stochastic ranking scheme. However, there is a parameter, P_f, (the probability of using only the objective function for ranking infeasible individuals) that must be set by the user.

Later, Wright & Farmani[27] proposed a method that requires no parameters and aggregates all constraint violations in a single infeasibility measure.

For constraint satisfaction problems, adaptive EAs have been developed succesfuly by Eiben and co-workers (see [28]).

3 The Proposed Method

In a previous paper[14] a penalty scheme was proposed which adaptively sizes the penalty coefficient of *each* constraint using information from the population

such as the average of the objective function and the level of violation of each constraint. The fitness function was written as:

$$F(x) = \begin{cases} f(x), & \text{if } x \text{ is feasible,} \\ \mathtt{h}(x) + \sum_{j=1}^{m} k_j v_j(x) & \text{otherwise} \end{cases} \tag{3}$$

where

$$\mathtt{h}(x) = \begin{cases} f(x), & \text{if } f(x) > \langle f(x) \rangle, \\ \langle f(x) \rangle & \text{otherwise} \end{cases} \tag{4}$$

and $\langle f(x) \rangle$ is the average of the objective function values in the current population. The penalty parameter was defined at each *generation* by:

$$k_j = |\langle f(x) \rangle| \frac{\langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \tag{5}$$

and $\langle v_l(x) \rangle$ is the violation of the l-th constraint averaged over the current population. The idea is that the penalty coefficients should be distributed in such a way that those constraints which are more difficult to be satisfied should have a relatively higher penalty coefficient. It is also clear that the notion of the superiority of any feasible over any infeasible solution[16] is not enforced here.

It must be observed that in all procedures where a penalty coefficient varies along the run one must ensure that the fitness value of all elements is computed with the same penalty coefficient(s) so that standard selection schemes remain valid. For a generational GA, one can simply update the coefficient(s) every, say g, generations. As the concept of generation does not hold for a steady-state GA extra care must be taken in order to ensure that selection (for reproduction as well as for replacement) works properly.

A straightforward extension of that penalty procedure[14] to the steady-state case would be to periodically update the penalty coefficients and the fitness function values for the population. However, in spite of using a real-coding, the results obtained were inferior to those of the binary-coded generational case[14].

Further modifications are then proposed here for the steady-state version of that penalty scheme. The fitness function is still computed according to (3). However, $\mathtt{h}$ and the penalty coefficients are redefined respectively as

$$\mathtt{h} = \begin{cases} f(x_{worst}) & \text{if there is no feasible element in the population,} \\ f(x_{bestfeasible}) & \text{otherwise} \end{cases} \tag{6}$$

$$k_j = \mathtt{h} \frac{\langle v_j(x) \rangle}{\sum_{l=1}^{m} [\langle v_l(x) \rangle]^2} \tag{7}$$

Also, every time a better feasible element is found (or the number of new elements inserted into the population reaches a certain level) $\mathtt{h}$ is redefined and all fitness values are recomputed using the updated penalty coefficients. The updating of each penalty coefficient is performed in such a way that no reduction in its value is allowed. For convenience one should keep, for each individual in the population, the objective function value and all constraint violations . The fitness function value is then computed using (6), (7), and (3).

It is clear from the definition of `h` in (6) that if no feasible element is present in the population one is actually minimizing a measure of the distance of the individuals to the feasible set since the actual value of the objective function is not taken into account. However, when a feasible element is found then it immediately enters the population since, after updating all fitness values using (6), (7), and (3), it becomes the element with the best fitness value.

A pseudo-code for the proposed adaptive penalty scheme for a steady-state GA can be written as shown in Figure 1. Numerical experiments are then presented in the following section.

```
Begin
Initialize population
Compute objective function and constraint violation values
if there is no feasible element then
   h = worst objective function value
else
   h = objective function value of best feasible individual
endif
Compute penalty coefficients
Compute fitness values
ninser = 0
repeat
  Select operator
  Select parent(s)
  Generate offspring
  Evaluate offspring
  Keep best offspring
  if offspring is the new best feasible element then
     update penalty coefficients and fitness values
     ninser = 0
  endif
  if offspring is better than the worst in the population then
     worst is removed
     offspring is inserted
     ninser = ninser + 1
  endif
  if (ninser/popsize >= r) then
     update penalty coefficients and fitness values
     ninser = 0
  endif
until maximum number of evaluations is reached
End
```

Fig. 1. Pseudo-code for the steady-state GA with adaptive penalty scheme. (`ninser` is a counter for the number of offspring inserted in the population, `popsize` is the population size and `r` is a fixed constant that was set to 3 in all cases)

4 Numerical Experiments

In order to investigate the performance of the proposed penalty procedure, the 11 well known G1-G11 test-functions presented by Koziel & Michalewicz[2] are considered. The G-Suite is made up of different kinds of functions and involves constraints given by linear inequalities, nonlinear equalities, and nonlinear inequalities. An extended discussion involving each one of these problems and other techniques from the evolutionary computation literature can be found in [29].

A simple real-coded steady-state GA with a linear ranking selection scheme was implemented. The operators used were: (i) random mutation (which modifies a randomly chosen variable of the selected parent to a random value uniformly distributed between the lower and upper bounds of the corresponding variable), (ii) non-uniform mutation (as proposed by Michalewicz[30]), (iii) Muhlenbein's mutation (as described in [31]), (iv) multi-parent discrete crossover (which generates an offspring by randomly taking each allele from one of the n_p selected parents), and (v) Deb's SBX crossover as described in [32].

No parameter tuning was attempted. The same probability of application (namely 0.2) was assigned to all operators above, n_p was set to 4, and η was set to 2 in SBX. This set of values was applied to *all* test-problems in order to demonstrate the robustness of the procedure. Each equality constraint was converted into one inequality constraint of the form $|h(x)| \leq 0.0001$. Enlarging the set of operators, changing the relative probabilities of application, population size, or parameters associated with operators in each case could of course lead to local performance gains.

The Tables 1, 2, 3, and 4 show the results obtained for the G1-G11 test-functions, in 20 independent runs, using a population containing 800 individuals and a maximum number of function evaluations n_{eval} set to 320000, 640000, 1120000, and 1440000, respectively. It is clear that good results were found for all test-functions and at all levels of n_{eval}.

The Table 5 displays a comparison of results found in the Experiment 3 (Table 3) –where $n_{eval} = 1120000$– and the results found in the Experiment #2 of [14] where a generational binary-coded GA –with `popsize` $= 70$ and $n_{eval} = 1400000$– was used in 20 independent runs.

The Table 6 compares the results from Experiment 3 with those presented by Hamida & Shoenauer[26] using a $(100 + 300)$–ES segregational selection scheme with an adaptive penalty and a niching strategy. They performed 31 independent runs comprising 5000 generations ($n_{eval} = 1500000$) each.

The Tables 5 and 6 show that better results are obtained with the proposed adaptive steady-state GA using less function evaluations. The interested reader can find additional results in [2,33,27,29], and verify that they are not superior to those presented here.

Finally, in Table 7 we compare the results obtained with the parameter-less scheme proposed here, using `popsize` $= 700$, with those of Runarsson & Yao[8], both with $n_{eval} = 350000$. It must be observed that the results in Table 7 are the best in [8] (and probably the best in the evolutionary computation literature) and correspond to the choice $P_f = 0.45$. However, one can see in [8] that slightly

Table 1. Exp. 1: $n_{eval} = 320000$.

$f(x)$	Experiment 1		
	worst	best	average
G1	−15.00	−15.00	−15.00
G2	0.7701039	0.7980134	0.7894922
G3	0.7468729	0.9970834	0.8733876
G4	−30665.54	−30665.54	−30665.54
G5	5667.431	5126.484	5829.603
G6	−6961.811	−6961.811	−6961.811
G7	27.05797	24.31103	24.86856
G8	0.0958250	0.0958250	0.0958250
G9	680.7184	680.6303	680.64824
G10	10864.27	7139.031	7679.41880
G11	0.749	0.749	0.74899

Table 2. Exp. 2: $n_{eval} = 640000$.

$f(x)$	Experiment 2		
	worst	best	average
G1	−13.00	−15.00	−14.90
G2	0.7624246	0.8036177	0.7904785
G3	0.9318285	1.000491	0.9890722
G4	−30665.54	−30665.54	−30665.54
G5	5632.585	5126.484	5257.531
G6	−6961.811	−6961.811	−6961.811
G7	25.77410	24.32803	24.70925
G8	0.0958250	0.0958250	0.0958250
G9	680.6932	680.6305	680.6385
G10	7786.534	7098.464	7413.0185
G11	0.749	0.749	0.74899

Table 3. Exp. 3: $n_{eval} = 1120000$.

$f(x)$	Experiment 3		
	worst	best	average
G1	−15.00	−15.00	−15.00
G2	0.7778333	0.8036125	0.7900538
G3	0.9593665	1.000498	0.9981693
G4	−30665.54	−30665.54	−30665.54
G5	5639.265	5126.484	5205.561
G6	−6961.811	−6961.811	−6961.811
G7	25.24219	24.31465	24.58272
G8	0.0958250	0.0958250	0.0958250
G9	680.6494	680.6301	680.6333
G10	8361.596	7049.360	7339.957
G11	0.749	0.749	0.74899

Table 4. Exp. 4: $n_{eval} = 1440000$.

$f(x)$	Experiment 4		
	worst	best	average
G1	−15.00	−15.00	−15.00
G2	0.7778334	0.8036024	0.7908203
G3	1.000340	1.000499	1.000460
G4	−30665.54	−30665.54	−30665.54
G5	5672.701	5126.484	5206.389
G6	−6961.811	−6961.811	−6961.811
G7	25.51170	24.30771	24.52875
G8	0.0958250	0.0958250	0.0958250
G9	680.7122	680.6301	680.6363
G10	7942.683	7072.100	7300.013
G11	0.749	0.749	0.74899

Table 5. Results from this study (SSGA) and the generational GA (GGA) of [14].

$f(x)$	optimum	best values		average values		worst values	
		SSGA	GGA	SSGA	GGA	SSGA	GGA
G1	−15.0	−15.00	−15.00	−15.00	−15.00	−15.00	−15.00
G2	0.803619	0.8036125	0.7918570	0.7900538	0.7514353	0.7778333	0.6499022
G3	1.0	1.000498	1.000307	0.9981693	0.9997680	0.9593665	0.9983935
G4	−30655.539	−30665.54	−30665.51	−30665.54	−30665.29	−30665.54	−30664.91
G5	5126.4981	5126.484	5126.571	5205.561	5389.347	5639.265	6040.595
G6	−6961.814	−6961.811	−6961.796	−6961.811	−6961.796	−6961.811	−6961.796
G7	24.306	24.31465	24.85224	24.58272	27.90973	25.24219	33.07581
G8	0.0958250	0.0958250	0.0958250	0.0958250	0.0942582	0.0958250	0.0795763
G9	680.630	680.6301	680.6678	680.6333	680.9640	680.6494	681.6396
G10	7049.33	7049.360	7080.107	7339.957	8018.938	8361.596	9977.767
G11	0.75	0.749	0.75	0.74899	0.75	0.749	0.75

changing that parameter to $P_f = 0.475$ produces changes in the second most relevant digit of the best values found for functions G6 and G10, and severely degrades the mean value for functions G1, G6 and G10. It is clear that our first results presented in this paper are very competitive.

Table 6. Comparison between this study (SSGA) and Hamida & Schoenauer[26]. Average values for this study were computed with feasible and infeasible final solutions. Those in [26] considered only feasible solutions. Worst values were not given in [26].

$f(x)$	optimum	best values		average values	
		SSGA	H&S	SSGA	H&S
G1	−15.0	−15.00	−15.00	−15.00	−14.84
G2	0.803619	0.8036125	0.785	0.7900538	0.59
G3	1.0	1.000498	1.0	0.9981693	0.99989
G4	−30655.539	−30665.54	−30665.5	−30665.54	−30665.5
G5	5126.4981	5126.484	5126.5	5205.561	5141.65
G6	−6961.814	−6961.811	−6961.81	−6961.811	−6961.81
G7	24.306	24.31465	24.3323	24.58272	24.6636
G8	0.0958250	0.0958250	0.095825	0.0958250	0.095825
G9	680.630	680.6301	680.630	680.6333	680.641
G10	7049.33	7049.360	7061.13	7339.957	7497.434
G11	0.75	0.749	0.75	0.74899	0.75

Table 7. Comparison of results between this study (SSGA) and Runarsson & Yao[8].

$f(x)$	optimum	best values		worst values	
		SSGA	R&Y	SSGA	R&Y
G1	−15.0	−15.00	−15.00	−15.00	−15.00
G2	0.803619	0.8035839	0.803515	0.7777818	0.726288
G3	1.0	0.9960645	1.0	0.6716288	1.00
G4	−30655.539	−30665.54	−30665.539	−30644.32	−30665.539
G5	5126.4981	5126.484	5126.497	5624.208	5142.472
G6	−6961.814	−6961.811	−6961.814	−6961.811	−6350.262
G7	24.306	24.32190	24.307	29.82257	24.642
G8	0.0958250	0.0958250	0.095825	0.0958250	0.095825
G9	680.630	680.6304	680.630	680.6886	680.763
G10	7049.33	7102.265	7054.316	7229.3908	8835.655
G11	0.75	0.749	0.75	0.749	0.75

5 Conclusions

A new adaptive parameter-less penalty scheme which is suitable for implementation within steady-state genetic algorithms has been proposed in order to tackle constrained optimization problems. Its main feature, besides being adaptive and not requiring any parameter, is to automatically define a different penalty coefficient for each constraint. The scheme was introduced in a real-coded steady-state

GA and, using available operators from the literature, produced results competitive with the best available in the EC literature, besides alleviating the user from the delicate and time consuming task of setting penalty parameters.

Acknowledgements. The authors acknowledge the support received from CNPq and FAPEMIG. The authors would also like to thank the reviewers for the corrections and suggestions which helped improve the quality of the paper.

References

1. M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In *Parallel Problem Solving from Nature - PPSN IV*, volume 1141, pages 245–254. Springer-Verlag, 1996. LNCS.
2. S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.
3. G.E. Liepins and W.D. Potter. A genetic algorithm approach to multiple-fault diagnosis. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 17, pages 237–250. Van Nostrand Reinhold, New York, New York, 1991.
4. D. Orvosh and L. Davis. Using a genetic algorithm to optimize problems with feasibility contraints. In *Proc. of the First IEEE Conf. on Evolutionary Computation*, pages 548–553, 1994.
5. H. Adeli and N-T. Cheng. Augmented lagrangian genetic algorithm for structural optimization. *Journal of Aerospace Engineering*, 7(1):104–118, January 1994.
6. 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.
7. P.D. Surry and N.J. Radcliffe. The COMOGA method: Constrained optimisation by multiobjective genetic algorithms. *Control and Cybernetics*, 26(3), 1997.
8. T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Trans. on Evolutionary Computation*, 4(3):284–294, 2000.
9. A.H.C. van Kampen, C.S. Strom, and L.M.C. Buydens. Lethalization, penalty and repair functions for constraint handling in the genetic algorithm methodology. *Chemometrics and Intelligent Laboratory Systems*, 34:55–68, 1996.
10. Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.
11. 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.
12. S. Koziel and Z. Michalewicz. A decoder-based evolutionary algorithm for constrained optimization problems. In *Proc. of the Fifth Parallel Problem Solving from Nature*. Springer-Verlag, 1998. Lecture Notes in Computer Science.
13. J.-H. Kim and H. Myung. Evolutionary programming techniques for constrained optimization problems. *IEEE Trans. on Evolutionary Computation*, 2(1):129–140, 1997.
14. H.J.C. Barbosa and A.C.C. Lemonge. An adaptive penalty scheme in genetic algorithms for constrained optimization problems. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 287–294. Morgan Kaufmann Publishers, 2002.

15. 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.
16. D. Powell and M.M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 424–430. Morgan Kaufmann, 1993.
17. K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311–338, June 2000.
18. 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.
19. 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.
20. R.G. Le Riche, C. Knopf-Lenoir, and R.T. Haftka. A segregated genetic algorithm for constrained structural optimization. In *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 558–565, 1995.
21. H. Homaifar, S.H.-Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.
22. J.A Joines and C.R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proc. of the First IEEE Int. Conf. on Evolutionary Computation*, pages 579–584, June 19–23 1994.
23. J.C. Bean and A.B. Alouane. A dual genetic algorithm for bounded integer programs. Dept. of Industrial and Operations Engineering, The University of Michigan, Tech. Rep. 92-53 1992.
24. 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.
25. M. Schoenauer and S. Xanthakis. Constrained GA optimization. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580. Morgan Kaufmann Publishers, 1993.
26. S. Ben Hamida and M. Schoenauer. ASCHEA: new results using adaptive segregational constraint handling. In *Proc. of the 2002 Congress on Evolutionary Computation*, volume 1, pages 884–889, May 2002.
27. J.A. Wright and R. Farmani. Genetic algorithms: A fitness formulation for constrained minimization. In *GECCO 2001: Proc. of the Genetic and Evolutionary Computation Conference*, pages 725–732. Morgan Kaufmann, 2001.
28. A.E. Eiben and J. I. van Hemert. Saw-ing EAs: adapting the fitness function for solving constrained problems. In D. Corne, M. Dorigo, and F. Glover, editors, *New ideas in optimization, chapter 26*, pages 389–402. McGraw-Hill, London, 1999.
29. Z. Michalewicz and D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 1999.
30. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1992.
31. H. Muhlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17(6-7):619–632, Sep 1991.
32. K. Deb and H.G. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evolutionary Computation Journal*, 9(2):197–221, 2001.
33. S.B. Hamida and M. Schoenauer. An adaptive algorithm for constrained optimization problems. In *PPSN VI – LNCS*, volume 1917, pages 529–538. Springer-Verlag, 2000.