A Survey of Constraint Handling Techniques used with Evolutionary Algorithms

Carlos A. Coello Coello
ccoello@xalapa.lania.mx
Laboratorio Nacional de Informática Avanzada
Rébsamen 80, Xalapa, Veracruz 91090, México

Abstract

Despite the extended applicability of evolutionary algorithms to a wide range of domains, the fact that these algorithms are unconstrained optimization techniques leaves open the issue regarding how to incorporate constraints of any kind (linear, non-linear, equality and inequality) into the fitness function as to search efficiently. The main goal of this paper is to provide a detailed and comprehensive survey of the many constraint handling approaches that have been proposed for evolutionary algorithms, analyzing in each case their advantages and disadvantages, and concluding with some of the most promising paths of research.

Keywords: evolutionary algorithms, optimization, genetic algorithms, evolutionary optimization, constraint handling.

1 Introduction

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) [46], in which the main goal is to simulate the evolutionary process in a computer. There are 3 main paradigms within evolutionary algorithms, whose motivations and origins were completely independent from each other: evolution strategies [123], evolutionary programming [47], and genetic algorithms [63].

In general, evolutionary algorithms simulate evolution using 4 main elements [45]: (1) a encoding structure that will be replicated, (2) operators that affect the individuals of a population, (3) a fitness function that indicates how "good" a certain individual is with respect to the others, and (4) a selection mechanism.

The 3 main paradigms mentioned before differ mainly in terms of the importance given to the operators (recombination vs. mutation), the way in which selection is performed (stochastic vs. deterministic), the level of abstraction at which evolution is modeled (species vs. individual), and consequently, the level at which the operators are applied (phenotypic vs. genotypic).

Regardless of their differences, evolutionary algorithms have been quite successful in a wide range of applications [56, 91, 83, 2, 108, 53, 45, 109, 124]. However, an aspect normally disregarded when using evolutionary algorithms for optimization (a rather common trend) is that these algorithms are unconstrained optimization procedures, and therefore is necessary to device ways of incorporating the constraints (normally existing in any real-world application) into the fitness function.

Despite the relative success of penalty functions [115, 56] in many optimization problems, researchers in evolutionary computing have developed a considerable amount of alternative approaches to handle constraints and have proposed different ways to automate the definition of good penalty factors, which remains as the main drawback of using penalty functions.

In this paper, we provide a comprehensive survey of contraint-handling techniques that have been adopted over the years to handle all sorts of constraints (linear, non-linear, equality, inequality, explicit and implicit) in

evolutionary algorithms. A brief criticism of each of them will also be provided, showing their advantages and disadvantages, and we will conclude with some of the most promising paths of future research.

It should be mentioned that despite the fact that there are other surveys on constraint handling techniques available in the specialized literature (see for example [86, 85, 52, 23, 127]), they are either too narrow (i.e., they cover a single constraint handling technique) or they focus more on empirical comparisons and on the design of interesting test functions and do not attempt to be comprehensive as we pretend 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).

2 Basic definitions

The problem that is of interest to us is the general non-linear programming problem in which we want to:

$$\text{Find} \quad \mathbf{X} \text{ which optimizes } f(\mathbf{X}) \tag{1}$$

subject to:

$$g_i(\mathbf{X}) \leq 0, \quad i = 1, \ldots, n \tag{2}$$

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

where $\mathbf{X}$ is the vector of solutions ($\mathbf{X} = x_1, x_2, \ldots, x_r$), 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 $\mathcal{F}$ to the feasible region and with $\mathcal{S}$ to the whole search space, then it should be clear that $\mathcal{F} \subseteq \mathcal{S}$.

For an inequality constaint that satisfies $g_i(\mathbf{X}) = 0$, then we will say that is active at $\mathbf{X}$. All equality constraints h_j (regardless of the value of $\mathbf{X}$ used) are considered active at all points of $\mathcal{S}$.

For simplicity, we will use $\phi(\mathbf{X})$, $i = 1, \ldots, m$ (m is the total number of constraints) to denote constraints of both kinds (inequality and equality), unless the approach analyzed makes a clear distinction between equality and inequality constraints.

3 A Taxonomy of Approaches

Focusing only on numerical optimization, particularly regarding non-linear optimization problems, the constraint-handling techniques developed over the years can be roughly classified as follows [23]:

- Use of penalty functions.
- Maintaining a feasible population by special representations and genetic operators.
- Separation of objectives and constraints.
- Hybrid methods.
- Novel approaches.

4 Use of penalty functions

The most common approach in the EA (mainly with genetic algorithms) community to handle constraints (particularly, inequality constraints) is to use penalties. The basic approach is to define the fitness value of an individual i by extending the domain of the objective function $f(\mathbf{X})$ using [23]

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) \pm Q_i \tag{4}$$

where Q_i represents either a penalty for an infeasible individual i, or a cost for repairing such an individual (i.e., the cost for making it feasible). It is assumed that if i is feasible then $Q_i = 0$ (i.e., we do not penalize feasible individuals). In genetic algorithms, we typically have

$$Q_i = c \times \sum_{i=1}^{m} \Omega[\phi_i(\mathbf{X})] \tag{5}$$

where $\Omega[\phi_i(\mathbf{X})] = \phi_i(\mathbf{X})^2$ for all violated constraints i, and c is a penalty coefficient defined by the user [56].

Although evolution strategies normally use death penalties [123], there are also some proposals to incorporate penalty functions into this approach. For example, Hoffmeister & Sprave have proposed to use [62]:

$$Q_i = \sqrt{\sum_{i=0}^{m} H(-\phi_i(\mathbf{X}))\phi_i(\mathbf{X})^2} \tag{6}$$

where $H : \mathcal{R} \rightarrow \{0, 1\}$ is the Heavyside function:

$$H(y) = \begin{cases} 1 & : \quad y > 0 \\ 0 & : \quad y \leq 0 \end{cases} \tag{7}$$

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* [28, 116, 128]). This is due to the fact that if the penalty is too high or too low, then the problem becomes GA-hard [28, 116, 118]. However, although conceptually very simple, in practice it is quite difficult to implement this rule, because the exact location of the boundary between the feasible and infeasible regions is unknown in most problems.

It is known that the relationship between an infeasible individual and the feasible part of the search space plays a significant role in penalizing such individual [115]. However, it is not completely clear how to exploit this relationship to guide the search in the most desirable direction.

There are at least 3 main choices to define a relationship between an infeasible individual and the feasible region of the search space [23]:

1. an individual might be penalized just for being infeasible (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. [115] 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 fully 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* is the cost of making feasible an infeasible solution.

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 analized 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. [127]). Also, it is not clear how to combine the two quantities indicated by Richardson et al. [115] and how to design a fitness function that uses accurate penalties.

4.1 Static Penalties

Homaifar, Lai and Qi [64] 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 [86]:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + \sum_{j=1}^{m} R_{k,j}\phi_j^2(\mathbf{X}) \tag{8}$$

where $R_{i,j}$ are the penalty coefficients used, m is the number of constraints, $f(\mathbf{X})$ 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.

Criticism

The main drawback of this technique is the high number of parameters required [84]. For m constraints, this approach requires $m(2l + 1)$ parameters in total. So, if we have for example 6 constraints and 2 levels, we would need 30 parameters, which is a very high number considering the small size of the problem.

Other researchers have used different distance based static penalty functions [56, 4, 65, 102, 115, 13, 136], but in all cases these metrics rely on some extra parameter which is difficult to generalize and normally remains problem-dependent.

4.2 Dynamic Penalties

Joines and Houck [68] proposed a technique in which dynamic penalties (i.e., penalties that change over time) are used. Individuals are evaluated (at generation t) using:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} |\phi_j(\mathbf{X})|^{\beta} \tag{9}$$

where C, α and β are constants defined by the user, m is the number of constraints, and β is a constant (the authors used $\beta = 1$ and 2) defined by the user. This dynamic function increases the penalty as we progress through generations. In their experiments, Joines and Houck [68] used a real-coded GA with arithmetical crossover and non-uniform mutation [83]. The

Criticism

Some researchers [125] 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 to produce good penalty factors for static functions. For example, in the approach proposed by Joines and Houck [68], the quality of the solution found was very sensitive

to changes in the values of the parameters. Even when a certain set of values for these parameters ($C = 0.5$, $\beta = 1$ or 2) were found by the authors of this method to be a reasonable choice, Michalewicz [84, 88] reported that these values 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 [84, 23].

Apparently, this technique provides very good results only when the objective function is quadratic [89].

4.3 Annealing Penalties

Michalewicz and Attia [82] considered a method based on the idea of simulated annealing [72]: 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 hardly penalized in the last generations.

The method of Michalewicz and Attia [82] requires that constraints are divided into 4 groups: linear equalities, linear inequalities, nonlinear equalities and nonlinear inequalities. Also, a set of active constraints $\mathcal{A}$ has to be created, and all nonlinear equalities together with all violated nonlinear inequalities have to be included there. The population is evolved using [84]:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + \frac{1}{2\tau} \sum_{j \in \mathcal{A}} \phi_j^2(\mathbf{X}) \tag{10}$$

where τ is the cooling schedule [72].

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 [89]). 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.

Another, similar proposal was made by Carlson et al. [126]. In this case, the fitness function of an individual is computed using:

$$\text{fitness}_i(\mathbf{X}) = \mathcal{A} \cdot f_i(\mathbf{X}) \tag{11}$$

where $\mathcal{A}$ depends on 2 parameters: M, which measures the amount in 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. [126] defined $\mathcal{A}$ as:

$$\mathcal{A} = e^{-M/T} \tag{12}$$

so that the initial penalty factor is small and it increases over time so that infeasible solutions are totally discarded in the last generations.

To define T (the cooling schedule), Carlson et al. [126] use:

$$T = \frac{1}{\sqrt{t}} \tag{13}$$

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

Finally, it should be mentioned that Joines and Houck [68] also experimented with a penalty function based on simulated annealing:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + e^{(C \times t)^\alpha \times \sum_{j=1}^{m} |\phi_j(\mathbf{X})|^\beta} \tag{14}$$

where t is the generation number, $C = 0.05$, and $\alpha = \beta = 1$.

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

Criticism

One of the main drawbacks of Michalewicz and Attia's approach is its extreme sensitivity to the values of its parameters, and it is also well known that it is normally difficult to choose an appropriate cooling scheme when solving a problem with simulated annealing [72]. Michalewicz and Attia [82] used $\tau_0 = 1$ and $\tau_f = 0.000001$ in their experiments, with increments $\tau_{i+1} = 0.1 \times \tau_i$. Carlson et al. [126] decided to use the mean constraint violation ($\bar{M}$) as the starting temperature value. For the final temperature, they decided to use one hundreth 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. [126], their definition remains as the most critical issue when using simulated annealing.

Also, 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.

Regarding Joines and Houck's approach [68], their main problems had to do with the overflows produced by the fact that they did not normalize their constraints and therefore, the exponential function would sometimes fall out of the valid numerical range of the computer. Furthermore, the definition of C was not justified, and the authors admitted that further experimentation regarding its effect was necessary.

4.4 Adaptive Penalties

Bean and Hadj-Alouane [7, 57] developed a method of adapting penalties that uses a penalty function which takes a feedback from the search process. Each individual is evaluated by the formula:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + \lambda(t) \sum_{j=1}^{m} \phi_j^2(\mathbf{X}) \tag{15}$$

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

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & \text{if case \#1} \\ \beta_2 \cdot \lambda(t), & \text{if case \#2} \\ \lambda(t), & \text{otherwise,} \end{cases} \tag{16}$$

where cases #1 and #2 denote situations where the best individual in the last k generation was always (case #1) or was never (case #2) feasible, $\beta_1, \beta_2 > 1$, 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 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 [128] proposed an approach later refined by Coit and Smith [20] and Coit et al. [21] 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:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + (B_{feasible} - B_{all}) \sum_{j=1}^{m} \left(\frac{\phi_j(\mathbf{X})}{NFT(t)} \right)^k \tag{17}$$

where $B_{feasible}$ is the best known objective function at generation t, B_{all} is the best (unpenalized) overall objective function at generation t, $\phi_j(\mathbf{X})$ is the amount by which the constraint ϕ_j 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 [20]), 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 [89, 52].

Norman & Smith [101] further applied Coit & Smith's approach to facility layout problems, and apparently the technique has been used only in combinatorial optimization problems.

Gen and Cheng [52] indicate that Yokota et al. [140] 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 Smith et al. [128]):

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) \times P(\mathbf{X}) \tag{18}$$

where $P(\mathbf{X})$ is defined as:

$$P(\mathbf{X}) = 1 - \frac{1}{m}\sum_{j=1}^{m}\left(\frac{\Delta b_j(\mathbf{X})}{b_j}\right)^k \tag{19}$$

and

$$\Delta b_j(\mathbf{X}) = max\{0, \phi_j(\mathbf{X}) - b_j\} \tag{20}$$

In this case, $\Delta b_j(\mathbf{X})$ refers to the violation of constraint j. Notice that this approach is really a special case of Smith et al.'s approach in which $NFT = b_j$, assuming that $g_j(\mathbf{X}) \leq b_j$ is required to consider a solution as fully feasible.

Gen and Cheng [51] later refined their approach introducing a more severe penalty for infeasible solutions. In the new version of their algorithm,

$$P(\mathbf{X}) = 1 - \frac{1}{m}\sum_{j=1}^{m}\left(\frac{\Delta b_j(\mathbf{X})}{\Delta b_j^{max}}\right)^k \tag{21}$$

$$\Delta b_j(\mathbf{X}) = max\{0, \phi_j(\mathbf{X}) - b_j\} \tag{22}$$

$$\Delta b_j^{max} = max\{\epsilon, \Delta b_j(\mathbf{X}); \mathbf{X} \in P(t)\} \tag{23}$$

where $\Delta b_j(\mathbf{X})$ is the value by which the constraint j is violated in the n-th chromosome. Δb_j^{max} is the maximum violation of constraint j in the whole (current) population, and ϵ is a small positive number used to avoid dividing by zero [52]. 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 [52].

Eiben & van der Hauw [41], Eiben et al. [42] and Eiben & Ruttkay [40] proposed an adaptive penalty function that was successfully applied to the graph 3-coloring problem. They used a fitness function of the form

$$\text{fitness}_i(\mathbf{X}) = \sum_{i=1}^{n} w_i \cdot \chi(\mathbf{X}, i) \tag{24}$$

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

$$\chi(\mathbf{X}, i) = \begin{cases} 1 & \text{if node } x_i \text{ is left uncolored because of a constraint violation} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

In this approach, originally introduced by Eiben et al. [39], 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 [11] and to a Grouping Genetic Algorithm [43].

Criticism

The obvious drawback of Bean and Hadj-Alouane' approach [7, 57] is how to choose the generational gap (i.e., the appropriate value of k) that provides reasonable information to guide the search, and more important, how do we define the values of β_1 and β_2 to penalyze fairly a given solution.

The most obvious drawback of Smith and Tate's [128] approach is how to choose NFT, since this parameter will be problem dependent. Coit and Smith [20] have proposed to define NFT as:

$$NFT = \frac{NFT_0}{1 + \lambda \times t} \tag{26}$$

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 [20]. Although Coit and Smith [20] have provided some alternatives for defining NFT, its value remains as an additional parameter to be determined by the user.

Additionally, the factor $B_{feasible} - B_{all}$ 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 [20] 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 they 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 [52] assigns a relatively mild penalty with respect to Coit et al. [21], but the authors of this method argue that their approach is problem-independent [52]. However, no information is provided by Gen and Cheng [52] 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 & van der Hauw [41] 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 use of adaptive penalties, therefore, remains to be extended to numerical optimization problems.

4.5 Self-Adaptive penalties

Michalewicz et al. [86] have recognized the importance of using adaptive penalties in evolutionary optimization, and considered this approach as a very promising direction of research on evolutionary optimization. Following this idea, Coello [19, 16] proposed the use of a penalty function of the form:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) - (coef \times w_1 + viol \times w_2) \tag{27}$$

where $f_i(\mathbf{X})$ is the value of the objective function for the given set of variable values encoded in the chromosome i; w_1 and w_2 are 2 penalty factors (considered as integers in this paper); $coef$ is the sum of all the amounts by which the constraints are violated (only inequality constraints were considered):

$$coef = \sum_{i=1}^{p} \phi_i(\mathbf{X}) \quad \forall \phi_i(\mathbf{X}) > 0 \tag{28}$$

$viol$ 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 GA has enough information not only about how many constraints were violated, but also about the amounts in which such constraints were violated. This follows Richardson's suggestion [115] about using penalties that are guided by the distance to feasibility.

Coello [19] used 2 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 genetic algorithm), and another to evolve the penalty factors w_1 and w_2. A graphical representation of this approach may be seen in Figure 1. Notice that 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 ($1 \leq j \leq M2$) 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 ($1 \leq k \leq M1$) is computed using Equation (27), keeping the penalty factors constant for every individual in the instance of $P1$ corresponding to the individual A_j being processed.

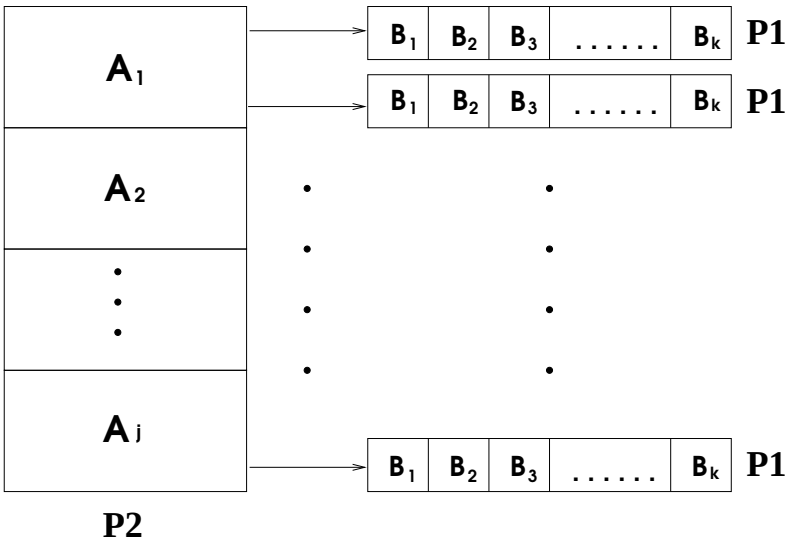

Figure 1: Graphical representation of approach to handle constraints proposed by Coello [19].

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:

$$average\_fitness_j = \sum_{i=1}^{M1} \left(\frac{\text{fitness}_i}{count\_feasible} \right) + count\_feasible \quad \forall \mathbf{X} \in \mathcal{F} \tag{29}$$

In Equation (29), 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 GA may bias the population towards regions of the search space where there are solutions with a very low weight combination (w_1 and w_2). 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 GA 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 GA gets trapped in local optima. However, such scaling factor is not very difficult to compute because Coello [19] assumes populations of constant size (such size must be defined before running the GA), and the range of the fitness values can be 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 GA from easily converging to a local optimum.

Criticism

The problem with this approach is that it introduces the definition of four additional parameters: $Gmax1$, $Gmax2$, $M1$ and $M2$. Coello [19, 18] argues that those parameters have to (empirically) determined for a GA 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 settle. 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.

4.6 Segregated genetic algorithm

Le Riche et al. [118] 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 individulas of the two populations interbreed, they are "segregated" in terms of satisfaction of a certain constraint.

The population is split into two cooperating groups, where individuals in each group are evaluated using either one of the two penalty parameters. Then, the best individuals of each group are chosen as parents for the next generation, which aims to combine feasible and infeasible individuals (if one of the penalties chosen is large and the other is small), and to help each other out of local minima.

Linear ranking was used to decrease the selection pressure that could cause premature convergence. This approach was used to solve a laminated design problem, providing excellent results [118].

Criticism

The problem with this approach is again the way of choosing the penalties for each of the 2 sub-populations, and even when some guidelines have been provided by the authors of this method [118] to define such penalties, 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.

4.7 Penalty function based on feasibility

Deb [31] proposed an interesting approach in which an individual is evaluated using:

$$\text{fitness}_i(\mathbf{X}) = \begin{cases} f_i(\mathbf{X}) & \text{if } \phi_j(\mathbf{X}) \geq 0, \quad \forall j = 1, 2, \ldots, m \\ f_{worst} + \sum_{j=1}^{m} \phi_j(\mathbf{X}) & \text{otherwise} \end{cases} \tag{30}$$

where f_{worst} is the objective function value of the worst feasible solution in the population, and $\phi_j(\mathbf{X})$ refers only to inequality constraints (equality constraints can be transformed to inequality constraints using a tolerance). If there are no feasible solutions in the population, then f_{worst} is set to zero.

Using binary tournament selection, Deb uses the following rules to compare two individuals [31]:

1. A feasible solution is always preferred over an infeasible one.
2. Between two feasible solutions, the one having better objective function 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.

Deb [31] used a real-coded GA with simulated binary crossover (SBX) [32] and a parameter-based mutation operator [34].

Criticism

Deb's results [31] are very encouraging, but the technique seems to have problems to maintain diversity in the population, and the use of niching methods [33] combined with higher than usual mutation rates is apparently necessary when using this approach.

4.8 Death penalty

The rejection of infeasible individuals (also called "death penalty") is probably the easiest and most efficient way to handle constraints, because when a certain solution violates a constraint, it is assigned a fitness of zero, and no further calculations are necessary to estimate the degree of infeasibility of the solution. The normal approach taken is to iterate recursively, generating a new point at each recursive call, until a feasible solution is found [62]. This might be a rather lengthy process in problems in which is very difficult to approach the feasible region.

An interesting death penalty approach has been used by Kuri [94]. Fitness of an individual is determined using:

$$\text{fitness}_i(\mathbf{X}) = \begin{cases} f(\mathbf{X}) & \text{if the solution is feasible} \\ K - \sum_{i=1}^{s} \left(\frac{K}{m}\right) & \text{otherwise} \end{cases} \tag{31}$$

where s is the number of constraints satisfied, and K is a large constant (it was set to 1×10^9 [93]in the experiments reported in [94]). Notice that when an individual is infeasible, its fitness is not computed (as in evolution strategies) 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.

Criticism

Death penalty is very popular within the evolution strategies community [123, 3], 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 evolutionary algorithm to guide the search. Michalewicz [84, 88, 89] has 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.

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 [115] about the definition of good penalty functions, apparently the self-adaptive GA 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 used, 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 the very similar or equal fitness), which seriously limits its applicability in highly constrained search spaces. It is interesting to mention that this is exactly the same problem that arises with death penalty, and that in some sense this sort of high penalty can be seen as a more elaborate form of death penalty, since infeasible individuals are not evaluated.

5 Special representations and operators

Some researchers have decided to develop special representation schemes to tackle a certain (particularly difficult) problem for which the conventional binary representation used in the GA 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 a certain solution at all times. The main application of this approach is naturally in problems in which it is extremelly difficult to locate at least a single feasible solution.

5.1 Davis' applications

Lawrence Davis' *Handbook of Genetic Algorithms* [29] contains several examples of GAs that use special representations and operators to solve complex real-world problems. For example, Yuval Davidor [26] (see also [25]) used a varying-length GA to generate robot trajectories, and defined a special crossover operator called *analogous crossover* [24], 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 the application presented by Davidor.

Other applications included in Davis' book are: schedule optimization [134], synthesis of neural networks architecture [61], and conformational analysis of DNA [80], among others.

Criticism

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.

5.2 GENOCOP

Another example of this approach is GENOCOP (GEnetic algorithm for Numerical Optimization for COnstrained Problems), developed by Michalewicz [83], which handles linear constraints by eliminating equalities and designing special genetic operators which guarantee to keep all chromosomes within the constrained solution space. Due to the complexity of this approach, the interested reader is referred to Michalewicz [83] for details.

Criticism

GENOCOP assumes a feasible starting point (or feasible initial population), which implies that the user must have a way of generating (in a reasonable time) such starting point. Also, since GENOCOP assumes the existence of only linear constraints, it is inherently restricted to convex search spaces [23].

5.3 Constraint Consistent GAs

Kowalczyk [73] proposed the use of constraint consistency [76] 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-coded GAs 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 fully aware that in many cases partially feasible solutions may be preferred because they contain better building blocks or because they are much easier to find.

Criticism

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

5.4 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* [54] and has been used in combinatorial and nonlinear optimization problems [55]. 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 [89].

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 [111, 83]: (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 [120], 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 [121] analyze in more detail the use of sphere operators in convex feasible search spaces.

Criticism

The main drawback of this approach is that the operators designed are either highly dependent on the chosen parameterization [120], or more complex calculations are required to perform crossover and mutation. Also, the use of such 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.

5.5 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 [23]. When using decoders, however, it is important that several conditions are satisfied [105]: (1) for each feasible solution s there is a decoded solution d, (2) each decoded solution d corresponds 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 [23].

Koziel and Michalewicz [74, 75] 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 GA.

Kim and Husbands [69, 70] 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 GA.

Criticism

Despite the several advantages of Koziel and Michalewicz's approach [75], it also has some disadvantages [75]:

- It uses an extra parameter v which has to be found empirically, performing a set of runs.
- It is expensive (computationally speaking), because of the sort of computations involved in the technique.

However, in the experiments reported by Koziel and Michalewicz [75], 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 [69, 70] could only be used with problems of low dimensionality (no more than 4 variables) and required the objective function to be given in explicit (algebraic) form. The mapping proposed by Koziel and Michalewicz [74, 75], however, can be used with problems of any dimensionality and does not require that the objective function is given in algebraic form.

5.6 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 et al. [78, 79] have shown, through an empirical test of GA 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 [88] also uses repair algorithms. The idea is to incorporate the original GENOCOP [87] system (which handles only linear constraints) and extend it by maintaining two separate populations, where results in one population influences 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 fully 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 et al. [90, 139, 138] used a repair algorithm to transform an infeasible path of a robot trying to move between two points in the presence of obstacles, such 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 [104], Mühlenbein [95], Le Riche and Haftka [117], and Tate and Smith [135].

There are no standard heuristics for the design of repair algorithms: normally, it is possible to use a greedy repair (a *greedy* algorithm is 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 repair or any other heuristic which would guide the repair process, and 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. Some researchers like Liepins et al. [78, 79] have taken the *never replacing* approach (that is, the repaired version is never returned to the population), while other authors such as Nakano [99] have taken the *always replacing* approach.

Orvosh and Davis [103, 104] reported a so-called 5% rule for combinatorial optimization problems, which means that genetic algorithms (applied to combinatorial optimization problems) with a repairing procedure provide the best result when 5% of the repaired chromosomes replace their infeasible originals. Michalewicz et al. [86] have reported, however, that a 15% replacement rule seems to be the best choice for numerical optimization problems with nonlinear constraints.

Criticism

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[1] and in some cases repair operators may severely disturb the good building blocks of the parent solutions carried in the children, harming the evolutionary process itself [127]. Furthermore, this approach is problem-dependant, since a specific repair algorithm has to be designed for each particular problem.

6 Separation of constraints and objectives

There are several approaches that handle constraints and objectives separately and we will review in this section some of the most representative proposals.

[1] In fact, in some cases the task of finding a feasible solution is by itself an NP-hard problem [127].

6.1 Co-evolution

Paredis [106] 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 [106].

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 (Paredis [106] 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.

Criticism

Paredis [106] indicated that his approach was similar to a self-adapting penalty function in which the relevance of a certain constraint can be changed over time, according to its difficulty. The results reported by Paredis [106] 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 invidual 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 effectivity of the approach in other combinatorial optimization problems, and apparently, it has not been extended to numerical optimization problems either.

6.2 Superiority of feasible points

Powell and Skolnick [110] incorporated a heuristic rule (suggested by Richardson et al. [115]) 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 [110]:

$$\text{fitness}_i(\mathbf{X}) = \begin{cases} f_i(\mathbf{X}) & \text{if feasible} \\ 1 + r \sum_{j=1}^{m} \phi(\mathbf{X}) & \text{otherwise} \end{cases} \tag{32}$$

$f_i(\mathbf{X})$ is scaled into the interval $(-\infty,1)$, $\phi(\mathbf{X})$ is scaled into the interval $(1, \infty)$, and r is a constant.

Powell and Skolnick [110] used linear ranking selection [5, 6, 29] 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.

Criticism

Although some might think that the definition of r 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 [110]. 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), the technique will fail unless a feasible point is introduced in the initial population [86].

6.3 Behavioral memory

Schoenauer and Xanthakis [122] proposed to extend a technique called *behavioral memory*, which was originally proposed for unconstrained optimization [30]. The main idea of this approach is that constraints are handled in a particular order. The algorithm is the following [122]:

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

$$\text{fitness}(\mathbf{X}) = M - \phi_1(\mathbf{X}) \tag{33}$$

  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 j-th constraint,

$$\text{fitness}(\mathbf{X}) = M - \phi_j(\mathbf{X}) \tag{34}$$

  During this phase, points that do not satisfy at least one of the 1st, 2nd, ... $(j-1)$-th constraints are eliminated from the population. The halting criterion is again the satisfaction of the j-th constraint by the flip threshold percentage 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. 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 its last step of the algorithm, Schoenauer and Xanthakis [122] use death penalty, because infeasible individuals are completely eliminated from the population.

Criticism

This method requires that there is a linear order of all constraints, and apparently, the order in which the constraints are processed influences the results provided by the algorithm (in terms of total running time and precision) [86].

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* [116, 118], it has a high computational cost (increased by the use of sharing to keep diversity in the population). As Schoenauer and Xanthakis [122] 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 Schoenauer and Xanthakis [122].

6.4 Multiobjective Optimization Techniques

The main idea is to redefine the single-objective optimization of f as a multiobjective optimization problem in which we will have $m+1$ objectives, where m is the number of constraints. Then, we can apply any multiobjective optimization technique [49] to the new vector $\bar{v} = (f, f_1, \ldots, f_m)$, where $f_1, \ldots, f_m$ are the original constraints of the problem. An ideal solution $\mathbf{X}$ would thus have $f_i(\mathbf{X})$=0 for $1 \leq i \leq m$ and $f(\mathbf{X}) \leq f(\mathbf{Y})$ for all feasible $\mathbf{Y}$ (assuming minimization).

Surry et al. [132] proposed the use of Pareto ranking [48] and VEGA [119] 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 [107] implemented a version of VEGA [119] that handled the constraints of a gas turbine problem as objectives to allow the GA 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 [107] opted to use specialized operators that would create a variable-size hypercube around each feasible point to help the GA to remain within the feasible region at all times.

Camponogara & Talukdar [12] 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:

$$\Phi(\mathbf{X}) = \sum_{i=1}^{k} max(0, \phi_i(\mathbf{X})) \tag{35}$$

Once the problem is redefined, non-dominated solutions with respect to the two new objectives are generated. The solutions found define a search direction $d = (x_i - x_j)/|x_i - xj|$, 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 [12]. 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 [12]. 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 [67] proposed the use of a min-max approach [14] 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 $\phi_j(\mathbf{X})$, for$j = 1, \ldots, m$). The individual with the lowest maximum violation wins.

Notice the great similarity between this approach and the technique proposed by Deb [31] that was described in a previous section. Jiménez and Verdegay [67] used a real-coded GA with uniform crossover [133], nonuniform mutation [83], and tournament selection.

Coello [17] proposed the use of a population-based multiobjective optimization technique such as VEGA [119] to handle each of the constraints of a single-objective optimization problem as an objective. The technique may be better illustrated by Figure 2. At each generation, the population is split into $m + 1$ sub-populations, where m refers to the number of constraints of the problem (we have to add one to consider also the objective function).

Using this scheme, a fraction of the population will be selected using the (unconstrained) objective function as its fitness; another fraction will use 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 $\mathbf{X}$ is used directly as the fitness function (probably multiplied by (-1) if it is a minimization problem), with no penalties of any sort. For all the other sub-populations, the algorithm used was the following [17]:

if $\phi_j(\mathbf{X}) < 0.0$	**then**	fitness $= \phi_j(\mathbf{X})$
else if $v \neq 0$	**then**	fitness $= -v$
else		fitness $= f$

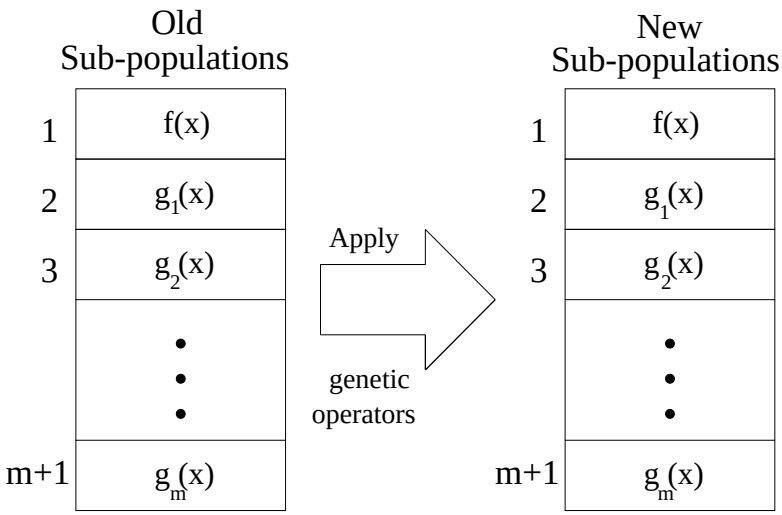

Figure 2: Graphical representation of the approach introduced in this paper.

where $\phi_j(\mathbf{X})$ 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), and v refers to the number of constraints that are violated ($\leq m$).

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 GA 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, because the number of constraints will normally be greater than one, the other sub-populations will drive the GA 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, the building blocks of these sub-populations 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 GA 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.

Criticism

COMOGA compared fairly with a penalty-based approach in a pipe-sizing problem, since the resulting GA was less sensitive to changes in the parameters, but the results achieved were not better than those found with a penalty function [132]. It should be added that COMOGA [132] required several extra parameters, from which the so-called p_{cost} was the most important (this parameter regulates the proportion of feasible and infeasible individuals that will exist in the population at any given time). Also, due to the use of dominance to impose an order on the constraints based on their violation, COMOGA involves the use of a more expensive process (in terms of CPU

time).

Parmee and Purchase's [107] approach was developed for a heavily constrained search space and it proved to be appropriate to reach the feasible region. However, this application of a multiobjective 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.

Camponogara & Talukdar's approach [12] has obvious problems to keep diversity (a common problem with using evolutionary multiobjective optimization techniques), as it 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 and it is not clear what is the impact of the segment chosen to search in the overall performance of the algorithm.

Jiménez and Verdegay's approach [67] can hardly be said to be using a multiobjective 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 [132]. 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 Parmee and Purchase's [107] technique) may be a good alternative to find a feasible point in a heavily constrained search space.

The main drawback of Coello's approach [17] may be 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 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 multiobjective 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 [119, 49]). 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 completely feasible, instead of good compromises that may not satisfy one of the constraints.

7 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 evolutionary algorithm.

7.1 Lagrangian multipliers

Adeli and Cheng [1] proposed a hybrid GA 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:

$$\text{fitness}_i = f_i(\mathbf{X}) + \frac{1}{2}\sum_{j=1}^{m} \gamma_j \left\{[\phi_j(\mathbf{X}) + \mu_j]^+\right\}^2 \tag{36}$$

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

$$[\phi_j(\mathbf{X}) + \mu_j]^+ = max[0, \phi_j(\mathbf{X}) + \mu_j] \tag{37}$$

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 which is defined by the user and has to be > 1. 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 as to ensure that the penalty is increased over generations.

Kim and Myung [71, 97] 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* [98, 96], 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:

$$\text{fitness}_i(\mathbf{X}) = f_i(\mathbf{X}) + \frac{C}{2}\left(\sum_{i=1}^{m} \phi_j^2(\mathbf{X})\right) \tag{38}$$

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 [81] 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].

Criticism

Adeli and Cheng's technique [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.

The main drawback of Kim and Myung;'s approach [71, 97] 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.

7.2 Constrained optimization by random evolution

Belur [8] 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 [100], but any other similar technique should work as well). Whenever a solution is not feasible, the following constraint functional is minimized:

$$C(\mathbf{X}) = \sum_{i \in C_1} h_i^2(\mathbf{X}) - \sum_{j \in C_2} g_j(\mathbf{X}) \tag{39}$$

where

$$C_1 = \{i = 1, \ldots, m / |h_i(\mathbf{X})| > \varepsilon_c\} \tag{40}$$

$$C_2 = \{j = 1, \ldots, q / g_j(\mathbf{X}) < 0\} \tag{41}$$

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

Criticism

This minimization process can be seen as a repair algorithm for numerical optimization, which implies that this technique has the same problems of this technique as we have seen in a previous section.

7.3 Fuzzy logic

T. Van Le [77] proposed the use of a combination of fuzzy logic and evolutionary programming to handle constraints. The main idea was to replace constraints of the form $\phi_i(\mathbf{X}) \leq b_i$ by a set of fuzzy constraints $C_1, \ldots, C_m$, $i = 1, \ldots, m$ defined as:

$$\mu C_i(\mathbf{X}) = \mu_{\sigma(b_i,\epsilon_i)}(\phi_i(\mathbf{X})), \quad i = 1, \ldots, m \tag{42}$$

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

$$\mu_{\sigma(a,s)}(\mathbf{X}) = \begin{cases} 1 & \text{if } x \leq a, \\ \frac{e^{-p\left(\frac{x-a}{s}\right)^2} - e^{-p}}{1-e^{-p}} & \text{if } a < x \leq a+s \\ 0 & \text{if } x > a+s \end{cases} \tag{43}$$

The rationale behind this fuzzification process is to allow a higher degree of tolerance if $\phi_i(\mathbf{X})$ 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:

$$F(\mathbf{X}) = f(\mathbf{X}) \times \min(\mu_{C_1}(\mathbf{X}), \ldots, \mu_{C_m}(\mathbf{X})) \tag{44}$$

Criticism

This approach seems very promising, but Van Le [77] provides very little empirical evidence of its performance, although this is certainly a research path that is worth exploring.

8 Novel approaches

There are some other proposals that do not quite fit into any of the previous categories and that are, therefore, considered separately.

8.1 Immune system

Forrest and Perelson [50] and Smith et al. [129, 130] 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 el al. [129, 128] proposed this approach as a way to keep diversity in multimodal optimization problems, Hajela and Lee [58, 59] extended it to handle constraints.

The idea of Hajela and Lee's technique is to separate any feasible individuals in a population (the antigens) from those that are infeasible (the antibodies). By using a simple matching function that computes the similarity (on a bit-per-bit basis, assuming binary encoding) between the two chromosomes, this approach co-evolves the population of antibodies until they become sufficiently similar to their antigens by maximizing the degree of matching between the antigens and the antibodies. Then, the two populations are mixed and evolved using a standard genetic algorithm, but without the use of a penalty function, since all the individuals are feasible at that point, and the population will be re-filled at certain intervals, to eliminate any infeasible individuals generated by the genetic operators.

A simpler instance of this technique, called *expression strategies* was proposed by Hajela and Yoo [60]. In this case, feasible and infeasible individuals are combined using uniform crossover [133] in such a way that their chromosomic material is exchanged.

It is worth mentioning that Hajela and Yoo [60] proposed the use of the Kreisselmeir-Steinhauser function [131] to handle equality constraints. The idea is that if h_i is the ith equality constraint, then it can be represented by a pair of inequality constraints as:

$$h_i \leq 0 \qquad -h_i \leq 0 \tag{45}$$

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

$$\Omega = (1/\rho)\ln(e^{\rho h_i} + e^{-\rho h_i}) - (1/\rho)\ln 2 + c_1 \tag{46}$$

where c_1 represents the width of a band that replaces the original strict equaliy constraint, and ρ is a user-defined constant. By reducing c_1 the solutions are forced to move closer to the equality constraint. This approach is just a way of converting equality constraints into inequality constraints which should be easier to satisfy.

Criticism

Since the bit matching process 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), nor how to proceed when there are no feasible solutions in the initial population.

8.2 Cultural algorithms

Some social researchers have suggested that culture might be symbolically encoded and transmitted within and between populations, as another inheritance mechanism [38, 112]. Using this idea, Reynolds [113] 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" [112], 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 [113] proposed the use of GAs to model the micro-evolutionary process, and Version Spaces [92] 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 [85]. Therefore, constraints can influence directly the search process, leading to an efficient optimization process. In fact, Reynolds et al. [114] and Chung & Reynolds [15] have explored this area of research with very encouraging results in numerical optimization.

The approach taken by Chung and Reynolds [15] was to use a hybrid of evolutionary programming and GENOCOP [87] in which they incorporated an interval constraint-network [27, 66] 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 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.

Criticism

Although interesting, the extension of the cultural operators to handle non-linear constraints is not obvious, since finding the boundary between the feasible and infeasible regions is a much harder problem (we have discussed previously this problem with repair algorithms in general). However, this is an interesting application of domain knowledge to handle constraints that turns out to be relatively general in scope (within numerical optimization, in this case).

8.3 Ant colony optimization

This technique was proposed by Dorigo et al. [22, 37, 36, 35] 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* [35]. 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 [9, 137] 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 [10] 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 [9, 10] proposed to make a food source "unacceptable" in case it violates 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. To make this model effective, 3 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 in 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).

In their model, Bilchev & Parmee [9, 10] used a real-coded GA with arithmetic crossover and non-uniform mutation [83]. Their results were very encouraging and showed clearly the high potential of this technique in multi-modal and/or heavily constrained search spaces.

Criticism

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 [9] suggest the use of a niching GA), 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. Also, the use of non-uniform mutation introduces the need to define a parameter b that indicates the degree of non-uniformity of the mutations performed.

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.

9 Future research paths

The most important issues that remain to be solved regarding constraint handling techniques for evolutionary algorithms are the following:

- **Efficiency**. Any constraint-handling technique proposed must deal with efficiency issues, since many real-world applications have fitness functions whose evaluation cost is very high (computationally speaking).
- **Generality**. Ideally, constraint-handling techniques should be as general as possible, requiring only minor (or no) modification to work in different problems.
- **Ease of use**. Most of the current approaches to handle constraints in evolutionary algorithms require the fine tuning of some parameters. This makes their use highly dependant on the setting of these parameters and increases the (already tedious) parameter tuning process (i.e., population size, crossover and mutation rates, etc.) normally required with evolutionary algorithms (particularly with genetic algorithms).

- **Limitations of a technique**. Since it is utopical to consider that a single constraint-handling technique will be able to deal with any sort of constraints efficiently, then it should be clear when it is possible to use it. Michalewicz and Schoenauer [89] discussed this issue, but the question remains open regarding the characteristics that we could use from a problem to decide what technique to use.

- **Comparisons of approaches**: Michalewicz [89, 83, 84, 85] has performed several (partial) comparisons of constraint-handling approaches, but more work in this area must be done. 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.

- **Test suites**: If it is necessary to compare more approaches, it is equally important to have good test suites publicly available. Regarding this issue, there is some literature that can be used (see for example[2] [44, 84]). Chung and Reynolds [15] have provided a test suite for cultural algorithms, and Michalewicz and Schoenauer [89] make reference to the construction of another (more general) test suite, but any other efforts in this direction will certainly be useful as well.

- **Incorporation of knowledge about the domain**. Even when it is well known that incorporating knowledge about an specific domain will reduce the generality of an evolutionary approach, in highly complex problems (e.g., heavily constrained search spaces) such knowledge can bring great benefits in terms of performance, and it is desirable that a new constraint-handling approach has the capability to incorporate efficiently such domain knowledge whenever available.

10 Conclusions

In this paper we have given a very comprehensive review of the most important constraint-handling techniques developed for evolutionary algorithms. The spectrum covered is quite wide, and it includes approaches that go from several variations of a simple penalty function to sophisticated mathematical techniques that aim at finding the boundary between the feasible and the infeasible region, and biologically inspired techniques that emulate the behavior of the immune system, culture, or even ant colonies. In the final section, we have provided some promising research paths (in the author's opinion) that may be worth exploring.

Acknowledgments

The author acknowledges support from CONACyT through project number I-29870 A.

References

[1] Hojjat Adeli and Nai-Tsang Cheng. Augmented lagrangian genetic algorithm for structural optimization. *Journal of Aerospace Engineering*, 7(1):104–18, jan 1994.

[2] Thomas Bäck, editor. *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, California, July 1997.

[3] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A Survey of Evolution Strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[4] Thomas Bäck and Sami Khuri. An Evolutionary Heuristic for the Maximum Independent Set Problem. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano, editors, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 531–535, Piscataway, New Jersey, 1994. IEEE Press.

[2] The web page `http://solon.cma.univie.ac.at/ neum/glopt/test.html` also contains test problems from constrained optimization.

[5] James Edward Baker. Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 100–111, Hillsdale, New Jersey, 1985. Lawrence Erlbaum.

[6] James Edward Baker. *An Analysis of the Effects of Selection in Genetic Algorithms.* PhD thesis, Vanderbilt University, Nashville, Tennessee, 1989.

[7] J. C. Bean and 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.

[8] Sheela V. Belur. CORE: Constrained Optimization by Random Evolution. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 280–286, Stanford University, California, July 1997. Stanford Bookstore.

[9] George Bilchev and Ian C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In Terence C. Fogarty, editor, *Evolutionary Computing*, pages 25–39. Springer VErlag, Sheffield, UK, April 1995.

[10] George Bilchev and Ian C. Parmee. Constrained and Multi-Modal Optimisation with an Ant Colony Search Model. In Ian C. Parmee and M. J. Denham, editors, *Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control.* University of Plymouth, Plymouth, UK, March 1996.

[11] D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[12] Eduardo Camponogara and Sarosh N. Talukdar. A Genetic Algorithm for Constrained and Multiobjective Optimization. In Jarmo T. Alander, editor, *3rd Nordic Workshop on Genetic Algorithms and Their Applications (3NWGA)*, pages 49–62, Vaasa, Finland, August 1997. University of Vaasa.

[13] Susan E. Carlson. A General Method for Handling Constraints in Genetic Algorithms. In *Proceedings of the Second Annual Joint Conference on Information Science*, pages 663–667, 1995.

[14] V. Chankong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology.* Systems Science and Engineering. North-Holland, 1983.

[15] Chan-Jin Chung and Robert G. Reynolds. A Testbed for Solving Optimization Problems using Cultural Algorithms. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, Cambridge, Massachusetts, 1996. MIT Press.

[16] Carlos A. Coello Coello. Self-Adaptive Penalties for GA-based Optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999. IEEE.

[17] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32, 1999. (Accepted for publication).

[18] Carlos A. Coello Coello. The use of a multiobjective optimization technique to handle constraints. In Alberto A. Ochoa Rodríguez, Marta R. Soto Ortiz, and Roberto Santana Hermida, editors, *Proceedings of the Second International Symposium on Artificial Intelligence (Adaptive Systems)*, pages 251–256, La Habana, Cuba, July 1999. Institute of Cybernetics, Mathematics and Physics, Ministry of Science, Technology and Environment.

[19] Carlos A. Coello Coello. Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, 1999. (Accepted for publication).

[20] David W. Coit and Alice E. Smith. Penalty guided genetic search for reliability design optimization. *Computers and Industrial Engineering*, 30(4):895–904, September 1996. Special Issue on Genetic Algorithms.

[21] David W. Coit, Alice E. Smith, and David M. Tate. Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems. *INFORMS Journal on Computing*, 8(2):173–182, Spring 1996.

[22] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In P. Bourgine and F. Varela, editors, *Proceedings of the First European Conference on Artificial Life*, Cambridge, Massachusetts, 1991. MIT Press/Bradford Books.

[23] Dipankar Dasgupta and Zbigniew Michalewicz, editors. *Evolutionary Algorithms in Engineering Applications.* Springer-Verlag, Berlin, 1997.

[24] Yuval Davidor. Analogous Crossover. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 98–103, San Mateo, California, 1989. Morgan Kaufmann Publishers.

[25] Yuval Davidor. *Genetic Algorithms and Robotics : A Heuristic Strategy for Optimization.* World Scientific Publishing Co., Singapore, 1990.

[26] Yuval Davidor. A Genetic Algorithm Applied To Robot Trajectory Generation. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 12, pages 144–165. Van Nostrand Reinhold, New York, New York, 1991.

[27] Ernest Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32:281–331, 1987.

[28] Lawrence Davis. *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[29] Lawrence Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, New York, 1991.

[30] Hugo de Garis. Genetic Programming: Building Artificial Nervous Systems using Genetically Programmed Neural Networks Modules. In R. Porter and B. Mooney, editors, *Proceedings of the 7th International Conference on Machine Learning*, pages 132–139. Morgan Kaufmann, 1990.

[31] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 1999. (in Press).

[32] Kalyanmoy Deb and R. W. Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9:115–148, 1995.

[33] Kalyanmoy Deb and David E. Goldberg. An investigation of niche and species formation in genetic function optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, jun 1989. George Mason University, Morgan Kaufmann Publishers.

[34] Kalyanmoy Deb and Mayank Goyal. A Combined Genetic Adaptive Search GeneAS for Engineering Design. *Computer Science and Informatics*, 26(4):30–45, 1996.

[35] M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization.* McGraw-Hill, 1989.

[36] M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[37] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26(1):29–41, 1996.

[38] W. H. Durham. *Co-evolution: Genes, Culture, and Human Diversity.* Stanford University Press, Stanford, California, 1994.

[39] A. E. Eiben, P.-E. Raué, and Zs. Ruttkay. GA-easy and GA-hard constraint satisfaction problems. In M. Meyer, editor, *Proceedings of the ECAI'94 Workshop on Constraint Processing*, pages 267–284. Springer-Verlag, 1995.

[40] A. E. Eiben and Zs. Ruttkay. Self-adaptivity for Constraint Satisfaction: Learning Penalty Functions. In *Proceedings of the 3rd IEEE Conference on Evolutionary Computation*, pages 258–261, Piscataway, New Jersey, 1996. IEEE Service Center.

[41] A. E. Eiben and J. K. van der Hauw. Adaptive penalties for evolutionary graph coloring. In *Artificial Evolution'97*, pages 95–106, Berlin, 1998. Springer-Verlag.

[42] A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert. Graph Coloring with Adaptive Evolutionary Algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[43] E. Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[44] C. A. Floudas and P. M. Pardalos. *A Collection of Test PRoblems for Constrained Global Optimization Algorithms*. Number 455 in Lecture Notes in Computer Science. Springer-Verlag, 1990.

[45] David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Engineers, New York, 1995.

[46] David B. Fogel, editor. *Evolutionary Computation. The Fossil Record. Selected Readings on the History of Evolutionary Algorithms*. The Institute of Electrical and Electronic Engineers, New York, 1998.

[47] Lawrence J. Fogel. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[48] Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[49] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[50] Stephanie Forrest and Alan S. Perelson. Genetic algorithms and the immune system. In Hans-Paul Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, pages 320–325. Springer-Verlag, Berlin, Germany, 1991.

[51] Mitsuo Gen and Runwei Cheng. Interval Programming using Genetic Algorithms. In *Proceedings of the Sixth International Symposium on Robotics and Manufacturing*, Montpellier, France, 1996.

[52] Mitsuo Gen and Runwei Cheng. A Survey of Penalty Techniques in Genetic Algorithms. In Toshio Fukuda and Takeshi Furuhashi, editors, *Proceedings of the 1996 International Conference on Evolutionary Computation*, pages 804–809, Nagoya, Japan, 1996. IEEE.

[53] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms & Engineering Design*. John Wiley & Sons, Inc, New York, 1997.

[54] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[55] Fred Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Proceedings of the International Conference on Metaheuristics for Optimization*, pages 113–133, Dordrecht, The Netherlands, 1995. Kluwer Publishing.

[56] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[57] A. B. Hadj-Alouane and J. C. Bean. A Genetic Algorithm for the Multiple-Choice Integer Program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

[58] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. In Niels Olhoff and George I. N. Rozvany, editors, *Proceedings of the First World Congress of Stuctural and Multidisciplinary Optimization*, pages 915–920, Goslar, Germany, 1995. Pergamon.

[59] P. Hajela and J. Lee. Constrained Genetic Search via Schema Adaptation. An Immune Network Solution. *Structural Optimization*, 12:11–15, 1996.

[60] P. Hajela and J. Yoo. Constraint Handling in Genetic Search Using Expression Strategies. *AIAA Journal*, 34(12):2414–2420, 1996.

[61] Steven A. Harp and Tariq Samad. Genetic Synthesis of Neural Network Architecture. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 15, pages 202–221. Van Nostrand Reinhold, New York, New York, 1991.

[62] Frank Hoffmeister and Joachim Sprave. Problem-independent handling of constraints by use of metric penalty functions. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, pages 289–294, San Diego, California, February 1996. The MIT Press.

[63] John H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Harbor : University of Michigan Press, 1975.

[64] A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained Optimization via Genetic Algorithms. *Simulation*, 62(4):242–254, 1994.

[65] W.-C. Huang, C.-Y. Kao, and J.-T. Horng. A Genetic Algorithm Approach for Set Covering Problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 569–573. IEEE Press, 1994.

[66] E. Hyvoenen. Constraint reasoning based on interval arithmetic—The tolerance propagation approach. *Artificial Intelligence*, 58:71–112, 1992.

[67] Fernando Jiménez and José L. Verdegay. Evolutionary techniques for constrained optimization problems. In *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99)*, Aachen, Germany, 1999. Springer-Verlag.

[68] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In David Fogel, editor, *Proceedings of the first IEEE Conference on Evolutionary Computation*, pages 579–584, Orlando, Florida, 1994. IEEE Press.

[69] Dae Gyu Kim and Phil Husbands. Riemann Mapping Constraint Handling Method for Genetic Algorithms. Technical Report CSRP 469, COGS, University of Sussex, UK, 1997.

[70] Dae Gyu Kim and Phil Husbands. Mapping Based Constraint Handling for Evolutionary Search; Thurston's Circle Packing and Grid Generation. In Ian Parmee, editor, *The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation*, pages 161–173. Springer-Verlag, Plymouth, United Kingdom, April 1998.

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

[72] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[73] Ryszard Kowalczyk. Constraint Consistent Genetic Algorithms. In *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*, pages 343–348, Indianapolis, USA, April 1997. IEEE.

[74] Slawomir Koziel and Zbigniew Michalewicz. A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In T. Bäck, A. E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V)*, pages 231–240, Amsterdam, September 1998. Springer-Verlag.

[75] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[76] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, pages 32–44, Spring 1992.

[77] T. Van Le. A Fuzzy Evolutionary Approach to Constrained Optimization Problems. In *Proceedings of the Second IEEE Conference on Evolutionary Computation*, pages 274–278, Perth, November 1995. IEEE.

[78] G. E. Liepins and Michael D. Vose. Representational Issues in Genetic Optimization. *Journal of Experimental and Theoretical Computer Science*, 2(2):4–30, 1990.

[79] Gunar E. Liepins and W. D. Potter. A Genetic Algorithm Approach to Multiple-Fault Diagnosis. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 17, pages 237–250. Van Nostrand Reinhold, New York, New York, 1991.

[80] C. B. Lucasius, M. J. J. Blommers, L. M. C. Buydens, and G. Kateman. A Genetic Algorithm for Conformational Analysis of DNA. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 18, pages 251–281. Van Nostrand Reinhold, New York, New York, 1991.

[81] C. Maa and M. Shanblatt. A two-phase optimization neural network. *IEEE Transactions on Neural Networks*, 3(6):1003–1009, 1992.

[82] Z. Michalewicz and N. Attia. Evolutionary Optimization of Constrained Problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108. World Scientific, 1994.

[83] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, second edition, 1992.

[84] Zbigniew Michalewicz. Genetic Algorithms, Numerical Optimization, and Constraints. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 151–158, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[85] Zbigniew Michalewicz. A Survey of Constraint Handling Techniques in Evolutionary Computation Methods. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155. The MIT Press, Cambridge, Massachusetts, 1995.

[86] Zbigniew Michalewicz, Dipankar Dasgupta, R. Le Riche, and Marc Schoenauer. Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(4):851–870, September 1996.

[87] Zbigniew Michalewicz and Cezary Z. Janikow. Handling Constraints in Genetic Algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[88] Zbigniew Michalewicz and G. Nazhiyath. Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints. In David B. Fogel, editor, *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 647–651, Piscataway, New Jersey, 1995. IEEE Press.

[89] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[90] Zbigniew Michalewicz and Jing Xiao. Evaluation of Paths in Evolutionary Planner/Navigator. In *Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems*, pages 45–52, Tokyo, Japan, May 1995.

[91] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts, 1996.

[92] Tom Mitchell. *Version Spaces: An Approach to Concept Learning.* PhD thesis, Computer Science Department, Stanford University, Stanford, California, 1978.

[93] Angel Fernando Kuri Morales. Personal Communication, 1999.

[94] Angel Kuri Morales and Carlos Villegas Quezada. A Univeral Eclectic Genetic Algorithm for Constrained Optimization. In *Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EUFIT'98*, pages 518–522, Aachen, Germany, September 1998. Verlag Mainz.

[95] Heinz Mühlenbein. Parallel Genetic Algorithms in Combinatorial Optimization. In O. Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research*, pages 441–456. Pergamon Press, New York, 1992.

[96] H. Myung and J.-H. Kim. Evolian: Evolutionary optimization based on Lagrangian with constraint scaling. In P.J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Proceedings of the Sixth Anual Conference on Evolutionary Programming*, pages 177–188, Indianapolis, April 1997. Springer-Verlag.

[97] H. Myung and J.-H. Kim. Hybrid Interior-Lagrangian Penalty Based Evolutionary Optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, editors, *Proceedings of the Seventh Anual Conference on Evolutionary Programming*, pages 85–94. Springer-Verlag, 1998.

[98] H. Myung, J.-H. Kim, and D. B. Fogel. Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 449–463, Cambridge, Massachusetts, 1995. MIT Press.

[99] Ryohei Nakano. Conventional Genetic Algorithm for Job Shop Problems. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 474–479, San Mateo, California, 1991. Morgan Kaufmann Publishers.

[100] A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7:308–313, 1965.

[101] Bryan A. Normal and Alice E. Smith. Random Keys Genetic Algorithm with Adaptive Penalty Function for Optimization of Constrained Facility Layout Problems. In Thomas Bäck, Zbigniew Michalewicz, and Xin Yao, editors, *Proceedings of the 1997 International Conference on Evolutionary Computation*, pages 407–411, Indianapolis, Indiana, 1997. IEEE.

[102] A. L. Olsen. Penalty Functions for the Knapsack Problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 554–558. IEEE Press, 1994.

[103] David Orvosh and Lawrence Davis. Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, page 650. Morgan Kauffman Publishers, San Mateo, California, July 1993.

[104] David Orvosh and Lawrence Davis. Using a Genetic Algorithm to Optimize Problems with Feasibility Contraints. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 548–553. IEEE Press, 1994.

[105] Charles C. Palmer and Aaron Kershenbaum. Representing Trees in Genetic Algorithms. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano, editors, *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 379–384, Piscataway, New Jersey, 1994. IEEE Press.

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

[107] I. C. Parmee and G. Purchase. The development of a directed genetic search technique for heavily constrained design spaces. In I. C. Parmee, editor, *Adaptive Computing in Engineering Design and Control-'94*, pages 97–102, Plymouth, UK, 1994. University of Plymouth, University of Plymouth.

[108] Ian Parmee, editor. *The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation.* Springer-Verlag, Plymouth, United Kingdom, 1998.

[109] V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors. *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science.* Springer-Verlag, San Diego, California, March 1998.

[110] David Powell and Michael M. Skolnick. Using genetic algorithms in engineering design optimization with nonlinear constraints. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, San Mateo, California, jul 1993. University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers.

[111] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–220, 1991.

[112] A. C. Renfrew. Dynamic Modeling in Archaeology: What, When, and Where? In S. E. van der Leeuw, editor, *Dynamical Modeling and the Study of Change in Archaeology.* Edinburgh University Press, Edinburgh, Scotland, 1994.

[113] Robert G. Reynolds. An Introduction to Cultural Algorithms. In A. V. Sebald, , and L. J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 131–139. World Scientific, River Edge, New Jersey, 1994.

[114] Robert G. Reynolds, Zbigniew Michalewicz, and M. Cavaretta. Using cultural algorithms for constraint handling in GENOCOP. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 298–305. MIT Press, Cambridge, Massachusetts, 1995.

[115] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, George Mason University, 1989. Morgan Kaufmann Publishers.

[116] Rodolphe G. Le Riche and Raphael T. Haftka. Optimization of Laminate Stacking Sequence for Buckling Load Maximization by Genetic Algorithm. *AIAA Journal*, 31(5):951–970, 1993.

[117] Rodolphe G. Le Riche and Raphael T. Haftka. Improved Genetic Algorithm for Minimum Thickness Composite Laminate Design. *Composites Engineering*, 3(1):121–139, 1994.

[118] Rodolphe G. Le Riche, Catherine Knopf-Lenoir, and Raphael T. Haftka. A Segregated Genetic Algorithm for Constrained Structural Optimization. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 558–565, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[119] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[120] Marc Schoenauer and Zbigniew Michalewicz. Evolutionary Computation at the Edge of Feasibility. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Proceedings of the Fourth Conference on Parallel Problem Solving from Nature*, pages 245–254. Springer-Verlag, Berlin, September 1996.

[121] Marc Schoenauer and Zbigniew Michalewicz. Sphere Operators and Their Applicability for Constrained Parameter Optimization Problems. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming*, volume 1447 of *Lecture Notes in Computer Science*, pages 241–250. Springer-Verlag, San Diego, California, March 1998.

[122] Marc Schoenauer and Spyros Xanthakis. Constrained GA Optimization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580. Morgan Kauffman Publishers, San Mateo, California, July 1993.

[123] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* John Wiley & Sons, Great Britain, 1981.

[124] Hans-Paul Schwefel. *Evolution and Optimum Seeking.* John Wiley & Sons, New York, 1995.

[125] W. Siedlecki and J. Sklanski. Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 141–150, San Mateo, California, jun 1989. George Mason University, Morgan Kaufmann Publishers.

[126] Susan Carlson Skalak, Ron Shonkwiler, Sani Babar, and M. Aral. Annealing a Genetic Algorithm over Constraints. Available at `http://vlead.mech.virginia.edu/publications/shenkpaper/shenkpaper.html`.

[127] Alice E. Smith and David W. Coit. Constraint Handling Techniques—Penalty Functions. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter C 5.2. Oxford University Press and Institute of Physics Publishing, 1997.

[128] Alice E. Smith and David M. Tate. Genetic Optimization Using a Penalty Function. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 499–503, San Mateo, California, July 1993. University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers.

[129] Robert E. Smith, Stephanie Forrest, and Alan S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. Technical Report TCGA No. 92002, University of Alabama, Tuscaloosa, Alabama, 1992.

[130] Robert E. Smith, Stephanie Forrest, and Alan S. Perelson. Population Diversity in an Immune System Model: Implications for Genetic Search. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 153–165. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[131] J. Sobieszanski-Sobieski. A Technique for Locating Function Roots and for Satisfying Equality Constraints in Optimization. *Structural Optimization*, 4(3–4):241–243, 1992.

[132] Patrick D. Surry, Nicholas J. Radcliffe, and Ian D. Boyd. A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Terence C. Fogarty, editor, *Evolutionary Computing. AISB Workshop. Selected Papers*, Lecture Notes in Computer Science, pages 166–180. Springer-Verlag, Sheffield, U.K., 1995.

[133] Gilbert Syswerda. Uniform Crossover in Genetic Algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, California, jun 1989. George Mason University, Morgan Kaufmann Publishers.

[134] Gilbert Syswerda. Schedule Optimization Using Genetic Algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 21, pages 332–349. Van Nostrand Reinhold, New York, New York, 1991.

[135] David M. Tate and Alice E. Smith. A Genetic Approach to the Quadratic Assignment Problem. *Computers and Operations Reseach*, 22(1):73–78, 1995.

[136] Sam R. Thangiah. An Adaptive Clustering Method using a Geometric Shape for Vehicle Routing Problems with Time Windows. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 536–543, San Mateo, California, July 1995. University of Pittsburgh, Morgan Kaufmann Publishers.

[137] M. Wodrich and G. Bilchev. Cooperative Distributed Search: The Ant's Way. *Control and Cybernetics*, 26(3):413–446, 1997.

[138] Jing Xiao, Zbigniew Michalewicz, and Krzysztof Trojanowski. Adaptive Evolutionary Planner/Navigator for Mobile Robots. *IEEE Transactions on Evolutionary Computation*, 1(1):18–28, 1997.

[139] Jing Xiao, Zbigniew Michalewicz, and Lixin Zhang. Evolutionary Planner/Navigator: Operator Performance and Self-Tuning. In *Proceedings of the 3rd IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, May 1996. IEEE Press.

[140] T. Yokota, M. Gen, K. Ida, and T. Taguchi. Optimal Design of System Reliability by an Improved Genetic Algorithm. *Transactions of Institute of Electronics, Information and Computer Engineering*, J78-A(6):702–709, 1995. (In Japanese).