ELSEVIER

Comput. Methods Appl. Mech. Engrg. 191 (2002) 1245–1287

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art

Carlos A. Coello Coello [1]

CINVESTAV-IPN, Depto. de Ingeniería Eléctrica, Sección de Computación, Av. Instituto Politécnico Nacional No. 2508, Col. San Pedro Zacatenco, México D.F. 07300, Mexico

Received 20 March 2000; received in revised form 19 June 2001

Abstract

This paper provides a comprehensive survey of the most popular constraint-handling techniques currently used with evolutionary algorithms. We review approaches that go from simple variations of a penalty function, to others, more sophisticated, that are biologically inspired on emulations of the immune system, culture or ant colonies. Besides describing briefly each of these approaches (or groups of techniques), we provide some criticism regarding their highlights and drawbacks. A small comparative study is also conducted, in order to assess the performance of several penalty-based approaches with respect to a dominance-based technique proposed by the author, and with respect to some mathematical programming approaches. Finally, we provide some guidelines regarding how to select the most appropriate constraint-handling technique for a certain application, and we conclude with some of the most promising paths of future research in this area.

Keywords: Evolutionary algorithms; Constraint handling; Evolutionary optimization

1. Introduction

The famous naturalist Charles Darwin defined *Natural Selection* or *Survival of the Fittest* as the *preservation of favorable individual differences and variations, and the destruction of those that are injurious* [33]. In nature, individuals have to adapt to their environment in order to survive in a process called *evolution*, in which those features that make an individual more suited to compete are preserved when it reproduces, and those features that make it weaker are eliminated. Such features are controlled by units called *genes* which form sets called *chromosomes*. Over subsequent generations not only the fittest individuals survive, but also their fittest genes which are transmitted to their descendants during the sexual recombination process which is called *crossover*.

Early analogies between the mechanism of natural selection and a learning (or optimization) process led to the development of the so-called "evolutionary algorithms" (EAs) [2], in which the main goal is to simulate the evolutionary process in a computer. There are three main paradigms within evolutionary algorithms, whose motivations and origins were independent from each other: evolution strategies [156], evolutionary programming [58], and genetic algorithms [77]. However, the current trend has been to

E-mail address: ccoello@cs.cinvestav.mx (C.A. Coello Coello).

[1] Present address: P.O. Box 60326-394, Houston, TX 77205, USA. Tel.: +1-011-52-28-181302; fax: +1-011-52-28-181508.

0045-7825/02/$ - see front matter
PII: S0045-7825(01)00323-1

decrease the difference among these three paradigms and refer (in generic terms) simply to evolutionary algorithms when talking about any of them.

In general, we need the following basic components to implement an EA in order to solve a problem [104]:

1. A representation of the potential solutions to the problem.
2. A way to create an initial population of potential solutions (this is normally done randomly, but deterministic approaches can also be used).
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness".
4. A selection procedure that chooses the parents that will reproduce.
5. Evolutionary operators that alter the composition of children (normally, crossover and mutation).
6. Values for various parameters that the evolutionary algorithm uses (population size, probabilities of applying evolutionary operators, etc.).

EAs have been quite successful in a wide range of applications [3,57,64,67,101,111,130,133,157]. However, an aspect normally disregarded when using them for optimization (a rather common trend) is that these algorithms are unconstrained optimization procedures, and therefore is necessary to find ways of incorporating the constraints (normally existing in any real-world application) into the fitness function.

The most common way of incorporating constraints into an EA have been penalty functions (we will be referring only to exterior penalty functions in this paper) [67,144]. However, due to the well-known difficulties associated with them [144], researchers in evolutionary computing have proposed different ways to automate the definition of good penalty factors, which remains as the main drawback of using penalty functions. Additionally, several researchers have developed a considerable amount of alternative approaches to handle constraints, mainly to deal with specific features of some complex optimization problems in which it is difficult to estimate good penalty factors or to even generate a single feasible solution.

In this paper, we provide a comprehensive survey of constraint-handling techniques that have been adopted over the years to handle all sorts of constraints (linear, non-linear, equality and inequality) in EAs. Each group of approaches is briefly described and discussed, indicating their main advantages and disadvantages. At the end, we conclude with some of the most promising paths of future research in this area.

There are several other surveys on constraint-handling techniques available in the specialized literature (see for example [34,63,103,104,109,161]), but they are either too narrow (i.e., they cover a single group of constraint-handling techniques) or they focus more on empirical comparisons and on the design of interesting test functions. None of these surveys attempt to focus on the discussion of the different aspects of each method or to be as comprehensive as we intend in this paper.

Our main goal is to provide enough (mainly descriptive) information as to allow newcomers in this area to get a very complete picture of the research that has been done and that is currently under way. Since trying to be exhaustive is as fruitless as it is ambitious, we have focused on papers in which the main emphasis is the way in which constraints are handled, and from this subset, we have selected the most representative work available (particularly, when dealing with very prolific authors).

We are interested in the general non-linear programming problem in which we want to:

$$\text{Find } \vec{x} \text{ which optimizes } f(\vec{x}) \tag{1}$$

subject to

$$g_i(\vec{x}) \leqslant 0, \quad i = 1, \ldots, n, \tag{2}$$

$$h_j(\vec{x}) = 0, \quad j = 1, \ldots, p, \tag{3}$$

where $\vec{x}$ is the vector of solutions $\vec{x} = [x_1, x_2, \ldots, x_r]^{\mathrm{T}}$, n is the number of inequality constraints and p is the number of equality constraints (in both cases, constraints could be linear or non-linear).

If we denote with $\mathscr{F}$ to the feasible region and with $\mathscr{S}$ to the whole search space, then it should be clear that $\mathscr{F} \subseteq \mathscr{S}$.

For an inequality constraint that satisfies $g_i(\vec{x}) = 0$, then we will say that is active at $\vec{x}$. All equality constraints h_j (regardless of the value of $\vec{x}$ used) are considered active at all points of $\mathscr{F}$.

The rest of this paper is organized as follows. Section 2 presents penalty functions in several of their variations that have been used with EAs (i.e., static, dynamic, annealing, adaptive, co-evolutionary, and death penalties). Penalty functions are the oldest approach used to incorporate constraints into unconstrained optimization algorithms (including EAs) and, therefore, they are discussed first. Section 3 discusses the use of special representations and genetic operators. The use of operators that preserve feasibility at all times and decoders that transform the shape of the search space are discussed, among other techniques. Section 4 discusses repair algorithms, which are normally used in combinatorial optimization problems in which the traditional genetic operators tend to generate infeasible solutions all (or at least most of) the time. Thus, "repair" refers, in this context, to make valid (or feasible) these individuals through the application of a certain (normally heuristic) procedure. Section 5 covers techniques that handle objectives and constraints separately. From these approaches, the use of multiobjective optimization techniques seems one of the most promising venues of future research in the area. Section 6 discusses approaches that use hybrids with other techniques such as Lagrangian multipliers or fuzzy logic. This section also contains some approaches that constitute very promising paths of future research (e.g., the use of cultural algorithms or the immune system). Section 7 presents a small comparative study in which several penalty-based techniques are compared against a technique based on dominance relations (i.e., one of the techniques discussed in Section 5). As a corollary to the results of this comparative study, Section 8 provides some final suggestions on the choice of constraint-handling techniques for a certain problem. Finally, Section 9 presents some conclusions and some possible paths of future research.

The detailed table of contents of the paper is the following:

2. Penalty functions

The most common approach in the EA community to handle constraints (particularly, inequality constraints) is to use penalties. Penalty functions were originally proposed by Courant in the 1940s [31] and later expanded by Carroll [18] and Fiacco and McCormick [55]. The idea of this method is to transform a constrained-optimization problem into an unconstrained one by adding (or subtracting) a certain value to/from the objective function based on the amount of constraint violation present in a certain solution.

In classical optimization, two kinds of penalty functions are considered: exterior and interior. In the case of exterior methods, we start with an infeasible solution and from there we move towards the feasible region. In the case of interior methods, the penalty term is chosen such that its value will be small at points away from the constraint boundaries and will tend to infinity as the constraint boundaries are approached. Then, if we start from a feasible point, the subsequent points generated will always lie within the feasible region since the constraint boundaries act as barriers during the optimization process [138].

The most common method used in EAs is the exterior penalty approach and therefore, we will concentrate our discussion only on such technique. The main reason why most researchers in the EA community tend to choose exterior penalties is because they do not require an initial feasible solution. This sort of requirement (an initial feasible solution) is precisely the main drawback of interior penalties. This is an important drawback, since in many of the applications for which EAs are intended the problem of finding a feasible solution is itself NP-hard [161].

The general formulation of the exterior penalty function is

$$\phi(\vec{x}) = f(\vec{x}) \pm \left[\sum_{i=1}^{n} r_i \times G_i + \sum_{j=1}^{p} c_j \times L_j \right], \tag{4}$$

where $\phi(\vec{x})$ is the new (expanded) objective function to be optimized, G_i and L_j are functions of the constraints $g_i(\vec{x})$ and $h_j(\vec{x})$, respectively, and r_i and c_j are positive constants normally called "penalty factors".

The most common form of G_i and L_j is

$$G_i = \max[0, g_1(\vec{x})]^{\beta}, \tag{5}$$

$$L_j = |h_j(\vec{x})|^{\gamma}, \tag{6}$$

where β and γ are normally 1 or 2.

Ideally, the penalty should be kept as low as possible, just above the limit below which infeasible solutions are optimal (this is called, the *minimum penalty rule* [39,145,162]). This is due to the fact that if the penalty is too high or too low, then the problem might become very difficult for an EA [39,145,147]. If the penalty is too high and the optimum lies at the boundary of the feasible region, the EA will be pushed inside the feasible region very quickly, and will not be able to move back towards the boundary with the infeasible region. A large penalty discourages the exploration of the infeasible region since the very beginning of the search process. If, for example there are several disjointed feasible regions in the search space, the EA would tend to move to one of them, and would not be able to move to a different feasible region unless they are very close to each other.

On the other hand, if the penalty is too low, a lot of the search time will be spent exploring the infeasible region because the penalty will be negligible with respect to the objective function [161]. These issues are very important in EAs, because many of the problems in which they are used have their optimum lying on the boundary of the feasible region [159,162].

The minimum penalty rule is conceptually simple, but it is not necessarily easy to implement. The reason is that the exact location of the boundary between the feasible and infeasible regions is unknown in many of the problems for which EAs are intended (e.g., in many cases the constraints are not given in algebraic form, but are the outcome generated by a simulator [27]).

It is known that the relationship between an infeasible individual and the feasible region of the search space plays a significant role in penalizing such an individual [144]. However, it is not clear how to exploit this relationship to guide the search in the most desirable direction.

There are at least three main choices to define a relationship between an infeasible individual and the feasible region of the search space [34]:

1. an individual might be penalized just for being infeasible regardless of its amount of constraint violation (i.e., we do not use any information about how close it is from the feasible region),
2. the 'amount' of its infeasibility can be measured and used to determine its corresponding penalty, or
3. the effort of 'repairing' the individual (i.e., the cost of making it feasible) might be taken into account.

Several researchers have studied heuristics on the design of penalty functions. Probably the most well-known of these studies is the one conducted by Richardson et al. [144] from which the following guidelines were derived:

1. Penalties which are functions of the distance from feasibility are better performers than those which are only functions of the number of violated constraints.
2. For a problem having few constraints, and few feasible solutions, penalties which are solely functions of the number of violated constraints are not likely to produce any solutions.
3. Good penalty functions can be constructed from two quantities: the *maximum completion cost* and the *expected completion cost*. The *completion cost* refers to the distance to feasibility.
4. Penalties should be close to the expected completion cost, but should not frequently fall below it. The more accurate the penalty, the better will be the solution found. When a penalty often underestimates the completion cost, then the search may fail to find a solution.

Based mainly on these guidelines, several researchers have attempted to derive good techniques to build penalty functions. The most important will be analyzed next. It should be kept in mind, however, that these guidelines are difficult to follow in some cases. For example, the expected completion cost sometimes has to be estimated using alternative methods (e.g., doing a relative scaling of the distance metrics of multiple constraints, estimating the degree of constraint violation, etc. [161]). Also, it is not clear how to combine the two quantities indicated by Richardson et al. [144] and how to design a fitness function that uses accurate penalties.

Penalty functions can deal both with equality and inequality constraints, and the normal approach is to transform an equality to an inequality of the form

$$|h_j(\vec{x})| - \epsilon \leqslant 0, \tag{7}$$

where ϵ is the tolerance allowed (a very small value).

Most of the approaches analyzed in this paper attempt to avoid this hand-tuning of the penalty factors and some even make unnecessary at all the use of a penalty function.

2.1. Static penalties

Under this category, we consider approaches in which the penalty factors do not depend on the current generation number in any way, and therefore, remain constant during the entire evolutionary process.

Homaifar et al. [78] proposed an approach in which the user defines several levels of violation, and a penalty coefficient is chosen for each in such a way that the penalty coefficient increases as we reach higher levels of violation. This approach starts with a random population of individuals (feasible or infeasible).

An individual is evaluated using [104]

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \sum_{i=1}^{m} (R_{k,i} \times \max[0, g_i(\vec{x})]^2), \tag{8}$$

where $R_{k,i}$ are the penalty coefficients used, m is the total number of constraints (Homaifar et al. [78] transformed equality constraints into inequality constraints), $f(\vec{x})$ is the unpenalized objective function and $k = 1, 2, \ldots, l$, where l is the number of levels of violation defined by the user. The idea of this approach is to balance individual constraints separately by defining a different set of factors for each of them through the application of a set of deterministic rules.

An interesting static penalty approach has been used by Kuri Morales [114]. Fitness of an individual is determined using

$$\text{fitness}(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if the solution is feasible,} \\ K - \sum_{i=1}^{s} \left(\frac{K}{m}\right) & \text{otherwise,} \end{cases} \tag{9}$$

where s is the number of constraints satisfied, m is the total number of (equality and inequality) constraints and K is a large constant (it was set to 1×10^9 [113] in the experiments reported in [114]). Notice that when an individual is infeasible, its fitness is not computed and all the individuals that violate the same number of constraints receive the same penalty regardless of how close they are from the feasible region.

Finally, Hoffmeister and Sprave [76] have proposed to use the following penalty function:

$$\text{fitness}(\vec{x}) = f(\vec{x}) \pm \sqrt{\sum_{i=0}^{m} H(-g_i(\vec{x}))g_i(\vec{x})^2}, \tag{10}$$

where $H : \mathscr{R} \rightarrow \{0, 1\}$ is the Heaviside function

$$H(y) = \begin{cases} 1, & y > 0, \\ 0, & y \leqslant 0. \end{cases} \tag{11}$$

This is equivalent to a partial penalty approach and was successfully used in some real-world problems [155].

2.1.1. Advantages and disadvantages

The main drawback of the approach of Homaifar et al. is the high number of parameters required. For m constraints, this approach requires $m(2l + 1)$ parameters in total [102]. So, if we have, for example, six constraints and two levels, we would need 30 parameters, which is a very high number considering the small size of the proposed problem. Also, this method requires prior knowledge of the degree of constraint violation present in a problem (to define the levels of violation), which might not be always given (or easy to obtain) in real-world applications.

Kuri's approach does not use information about the amount of constraint violation, but only about the number of constraints that were violated. Although this contradicts one of the basic rules stated by Richardson et al. [144] about the definition of good penalty functions, apparently the self-adaptive EA used by Kuri (called Eclectic Genetic Algorithm or EGA for short) could cope with this problem and was able to optimize several difficult nonlinear optimization problems. In one of the functions reported in [114], however, it was necessary to initialize the population with another EGA because no feasible solutions were present in the first generation. This problem was obviously produced by the lack of diversity in the population (not having a single feasible individual in the population, they all had a very similar or equal fitness), which seriously limits its applicability in highly constrained search spaces.

The problem with Hoffmeister and Sprave's approach is that it is based on the assumption that infeasible points will always be valuated worse than feasible ones, and that is not always the case [103].

Other researchers have used different distance-based static penalty functions [5,17,68,79,124,144,172], but in all cases these metrics rely on some extra parameter (namely one or more penalty factors) which are difficult to generalize and normally remain problem-dependent.

2.2. Dynamic penalties

Within this category, we will consider any penalty function in which the current generation number is involved in the computation of the corresponding penalty factors (normally the penalty function is defined in such a way that it increases over time – i.e., generations). Notice that although the two approaches described in the following sections (Sections 2.3 and 2.4) are also dynamic penalty approaches, they were considered separately for the sake of clarity.

Joines and Houck [83] proposed a technique in which individuals are evaluated (at generation t) using (we assume minimization)

$$\text{fitness}(\vec{x}) = f(\vec{x}) + (C \times t)^{\alpha} \times \text{SVC}(\beta, \vec{x}), \tag{12}$$

where C, α and β are constants defined by the user (the authors used $C = 0.5$, $\alpha = 1$ or 2 and $\beta = 1$ or 2) and $\mathrm{SVC}(\beta, \vec{x})$ is defined as [83]

$$\mathrm{SVC}(\beta, \vec{x}) = \sum_{i=1}^{n} D_i^{\beta}(\vec{x}) + \sum_{j=1}^{p} D_j(\vec{x}) \tag{13}$$

and

$$D_i(\vec{x}) = \begin{cases} 0, & g_i(\vec{x}) \leqslant 0, \\ |g_i(\vec{x})|, & \text{otherwise}, \end{cases} \quad 1 \leqslant i \leqslant n. \tag{14}$$

$$D_j(\vec{x}) = \begin{cases} 0, & -\epsilon \leqslant h_j(\vec{x}) \leqslant \epsilon, \\ |h_j(\vec{x})|, & \text{otherwise}, \end{cases} \quad 1 \leqslant j \leqslant p. \tag{15}$$

This dynamic function increases the penalty as we progress through generations.

Kazarlis and Petridis [85] performed a detailed study of the behavior of a dynamic penalty function of the form

$$\mathrm{fitness}(\vec{x}) = f(\vec{x}) + V(g) \times \left(A \sum_{i=1}^{m} (\delta_i \cdot w_i \cdot \Phi(d_i(S))) + B \right) \times \delta_s, \tag{16}$$

where A is a "severity" factor, m is the total number of constraints, δ_i is 1 if the constraint i is violated and 0 otherwise, w_i is a weight factor for constraint i, $d_i(S)$ is a measure of the degree of violation of constraint i introduced by solution S, $\Phi_i(\cdot)$ is a function of this measure, B is a penalty threshold factor, δ_s is a binary factor ($d_s = 1$ if S is infeasible and 0 otherwise), and $V(g)$ is an increasing function of g (the current generation) in the range $(0, \ldots, 1)$.

Using as test functions the cutting stock problem and the unit commitment problem, Kazarlis and Petridis experimented with different forms of $V(g)$ (linear, quadratic, cubic, quartic, exponential and 5-step), and found that the best overall performance was provided by a function of the form

$$V(g) = \left(\frac{g}{G} \right)^2, \tag{17}$$

where G is the total number of generations.

2.2.1. Advantages and disadvantages

Some researchers have argued that dynamic penalties work better than static penalties. However, it is difficult to derive good dynamic penalty functions in practice as it is difficult to produce good penalty factors for static functions [159]. For example, in the approach proposed by Joines and Houck [83], the quality of the solution found was very sensitive to changes in the values of α and β and there were no clear guidelines regarding the sensitivity of the approach to different values of C. Even when the values indicated above were found by the authors of this method to be a reasonable choice, Michalewicz [102,108] reported that these parameters produced premature convergence most of the time in other examples. Also, it was found that the technique normally either converged to an infeasible solution or to a feasible one that was far away from the global optimum [34,102]. Apparently, this technique provides very good results only when the objective function is quadratic [109].

The dynamic penalty function proposed by Kazarlis and Petridis [85] (called by them Varying Fitness Function Technique or VFF for short) requires several parameters that depend on the problem and whose definition is not at all clear (for example, $A = 1000$ and $B = 0$ in the experiments reported in [85], but no further explanation is provided about why these values were chosen). Also, their tests (although exhaustive for the two problems considered in their work) need to be extended to other functions before being able to make more general claims about this technique.

In general, the problems associated with static penalty functions are also present with dynamic penalties: if a bad penalty factor is chosen, the EA may converge to either non-optimal feasible solutions (if the penalty is too high) or to infeasible solutions (if the penalty is too low) [161].

2.3. Annealing penalties

Michalewicz and Attia [105] considered a method based on the idea of simulated annealing [89]: the penalty coefficients are changed once in many generations (after the algorithm has been trapped in a local optima). Only active constraints are considered at each iteration, and the penalty is increased over time (i.e., the temperature decreases over time) so that infeasible individuals are heavily penalized in the last generations.

The method of Michalewicz and Attia [105] requires that constraints are divided into four groups: linear equalities, linear inequalities, nonlinear equalities and nonlinear inequalities. Also, a set of active constraints $\mathscr{A}$ has to be created, and all nonlinear equalities together with all violated nonlinear inequalities have to be included there. The population is evolved using [102]

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \frac{1}{2\tau}\sum_{i\in\mathscr{A}}\phi_i^2(\vec{x}), \tag{18}$$

where τ is the cooling schedule [89],

$$\phi_i(\vec{x}) = \begin{cases} \max[0, g_i(\vec{x})] & \text{if } 1 \leqslant i \leqslant n, \\ |h_i(\vec{x})| & \text{if } n+1 \leqslant i \leqslant m \end{cases} \tag{19}$$

and m is the total number of constraints.

An interesting aspect of this approach is that the initial population is not really diverse, but consists of multiple copies of a single individual that satisfies all the linear constraints (a single instance of this feasible individual is really enough [109]). At each iteration, the temperature τ is decreased and the new population is created using the best solution found in the previous iteration as the starting point for the next iteration. The process stops when a pre-defined final 'freezing' temperature τ_f is reached.

A similar proposal was made by Carlson Skalak et al. [160]. In this case, the fitness function of an individual is computed using

$$\text{fitness}(\vec{x}) = \mathscr{A}\cdot f(\vec{x}), \tag{20}$$

where $\mathscr{A}$ depends on two parameters: M, which measures the amount by which a constraint is violated (it takes a zero value when no constraint is violated) and T, which is a function of the running time of the algorithm. T tends to zero as evolution progresses. Using the basic principle of simulated annealing, Carlson et al. [160] defined $\mathscr{A}$ as

$$\mathscr{A} = \mathrm{e}^{-M/T} \tag{21}$$

so that the initial penalty factor is small and it increases over time. This will discard infeasible solutions in the last generations.

To define T (the cooling schedule), Carlson et al. [160] use

$$T = \frac{1}{\sqrt{t}}, \tag{22}$$

where t refers to the temperature used in the previous iteration.

Finally, it should be mentioned that Joines and Houck [83] also experimented with a penalty function based on simulated annealing

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \mathrm{e}^{(C\times t)^{\alpha}\times \text{SVC}(\beta,\vec{x})}, \tag{23}$$

where t is the generation number, $\text{SVC}(\beta,\vec{x})$ is defined by Eq. (13), $C = 0.05$, and $\alpha = \beta = 1$.

This fitness function was proposed as another form of handling constraints in an EA, but their success was relative, mainly because they used unnormalized constraints.

2.3.1. Advantages and disadvantages

One of the main drawbacks of Michalewicz and Attia's approach is its extreme sensitivity to the values of its parameters (particularly the cooling schedule τ), and it is also well known that it is normally difficult to choose an appropriate cooling schedule when solving a problem with simulated annealing [89]. Michalewicz and Attia [105] used $\tau_0 = 1$ and $\tau_f = 0.000001$ in their experiments, with increments $\tau_{i+1} = 0.1\tau_i$. Carlson et al. [160] decided to use the mean constraint violation ($\bar{M}$) as the starting temperature value. For the final temperature, they decided to use one hundredth of the mean constraint violation at the last generation. However, these values are empirically derived and although proved to be useful in some engineering problems by Carlson et al. [160], their definition remains as the most critical issue when using this approach.

The approach used to handle linear constraints in Michalewicz and Attia's technique (treated separately by them) is very efficient, but it requires that the user provides an initial feasible point to the algorithm. The implementation of this technique might require the use of another program to generate a feasible starting point that satisfies all linear constraints (equalities and inequalities) and also requires special operators that produce always feasible offspring from feasible parents.

Regarding the approach of Joines and Houck [83], their main problems to make this approach work were due to the overflows produced by the fact that they did not normalize their constraints. Therefore, the exponential function would sometimes fall out of the valid numerical range of the computer. Furthermore, the definition of the constant C was not justified, and the authors admitted that further experimentation regarding its effect was necessary. On the other hand, the implementation of this technique is easier because it does not distinguish between linear and nonlinear constraints and its authors leave to the EA itself the task of generating feasible solutions from an initial set of random values.

2.4. Adaptive penalties

Bean and Hadj-Alouane [10,69] developed a method that uses a penalty function which takes a feedback from the search process. Each individual is evaluated by the formula

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \lambda(t)\left[\sum_{i=1}^{n} g_i^2(\vec{x}) + \sum_{j=1}^{p} |h_j(\vec{x})|\right], \tag{24}$$

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

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\cdot\lambda(t) & \text{if case \#1,} \\ \beta_2\cdot\lambda(t) & \text{if case \#2,} \\ \lambda(t) & \text{otherwise,} \end{cases} \tag{25}$$

where cases #1 and #2 denote situations where the best individual in the last k generations was always (case #1) or was never (case #2) feasible, $\beta_1, \beta_2 > 1$, $\beta_1 > \beta_2$, and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the penalty component $\lambda(t+1)$ for the generation $t+1$ is decreased if all the best individuals in the last k generations were feasible or is increased if they were all infeasible. If there are some feasible and infeasible individuals tied as best in the population, then the penalty does not change.

Smith and Tate [162] proposed an approach later refined by Coit and Smith [28] and Coit et al. [29] in which the magnitude of the penalty is dynamically modified according to the fitness of the best solution found so far. An individual is evaluated using the formula (only inequality constraints were considered in this work)

$$\text{fitness}(\vec{x}) = f(\vec{x}) + (B_{\text{feasible}} - B_{\text{all}})\sum_{i=1}^{n}\left(\frac{g_i(\vec{x})}{\text{NFT}(t)}\right)^k, \tag{26}$$

where B_{feasible} is the best-known objective function at generation t, B_{all} is the best (unpenalized) overall objective function at generation t, $g_i(\vec{x})$ is the amount by which the constraint i is violated, k is a constant that adjusts the severity of the penalty (a value of $k = 2$ has been previously suggested by Coit and Smith

[28]), and NFT is the so-called Near Feasibility Threshold, which is defined as the threshold distance from the feasible region at which the user would consider that the search is "reasonably" close to the feasible region [63,109].

Norman and Smith [123] further applied Coit and Smith's approach to facility layout problems, and apparently the technique has been used only in combinatorial optimization problems.

Gen and Cheng [63] indicate that Yokota et al. [177] proposed a variant of Smith, Tate and Coit's approach in which they use a multiplicative form of the fitness function (instead of an addition as in [162])

$$\text{fitness}(\vec{x}) = f(\vec{x}) \times P(\vec{x}), \tag{27}$$

where $P(\vec{x})$ is defined as

$$P(\vec{x}) = 1 - \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\Delta b_i(\vec{x})}{b_i}\right)^k \tag{28}$$

and

$$\Delta b_i(\vec{x}) = \max[0, g_i(\vec{x}) - b_i]. \tag{29}$$

In this case, $\Delta b_i(\vec{x})$ refers to the violation of constraint i. Notice that this approach is really a special case of Smith et al.'s approach in which NFT $= b_i$, assuming that $g_i(\vec{x}) \leqslant b_i$ is required to consider a solution as feasible.

Gen and Cheng [62] later refined their approach introducing a more severe penalty for infeasible solutions. In the new version of their algorithm,

$$P(\vec{x}) = 1 - \frac{1}{n}\sum_{i=1}^{n}\left(\frac{\Delta b_i(\vec{x})}{\Delta b_i^{\max}}\right)^k, \tag{30}$$

$$\Delta b_i(\vec{x}) = \max[0, g_i(\vec{x}) - b_i], \tag{31}$$

$$\Delta b_i^{\max} = \max[\epsilon, \Delta b_i(\vec{x}); \vec{x} \in P(t)], \tag{32}$$

where $\Delta b_i(\vec{x})$ is the value by which the constraint i is violated in the nth chromosome. $\Delta b_i^{\max}$ is the maximum violation of constraint i in the whole (current) population, and ϵ is a small positive number used to avoid dividing by zero [63]. The motivation of this technique was to preserve diversity in the population, avoiding at the same time overpenalizing infeasible solutions which will constitute most of the population at early generations in highly constrained optimization problems [63].

Eiben and van der Hauw [52], Eiben et al. [53] and Eiben and Ruttkay [51] proposed an adaptive penalty function that was successfully applied to the graph 3-coloring problem. They used a fitness function of the form

$$\text{fitness}(\vec{x}) = \sum_{i=1}^{n} w_i \cdot \chi(\vec{x}, i), \tag{33}$$

where w_i is a weight (or penalty) assigned to node i of a graph, and

$$\chi(\vec{x}, i) = \begin{cases} 1 & \text{if node } x_i \text{ is left uncolored because of a constraint violation,} \\ 0 & \text{otherwise.} \end{cases} \tag{34}$$

In this approach, originally introduced by Eiben et al. [50], the weights used in the fitness function are changed during the evolutionary process such that the search focuses on satisfying those constraints that are considered "harder" by giving higher rewards to the fitness function in those cases. This technique proved to be superior to a powerful (traditional) graph coloring technique called DSatur [15] and to a Grouping Genetic Algorithm [54].

Rasheed [139] proposed an approach in which the penalty factor would be small at the beginning of the evolutionary process, and it would be increased whenever the search gave too little attention to feasibility

(i.e., when the point with highest fitness in the population was infeasible). Conversely, the penalty factor would be decreased if the search gave too much attention to feasibility (i.e., if all individuals in the population were feasible). The rationale behind the approach was to insure proper search of the regions adjacent to constraint boundaries, since in many cases the optimum lies precisely there. This approach was successfully applied to several engineering optimization problems (e.g., supersonic transport aircraft design).

Crossley and Williams [32] experimented with several adaptive penalty coefficients based on the current generation number (this would really be a dynamic penalty function) and the standard deviation and variance of the population's fitness values. They tested their six different penalty coefficients (including a constant value) on four engineering problems. Their results showed superiority of the adaptive approaches over the use of a constant penalty coefficient. A coefficient whose variation was linear with respect to the current generation number was found to provide the best results overall. However, they concluded that the best adaptive penalty is really problem-dependent if we are concerned of finding the best result in the minimum number of generations.

2.4.1. Advantages and disadvantages

The obvious drawback of the approach of Bean and Hadj-Alouane [10,69] is how to choose the generational gap (i.e., the appropriate value of k) that provides reasonable information to guide the search. More important yet is how do we define the values of β_1 and β_2 to penalize fairly a given solution.

The most obvious drawback of the approach of Smith and Tate [162] is how to choose NFT, since this parameter will be problem-dependent. Coit and Smith [28] have proposed to define NFT as

$$\mathrm{NFT} = \frac{\mathrm{NFT}_0}{1 + \lambda t}, \tag{35}$$

where NFT_0 is an upper bound for NFT, t is the generation number, and λ is a constant that assures that the entire region between NFT_0 and zero (feasible region) is searched. Care should be taken that NFT does not approach zero either too quickly or too slowly [28]. Although Coit and Smith [28] have provided some alternatives for defining NFT, its value remains as an additional parameter to be determined by the user.

Additionally, the factor $B_{\mathrm{feasible}} - B_{\mathrm{all}}$ has some potential dangers: First, if B_{feasible} is much greater than B_{all}, then the penalty would be quite large for all individuals in the population. Coit and Smith [28] claim that this does not seem to happen too often in practice because they use selection strategies that preclude the possibility of selecting solution vectors sufficiently far from the feasible region for this to happen, but in any case, they propose changing the values of B_{feasible} and B_{all} for the initial generations.

The second potential danger is that if B_{feasible} and B_{all} are identical, then the penalty would be zero, which means that all infeasible individuals would go unpenalized in that generation. The underlying assumption here is that the best unpenalized individual in fact lies on the feasible region, but that might not be the case, and it could introduce a strong bias towards infeasible solutions.

The approach proposed by Gen and Cheng [63] assigns a relatively mild penalty with respect to Coit et al. [29], but the authors of this method argue that their approach is problem-independent [63]. However, no information is provided by Gen and Cheng [63] regarding the sort of problems used to test this technique, and apparently the approach was used only in one combinatorial optimization problem, which does not constitute enough evidence of this statement.

Similarly, the approach of Eiben and van der Hauw [52] also requires the definition of additional parameters (the weights w_i assigned to each node of the graph), and it has been applied only to combinatorial optimization problems.

The approach of Rasheed [139] was inspired by Smith and Tate [162], and it seems to be the first attempt to use adaptive penalties in numerical optimization. This approach is interesting, but it requires the definition of an initial value for the penalty factor. Rasheed provides a way of computing such a default value. However, his formula is based on the assumption that the numerical magnitude of the constraints is comparable to what he calls the "measure of merit" (i.e., the objective function). If this is not true, then a scaling function will be required. Also, certain limits have to be defined for the increments and decrements of the penalty factor, in order to avoid abrupt changes.

Crossley and Williams' study was inconclusive. For example, adaptive penalties based on the standard deviation and variance of the population's fitness values were found to be too expensive (computationally speaking). A penalty factor that increased quadratically with respect to the number of generations was also found to provide poor results. However, from the remaining approaches, none of them was found to provide the best possible results with the lowest number of fitness function evaluations for all test problems. Obviously, more studies of this sort are required.

2.5. Co-evolutionary penalties

Coello [25] proposed the use of a penalty function of the form

$$\text{fitness}(\vec{x}) = f(\vec{x}) - (\text{coef} \times w_1 + \text{viol} \times w_2), \tag{36}$$

where $f(\vec{x})$ is the value of the objective function for the given set of variable values encoded in a chromosome; w_1 and w_2 are two penalty factors (considered as integers); coef is the sum of all the amounts by which the constraints are violated (only inequality constraints were considered):

$$\text{coef} = \sum_{i=1}^{n} g_i(\vec{x}) \quad \forall g_i(\vec{x}) > 0, \tag{37}$$

where viol is an integer factor, initialized to zero and incremented by one for each constraint of the problem that is violated, regardless of the amount of violation (i.e., only the number of constraints violated is counted with this variable, but not the magnitude in which each constraint is violated).

In Coello's approach, the penalty is actually split into two values (coef and viol) so that the EA has enough information not only about how many constraints were violated, but also about the corresponding amounts of violation. This follows the suggestion of Richardson [144] about using penalties that are guided by the distance to feasibility.

Coello [25] used two different populations $P1$ and $P2$ with corresponding sizes $M1$ and $M2$. The second of these populations ($P2$) encoded the set of weight combinations (w_1 and w_2) that would be used to compute the fitness value of the individuals in $P1$ (i.e., $P2$ contained the penalty factors that would be used in the fitness function). The idea of Coello's approach is to use one population to evolve solutions (as in a conventional EA), and another to evolve the penalty factors w_1 and w_2. For each individual A_j in $P2$ there is an instance of $P1$. However, the population $P1$ is reused for each new element A_j processed from $P2$.

Each individual A_j ($1 \leqslant j \leqslant M2$) in $P2$ is decoded and the weight combination produced (i.e., the penalty factors) is used to evolve $P1$ during a certain number ($Gmax1$) of generations. The fitness of each individual B_k ($1 \leqslant k \leqslant M1$) is computed using Eq. (36), keeping the penalty factors constant for every individual in the instance of $P1$ corresponding to the individual A_j being processed.

After evolving each $P1$ corresponding to every A_j in $P2$ (there is only one instance of $P1$ for each individual in $P2$), the best average fitness produced is computed using

$$\text{average\_fitness}_j = \sum_{i=1}^{M1} \left(\frac{\text{fitness}(\vec{x})}{\text{count\_feasible}} \right) + \text{count\_feasible} \quad \forall \vec{x} \in \mathscr{F}. \tag{38}$$

In Eq. (38), the fitnesses of all feasible solutions in $P1$ are added, and an average of them is computed (the integer variable count_feasible is a counter that indicates how many feasible solutions were found in the population). The reason for considering only feasible individuals is that if infeasible solutions are not excluded from this computation, the selection mechanism of the EA may bias the population towards regions of the search space where there are solutions with a very low weight combination. Such solutions may have good fitness values, and still be infeasible. The reason for that is that low values of w_1 and w_2 may produce penalties that are not big enough to outweight the value of the objective function.

Notice also the use of count_feasible to avoid stagnation (i.e., loss of diversity in the population) at certain regions in which only very few individuals will have a good fitness or will be even feasible. By adding this quantity to the average fitness of the feasible individuals in the population, the EA is encouraged to move towards regions in which lie not only feasible solutions with good fitness values, but there are also lots

of them. In practice, it may be necessary to apply a scaling factor to the average of the fitness before adding (count_feasible), to avoid that the EA gets trapped in local optima. However, such scaling factor is not very difficult to compute because Coello [25] assumes populations of constant size (such size must be defined before running the EA). The range of the fitness values can be also easily obtained at each generation, because the maximum and minimum fitness values in the population are known at each generation.

The process indicated above is repeated until all individuals in $P2$ have a fitness value (the best average_fitness of their corresponding $P1$). Then, $P2$ is evolved one generation using conventional genetic operators (i.e., crossover and mutation) and the new $P2$ produced is used to start the same process all over again. It is important to notice that the interaction between $P1$ and $P2$ introduces diversity in both populations, which keeps the EA from easily converging to a local optimum.

2.5.1. Advantages and disadvantages

The problem with this approach is that it introduces the definition of four additional parameters: $Gmax1$, $Gmax2$, $M1$ and $M2$. Coello [22,25] argues that those parameters have to be (empirically) determined for an EA in any particular application, and showed that the approach was really more sensitive to changes in the parameters of $P1$ than to changes in the parameters of $P2$. However, the definition of these parameters remains as an additional issue to be settled. Furthermore, if these parameters are not carefully chosen, a lot of fitness function evaluations might be required due to the nested loops involved in the optimization process. A parallel algorithm may be a viable solution to this problem, but such an alternative has not been implemented yet.

2.6. Segregated genetic algorithm

Le Riche et al. [147] designed a (segregated) genetic algorithm which uses two penalty parameters (for each constraint) instead of one; these two values aim at achieving a balance between heavy and moderate penalties by maintaining two subpopulations of individuals instead of one. Even when individuals of the two populations interbreed (i.e., they are merged), they are "segregated" in terms of satisfaction of a certain constraint.

The procedure is the following [147]: a population of size $2 \times m$ is generated. Each individual is evaluated according to two penalty functions (one with heavy and one with moderate penalties). Two ranked lists are generated and then merged. Only m individuals are chosen from the new list to apply the genetic operators (crossover and mutation): the best individuals from the two original ranked lists are chosen to become parents for the next generation. This aims to combine feasible and infeasible individuals, and to help the genetic algorithm to stay out of local minima.

Another important difference of this approach with respect to a traditional genetic algorithm is that if the two penalties have the same value, the m children produced after applying the genetic operators are mixed with their m parents. Then the best m individuals from this merged list are chosen for further processing. This replacement strategy (called "super elitism" by Le Riche et al. [147]) was taken from evolution strategies [156] and allows to balance the influence of the two penalty factors used.

Linear ranking was used to decrease the high selection pressure that could cause premature convergence. This approach was used to solve a laminated design problem, providing excellent results [147].

2.6.1. Advantages and disadvantages

The problem with this approach is again the way of choosing the penalties for each of the two subpopulations. Even when some guidelines have been provided by the authors of this method to define such penalties [147], they also admit that it is difficult to produce generic values that can be used in any problem for which no previous information is available.

2.7. Death penalty

The rejection of infeasible individuals (also called "death penalty") is probably the easiest way to handle constraints and it is also computationally efficient, because when a certain solution violates a constraint, it is assigned a fitness of zero. Therefore, no further calculations are necessary to estimate the degree of

infeasibility of such a solution. The normal approach taken is to iterate recursively, generating a new point at each recursive call, until a feasible solution is found [76]. This might be a rather lengthy process in problems in which is very difficult to approach the feasible region.

2.7.1. Advantages and disadvantages

Death penalty is very popular within the evolution strategies community [4,156], but it is limited to problems in which the feasible search space is convex and constitutes a reasonably large portion of the whole search space. This approach has the drawback of not exploiting any information from the infeasible points that might be generated by the EA to guide the search.

One potential problem of this approach is that if there are no feasible solutions in the initial population (which is normally generated at random) then the evolutionary process will "stagnate" because all the individuals will have the same fitness (i.e., zero).

There are well-documented experiments in which the use of death penalty with EAs is not a good choice. For example, Coit and Smith [28] compared this approach against an adaptive penalty in a reliability design optimization problem (a problem with highly constrained search spaces), finding that the adaptive penalty was superior in terms of both the quality of the final solutions found and the convergence of the EA to the best solution found. Michalewicz [102,108,109] has also shown that the use of death penalty is inferior to the use of penalties that are defined in terms of the distance to the feasible region.

3. Special representations and operators

Some researchers have decided to develop special representation schemes to tackle a certain (particularly difficult) problem for which a generic representation scheme (e.g., the binary representation used in the traditional genetic algorithm) might not be appropriate. Due to the change of representation, it is necessary to design special genetic operators that work in a similar way than the traditional operators used with a binary representation.

A change of representation is aimed at simplifying the shape of the search space and the special operators are normally used to preserve the feasibility of solutions at all times. The main application of this approach is naturally in problems in which it is extremely difficult to locate at least a single feasible solution.

3.1. Davis' applications

Lawrence Davis' *Handbook of Genetic Algorithms* [40] contains several examples of EAs that use special representations and operators to solve complex real-world problems. For example, Davidor [37] (see also [36]) used a varying-length genetic algorithm to generate robot trajectories, and defined a special crossover operator called *analogous crossover* [35], which uses phenotypic similarities to define crossover points in the parent strings. Davidor also used Lamarckian probabilities for crossover and mutation. This means that the crossover and mutation points were chosen according to the error distribution along the string, which was relatively easy to estimate in this particular application.

Other applications included in Davis' book are: schedule optimization [170], synthesis of neural networks architecture [73], and conformational analysis of DNA [98], among others.

3.1.1. Advantages and disadvantages

The use of special representations and operators is, with no doubt, quite useful for the intended application for which they were designed, but their generalization to other (even similar) problems is by no means obvious.

3.2. Random keys

Bean [8,9] proposed a special representation called "random keys encoding" which (in contrast with the approaches reported in Davis' book) is used to eliminate the need of special crossover and mutation

operators in certain sequencing and optimization problems (e.g., job shop scheduling, parallel machine tool scheduling, and facility layout), because it maintains the feasibility of the permutations used in these domains at all times. It also adds no computational overhead to the search.

The idea is to encode a solution with random numbers. Such random numbers are used as sort keys to decode the solution. For example, to represent an n-job m-machine scheduling problem using this approach, each allele is a real number in which the integer part belongs to the set $\{1, 2, \ldots, m\}$, whereas the decimal fraction is randomly generated within the interval $(0, 1)$. The integer part of the number is then interpreted as the machine assignment for that job, whereas the sorted fractional parts provide the job sequence on each machine [121,122].

3.2.1. Advantages and disadvantages

This approach is with no doubt interesting, although some researchers have reported poor performance of the technique in some applications. For example, Parsons et al. [131,132] found that the random keys genetic algorithm did not perform as well as a standard permutation representation with special-purpose operators (transposition and a form of inversion) in a DNA fragment-assembly problem (a TSP problem with noise, errors, and some other complications).

3.3. GENOCOP

Another example of this approach is GEnetic algorithm for Numerical Optimization for COnstrained Problems (GENOCOP), developed by Michalewicz [101]. GENOCOP eliminates equality constraints together with an equal number of problem variables. This removes part of the space to be searched and simplifies the problem for the EA. The remaining constraints are linear inequalities, which form a convex set that must be searched by the EA. GENOCOP tries to locate an initial (feasible) solution by sampling the feasible region. If it does not succeed after a certain number of trials, the user is asked to provide such a starting point. The initial population will then consist of identical copies of this starting point. The genetic operators adopted perform linear combinations of individuals to ensure that their offspring will also be feasible (these operators rely on properties of convex sets).

3.3.1. Advantages and disadvantages

GENOCOP assumes a feasible starting point (or feasible initial population), which implies that the user or the EA must have a way of generating (in a reasonable time) such starting point. Also, the fact that GENOCOP only allows linear constraints, limits its applications to convex search spaces [34].

3.4. Constraint consistent GAs

Kowalczyk [90] proposed the use of constraint consistency [93] to prune the search space by preventing variable instantiations that are not consistent with the constraints of the problem (i.e., making sure that variables produce only feasible solutions).

Kowalczyk used real-numbers representation and defined special genetic operators and a special initialization procedure that incorporated the concept of constraint consistency. He indicated that his approach can be used in combination with any other constraint-handling technique, and was aware that in many cases partially feasible solutions may be preferred because they can guide the search in a more appropriate way or because they are much easier to find.

3.4.1. Advantages and disadvantages

The main drawback of this approach is the extra computational cost required to propagate constraints, which may become a process more expensive than the optimization itself. In any case, the approach deserves some attention and more experimentation is required, since Kowalczyk illustrated its performance with only two optimization problems.

3.5. Locating the boundary of the feasible region

The main idea of this technique is to search areas close to the boundary of the feasible region. Since in many nonlinear optimization problems at least some constraints are active at the global optimum, it is perfectly justified to focus the search to the boundary between the feasible and infeasible regions.

The idea was originally proposed in an Operations Research technique known as *strategic oscillation* [65] and has been used in combinatorial and nonlinear optimization problems [66]. The basic approach is to use adaptive penalties or other similar mechanism (e.g., gradients) to cross the feasibility boundary back and forth by relaxing or tightening a certain factor that determines the direction of movement [109].

The two basic components of this approach are: (a) an initialization procedure that can generate feasible points, and (b) genetic operators that explore the feasible region.

Additionally, the genetic operators must satisfy the following conditions [101,136]: (1) crossover should be able to generate all points "between" the parents, (2) small mutations must result in small changes in the fitness function.

In the work done by Schoenauer and Michalewicz [152], several examples are presented and special genetic operators are designed for each using geodesical curves and plane-based operators. In a further paper, Schoenauer and Michalewicz [153] analyze in more detail the use of sphere operators in convex feasible search spaces.

3.5.1. Advantages and disadvantages

The main drawback of this approach is that the operators designed are either highly dependent on the chosen parameterization [152], or more complex calculations are required to perform crossover and mutation. Also, many problems have disjoint feasible regions and the use of operators of this sort would not be of much help in those cases since they would explore only one of those feasible regions.

Finally, the use of these operators is limited to a single problem, although some of the concepts involved can be generalized. Whenever applicable, however, the approach is quite efficient and produces very good results.

3.6. Decoders

In this case, a chromosome "gives instructions" on how to build a feasible solution. Each decoder imposes a relationship T between a feasible solution and a decoded solution [34]. When using decoders, however, it is important that several conditions are satisfied [127]: (1) for each feasible solution s there must be a decoded solution d, (2) each decoded solution d must correspond to a feasible solution s and (3) all feasible solutions should be represented by the same number of decodings d. Additionally, it is reasonable to request that (4) the transformation T is computationally fast and (5) it has locality feature in the sense that small changes in the decoded solution result in small changes in the solution itself [34].

Koziel and Michalewicz [91,92] have recently proposed a homomorphous mapping between an n-dimensional cube and a feasible search space (either convex or non-convex). The main idea of this approach is to transform the original problem into another (topologically equivalent) function that is easier to optimize by the EA.

Kim and Husbands [86,87] had an earlier proposal of a similar approach that used Riemann mappings to transform the feasible region into a shape that facilitated the search for the EA.

3.6.1. Advantages and disadvantages

Despite the several advantages of the approach of Koziel and Michalewicz [92], it also has some disadvantages:

- It uses an extra parameter v which has to be found empirically, performing a set of runs.
- Requires extra computational effort because of the binary search required to find the intersection of a line with the boundary of the feasible region (which is the core of the technique).
- It violates the locality feature mentioned before when used in non-convex search spaces: small changes in the encoded solution may result in huge changes in the decoded value (e.g., when dealing with disjoint search spaces).

However, in the experiments reported by Koziel and Michalewicz [92], this technique provided much better results than those reported with any other constraint-handling method, and seems a very promising area of research.

Kim and Husbands' approach [86,87] could only be used with problems of low dimensionality (no more than four variables) and required the objective function to be given in algebraic form. The mapping proposed by Koziel and Michalewicz [91,92], however, can be used with problems of any dimensionality and does not require that the objective function is given in algebraic form.

4. Repair algorithms

In many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) is relatively easy to 'repair' an infeasible individual (i.e., to make feasible an infeasible individual). Such a repaired version can be used either for evaluation only, or it can also replace (with some probability) the original individual in the population.

Liepins and co-workers [96,97] have shown, through an empirical test of EA performance on a diverse set of constrained-combinatorial optimization problems, that a repair algorithm is able to surpass other approaches in both speed and performance.

GENOCOP III [108] also uses repair algorithms. The idea is to incorporate the original GENOCOP system [107] (which handles only linear constraints) and extend it by maintaining two separate populations, where results in one population influence evaluations of individuals in the other population. The first population consists of the so-called search points which satisfy linear constraints of the problem; the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators. The second population consists of feasible reference points. Since these reference points are already feasible, they are evaluated directly by the objective function, whereas search points are "repaired" for evaluation.

Xiao and co-workers [110,175,176] used a repair algorithm to transform an infeasible path of a robot trying to move between two points in the presence of obstacles, so that the path would become feasible (i.e., collision-free). The repair algorithm was implemented through a set of carefully designed genetic operators that used knowledge about the domain to bring infeasible solutions into the feasible region in an efficient way.

Other authors that have used repair algorithms are Orvosh and Davis [126], Mühlenbein [115], Le Riche and Haftka [146], and Tate and Smith [171].

There are no standard heuristics for the design of repair algorithms: normally, it is possible to use a greedy algorithm (i.e., an optimization algorithm that proceeds through a series of alternatives by making the best decision, as computed locally, at each point in the series), a random algorithm or any other heuristic which would guide the repair process. However, the success of this approach relies mainly on the ability of the user to come up with such a heuristic.

Another interesting aspect of this technique is that normally an infeasible solution that is repaired is only used to compute its fitness, but the repaired version is returned to the population only in certain cases (using a certain probability). The question of replacing repaired individuals is related to the so-called *Lamarckian evolution*, which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome [166]. Some researchers like Liepins and co-workers [96,97] have taken the *never replacing* approach (that is, the repaired version is never returned to the population), while other authors such as Nakano [119] have taken the *always replacing* approach.

Orvosh and Davis [125,126] reported a so-called 5% rule for combinatorial optimization problems, which means that EAs (applied to combinatorial optimization problems) with a repairing procedure provide the best result when 5% of the repaired chromosomes replace their infeasible originals. Michalewicz [104] have reported, however, that a 15% replacement rule seems to be the best choice for numerical optimization problems with nonlinear constraints.

4.1. Advantages and disadvantages

When an infeasible solution can be easily (or at least at a low computational cost) transformed into a feasible solution, repair algorithms are a good choice. However this is not always possible and in some cases

repair operators may introduce a strong bias in the search, harming the evolutionary process itself [161]. Furthermore, this approach is problem-dependent, since a specific repair algorithm has to be designed for each particular problem.

5. Separation of constraints and objectives

There are several approaches that handle constraints and objectives separately (i.e., without combining the amount of constraint violation and the objective function value). In this section we will review some of the most representative proposals.

5.1. Co-evolution

Paredis [128] proposed a technique based on a co-evolutionary model in which there are two populations: the first contains the constraints to be satisfied (in fact, this is not a population in the general sense of the term, since its contents does not change over time) and the second contains potential (and possibly invalid) solutions to the problem to be solved. Using an analogy with a predator–prey model, the selection pressure on members of one population depends on the fitness of the members of the other population [128].

An individual with high fitness in the second population represents a solution that satisfies a lot of constraints whereas an individual with high fitness in the first population represents a constraint that is violated by a lot of solutions.

Solutions and constraints have *encounters* in which individuals belonging to both populations are evaluated. Each individual keeps a history of its encounters, and its fitness is computed according to the sum of the last n encounters ([128] used $n = 25$). The idea of the approach is to increase the fitness of those constraints that are harder to satisfy so that the evolutionary search concentrates on them. In fact, the relevance of a certain constraint can be changed over time using this approach.

5.1.1. Advantages and disadvantages

Paredis [128] indicated that his approach was similar to a self-adaptive penalty function in which the relevance of a certain constraint can be changed over time, according to its difficulty. The results reported by Paredis [128] are very impressive, and the approach seems very efficient because not all constraints have to be checked at all times. One problem with this approach is that the use of a historical record to compute fitness of an individual might introduce "stagnation" (i.e., the search may not progress anymore) if all the constraints (or at least most of them) are equally difficult to satisfy. Also, there is no further evidence of the effectiveness of the approach in other combinatorial optimization problems, and apparently, it has not been extended to numerical optimization problems either.

5.2. Superiority of feasible points

Powell and Skolnick [134] incorporated a heuristic rule (suggested by Richardson et al. [144]) for processing infeasible solutions: evaluations of feasible solutions are mapped into the interval $(-\infty, 1)$, and infeasible solutions into the interval $(1, \infty)$. Individuals are evaluated using [134]:

$$\text{fitness}(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if feasible,} \\ 1 + r\left(\sum_{i=1}^{n} g_i(\vec{x}) + \sum_{j=1}^{p} h_j(\vec{x})\right) & \text{otherwise.} \end{cases} \tag{39}$$

$f(\vec{x})$ is scaled into the interval $(-\infty, 1)$, $g_i(\vec{x})$ and $h_j(\vec{x})$ are scaled into the interval $(1, \infty)$, and r is a constant. Notice that in this approach the objective function and the amount of constraint violation are not combined when an individual is infeasible (as when using penalty functions).

Powell and Skolnick [134] used linear ranking selection [6,7,40] in such a way that at early generations there would be slow convergence, and later on convergence could be forced by increasing the number of copies of the highest ranked individuals.

Deb [44] proposed more recently a similar approach in which an individual is evaluated using:

$$\text{fitness}(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } g_i(\vec{x}) \geqslant 0 \quad \forall i = 1, 2, \ldots, n, \\ f_{\text{worst}} + \sum_{i=1}^{n} g_i(\vec{x}) & \text{otherwise,} \end{cases} \tag{40}$$

where f_{worst} is the objective function value of the worst feasible solution in the population, and $g_i(\vec{x})$ refers only to inequality constraints (Deb transformed equality constraints to inequality constraints using Eq. (7)). If there are no feasible solutions in the population, then f_{worst} is set to zero.

Using binary tournament selection, Deb applies the following rules to compare two individuals [44]:

1. A feasible solution is always preferred over an infeasible one.
2. Between two feasible solutions, the one having a better objective function value is preferred.
3. Between two infeasible solutions, the one having smaller constraint violation is preferred.

No penalty factor is required, since the selection procedure only performs pairwise comparisons. Therefore, feasible solutions have a fitness equal to their objective function value, and the use of constraint violation in the comparisons aims to push infeasible solutions towards the feasible region. Due to the fact that constraints are normally non-commensurable (i.e., they are expressed in different units), Deb normalized them to avoid any sort of bias toward any of them.

The main difference between these two approaches (Powell and Skolnick's and Deb's) is that the second does not require a penalty factor r, because of the pairwise comparisons performed during the selection process. However, the apprach of Deb [100] requires niching to maintain diversity in the population. This means that in this approach the search is focused initially on finding feasible solutions and then uses techniques to maintain diversity to approach the optimum.

Another similar approach called COnstraint based Numeric Genetic Algorithm (CONGA) was proposed by Hinterding and Michalewicz [75]. The idea is to perform the search in two phases, as Schoenauer and Xanthakis' behavioral memory algorithm [154]. In the first phase, the search concentrates on finding feasible individuals (assuming that there is none in the initial population) and the objective function value is not used (only the information about constraint violation of each individual). As the amount of feasible individuals increases, the search focuses on fine-tuning the best of them. Hinterding and Michalewicz [75] use two selection functions: one that selects an individual for mutation or the first parent for crossover (only one operator can be applied) using the same criteria as Deb [44] (an individual is randomly chosen when there is a tie). The second selection function finds a mate for a parent selected with the first function. This second selection function chooses the individual with the least number of satisfied constraints in common with the parent already selected. The idea is to select the mate who best "complements" the parent previously selected. This mate should satisfy the constraints than the first selected parent does not satisfy. Therefore, the aim is that crossover will create new individuals who satisfy more constraints than any of their parents. The idea of complementary matching was borrowed from Ronald [148], only that in his case, the selection of the second parent did not depend on the first one but on a different global criterion.

5.2.1. Advantages and disadvantages

Although some might think that the definition of r in Powell and Skolnick's approach introduces the traditional problems of using a penalty function, this is not true, since the linear ranking selection scheme used makes irrelevant the value of this constant. The approach has, however, other problems.

The key concept of this approach is the assumption of the superiority of feasible solutions over infeasible ones, and as long as such assumption holds, the technique is expected to behave well [134]. However, in cases where the ratio between the feasible region and the whole search space is too small (for example, when there are constraints very difficult to satisfy), this technique will fail unless a feasible point is introduced in the initial population [104].

The results of Deb [44] are very encouraging, but his technique seems to have problems to maintain diversity in the population, and the use of niching methods [45] combined with higher than usual mutation rates is apparently necessary to avoid stagnation. Sharing is an expensive process ($O(n^2)$), and its use

introduces an extra parameter (σ_{share}), whose definition is normally determined using an empirical procedure similar to the one used with the other parameters of an EA (e.g., crossover and mutation rates, population size, etc.).

The approach of Hinterding and Michalewicz relies on the same assumption as Powell and Skolnick's technique: feasible individuals are always better than infeasible ones. Therefore, it shares its same problems. The other problem with this approach is how to keep diversity in the population, since the tournament selection strategy adopted might introduce a high selection pressure (e.g., if there is only one feasible individual in the population, it will drive the others to a possible local optimum). The authors used a very high replacement rate (the 97% worst individuals from each generation are replaced by new individuals, and duplicates are not allowed in the population). This tries to keep a large number of infeasible individuals in the population when at least one feasible individual has been found, as to decrease the selection pressure. However, the approach still needs further refinement and validation (it was tested only with five benchmark functions and compared against GENOCOP II and III).

5.3. Behavioral memory

Schoenauer and Xanthakis [154] proposed to extend a technique called *behavioral memory*, which was originally proposed for unconstrained optimization [41]. The main idea of this approach is that constraints are handled in a particular order. The algorithm is the following [154]:

- Start with a random population of individuals.
- Set $j = 1$ (j is the constraint counter).
- Evolve this population to minimize the violation of the jth constraint, until a given percentage of the population (this is called the flip threshold Φ) is feasible for this constraint. In this case

$$\text{fitness}(\vec{x}) = M - g_1(\vec{x}), \tag{41}$$

  where M is a sufficiently large positive number which is dynamically adjusted at each generation.
- $j = j + 1$.
- The current population is the starting point for the next phase of the evolution, minimizing the violation of the jth constraint,

$$\text{fitness}(\vec{x}) = M - g_j(\vec{x}). \tag{42}$$

  During this phase, points that do not satisfy at least one of the $1\text{st}, 2\text{nd}, \ldots, (j-1)\text{th}$ constraints are eliminated from the population. The condition required to stop this stage of the algorithm is again the satisfaction of the jth constraint by the flip threshold percentage Φ of the population.
- If $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the objective function f rejecting infeasible individuals.

The idea of this technique is to satisfy sequentially (one-by-one) the constraints imposed on the problem. This is similar to an approach called "lexicographic ordering" that is used in multiobjective optimization [21]. Once a certain percentage of the population (defined by the flip threshold) satisfies the first constraint, an attempt to satisfy the second constraint (while still satisfying the first) will be made. Notice that in the last step of the algorithm, Schoenauer and Xanthakis [154] use death penalty, because infeasible individuals are eliminated from the population.

5.3.1. Advantages and disadvantages

This method requires that there is a linear order of all constraints, and the order in which the constraints are processed influences the results provided by the algorithm (in terms of total running time and precision achieved) [104].

Schoenauer and Xanthakis also recommended the use of a sharing scheme (to keep diversity in the population), which adds to the flip threshold Φ and the order of the constraints as extra parameters required by the algorithm.

Furthermore, since this approach violates the *minimum penalty rule* [145,147], it has a high computational cost (increased by the use of sharing to keep diversity in the population). As Schoenauer and

Xanthakis [154] admit, the extra computational cost of this approach is not justified when the feasible region is quite large. However, it is particularly suitable for applications in which constraints have a natural hierarchy of evaluation, like the problem of generating software test data used by the authors of this technique [154].

5.4. Multiobjective optimization techniques

The main idea is to redefine the single-objective optimization of $f(\vec{x})$ as a multiobjective optimization problem in which we will have $m+1$ objectives, where m is the total number of constraints. Then, we can apply any multiobjective optimization technique [60] to the new vector $\bar{v} = (f(\vec{x}), f_1(\vec{x}), \ldots, f_m(\vec{x}))$, where $f_1(\vec{x}), \ldots, f_m(\vec{x})$ are the original constraints of the problem. An ideal solution $\vec{x}$ would thus have $f_i(\vec{x}) = 0$ for $1 \leqslant i \leqslant m$ and $f(\vec{x}) \leqslant f(\vec{y})$ for all feasible $\vec{y}$ (assuming minimization).

Surry and co-workers [167,168] proposed the use of Pareto ranking [59] and VEGA [151] to handle constraints using this technique. In their approach, called COMOGA, the population was ranked based on constraint violations (counting the number of individuals dominated by each solution). Then, one portion of the population was selected based on constraint ranking, and the rest based on real cost (fitness) of the individuals.

Parmee and Purchase [129] implemented a version of VEGA [151] that handled the constraints of a gas turbine problem as objectives to allow an EA to locate a feasible region within the highly constrained search space of this application. However, VEGA was not used to further explore the feasible region, and instead Parmee and Purchase [129] opted to use specialized operators that would create a variable-size hypercube around each feasible point to help the EA to remain within the feasible region at all times.

Camponogara and Talukdar [16] proposed the use of a procedure based on an evolutionary multiobjective optimization technique. Their proposal was to restate a single objective optimization problem in such a way that two objectives would be considered: the first would be to optimize the original objective function and the second would be to minimize

$$\Phi(\vec{x}) = \sum_{i=1}^{n} \max[0, g_i(\vec{x})]. \tag{43}$$

Once the problem is redefined, non-dominated solutions with respect to the two new objectives are generated. The solutions found define a search direction $d = (x_i - x_j)/|x_i - x_j|$, where $x_i \in S_i$, $x_j \in S_j$ and S_i and S_j are Pareto sets. The direction search d is intended to simultaneously minimize all the objectives [16]. Line search is performed in this direction so that a solution x can be found such that x dominates x_i and x_j (i.e., x is a better compromise than the two previous solutions found). Line search takes the place of crossover in this approach, and mutation is essentially the same, where the direction d is projected onto the axis of one variable j in the solution space [16]. Additionally, a process of eliminating half of the population is applied at regular intervals (only the less fitted solutions are replaced by randomly generated points).

Jiménez and Verdegay [81] proposed the use of a min–max approach [19] to handle constraints. The main idea of this approach is to apply a set of simple rules to decide the selection process:

1. If the two individuals being compared are both feasible, then select based on the minimum value of the objective function.
2. If one of the two individuals being compared is feasible and the other one is infeasible, then select the feasible individual.
3. If both individuals are infeasible, then select based on the maximum constraint violation ($\max g_j(\vec{x})$ for $j = 1, \ldots, m$, and m is the total number of constraints). The individual with the lowest maximum violation wins.

Notice the great similarity between this approach and the technique proposed by Deb [44] that was described in Section 5.2. The main difference is that in this case, no extra mechanism is used to preserve diversity in the population.

Coello [24] proposed the use of a population-based multiobjective optimization technique such as VEGA [151] to handle each of the constraints of a single-objective optimization problem as an objective. At each

generation, the population is split into $m+1$ sub-populations (m is the number of constraints), so that a fraction of the population is selected using the (unconstrained) objective function as its fitness and another fraction uses the first constraint as its fitness and so on.

For the sub-population guided by the objective function, the evaluation of such objective function for a given vector $\vec{x}$ is used directly as the fitness function (multiplied by (-1) if it is a minimization problem), with no penalties of any sort. For all the other sub-populations, the algorithm used is the following [24]:

if $g_j(\vec{x}) < 0.0$ **then** fitness $= g_j(\vec{x})$
else if $v \neq 0$ **then** fitness $= -v$
else fitness $= f(\vec{x})$

where $g_j(\vec{x})$ refers to the constraint corresponding to sub-population $j+1$ (this is assuming that the first sub-population is assigned to the objective function $f(\vec{x})$), and v refers to the number of constraints that are violated ($\leqslant m$).

There are a few interesting things that can be observed from this procedure. First, each sub-population associated with a constraint will try to reduce the amount in which that constraint is violated. If the solution evaluated does not violate the constraint corresponding to that sub-population, but it is infeasible, then the sub-population will try to minimize the total number of violations, joining then the other sub-populations in the effort of driving the EA to the feasible region. This aims at combining the distance from feasibility with information about the number of violated constraints, which is the same heuristic normally used with penalty functions.

Finally, if the solution encoded is feasible, then this individual will be 'merged' with the first sub-population, since it will be evaluated with the same fitness function (i.e., the objective function).

It is interesting to notice that the use of the unconstrained objective function in one of the sub-populations may assign good fitness values to infeasible individuals. However, since the number of constraints will normally be greater than one, the other sub-populations will drive the EA to the feasible region. In fact, the sub-population evaluated with the objective function will be useful to keep diversity in the population, making then unnecessary the use of sharing techniques. The behavior expected under this scheme is to have few feasible individuals at the beginning, and then gradually produce solutions that may be feasible with respect to some constraints but not with respect to others. Over time, these solutions will combine to produce individuals that are feasible, but not necessarily optimum. At that point the direct use of the objective function will help the EA to approach the optimum, but since some infeasible solutions will still be present in the population, those individuals will be responsible to keep the diversity required to avoid stagnation.

More recently, Coello [23] proposed another approach based on non-dominance. In this case, fitness is assigned to an individual using the following algorithm:

Let the vector $\vec{x}_i$ ($i = 1, \ldots, pop\_size$) be an individual in the current population whose size is pop_size. The proposed algorithm is the following:

- To compute the rank of an individual $\vec{x}_i$ is feasible, following procedure is used:

$$\text{rank}(\vec{x}_i) = \text{count}(\vec{x}_i) + 1, \tag{44}$$

where $\text{count}(\vec{x}_i)$ is computed according to the following rules:

1. Compare $\vec{x}_i$ against every other individual in the population. Assuming pairwise comparisons, we will call $\vec{x}_j$ ($j = 1, \ldots, pop\_size$ and $j \neq i$) the other individual against which x_i is being compared at any given time.
2. Initialize $\text{count}(\vec{x}_i)$ (for $i = 1, \ldots, pop\_size$) to zero.
3. If both $\vec{x}_i$ and $\vec{x}_j$ are feasible, then both are given a rank of zero and $\text{count}(\vec{x}_i)$ remains without changes.
4. If $\vec{x}_i$ is infeasible and $\vec{x}_j$ is feasible, then $\text{count}(\vec{x}_i)$ is incremented by one.
5. If both $\vec{x}_i$ and $\vec{x}_j$ are infeasible, but $\vec{x}_i$ violates more constraints than $\vec{x}_j$, then $\text{count}(\vec{x}_i)$ is incremented by one.
6. If both $\vec{x}_i$ and $\vec{x}_j$ are infeasible, and both violate the same number of constraints, but $\vec{x}_i$ has a total amount of constraint violation larger than the constraint violation of $\vec{x}_j$, then $\text{count}(\vec{x}_i)$ is incremented by one.

If any constraint $g_k(\vec{x})$ ($k = 1, \ldots, m$, where m is the total amount of constraints) is considered satisfied if $g_i(\vec{x}) \leqslant 0$, then, the total amount of constraint violation for an individual $\vec{x}_i$ (denoted as $\text{coef}(\vec{x}_i)$) is given by

$$\text{coef}(\vec{x}_i) = \sum_{k=1}^{p} g_k(\vec{x}_i) \quad \text{for all } g_k(\vec{x}_i) > 0. \tag{45}$$

- Compute fitness using the following rules:
  7. If $\vec{x}_i$ is feasible, then $\text{rank}(\vec{x}_i) = \text{fitness}(\vec{x}_i)$, else
  8. $\text{rank}(\vec{x}_i) = 1/\text{rank}(\vec{x}_i)$.
- Individuals are selected based on $\text{rank}(\vec{x}_i)$ (stochastic universal sampling is used).
- Values produced by $\text{fitness}(\vec{x}_i)$ must be normalized to ensure that the rank of feasible individuals is always higher than the rank of infeasible ones.

This approach uses a real-coded GA with a simple self-adaptive mechanism for crossover and mutation (see [23] for details) and it does not require any additional parameters to maintain diversity in the population (as is normally the case of evolutionary multiobjective optimization techniques [21]).

Ray et al. [140] proposed an approach in which solutions are ranked separately based on the value of their objective functions and their constraints. Then, a set of mating restrictions are applied based on the information that each individual has of its own feasibility (this idea was inspired on an earlier approach by Hinterding and Michalewicz [75]), so that the global optimum can be reached through cooperative learning.

Finally, Runarsson and Yao [149] proposed a constraint-handling approach based on stochastic ranking that has some resemblance with the technique of Surry and Radcliffe [168]. In this case, however, the population is ranked using a stochastic version of bubble sort in which individuals are compared to their adjacent neighbors through a certain number of sweeps (this number is probabilistically determined). The approach aims to find whether the objective function or the penalty function is dominating the search so that an appropriate balance can be found and the evolutionary algorithm can be guided to the global optimum in an efficient way. The authors used a multi-member evolution strategy with this approach and were able to match (and even improve in some cases) the results produced by Koziel and Michalewicz [92] in the benchmark functions of Michalewicz [104], at a lower computational cost.

5.4.1. Advantages and disadvantages

COMOGA compared fairly with a penalty-based approach in a pipe-sizing problem, since the resulting EA was less sensitive to changes in the parameters. However, the results achieved were not better than those found with a penalty function [167]. It should be added that COMOGA [167,168] requires several extra parameters, although its authors argue that the technique is not particularly sensitive to their values [167]. This technique uses Pareto ranking based on constraint violation [168]. From Operations Research we know that determining which solutions in some set are Pareto optimal is a computationally expensive process (it is $O(k \times M^2)$, where k is the number of objectives and M is the population size)).

The approach of Parmee and Purchase [129] was developed for a heavily constrained search space and it proved to be appropriate to reach the feasible region. However, this application of a multi-objective optimization technique does not aim at finding the global optimum of the problem, and the use of special operators suggested by the authors certainly limits the applicability of their approach.

The approach of Camponogara and Talukdar [16] has obvious problems to keep diversity (a common problem when using evolutionary multi-objective optimization techniques [21]). This is indicated by the fact that the technique discards the worst individuals at each generation. Also, the use of line search increases the cost (computationally speaking) of the approach. Finally, it is not clear what is the impact of the segment chosen to search in the overall performance of the algorithm.

The approach of Jiménez and Verdegay [81] can hardly be said to be using a multi-objective optimization technique since it only ranks infeasible individuals based on constraint violation. A subtle problem with this approach is that the evolutionary process first concentrates only on the constraint satisfaction problem and therefore it samples points in the feasible region essentially at random [168]. This means that in some cases (e.g., when the feasible region is disjoint) we might land in an inappropriate part of the feasible region from which we will not be able to escape. However, this approach (as in the case of the technique Parmee and Purchase [129]) may be a good alternative to find a feasible point in a heavily constrained search space.

The main drawback of the population-based approach Coello [24] is the number of sub-populations that may be needed in larger problems, since they will increase linearly with the number of constraints. However, it is possible to deal with that problem in two different ways: first, some constraints could be tied; that means that two or more constraints could be assigned to the same sub-population. That would significantly reduce the number of sub-populations in highly constrained problems. Second, the approach could be parallelized, in which case a high number of sub-populations would not be a serious drawback, since they could be processed concurrently. The current algorithm would however need modifications as to decide the sort of interactions between a master process (responsible for actually optimizing the whole problem) and the slave sub-processes (all the sub-populations responsible for the constraints of the problem).

Specialists in evolutionary multi-objective optimization may indicate that VEGA is not a very good choice because of its well-known limitations (it tries to find individuals that excel only in one dimension regardless of the others [60,151]). However, that drawback turns out to be an advantage in the context of constraint-handling, because what we want to find are precisely solutions that are feasible, instead of good compromises that may not satisfy one of the constraints.

Coello's approach based on non-dominance [25] tends to perform well. However, as it is normally the case of constraint-handling techniques based on evolutionary multi-objective optimization concepts, this approach tends to generate good trade-offs that may be more beneficial in highly constrained search spaces (since they will allow us to approach the feasible region more efficiently). This implies that this approach may have more difficulties to reach the global optimum efficiently.

The approach of Ray et al. [140] is a promising venue of future research in constraint-handling, since it uses not only concepts from multiobjective optimization, but it also incorporates specific domain knowledge into the constraint-handling mechanism of their GA. This makes the approach very efficient (computationally speaking) with respect to other constraint-handling techniques, although there are some sacrifices (as in Coello's approach) in terms of quality of the solutions produced.

The approach of Runarsson and Yao [149] constitutes another promising path of future research in constraint-handling. Their approach is efficient and highly competitive with other (more sophisticated) techniques. Its only current drawback is the need of a parameter (called P_f by the authors of the technique) that defines the probability of using only the objective function for comparisons in the ranking process (when lying in the infeasible region). The authors of the technique, however, have provided some guidelines to compute the most appropriate value of this parameter [149]. [2]

6. Hybrid methods

Within this category we are considering methods that are coupled with another technique (normally a numerical optimization approach) to handle constraints in an EA.

6.1. Lagrangian multipliers

Adeli and Cheng [1] proposed a hybrid EA that integrates the penalty function method with the primal–dual method. This approach is based on sequential minimization of the Lagrangian method, and uses a fitness function of the form

$$\text{fitness} = f(\vec{x}) + \frac{1}{2}\sum_{j=1}^{m}\gamma_j\left\{[g_j(\vec{x}) + \mu_j]^+\right\}^2, \tag{46}$$

where $\gamma_i > 0$, μ_i is a parameter associated with the ith constraint, and m is the total number of constraints. Also

$$[g_j(\vec{x}) + \mu_j]^+ = \max[0, g_j(\vec{x}) + \mu_j]. \tag{47}$$

[2] The technique also requires another parameter (the number of sweeps to be performed) which, however, can be fixed.

The proposal of Adeli and Cheng [1] was to define μ_j in terms of the previously registered maximum violation of its associated constraint and scale it using a parameter β. This parameter is defined by the user and has to be greater than one. γ_j is increased using also the parameter β, whose value (kept constant through the entire process) is multiplied by the previous value adopted for γ_j. This is to ensure that the penalty is increased over generations.

This approach follows Powell's early proposal [135] of combining the penalty function method with the primal dual method. By using an outer loop we can update the Lagrange multiplier $\lambda_i = \gamma_i \mu_i$ automatically according to the information obtained in previous iterations. This makes unnecessary that penalty function coefficients or Lagrange multipliers go to infinity to ensure convergence.

Additionally, no derivatives of the objective function or the constraints are required to update the coefficients used by the Lagrange multipliers [1].

Kim and Myung [88,117] proposed the use of an evolutionary optimization method combined with an augmented Lagrangian function that guarantees the generation of feasible solutions during the search process. This proposal is an extension of a system called *Evolian* [116,118], which uses evolutionary programming with a multi-phase optimization procedure in which the constraints are scaled. During the first phase of the algorithm, the objective is to optimize

$$\text{fitness}(\vec{x}) = f(\vec{x}) + \frac{C}{2}\left(\sum_{i=1}^{n}(\max[0, g_i])^2(\vec{x}) + \sum_{j=1}^{p}|h_j(\vec{x})|^2\right), \tag{48}$$

where C is a constant. Once this phase is finished (i.e., once constraint violations have been decreased as much as the user wants), the second phase starts. During this second phase, the optimization algorithm of Maa and Shanblatt [99] is applied to the best solution found during the first phase.

The second phase uses Lagrange multipliers to adjust the penalty function according to the feedback information received from the environment during the evolutionary process, in a way akin to the proposal of Adeli and Cheng [1].

6.1.1. Advantages and disadvantages

The technique of Adeli and Cheng [1] provided them with good results, but the additional parameters needed to make it work properly do not seem to overcome the most serious drawbacks of a traditional penalty function. They initialize these parameters following the recommendations of Belegundu and Arora [11], but it is not clear what is the impact of these parameters when chosen in an arbitrary manner.

The main drawback of the appraoch of Kim and Myung [88,117] is the same as before: despite the fact that they provide more guidelines regarding the definition of some of the extra parameters needed by their procedure, there are still several values that have to be adjusted using an empirical procedure.

6.2. Constrained optimization by random evolution

Belur [12] proposed a hybrid technique called Constrained Optimization by Random Evolution (CORE). The main idea of this approach is to use random evolutionary search combined with a mathematical programming technique for unconstrained optimization (the author used the Nelder and Mead's simplex method [120], but any other similar technique should work as well). Whenever a solution is not feasible, the following constraint functional is minimized:

$$C(\vec{x}) = \sum_{i \in C_1} h_i^2(\vec{x}) - \sum_{j \in C_2} g_j(\vec{x}), \tag{49}$$

where

$$C_1 = \left\{ i = 1, \ldots, n / |h_i(\vec{x})| > \varepsilon_c \right\}, \tag{50}$$

$$C_2 = \left\{ j = 1, \ldots, q / g_j(\vec{x}) < 0 \right\} \tag{51}$$

and ε_c is the tolerance allowed in the equality constraints $h_i(\vec{x})$.

6.2.1. Advantages and disadvantages

This minimization process can be seen as a repair algorithm for numerical optimization, which implies that this technique has the same problems of the repair algorithms described in Section 4.

6.3. Fuzzy logic

Van Le [95] proposed a combination of fuzzy logic and evolutionary programming to handle constraints. The main idea was to replace constraints of the form $g_i(\vec{x}) \leqslant b_i$ by a set of fuzzy constraints $C_1, \ldots, C_m$, $i = 1, \ldots, m$ defined as

$$\mu C_i(\vec{x}) = \mu_{\sigma(b_i,\epsilon_i)}(g_i(\vec{x})), \quad i = 1, \ldots, m, \tag{52}$$

where ϵ_i is a positive real number that represents the tolerable violation of the constraints, and

$$\mu_{\sigma(a,s)}(\vec{x}) = \begin{cases} 1 & \text{if } x \leqslant a, \\ \dfrac{e^{-p((x-a)/s)^2} - e^{-p}}{1 - e^{-p}} & \text{if } a < x \leqslant a + s, \\ 0 & \text{if } x > a + s. \end{cases} \tag{53}$$

The rationale behind this fuzzification process is to allow a higher degree of tolerance if $g_i(\vec{x})$ is (greater than b_i but) close to b_i and then the tolerance decreases rapidly when the error increases.

The fitness function is then redefined as

$$\text{fitness}(\vec{x}) = f(\vec{x}) \times \min\left(\mu_{C_1}(\vec{x}), \ldots, \mu_{C_m}(\vec{x})\right). \tag{54}$$

6.3.1. Advantages and disadvantages

The idea of using degrees of constraint satisfaction as weight factors for the fitness of potential solutions is interesting and the use of fuzzy logic to determine the acceptability of a certain solution seems a natural way of processing constraints. However, the main problem with this approach is that it requires the definition of ϵ_i (the tolerable violation of constraints) and p for each particular problem. Furthermore, Van Le provides very little empirical evidence of the performance of his technique, although this is certainly a research path that is worth exploring.

6.4. Immune system

Forrest and Perelson [61] and Smith et al. [163,164] explored the use of a computational model of the immune system in which a population of antibodies is evolved to cover a set of antigens. In this proposal, binary strings were proposed to model both antibodies and antigens, and an antibody was said to match an antigen if their bit strings were complementary (maximally different).

Although Smith et al. [163,164] proposed this approach as a way to keep diversity in multi-modal optimization problems, Hajela and Lee [70,71] extended it to handle constraints.

The algorithm proposed by Hajela and Lee [70] is the following:

1. Generate a random population. Compute objective function values and a cumulative measure of constraint violation.
2. Separate feasible and infeasible individuals. Rank individuals within each group based on their objective function values. Compute an average objective function value of a subset of feasible individuals.
3. Choose a number of feasible individuals with objective function value closest to the average value computed in the previous step. Sort these individuals. They are called the *antigen* population.
4. Infeasible individuals are subject to an immune system simulation, generating antibodies to the antigen population of the previous step. This simulation yields a subpopulation of designs with a reduction in the level of constraint violations.
5. Conduct a traditional simulation of an EA with the objective function as the only measure of fitness. The population is seeded with all currently feasible individuals from step (2), and enough copies of constraint

conditioned individuals obtained in step (4). Several approaches are possible to introduce these constraint conditioned individuals. Hajela and Lee used two: (a) introduce multiple copies of the best constraint conditioned individual, and (b) introduce multiple copies, drawn at random from the best 25% of constraint conditioned individuals.

The immune system simulation consists of using a simple matching function that computes the similarity (on a bit-per-bit basis, assuming binary encoding) between two chromosomes. Then, the population of antibodies is co-evolved until they become sufficiently similar to their antigens by maximizing their degree of matching.

The idea is to adapt infeasible solutions to the current feasible individuals. The performance of the approach depends on the selection of antibodies (infeasible individuals) that are exposed to the antigens during the simulation. There are several choices. For example, all the infeasible individuals could be included in the antibody group that is exposed to the antigens from step (3). In this case, we would try to adapt infeasible individuals to the characteristics of the average feasible population. Another approach could be to use only those infeasible individuals that are close to the average objective function value of the antigen population. Such an approach would be based on the premise that individual features that determine objective function value are similar for the antibodies and antigens. Therefore the antibodies would inherit those features from the antigens that promote constraint satisfaction [70,71].

A simpler instance of this technique, called *expression strategies*, was proposed by Hajela and Yoo [72]. In this case, feasible and infeasible individuals are combined using uniform crossover [169] in such a way that their chromosomic material is exchanged.

It is worth mentioning that Hajela and Yoo [72] proposed the use of the Kreisselmeir–Steinhauser function [165] to handle equality constraints. The idea is that if $h_i(\vec{x})$ is the ith equality constraint, then it can be represented by a pair of inequality constraints as

$$h_i(\vec{x}) \leqslant 0, \quad -h_i(\vec{x}) \leqslant 0. \tag{55}$$

The Kreisselmeir–Steinhauser function can then be used to fold these constraints into a cumulative measure Ω:

$$\Omega = (1/\rho) \ln \left(e^{\rho h_i(\vec{x})} + e^{-\rho h_i(\vec{x})} \right) - (1/\rho) \ln 2 + c_1, \tag{56}$$

where c_1 represents the width of a band that replaces the original strict equality constraint, and ρ is a user-defined constant that scales the amount of constraint violation (ρ must take a non-zero non-negative value). As ρ grows, the scaling factor becomes one (i.e., there is no scaling of the constraint violation). If the equality constraint $h_i(\vec{x})$ is satisfied, then $h_i(\vec{x}) = 0$, and thus $\Omega = c_1$. By reducing c_1 the solutions are forced to move closer to the equality constraint. Therefore, we can see c_1 as a tolerance value. The idea then is to replace constraints of the form $h_i(\vec{x}) = 0$, by constraints of the form $\Omega \leqslant c_1$.

6.4.1. Advantages and disadvantages

Since the bit matching process used by this approach does not require evaluating the fitness function, its computational cost is not really significant. However, some other issues remain to be solved. For example, it is not clear what is the effect (in terms of performance) of mixing different proportions of each population (antibodies and antigens). It is also unclear what is the behavior of the algorithm when there are no feasible individuals in the initial population.

The underlying assumption of this approach might rise some controversy: by making the genotype of an infeasible individual more similar to the genotype of a feasible individual we can actually decrease its amount of constraint violation. Smith et al. [164] provide some theoretical analysis regarding the expected fitness of an individual when either perfect or partial matching is required. However, their work was done in the context of fitness sharing (where the emphasis is to keep diversity in the population), and is not necessarily applicable to constraint handling. Therefore, the only support to this hypothesis are the empirical results reported by Hajela and Lee [70,71].

Finally, although it is always possible to compute genotypic distances regardless of the encoding used by the EA, it is not entirely clear if it is possible to use this approach with non-binary representations.

6.5. Cultural algorithms

Some social researchers have suggested that culture might be symbolically encoded and transmitted within and between populations, as another inheritance mechanism [49,141]. Using this idea, Reynolds [142] developed a computational model in which cultural evolution is seen as an inheritance process that operates at two levels: the micro-evolutionary and the macro-evolutionary levels.

At the micro-evolutionary level, individuals are described in terms of "behavioral traits" (which could be socially acceptable or unacceptable). These behavioral traits are passed from generation to generation using several socially motivated operators. At the macro-evolutionary level, individuals are able to generate "mappa" [141], or generalized descriptions of their experiences. Individual mappa can be merged and modified to form "group mappa" using a set of generic or problem specific operators. Both levels share a communication link.

Reynolds [142] proposed the use of genetic algorithms to model the micro-evolutionary process, and Version Spaces [112] to model the macro-evolutionary process of a cultural algorithm.

The main idea behind this approach is to preserve beliefs that are socially accepted and discard (or prune) unacceptable beliefs. The acceptable beliefs can be seen as constraints that direct the population at the micro-evolutionary level [103]. Therefore, constraints can influence directly the search process, leading to an efficient optimization process. In fact, Reynolds et al. [143] and Chung and Reynolds [20] have explored this area of research with very encouraging results in numerical optimization. A cultural algorithm models the evolution of the culture component of an evolutionary computational system over time. This culture component provides an explicit mechanism for acquisition, storage and integration of individual and group's problem solving experience and behavior [82]. In contrast, traditional EAs only use implicit mechanisms for representing and storing individual's global acquired knowledge, which is passed from generation to generation.

The approach taken by Chung and Reynolds [20] was to use a hybrid of evolutionary programming and GENOCOP [107] in which they incorporated an interval constraint-network [38,80] to represent the constraints of the problem at hand. An individual is considered as "acceptable" when it satisfies all the constraints of the problem. When that does not happen, then the belief space is adjusted (the intervals associated with the constraints are adjusted). This approach is really a more sophisticated version of a repair algorithm in which an infeasible solution is made feasible by replacing its genes by a different value between its lower and upper bounds. Since GENOCOP assumes a convex search space, it is relatively easy to design operators that can exploit a search direction towards the boundary between the feasible and infeasible regions.

In more recent work, Jin and Reynolds [82] proposed an n-dimensional regional-based schema, called *belief-cell*, as an explicit mechanism that supports the acquisition, storage and integration of knowledge about non-linear constraints in a cultural algorithm. This *belief-cell* can be used to guide the search of an EA (evolutionary programming in this case) by pruning the instances of infeasible individuals and promoting the exploration of promising regions of the search space. The key aspect of this work is precisely how to represent and save the knowledge about the problem constraints in the belief space of the cultural algorithm.

The idea of Jin and Reynolds' approach is to build a map of the search space similar to the "Divide-and-Label" approaches used for robot motion planning [94]. This map is built using information derived from evaluating the constraints of each individual in the population of the EA. The map is formed by dividing the search space in sub-areas called *cells*. Each cell can be classified as: feasible (if it lies completely on a feasible region), infeasible (if it lies completely on an infeasible region), semi-feasible (if it occupies part of the feasible and part of the infeasible regions), or unknown (if that region has not been explored yet). This map is used to derive rules about how to guide the search of the EA (avoiding infeasible regions and promoting the exploration of feasible regions). In other words, these cells are used to form a "navigation map" for the EA.

6.5.1. Advantages and disadvantages

This approach presents an interesting hybrid of knowledge-based approaches and evolutionary computation techniques. However, it does not require the explicit definition of rules by the user, since the

algorithm is able to learn its own rules over time. The approach has been refined in the last few years, and proposals such as the one contained in Jin and Reynolds' paper [82] are applicable even to problems with disjoint feasible regions (normally quite difficult for most constraint-handling techniques). However, the technique requires more refinement and validation. For example, in Jin and Reynolds' paper, only one test function was used. Also, they had to experiment with different strategies to update the constraint knowledge of the problem. The other issue that deserves consideration is the efficiency of the method. Jin and Reynolds' do not discuss the computation cost of building belief maps in the presence of non-linear optimization constraints, and their approach might be sensitive to high dimensionality.

6.6. Ant colony optimization

This technique was proposed by Dorigo et al. [30,46–48] and it consists of a meta-heuristic intended for hard combinatorial optimization problems such as the traveling salesperson. The main algorithm is really a multi-agent system where low level interactions between single agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. The idea was inspired by colonies of real ants, which deposit a chemical substance on the ground called *pheromone* [46]. This substance influences the behavior of the ants: they will tend to take those paths where there is a larger amount of pheromone.

Recently, some researchers [13,173] have extended this technique to numerical optimization problems, with very promising results. The main issue when extending the basic approach to deal with continuous search spaces is how to model a continuous nest neighborhood with a discrete structure. Bilchev and Parmee [14], for example, proposed to represent a finite number of directions whose origin is a common base point called the *nest*. Since the idea is to cover eventually all the continuous search space, these vectors evolve over time according to the fitness values of the ants.

To handle constraints, Bilchev and Parmee [13,14] proposed to make a food source "unacceptable" in case it violated a constraint regardless of the value of its objective function (i.e., death penalty). As evolution progresses, some food sources that were acceptable before, will vanish, as constraints are tightened (i.e., the amount of "acceptable" constraint violation is decreased).

To make this model effective, three different levels of abstraction were considered: (a) the individual search agent (the lowest level in which any local search technique could be used), (b) the cooperation between agents (the middle level, which consists of a joint search effort in a certain direction), and (c) the meta-cooperation between agents (the highest level, which determines cooperation among different paths rather than just among different individuals).

The results obtained by Bilchev and Parmee [13,14] were very encouraging and showed clearly the high potential of this technique in multi-modal and/or heavily constrained search spaces.

6.6.1. Advantages and disadvantages

The first drawback of this approach is that it needs several parameters to work: first, an additional procedure has to be used to locate the *nest* (Bilchev and Parmee [13] suggest the use of a niching EA), which implies extra computational effort. Second, it requires a search radius R, which defines the portion of the search space that will be explored by the ants and has an obvious impact on the performance of the algorithm. Third, it is necessary to provide a model for the exhaustion of the food source to avoid that the ants pass through the same (already exhausted) path more than once.

Finally, it is necessary to be very careful about the equilibrium between local and global exploration, because in some cases (e.g., highly multi-modal landscapes), too much CPU time could be spent in local searches.

7. Some experimental results

To have an idea of the differences among some of the techniques discussed in this paper, we have conducted a small experimental study in which we implemented and tested six different penalty-based approaches coupled to a genetic algorithm and an approach based on non-dominance. The techniques selected are the following:

- Static penalty [78] (see Section 2.1),
- Dynamic penalty [83] (see Section 2.2),
- Annealing penalty [105] (see Section 2.3),
- Adaptive penalty [10,69] (see Section 2.4),
- Death penalty (see Section 2.7),
- Co-evolutionary penalty [25] (see Section 2.5),
- Use of non-dominance [23] (see Section 5.4).

Additionally, we will compare results against those found by other researchers using mathematical programming techniques and/or other types of GAs.

The first five penalty-based approaches previously indicated are representative of the techniques most commonly used in the standard literature on evolutionary optimization. The sixth and seventh approaches are proposals of the author. The co-evolutionary penalty uses two nested GAs so that one tries to adjust the penalty factors that the other one uses to optimize the objective function. The last approach (which we will denote as MGA, for multi-objective genetic algorithm) was proposed as an alternative to the manual fine tuning of the penalty factors. This last approach consists of a real-coded GA with arithmetical crossover [104], non-uniform mutation, elitism, tournament selection, and a simple self-adaptation mechanism for defining the crossover and mutation rates along the evolutionary process (see [23] for details).

All the penalty-based approaches indicated above (except for the co-evolutionary penalty) were implemented using a GA with binary representation, two-point crossover, tournament selection, and uniform mutation. The co-evolutionary penalty was implemented using a GA with fixed point representation [26], uniform crossover and non-uniform mutation [104].

Three test functions were selected to perform our small comparative study. Their corresponding description together with our comparison of results follows.

7.1. Example 1: Himmelblau's nonlinear optimization problem

This problem was originally proposed by Himmelblau [74], and it has been used before as a benchmark for several other GA-based techniques that use penalties [64]. In this problem, there are five design variables $(x_1, x_2, x_3, x_4, x_5)$, six nonlinear inequality constraints and 10 boundary conditions. The problem can be stated as follows:

$$\text{Minimize} \quad f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{57}$$

$$\text{Subject to :} \quad g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5, \tag{58}$$

$$g_2(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2, \tag{59}$$

$$g_3(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4, \tag{60}$$

$$0 \leqslant g_1(\vec{x}) \leqslant 92, \tag{61}$$

$$90 \leqslant g_2(\vec{x}) \leqslant 110, \tag{62}$$

$$20 \leqslant g_3(\vec{x}) \leqslant 25, \tag{63}$$

$$78 \leqslant x_1 \leqslant 102, \tag{64}$$

$$33 \leqslant x_2 \leqslant 45, \tag{65}$$

$$27 \leqslant x_3 \leqslant 45, \tag{66}$$

$$27 \leqslant x_4 \leqslant 45, \tag{67}$$

$$27 \leqslant x_5 \leqslant 45. \tag{68}$$

Table 1
Comparison of several constraint-handling techniques for the first example (Himmelblau's function) (N/A = not available) (PART I)

Results	MGA [23]	Gen [64]	Static penalty [78]	GRG [74]	Co-evolutionary penalty [25]
Best	−31005.7966	−30183.576	−30790.27159	−30373.949	−31020.859
Mean	−30862.8735	N/A	−30446.4618	N/A	−30984.2407
Worst	−30721.0418	N/A	−29834.3847	N/A	−30792.4077
S.D.	73.240	N/A	226.3428	N/A	73.6335

Table 2
Comparison of several constraint-handling techniques for the first example (Himmelblau's function) (PART II)

Results	Dynamic [83]	Annealing [105]	Adaptive [10,69]	Death penalty
Best	−30903.877	−30829.201	−30903.877	−30790.271
Mean	−30539.9156	−30442.126	−30448.007	−30429.371
Worst	−30106.2498	−29773.085	−29926.1544	−29834.385
S.D.	200.035	244.619	249.485	234.555

The comparison of results for several constraint-handling approaches for the first example are shown in Tables 1 and 2. This problem was originally solved using the Generalized Reduced Gradient method (GRG) [74]. Gen and Cheng [64] solved this problem using a genetic algorithm based on both local and global reference (they used a population size of 400 individuals, a crossover rate of 0.8, and a mutation rate of 0.088). [3] The solutions reported for the penalty-based approaches (static penalty, dynamic penalty, annealing penalty, adaptive penalty and death penalty) in Table 1 were produced after performing 30 runs, using the following parameters: population size = 50, crossover rate = 0.8, mutation rate = 0.005, maximum number of generations = 100. Specific parameters for the dynamic penalty are: $C = 0.5$, $\alpha = \beta = 2.0$ (Eq. (12) was used to assign fitness). Specific parameters for annealing penalties are: $\tau_0 = 1.0$, $\tau_f = 0.000001$, and τ is updated every 20 generations (Eq. (18) is used to assign fitness). Specific parameters for the adaptive penalty are: $\beta_1 = 1.0$, $\beta_2 = 2.0$, $k = 20$, $\lambda(0) = 100.0$ (Eq. (24) is used to assign fitness). For the static penalty, local and global penalty factors were defined as indicated by Homaifar et al. [78, p. 253] for this example.

The co-evolutionary penalty used the following parameters: crossover rate = 0.8, initial mutation rate = 0.1, $pop\_size_1 = 60$, $pop\_size_2 = 30$, $Gmax_1 = 25$, $Gmax_2 = 20$.

The solutions shown for the MGA were produced after performing 30 runs, and using the following parameters: population size = 50, and maximum number of generations = 100 (crossover and mutation rates were obtained through self-adaptation along the evolutionary process).

As expected, the death penalty, which does not use any constraint-violation information, had a poorer performance than the other GA-based approaches. Also, the dynamic penalty approach was better than a static penalty, and there was not much difference between using an adaptive penalty function and the dynamic penalty suggested by Joines and Houck [83]. The annealing penalty, however, had a poorer performance than the dynamic and adaptive penalties.

The best approaches were the co-evolutionary penalty and the MGA, with the first reporting slightly better results than the second. Note however that while all penalty-based approaches and the MGA performed only 5000 fitness function evaluations, the co-evolutionary penalty technique performed a considerably higher number of fitness function evaluations (900 000). One of the main advantages of the MGA is that no fine-tuning of the penalty factors are required. The co-evolutionary penalty also presents this advantage, but its use implies a significantly higher computational cost.

Also, note that the other penalty-based approaches can provide better results if some fine-tuning of their parameters (including their penalty factors) takes place. Finally, it should be clear from these results that all GA-based approaches performed better than the mathematical programming technique used in this case (GRG).

[3] The maximum number of generations used is unknown.

7.2. Example 2: Welded beam design

A welded beam is designed for minimum cost subject to constraints on shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), end deflection of the beam (δ), and side constraints [138]. There are four design variables as shown in Fig. 1 [138]: $h(x_1)$, $l(x_2)$, $t(x_3)$ and $b(x_4)$.

The problem can be stated as follows:

$$\text{Minimize} \quad f(\vec{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2) \tag{69}$$

$$\text{Subject to :} \quad g_1(\vec{x}) = \tau(\vec{x}) - \tau_{\max} \leqslant 0, \tag{70}$$

$$g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{\max} \leqslant 0, \tag{71}$$

$$g_3(\vec{x}) = x_1 - x_4 \leqslant 0, \tag{72}$$

$$g_4(\vec{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leqslant 0, \tag{73}$$

$$g_5(\vec{x}) = 0.125 - x_1 \leqslant 0, \tag{74}$$

$$g_6(\vec{x}) = \delta(\vec{x}) - \delta_{\max} \leqslant 0, \tag{75}$$

$$g_7(\vec{x}) = P - P_c(\vec{x}) \leqslant 0, \tag{76}$$

where

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}, \tag{77}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2}, \quad \tau'' = \frac{MR}{J}, \quad M = P\left(L + \frac{x_2}{2}\right), \tag{78}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}, \tag{79}$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}, \tag{80}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2}, \quad \delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4}, \tag{81}$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{x_3^2x_4^6/36}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right), \tag{82}$$

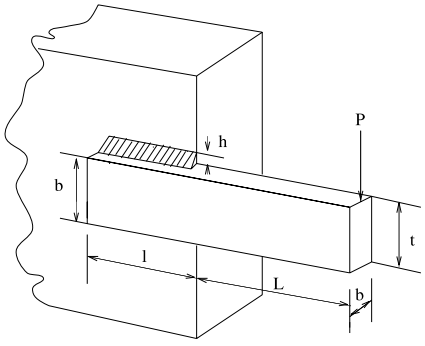

Fig. 1. The welded beam used for the second example.

Table 3
Comparison of several constraint-handling techniques for the second example (welded beam) (N/A = not available) (PART I)

Results	MGA [23]	Deb [42]	Siddall [158]	Ragsdell [137]	Co-evolutionary penalty [25]
Best	1.8245	2.4331	2.3815	2.3859	1.7483
Mean	1.9190	N/A	N/A	N/A	1.7720
Worst	1.9950	N/A	N/A	N/A	1.7858
S.D.	0.05377	N/A	N/A	N/A	0.01122

Table 4
Comparison of several constraint-handling techniques for the second example (welded beam) (PART II)

Results	Static [78]	Dynamic [83]	Annealing [105]	Adaptive [10,69]	Death penalty
Best	2.0469	2.1062	2.0713	1.9589	2.0821
Mean	2.9728	3.1556	2.9533	2.9898	3.1158
Worst	4.5741	5.0359	4.1261	4.84036	4.5138
S.D.	0.6196	0.7006	0.4902	0.6515	0.6625

$$P = 6000 \text{ lb}, \quad L = 14 \text{ in}, \quad \delta_{\max} = 0.25 \text{ in}, \quad E = 30 \times 10^6 \text{ psi}, \quad G = 12 \times 10^6 \text{ psi},$$
$$\tau_{\max} = 13\,600 \text{ psi}, \quad \sigma_{\max} = 30\,000 \text{ psi}.$$

For this example, we used the same parameters for all the approaches, except the static penalty, for which we used a value of 50.0 for all cases (local and global penalty factors).

The comparison of results for several constraint-handling approaches for the second example are shown in Tables 3 and 4. This problem has been solved before by Deb [42] using a simple genetic algorithm with binary representation, and a traditional penalty function as suggested by Goldberg [67], and by Ragsdell and Phillips [137] using geometric programming. Ragsdell and Phillips also compared their results with those produced by the methods contained in a software package called "Opti-Sep" [158], which includes the following numerical optimization techniques: ADRANS (Gall's adaptive random search with a penalty function), APPROX (Griffith and Stewart's successive linear approximation), DAVID (Davidon–Fletcher–Powell with a penalty function), MEMGRD (Miele's memory gradient with a penalty function), SEEK1 and SEEK2 (Hooke and Jeeves with two different penalty functions), SIMPLX (Simplex method with a penalty function) and RANDOM (Richardson's random method). In the case of Siddall's techniques [158], only the best solution produced by the techniques contained in "Opti-Sep" is displayed.

In this case, the results were somewhat more surprising. The dead penalty turned out to be better than the dynamic penalty. This may due to the use of an inappropriate penalty factor, but it illustrates well the idea of why the fine tuning of the penalty factors becomes an important issue when using penalty-based constraint-handling techniques. Regarding the other approaches, the use of a static penalty was again no better than using an adaptive penalty or a death penalty. However, the static penalty was better than the annealing penalty in this example. This is due to an inappropriate cooling schedule for the annealing penalty. The best results were produced by the co-evolutionary penalty (even its worst results was better than the best result of the MGA). Note however that the computational cost of this technique remains significantly higher (900 000 fitness function evaluations vs. 5000 of the other approaches). Once again, all the mathematical programming techniques provided much poorer results than any of the GA-based approaches.

7.3. Example 3: Design of a pressure vessel

A cylindrical vessel is capped at both ends by hemispherical heads as shown in Fig. 2. The objective is to minimize the total cost, including the cost of the material, forming and welding. There are four design variables: T_s (thickness of the shell), T_h (thickness of the head), R (inner radius) and L (length of the cylindrical section of the vessel, not including the head). T_s and T_h are integer multiples of 0.0625 inch, which are the available thicknesses of rolled steel plates, and R and L are continuous. Using the same notation given by Kannan and Kramer [84], the problem can be stated as follows:

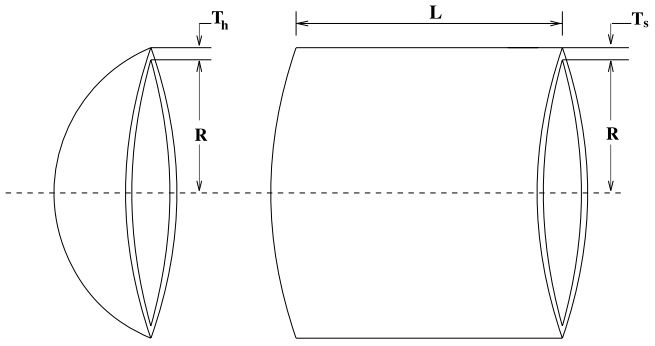

Fig. 2. Center and end section of the pressure vessel used for the first example.

$$\text{Minimize} \quad f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \tag{83}$$

$$\text{Subject to :} \quad g_1(\vec{x}) = -x_1 + 0.0193x_3 \leqslant 0, \tag{84}$$

$$g_2(\vec{x}) = -x_2 + 0.00954x_3 \leqslant 0, \tag{85}$$

$$g_3(\vec{x}) = -\pi x_3^2 x_4 - \tfrac{4}{3}\pi x_3^3 + 1\,296\,000 \leqslant 0, \tag{86}$$

$$g_4(\vec{x}) = x_4 - 240 \leqslant 0. \tag{87}$$

The comparison of results for several constraint-handling approaches for the second example are shown in Tables 5 and 6. This problem has been solved before by Deb [43] using Genetic Adaptive Search (GeneAS), by Kannan and Kramer [84] using an augmented Lagrangian Multiplier approach, and by Sandgren [150], using Branch and Bound.

All the penalty-based techniques (except for the co-evolutionary penalty that kept the same parameters indicated before) used a population size of 500 and a maximum number of generations of 5000 for this example. This change was required so that these approaches could provide competitive results (the population size and maximum number of generations were empirically determined). All their other parameters remained the same (local and global penalties were defined with a value of 50 for the static penalty approach, as in the previous example). For the MGA, we only extended the maximum number of generations to 1000 (using the same population size of 50, as before).

This example illustrates how the use of penalty-based approaches is highly dependant on the problem at hand. Despite the fact that all the penalty-based approaches performed 2 500 000 fitness function evaluations (except for the co-evolutionary penalty approach that performed 900 000 evaluations, as before), they

Table 5
Comparison of several constraint-handling techniques for the third example (pressure vessel) (N/A = not available) (PART I)

Results	MGA [23]	Deb [43]	Kannan [84]	Sandgren [150]	Co-evolutionary penalty [25]
Best	6069.3267	6410.3811	7198.0428	8129.1036	6288.7445
Mean	6263.7925	N/A	N/A	N/A	6293.8432
Worst	6403.4500	N/A	N/A	N/A	6308.1497
S.D.	97.9445	N/A	N/A	N/A	7.4133

Table 6
Comparison of several constraint-handling techniques for the third example (pressure vessel) (PART II)

Results	Static [78]	Dynamic [83]	Annealing [105]	Adaptive [10,69]	Death penalty
Best	6110.8117	6213.6923	6127.4143	6110.8117	6127.4143
Mean	6656.2616	6691.5606	6660.8631	6689.6049	6616.9333
Worst	7242.2035	7445.6923	7380.4810	7411.2532	7572.6591
S.D.	320.8196	322.7647	330.7516	330.4483	358.8497

were not able to match the results of the dominance-based approach (MGA), which only performed 50,000 fitness function evaluations. Note that the co-evolutionary penalty approach did not perform very well in this example, mainly due to its choice of parameters (allowing a larger number of fitness function evaluations could slightly improve its performance). Again, the mathematical programming techniques produced poorer results than any of the GA-based approaches.

8. Some recommendations

Having such a wide variety of possible techniques to handle constraints in evolutionary optimization may be overwhelming for a newcomer. However, as suggested by our small comparative study, even the simple use of a death penalty may be sufficient in some applications, if nothing about the problem is known. Our suggestion for beginners in the use of evolutionary algorithms is therefore to use penalty-based approaches first (maybe a simple static or dynamic penalty approach), since they are the easiest to implement and are also quite efficient. Later on, and depending on the application at hand, other techniques may be desirable. For example, if a combinatorial optimization problem has to be solved, then repair algorithms (see Section 4) may be the best choice. If dealing with linear constraints, then the use of special representations and operators (see Section 3) may become necessary. If dealing with highly constrained search spaces, then the use of techniques that separate constraints and objectives (see Section 5) may be useful. If something about the problem is known, or if there is a need of saving time fine tuning the penalty factors of a penalty function of any type, then one can consider the use of approaches such as those discussed in Section 5.4 or Section 6. More sophisticated techniques are normally reserved for more complex problems in which the results found by penalty-based approaches are far from satisfactory, or when the computational costs related to these techniques are too high.

Also, it is important to add that most of the comparative studies of constraint-handling techniques reported in the literature are inconclusive. Whereas some technique may perform better in a certain class of functions (e.g., nonlinear optimization problems), it will tend to be inferior in a different domain (e.g., combinatorial optimization). Despite the goal of generality that should characterize new constraint-handling techniques, it is known that because of the No Free-Lunch Theorems [174], it is expected that the best constraint-handling techniques for a certain type of problems will tend to exploit specific domain knowledge.

9. Conclusions and future research paths

In this paper we have given a very comprehensive review of the most important constraint-handling techniques developed for evolutionary algorithms. We reviewed a wide variety of techniques that go from several variations of a simple penalty function to biologically inspired techniques that emulate the behavior of the immune system, culture, or ant colonies. However, there is still plenty of room for new techniques and more research in this area. For example, regarding the development of new approaches, the following issues deserve special attention:

- *Generality*. Ideally, the same constraint-handling approach should work with any kind of problem and constraints. If modifications are required, they should be minor. There are several approaches such as decoders and the use of special representations, that depend on certain characteristics of the problem and cannot be easily generalized. Although we should not aim to produce a single (universal) constraint-handling technique that will defeat any other [174], it is reasonable to aim to make it easier to be adapted to different types of problems/constraints.
- *Minimum fine tuning*. Finding an appropriate penalty function for an optimization problem in general normally requires a lot of fine tuning. Ideally, a good constraint-handling technique should minimize the requirement of this fine tuning, or should not need it at all. When fine tuning is necessary, the performance of the algorithm tends to depend on it. Furthermore, this trial and error process adds up to the parameter tuning required by most EAs (i.e., how to define the values of: population size, crossover and mutation rates, maximum number of generations, etc.).

- *Efficiency*. In many real-world applications, a single evaluation of the fitness function might be very expensive. Therefore, a good constraint-handling technique should not require a high evaluation cost. In Section 2.5 we saw an example of a technique that requires a high number of fitness function evaluations to obtain the information that will guide the search. As we mentioned before, in some applications, the problem of finding a feasible solution might be itself NP-hard [161].
- *Well-known limitations*. If we assume that no single constraint-handling technique will be the best for all kinds of problems, then it is important to identify clearly the limitations of each available technique to know when to use them. Michalewicz and Schoenauer [109] discussed this issue, but the question remains open regarding the characteristics that we could use from a problem to decide what technique to use.
- *Incorporation of knowledge about the domain*. Incorporating knowledge about an specific domain reduces the generality of an evolutionary approach [67]. However, in highly complex problems (e.g., heavily constrained search spaces) some knowledge about the domain can considerably improve performance of an EA. Therefore, it is desirable that a good constraint-handling approach has the capability to incorporate efficiently such domain knowledge whenever is available.

The "utopical" constraint-handling technique for EAs should combine the best of these issues. The development of such a technique, however, might prove impossible in practice [174]. For example, if we emphasize efficiency, our constraint-handling technique might lose generality. The converse is also normally true. Nevertheless, even if these issues are incompatible to a certain extent, they should at least be taken into consideration when developing a new approach and aim to obtain reasonable trade-offs among these objectives.

Regarding open areas of research, the following are particularly important:

- *Comparisons of approaches*. Despite the several comparative studies of constraint-handling techniques used with EAs reported in the literature (see for example [101–103,109]), more work is required. It is desirable, for example, to study in more detail the behavior of certain approaches under different sorts of constraints (linear, non-linear, etc.), so that we can establish under what conditions is more convenient to use them.

Michalewicz et al. [106] argue that any problem can be characterized by a certain set of parameters including the following: number of linear and nonlinear constraints, number of equality constraints, number of active constraints, ratio between the feasible search space and the whole search space, and the type of objective function (number of variables, number of local optima, continuity of the function, etc.). However, tests performed in the past regarding eleven (now considered classical) test functions (see [109]) have produced inconclusive evidence about the behavior of several constraint-handling techniques. This means that the appropriate choice of a certain technique in the absence of knowledge about the domain remains as an open research problem [106,109].

- *Test suites*. A very important issue closely related to the previous one is the existence of good test suites that are publicly available. Regarding this issue, there is some literature that can be used (see for example [56,102]). [4] Chung and Reynolds [20] have provided a test suite for cultural algorithms. More recently, Michalewicz et al. [106] have proposed the design of a scalable test suite of constrained optimization problems in which many features can be easily tuned to allow the evaluation of the advantages and disadvantages of a certain constraint-handling technique. The test case generator proposed by Michalewicz et al. [106] has six parameters that can be tuned to investigate advantages and disadvantages of a certain constraint-handling technique: dimensionality of the search space, multimodality of the search space, number of constraints used, connectedness of the feasible subspaces, ratio between the feasible search space and the whole search space, and function ruggedness. However, more work in this direction is desirable.
- *Metrics*. Closely related to the previous issue is the development of good metrics that allow to compare different techniques in a quantitative way. Beyond the obvious comparative issues such as quality of the final solution found and amount of fitness function evaluations required, there are other aspects of a certain technique that might be relevant in certain cases. For example, it would be interesting to have

[4] The web page http://solon.cma.univie.ac.at/~neum/glopt/test.html also contains test problems for constrained optimization algorithms.

a metric that traces down the behavior of a technique in terms of the number of feasible solutions found. Also, metrics that determine that robustness and convergence rate of a certain technique are highly desirable. These metrics would be very useful to determine the limitations of a constraint-handling approach in quantitative form.

- *Multi-objetive optimization*. Despite the considerably large amount of research on evolutionary multiobjective optimization (EMO) [21], little emphasis has been made on constraint-handling. In fact, many of the early EMO approaches considered only unconstrained problems. As we saw in Section 5.4, EMO techniques can be used also to handle constraints, but ironically, their use in multiobjective optimization has been very limited until now. Most EMO researchers tend to use traditional (static) penalty functions instead of trying to exploit the power of EMO techniques to handle constraints as additional objectives.

Acknowledgements

The author thanks the two anonymous reviewers for their comments that greatly helped him to improve the contents of this paper. He also acknowledges the support from CONACyT through NSF-CONACyT project No. 32999-A.

References

[1] H. Adeli, N.-T. Cheng, Augmented Lagrangian genetic algorithm for structural optimization, J. Aerospace Engrg. 7 (1) (1994) 104–118.
[2] T. Bäck (Ed.), Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.
[3] T. Bäck (Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, July 1997.
[4] T. Bäck, F. Hoffmeister, H.-P. Schwefel, A survey of evolution strategies, in: R.K. Belew, L.B. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 2–9.
[5] T. Bäck, S. Khuri, An evolutionary heuristic for the maximum independent set problem, in: Z. Michalewicz, J.D. Schaffer, H.-P. Schwefel, D.B. Fogel, H. Kitano (Eds.), Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1994, pp. 531–535.
[6] J.E. Baker, Adaptive selection methods for genetic algorithms, in: J.J. Grefenstette (Ed.), Proceedings of an International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 100–111.
[7] J.E. Baker, An analysis of the effects of selection in genetic algorithms, PhD thesis, Vanderbilt University, Nashville, TN, 1989.
[8] J.C. Bean, Genetics and random keys for sequencing and optimization, Technical Report TR 92-43, Department of Industrial and Operations Engineering, The University of Michigan, 1992.
[9] J.C. Bean, Genetics and random keys for sequencing and optimization, ORSA J. Comput. 6 (2) (1994) 154–160.
[10] J.C. Bean, A.B. Hadj-Alouane, A dual genetic algorithm for bounded integer programs, Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan, 1992 (R.A.I.R.O.-R.O, invited submission to special issue on GAs and OR), to appear.
[11] A.D. Belegundu, J.S. Arora, A computational study of transformation methods for optimal design, AIAA J. 22 (4) (1984) 535–542.
[12] S.V. Belur, CORE: Constrained optimization by random evolution, in: J.R. Koza (Ed.), Late Breaking Papers at the Genetic Programming 1997 Conference, Stanford Bookstore, Stanford University, CA, July 1997, pp. 280–286.
[13] G. Bilchev, I.C. Parmee, The ant colony metaphor for searching continuous design spaces, in: T.C. Fogarty (Ed.), Evolutionary Computing, Springer, Sheffield, UK, April 1995, pp. 25–39.
[14] G. Bilchev, I.C. Parmee, Constrained and multi-modal optimisation with an ant colony search model, in: I.C. Parmee, M.J. Denham (Eds.), Proceedings of the 2nd International Conference on Adaptive Computing in Engineering Design and Control, University of Plymouth, Plymouth, UK, March 1996.
[15] D. Brélaz, New methods to color vertices of a graph, Commun. ACM 22 (1979) 251–256.
[16] E. Camponogara, S.N. Talukdar, A genetic algorithm for constrained and multiobjective optimization, in: J.T. Alander (Ed.), 3rd Nordic Workshop on Genetic Algorithms and their Applications (3NWGA), University of Vaasa, Vaasa, Finland, August 1997, pp. 49–62.
[17] S.E. Carlson, A general method for handling constraints in genetic algorithms, in: Proceedings of the Second Annual Joint Conference on Information Science, 1995, pp. 663–667.
[18] C.W. Carroll, The created response surface technique for optimizing nonlinear restrained systems, Operations Research 9 (1961) 169–184.
[19] V. Chankong, Y.Y. Haimes, Multiobjective decision making: theory and methodology, in: Systems Science and Engineering, North-Holland, Amsterdam, 1983.

[20] C.-J. Chung, R.G. Reynolds, A testbed for solving optimization problems using cultural algorithms, in: L.J. Fogel, P.J. Angeline, T. Bäck (Eds.), Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, 1996.
[21] C.A.C. Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge Inf. Syst., Int. J. 1 (3) (1999) 269–308.
[22] C.A.C. Coello, The use of a multiobjective optimization technique to handle constraints, in: A.A.O. Rodríguez, M.R.S. Ortiz, R.S. Hermida (Eds.), Proceedings of the Second International Symposium on Artificial Intelligence (Adaptive Systems), Institute of Cybernetics, Mathematics and Physics, Ministry of Science Technology and Environment, La Habana, Cuba, 1999, pp. 251–256.
[23] C.A.C. Coello, Constraint-handling using an evolutionary multiobjective optimization technique, Civil Engrg. Environ. Syst. 17 (2000) 319–346.
[24] C.A.C. Coello, Treating constraints as objectives for single-objective evolutionary optimization, Engrg. Optim. 32 (3) (2000) 275–308.
[25] C.A.C. Coello, Use of a self-adaptive penalty approach for engineering optimization problems, Comput. Ind. 41 (2) (2000) 113–127.
[26] C.A.C. Coello, A.D. Christiansen, A simple genetic algorithm for the design of reinforced concrete beams, Engrg. Comput. 13 (4) (1997) 185–196.
[27] C.A.C. Coello, M. Rudnick, A.D. Christiansen, Using genetic algorithms for optimal design of trusses, in: Proceedings of the Sixth International Conference on Tools with Artificial Intelligence, IEEE Computer Society Press, New Orleans, LA, USA, November 1994, pp. 88–94.
[28] D.W. Coit, A.E. Smith, Penalty guided genetic search for reliability design optimization, Comput. Ind. Engrg. 30 (4) (September 1996) 895–904 (special issue on genetic algorithms).
[29] D.W. Coit, A.E. Smith, D.M. Tate, Adaptive penalty methods for genetic optimization of constrained combinatorial problems, INFORMS J. Comput. 8 (2) (1996) 173–182.
[30] A. Colorni, M. Dorigo, V. Maniezzo, Distributed optimization by ant colonies, in: P. Bourgine, F. Varela (Eds.), Proceedings of the First European Conference on Artificial Life, MIT Press/Bradford Books, Cambridge, MA, 1991.
[31] R. Courant, Variational methods for the solution of problems of equilibrium and vibrations, Bull. Am. Math. Soc. 49 (1943) 1–23.
[32] W.A. Crossley, E.A. Williams, A study of adaptive penalty functions for constrained genetic algorithm based optimization, in: AIAA 35th Aerospace Sciences Meeting and Exhibit, AIAA Paper 97-0083, Reno, Nevada, January 1997.
[33] C. Darwin, The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life, The Book League of America, 1929 (originally published in 1859).
[34] D. Dasgupta, Z. Michalewicz (Eds.), Evolutionary Algorithms in Engineering Applications, Springer, Berlin, 1997.
[35] Y. Davidor, Analogous crossover, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1989, pp. 98–103.
[36] Y. Davidor, Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization, World Scientific, Singapore, 1990.
[37] Y. Davidor, A genetic algorithm applied to robot trajectory generation, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 144–165 (Chapter 12).
[38] E. Davis, Constraint propagation with interval labels, Artif. Intell. 32 (1987) 281–331.
[39] L. Davis, Genetic Algorithms and Simulated Annealing, Pitman, London, 1987.
[40] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.
[41] H. de Garis, Genetic programming: Building artificial nervous systems using genetically programmed neural networks modules, in: R. Porter, B. Mooney (Eds.), Proceedings of the 7th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1990, pp. 132–139.
[42] K. Deb, Optimal design of a welded beam via genetic algorithms, AIAA J. 29 (11) (1991) 2013–2015.
[43] K. Deb, GeneAS: A robust optimal design technique for mechanical component design, in: D. Dasgupta, Z. Michalewicz (Eds.), Evolutionary Algorithms in Engineering Applications, Springer, Berlin, 1997, pp. 497–514.
[44] K. Deb, An efficient constraint handling method for genetic algorithms, Comput. Methods Appl. Mech. Engrg. 186 (2–4) (2000) 311–338.
[45] K. Deb, D.E. Goldberg, An investigation of niche and species formation in genetic function optimization, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann, San Mateo, CA, June 1989, pp. 42–50.
[46] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, New York, 1989.
[47] M. Dorigo, L.M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, IEEE Trans. Evolutionary Comput. 1 (1) (1997) 53–66.
[48] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: Optimization by a colony of cooperating agents, IEEE Trans. Syst. Man Cybern. Part B 26 (1) (1996) 29–41.
[49] W.H. Durham, Co-evolution: Genes, Culture, and Human Diversity, Stanford University Press, Stanford, CA, 1994.
[50] A.E. Eiben, P.-E. Raué, Zs. Ruttkay, GA-easy and GA-hard constraint satisfaction problems, in: M. Meyer (Ed.), Proceedings of the ECAI'94 Workshop on Constraint Processing, Springer, Berlin, 1995, pp. 267–284.
[51] A.E. Eiben, Zs. Ruttkay, Self-adaptivity for constraint satisfaction: Learning penalty functions, in: Proceedings of the 3rd IEEE Conference on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, 1996, pp. 258–261.

[52] A.E. Eiben, J.K. van der Hauw, Adaptive penalties for evolutionary graph coloring, in: Artificial Evolution'97, Springer, Berlin, 1998, pp. 95–106.
[53] A.E. Eiben, J.K. van der Hauw, J.I. van Hemert, Graph coloring with adaptive evolutionary algorithms, J. Heuristics 4 (1) (1998) 25–46.
[54] E. Falkenauer, A new representation and operators for genetic algorithms applied to grouping problems, Evol. Comput. 2 (2) (1994) 123–144.
[55] A.V. Fiacco, G.P. McCormick, Extensions of SUMT for nonlinear programming: Equality constraints and extrapolation, Manage. Sci. 12 (11) (1968) 816–828.
[56] C.A. Floudas, P.M. Pardalos, A Collection of Test Problems for Constrained Global Optimization Algorithms, Lecture Notes in Computer Science, Springer, Berlin, 1990.
[57] D.B. Fogel, Evolutionary Computation. Toward a New Philosophy of Machine Intelligence, The Institute of Electrical and Electronic Engineers, New York, 1995.
[58] L.J. Fogel, Artificial Intelligence through Simulated Evolution, Wiley, New York, 1966.
[59] C.M. Fonseca, P.J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, University of Illinois at Urbana-Champaign, Morgan Kauffman, San Mateo, CA, 1993, pp. 416–423.
[60] C.M Fonseca, P. Fleming, An overview of evolutionary algorithms in multiobjective optimization, Evol. Comput. 3 (1) (1995) 1–16.
[61] S. Forrest, A.S. Perelson, Genetic algorithms and the immune system, in: H.-P. Schwefel, R. Männer (Eds.), Parallel Problem Solving from Nature, Springer, Berlin, Germany, 1991, pp. 320–325.
[62] M. Gen, R. Cheng, Interval programming using genetic algorithms, in: Proceedings of the Sixth International Symposium on Robotics and Manufacturing, Montpelleir, France, 1996.
[63] M. Gen, R. Cheng, A survey of penalty techniques in genetic algorithms, in: T. Fukuda, T. Furuhashi (Eds.), Proceedings of the 1996 International Conference on Evolutionary Computation, IEEE, Nagoya, Japan, 1996, pp. 804–809.
[64] M. Gen, R. Cheng, Genetic Algorithms & Engineering Design, Wiley, New York, 1997.
[65] F. Glover, Heuristics for integer programming using surrogate constraints, Decision Sciences 8 (1) (1977) 156–166.
[66] F. Glover, G. Kochenberger, Critical event tabu search for multidimensional knapsack problems, in: Proceedings of the International Conference on Metaheuristics for Optimization, Kluwer Academic Publishers, Dordrecht, Netherlands, 1995, pp. 113–133.
[67] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[68] D.E. Goldberg, M.P. Samtani, Engineering optimization via genetic algorithm, in: Ninth Conference on Electronic Computation, ASCE, New York, 1986, pp. 471–482.
[69] A.B. Hadj-Alouane, J.C. Bean, A genetic algorithm for the multiple-choice integer program, Operations Research 45 (1997) 92–101.
[70] P. Hajela, J. Lee, Constrained genetic search via schema adaptation. An immune network solution, in: N. Olhoff, G.I.N. Rozvany (Eds.), Proceedings of the First World Congress of Structural and Multidisciplinary Optimization, Pergamon, Goslar, Germany, 1995, pp. 915–920.
[71] P. Hajela, J. Lee, Constrained genetic search via schema adaptation. An immune network solution, Struct. Optim. 12 (1996) 11–15.
[72] P. Hajela, J. Yoo, Constraint handling in genetic search using expression strategies, AIAA J. 34 (12) (1996) 2414–2420.
[73] S.A. Harp, T. Samad, Genetic synthesis of neural network architecture, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 202–221 (Chapter 15).
[74] D.M. Himmelblau, Applied Nonlinear Programming, McGraw-Hill, New York, 1972.
[75] R. Hinterding, Z. Michalewicz, Your brains and my beauty: Parent matching for constrained optimisation, in: Proceedings of the 5th International Conference on Evolutionary Computation, Anchorage, AK, May 1998, pp. 810–815.
[76] F. Hoffmeister, J. Sprave, Problem-independent handling of constraints by use of metric penalty functions, in: L.J. Fogel, P.J. Angeline, T. Bäck (Eds.), Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96), MIT Press, San Diego, CA, February 1996, pp. 289–294.
[77] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Harbor, 1975.
[78] A. Homaifar, S.H.Y. Lai, X. Qi, Constrained optimization via genetic algorithms, Simulation 62 (4) (1994) 242–254.
[79] W.-C. Huang, C.-Y. Kao, J.-T. Horng, A genetic algorithm approach for set covering problem, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, New York, 1994, pp. 569–573.
[80] E. Hyvoenen, Constraint reasoning based on interval arithmetic – The tolerance propagation approach, Artif. Intell. 58 (1992) 71–112.
[81] F. Jiménez, J.L. Verdegay, Evolutionary techniques for constrained optimization problems, in: Seventh European Congress on Intelligent Techniques and Soft Computing, Springer, Aachen, Germany, 1999.
[82] X. Jin, R.G. Reynolds, Using knowledge-based evolutionary computation to solve nonlinear constraint optimization problems: A cultural algorithm approach, in: 1999 Congress on Evolutionary Computation, IEEE Service Center, Washington, DC, July 1999, pp. 1672–1678.
[83] J. Joines, C. Houck, On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs, in: D. Fogel (Ed.), Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, Orlando, FL, 1994, pp. 579–584.

[84] B.K. Kannan, S.N. Kramer, An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design, J. Mech. Des. Trans. ASME 116 (1994) 318–320.
[85] S. Kazarlis, V. Petridis, Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms, in: A.E. Eiben, T. Bäck, M. Schoenauer, H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature V – PPSN V, Springer, Amsterdam, Netherlands, 1998.
[86] D.G. Kim and P. Husbands, Riemann mapping constraint handling method for genetic algorithms, Technical Report CSRP 469, COGS, University of Sussex, UK, 1997.
[87] D.G. Kim, P. Husbands, Mapping based constraint handling for evolutionary search; Thurston's circle packing and grid generation, in: I. Parmee (Ed.), The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation, Springer, Plymouth, UK, April 1998, pp. 161–173.
[88] J.-H. Kim, H. Myung, Evolutionary programming techniques for constrained optimization problems, IEEE Trans. Evol. Comput. 1 (1997) 129–140.
[89] S. Kirkpatrick Jr., C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[90] R. Kowalczyk, Constraint consistent genetic algorithms, in: Proceedings of the 1997 IEEE Conference on Evolutionary Computation, IEEE, Indianapolis, USA, April 1997, pp. 343–348.
[91] S. Koziel, Z. Michalewicz, A decoder-based evolutionary algorithm for constrained parameter optimization problems, in: T. Bäck, A.E. Eiben, M. Schoenauer, H.-P. Schwefel (Eds.), Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V), Springer, Amsterdam, September 1998, pp. 231–240.
[92] S. Koziel, Z. Michalewicz, Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization, Evol. Comput. 7 (1) (1999) 19–44.
[93] V. Kumar, Algorithms for constraint-satisfaction problems: a survey, AI Mag. (1992) 32–44.
[94] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Norwell, MA, 1993.
[95] T. Van Le, A fuzzy evolutionary approach to constrained optimization problems, in: Proceedings of the Second IEEE Conference on Evolutionary Computation, IEEE, Perth, November 1995, pp. 274–278.
[96] G.E. Liepins, M.D. Vose, Representational issues in genetic optimization, J. Exp. Theoret. Comput. Sci. 2 (2) (1990) 4–30.
[97] G.E. Liepins, W.D. Potter, A genetic algorithm approach to multiple-fault diagnosis, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 237–250 (Chapter 17).
[98] C.B. Lucasius, M.J.J. Blommers, L.M.C. Buydens, G. Kateman, A genetic algorithm for conformational analysis of DNA, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 251–281 (Chapter 18).
[99] C. Maa, M. Shanblatt, A two-phase optimization neural network, IEEE Trans. Neural Networks 3 (6) (1992) 1003–1009.
[100] Z. Michalewicz, K. Deb, M. Schmidt, Th. Stidsen, Evolutionary algorithms for engineering applications, in: K. Miettinen, P. Neittaanmäki, M.M. Mäkelä, J. Périaux (Eds.), Evolutionary Algorithms in Engineering and Computer Science, Wiley, Chichester, England, 1999, pp. 73–94.
[101] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, second ed., Springer, Berlin, 1992.
[102] Z. Michalewicz, Genetic algorithms, numerical optimization, and constraints, in: L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, University of Pittsburgh, Morgan Kaufmann, San Mateo, CA, July 1995, pp. 151–158.
[103] Z. Michalewicz, A survey of constraint handling techniques in evolutionary computation methods, in: J.R. McDonnell, R.G. Reynolds, D.B. Fogel (Eds.), Proceedings of the Fourth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, 1995, pp. 135–155.
[104] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, Berlin, 1996.
[105] Z. Michalewic, N.F. Attia, Evolutionary optimization of constrained problems, in: Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scientific, Singapore, 1994, pp. 98–108.
[106] Z. Michalewicz, K. Deb, M. Schmidt, T.J. Stidsen, Towards understanding constraint-handling methods in evolutionary algorithms, in: 1999 Congress on Evolutionary Computation, IEEE Service Center, Washington, DC, July 1999, pp. 581–588.
[107] Z. Michalewicz, C.Z. Janikow, Handling constraints in genetic algorithms, in: R.K. Belew, L.B. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 151–157.
[108] Z. Michalewicz, G. Nazhiyath, Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints, in: D.B. Fogel (Ed.), Proceedings of the Second IEEE International Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1995, pp. 647–651.
[109] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evol. Comput. 4 (1) (1996) 1–32.
[110] Z. Michalewicz, J. Xiao, Evaluation of paths in evolutionary planner/navigator, in: Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 1995, pp. 45–52.
[111] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1996.
[112] T. Mitchell, Version spaces: An approach to concept learning, PhD thesis, Computer Science Department, Stanford University, Stanford, CA, 1978.
[113] A.F. Kuri Morales, Personal Communication, 1999.
[114] A. Kuri Morales, C.V. Quezada, A universal eclectic genetic algorithm for constrained optimization, in: Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing, EUFIT'98, Verlag Mainz, Aachen, Germany, September 1998, pp. 518–522.
[115] H. Mühlenbein, Parallel genetic algorithms in combinatorial optimization, in: O. Balci, R. Sharda, S. Zenios (Eds.), Computer Science and Operations Research, Pergamon Press, New York, 1992, pp. 441–456.

[116] H. Myung, J.-H. Kim, Evolian: Evolutionary optimization based on Lagrangian with constraint scaling, in: P.J. Angeline, R.G. Reynolds, J.R. McDonnell, R. Eberhart (Eds.), Proceedings of the Sixth Annual Conference on Evolutionary Programming, Springer, Indianapolis, April 1997, pp. 177–188.
[117] H. Myung, J.-H. Kim, Hybrid interior-Lagrangian penalty based evolutionary optimization, in: V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), Proceedings of the Seventh Annual Conference on Evolutionary Programming, Springer, Berlin, 1998, pp. 85–94.
[118] H. Myung, J.-H. Kim, D.B. Fogel, Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems, in: J.R. McDonnell, R.G. Reynolds, D.B. Fogel (Eds.), Proceedings of the Fourth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, 1995, pp. 449–463.
[119] R. Nakano, Conventional genetic algorithm for job shop problems, in: R.K. Belew, L.B. Booker (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1991, pp. 474–479.
[120] A. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (1965) 308–313.
[121] B.A. Norman, J.C. Bean, Random keys genetic algorithm for scheduling: Unabridged version, Technical Report 95-10, University of Michigan, Ann Harbor, 1995.
[122] B.A. Norman, J.C. Bean, A random keys genetic algorithm for job shop scheduling, Technical Report 96-10, University of Michigan, Ann Harbor, 1996.
[123] B.A. Norman, A.E. Smith, Random keys genetic algorithm with adaptive penalty function for optimization of constrained facility layout problems, in: T. Bäck, Z. Michalewicz, X. Yao (Eds.), Proceedings of the 1997 International Conference on Evolutionary Computation, IEEE, Indianapolis, Indiana, 1997, pp. 407–411.
[124] A.L. Olsen, Penalty functions for the knapsack problem, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, New York, 1994, pp. 554–558.
[125] D. Orvosh, L. Davis, Shall we repair? genetic algorithms, combinatorial optimization and feasibility constraints, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kauffman, San Mateo, CA, July 1993, p. 650.
[126] D. Orvosh, L. Davis, Using a genetic algorithm to optimize problems with feasibility constraints, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, New York, 1994, pp. 548–553.
[127] C.C. Palmer, A. Kershenbaum, Representing trees in genetic algorithms, in: Z. Michalewicz, J.D. Schaffer, H.-P. Schwefel, D.B. Fogel, H. Kitano (Eds.), Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway, NJ, 1994, pp. 379–384.
[128] J. Paredis, Co-evolutionary constraint satisfaction, in: Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, Springer, New York, 1994, pp. 46–55.
[129] I.C. Parmee, G. Purchase, The development of a directed genetic search technique for heavily constrained design spaces, in: I.C. Parmee (Ed.), Adaptive Computing in Engineering Design and Control-'94, University of Plymouth, Plymouth, UK, 1994, pp. 97–102.
[130] I. Parmee (Ed.), The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation, Springer, Plymouth, UK, 1998.
[131] R. Parsons, S. Forrest, C. Burks, Genetic Algorithms for DNA Sequence Assembly, in: Proceedings of the 1st International Conference on Intelligent Systems in Molecular Biology, AAAI Press, July 1993.
[132] R.J. Parsons, S. Forrest, C. Burks, Genetic algorithms, operators and DNA fragment assembly, Machine Learning 21 (1–2) (1995) 11–33.
[133] V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, Lecture Notes in Computer Science, vol. 1447, Springer, San Diego, CA, March 1998.
[134] D. Powell, M.M. Skolnick, Using genetic algorithms in engineering design optimization with non-linear constraints, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, University of Illinois at Urbana-Champaign, Morgan Kaufmann, San Mateo, CA, July 1993, pp. 424–431.
[135] M.J.D. Powell, A method for nonlinear constraints in minimization problems, in: R. Fletcher (Ed.), Optimization, Academic Press, London, England, 1969.
[136] N.J. Radcliffe, Equivalence class analysis of genetic algorithms, Complex Systems 5 (1991) 183–220.
[137] K.M. Ragsdell, D.T. Phillips, Optimal design of a class of welded structures using geometric programming, ASME J. Eng. Ind., Series B 98 (3) (1976) 1021–1025.
[138] S.S. Rao, Engineering Optimization, third ed., Wiley, New York, 1996.
[139] K. Rasheed, An adaptive penalty approach for constrained genetic-algorithm optimization, in: J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, R.L. Riolo (Eds.), Proceedings of the Third Annual Genetic Programming Conference, Morgan Kaufmann, San Francisco, CA, 1998, pp. 584–590.
[140] T. Ray, T. Kang, S.K. Chye, An evolutionary algorithm for constrained optimization, in: D. Whitley, D. Goldberg, E. Cantú-Paz, L. Spector, I. Parmee, H.-G. Beyer (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000), Morgan Kaufmann, San Francisco, CA, 2000, pp. 771–777.
[141] A.C. Renfrew, Dynamic modeling in archaeology: what, when, and where?, in: S.E. van der Leeuw (Ed.), Dynamical Modeling and the Study of Change in Archaelogy, Edinburgh University Press, Edinburgh, Scotland, 1994.
[142] R.G. Reynolds, An introduction to cultural algorithms, in: A.V. Sebald, L.J. Fogel (Eds.), Proceedings of the Third Annual Conference on Evolutionary Programming, World Scientific, River Edge, NJ, 1994, pp. 131–139.
[143] R.G. Reynolds, Z. Michalewicz, M. Cavaretta, Using cultural algorithms for constraint handling in GENOCOP, in: J.R. McDonnell, R.G. Reynolds, D.B. Fogel (Eds.), Proceedings of the Fourth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, 1995, pp. 298–305.

[144] J.T. Richardson, M.R. Palmer, G. Liepins, M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann, Reading, MA, 1989, pp. 191–197.
[145] R.G. Le Riche, R.T. Haftka, Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm, AIAA J. 31 (5) (1993) 951–970.
[146] R.G. Le Riche, R.T. Haftka, Improved genetic algorithm for minimum thickness composite laminate design, Compos. Engrg. 3 (1) (1994) 121–139.
[147] R.G. Le Riche, C. Knopf-Lenoir, R.T. Haftka, A segregated genetic algorithm for constrained structural optimization, in: L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, University of Pittsburgh, Morgan Kaufmann, San Mateo, CA, July 1995, pp. 558–565.
[148] E. Ronald, When selection meets seduction, in: L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, July 1995, pp. 167–173.
[149] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Trans. Evol. Comput. 4 (3) (2000) 284–294.
[150] E. Sandgren, Nonlinear integer and discrete programming in mechanical design, in: Proceedings of the ASME Design Technology Conference, Kissimine, Florida, 1988, pp. 95–105.
[151] J.D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum, London, 1985, pp. 93–100.
[152] M. Schoenauer, Z. Michalewicz, Evolutionary computation at the edge of feasibility, in: H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (Eds.), Proceedings of the Fourth Conference on Parallel Problem Solving from Nature, Springer, Berlin, September 1996, pp. 245–254.
[153] M. Schoenauer, Z. Michalewicz, Sphere operators and their applicability for constrained parameter optimization problems, in: V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben (Eds.), Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, Lecture Notes in Computer Science, vol. 1447, Springer, San Diego, CA, March 1998, pp. 241–250.
[154] M. Schoenauer, S. Xanthakis, Constrained GA optimization, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, July 1993, pp. 573–580.
[155] M. Schütz, J. Sprave, Application of partially mixed-integer evolution strategies with mutation rate pooling, in: L.J. Fogel, P.J. Angeline, T. Bäck (Eds.), Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96), MIT Press, San Diego, CA, February 1996, pp. 345–354.
[156] H.-P. Schwefel, Numerical Optimization of Computer Models, Wiley, Great Britain, 1981.
[157] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.
[158] J.N. Siddall, Analytical Design-Making in Engineering Design, Prentice-Hall, Englewood Cliffs, NJ, 1972.
[159] W. Siedlecki, J. Sklanski, Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann, San Mateo, CA, June 1989, pp. 141–150.
[160] S. Carlson Skalak, R. Shonkwiler, S. Babar, M. Aral, Annealing a genetic algorithm over constraints. Available from http://vlead.mech.virginia.edu/publications/shenkpaper/shenkpaper.html.
[161] A.E. Smith, D.W. Coit, Constraint handling techniques – penalty functions, in: T. Bäck, D.B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press and Institute of Physics Publishing, Oxford, 1997 (Chapter C 5.2).
[162] A.E. Smith, D.M. Tate, Genetic optimization using a penalty function, in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, University of Illinois at Urbana-Champaign, Morgan Kaufmann, San Mateo, CA, July 1993, pp. 499–503.
[163] R.E. Smith, S. Forrest, A.S. Perelson, Searching for diverse, cooperative populations with genetic algorithms, Technical Report TCGA No. 92002, University of Alabama, Tuscaloosa, AL, 1992.
[164] R.E. Smith, S. Forrest, A.S. Perelson, Population diversity in an immune system model: Implications for genetic search, in: L.D. Whitley (Ed.), Foundations of Genetic Algorithms, vol. 2, Morgan Kaufmann, San Mateo, CA, 1993, pp. 153–165.
[165] J. Sobieszanski-Sobieski, A technique for locating function roots and for satisfying equality constraints in optimization, Struct. Optim. 4 (3–4) (1992) 241–243.
[166] E.J. Steele, R.A. Lindley, R.V. Blanden, Lamarck's Signature. How Retrogenes Are Changing Darwin's Natural Selection Paradigm, Perseus Books, Reading, MA, 1998.
[167] P.D. Surry, N.J. Radcliffe, The COMOGA method: Constrained optimisation by multiobjective genetic algorithms, Control Cybern. 26 (3) (1997).
[168] P.D. Surry, N.J. Radcliffe, I.D. Boyd, A multi-objective approach to constrained optimisation of gas supply networks: The COMOGA method, in: T.C. Fogarty (Ed.), Evolutionary Computing. AISB Workshop. Selected Papers, Springer, Sheffield, UK, 1995, pp. 166–180.
[169] G. Syswerda, Uniform crossover in genetic algorithms, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann, San Mateo, CA, June 1989, pp. 2–9.
[170] G. Syswerda, Schedule optimization using genetic algorithms, in: L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp. 332–349 (Chapter 21).

[171] D.M. Tate, A.E. Smith, A genetic approach to the quadratic assignment problem, Computers and Operations Research 22 (1) (1995) 73–78.
[172] S.R. Thangiah, An adaptive clustering method using a geometric shape for vehicle routing problems with time windows, in: L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, University of Pittsburgh, Morgan Kaufmann, San Mateo, CA, July 1995, pp. 536–543.
[173] M. Wodrich, G. Bilchev, Cooperative distributed search: The ant's way, Control and Cybernetics 26 (3) (1997) 413–446.
[174] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[175] J. Xiao, Z. Michalewicz, K. Trojanowski, Adaptive evolutionary planner/navigator for mobile robots, IEEE Trans. Evol. Comput. 1 (1) (1997) 18–28.
[176] J. Xiao, Z. Michalewicz, L. Zhang, Evolutionary planner/navigator: Operator performance and self-tuning, in: Proceedings of the 3rd IEEE International Conference on Evolutionary Computation, IEEE Press, Nagoya, Japan, May 1996.
[177] T. Yokota, M. Gen, K. Ida, T. Taguchi, Optimal design of system reliability by an improved genetic algorithm, Trans. Inst. Electron. Inf. Comput. Engrg. J78-A (6) (1995) 702–709 (in Japanese).