Constraint-Handling using an Evolutionary Multiobjective Optimization Technique

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

Abstract

In this paper, we introduce the concept of non-dominance (commonly used in multiobjective optimization) as a way to incorporate constraints into the fitness function of a genetic algorithm. Each individual is assigned a rank based on its degree of dominance over the rest of the population. Feasible individuals are always ranked higher than infeasible ones, and the degree of constraint violation determines the rank among infeasible individuals. The proposed technique does not require fine tuning of factors like the traditional penalty function and uses a self-adaptation mechanism that avoids the traditional empirical adjustment of the main genetic operators (i.e., crossover and mutation).

Keywords: genetic algorithms, constraint handling, multiobjective optimization, self-adaptation, evolutionary optimization, numerical optimization.

1 Introduction

Despite the wide success of genetic algorithms (GAs) in a wide range of applications [25, 3, 36, 34], their use in constrained optimization requires the incorporation of constraints of any sort (linear, non-linear, equality or inequality) into the fitness function as to guide the search properly. The approach most commonly used to incorporate constraints is the penalty function, and there have been many successful applications of this approach reported in the literature [1, 15, 35, 31, 12]. However, penalty functions have some well-known limitations [41], from which the most remarkable is the difficulty to define good penalty factors. These penalty factors are normally generated by trial and error, although their definition may severely affect the results produced by the GA [41].

In this paper, we propose a new constraint-handling approach that guides the search of a GA using the concept of non-dominance (used in multiobjective optimization). Additionally, we introduce a simple self-adaptation mechanism that allows the GA to find its own crossover and mutation rates without the need of any human intervention. This aims at reducing the burden that is normally associated with the use of this heuristic, since it is traditionally required to perform several experiments to fine-tune its parameters.

The remainder of this paper is organized as follows: first, a short review of GAs in general is provided. Then, we review some of the previous (related) work in this area before introducing our own approach. Then, we describe the representation and genetic operators used by our approach, and the self-adaptation mechanism proposed. After that, we compare the results produced by our approach with those reported in the literature for several optimization problems that have been previously solved with GA-based and mathematical programming techniques. Finally, we discuss the results obtained and draw some possible paths of future research.

2 Genetic Algorithms

The optimization capabilities of evolutionary techniques (i.e., techniques based on the natural selection principle) within a wide variety of domains have been recognized over the years, and have received much attention from scientists working in many different disciplines. Perhaps the most widely used technique is the genetic algorithm (GA) [25, 36, 34]. Being a stochastic, heuristic technique, the GA does not need specific information to guide the search. Its structure is analogous to biological evolution theory using the principle of survival of the fittest [27]. Therefore, the GA can be seen as a "black box" that can be attached to any particular application.

In general, we need the following basic components to implement a GA in order to solve a problem [34]:

1. A representation for potential solutions to the problem.

2. A way to create an initial population of potential solutions (this is normally done randomly, but deterministic approaches can also be used).

3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness".

4. Genetic operators that alter the composition of children (normally, crossover and mutation).

5. Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

In this paper, we will use GAs as a numerical optimization tool, and we will propose a constraint-handling technique that makes unnecessary the use of a penalty function, which is the most common approach used to incorporate constraints into the fitness function of a GA [41].

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

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

subject to:

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

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

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

3 Related Work

The idea of using evolutionary multiobjective optimization techniques [9] to handle constraints is not entirely new. A few researchers have reported approaches that rely on the use of multiobjective optimization techniques as we will see in this section.

The most common approach 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 [23] 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 $\vec{x}$ would thus have $f_i(\vec{x})$=0 for $1 \leq i \leq m$ and $f(\vec{x}) \leq f(\vec{y})$ for all feasible $\vec{y}$ (assuming minimization).

Surry et al. [50, 49] proposed the use of Pareto ranking [22] and VEGA [43] 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. COMOGA compared fairly with a penalty-based approach in a pipe-sizing problem, since the resulting EA was less sensitive to changes in the parameters, but the results achieved were not better than those found with a penalty function [50]. It should be added that COMOGA [50] required several extra parameters, although its authors argue [50] that the technique is not particularly sensitive to the values of such parameters.

Parmee and Purchase [37] implemented a version of the *Vector Evaluated Genetic Algorithm* (VEGA) [43] that handled the constraints of a gas turbine problem as objectives to allow a genetic algorithm 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 [37] opted to use specialized operators that would create a variable-size hypercube around each feasible point to help the genetic algorithm to remain within the feasible region at all times. This approach was specially 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 the approach.

Camponogara & Talukdar [5] proposed the use of a procedure based on an evolutionary multiobjective optimization technique. Their proposal was to restate a single objective optimization problem in such a way that two objectives would be considered: the first would be to optimize the original objective function and the second would be to minimize:

$$\Phi(\vec{x}) = \sum_{i=1}^{n} \max[0, g_i(\vec{x})]^{\beta} \tag{4}$$

where β is normally 1 or 2.

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 [5]. 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 [5]. 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).

Camponogara & Talukdar's approach [5] 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 [30] proposed the use of a min-max approach [6] to handle constraints. The main idea of this approach is to apply a set of simple rules to decide the selection process:

1. If the two individuals being compared are both feasible, then select based on the minimum value of the objective function.

2. If one of the two individuals being compared is feasible and the other one is infeasible, then select the feasible individual.

3. If both individuals are infeasible, then select based on the maximum constraint violation ($\max g_j(\vec{x})$, for $j = 1, \ldots, m$, and m is the total number of constraints). The individual with the lowest maximum violation wins.

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 [50]. 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 [37] technique) may be a good alternative to find a feasible point in a heavily constrained search space.

Coello [10] proposed the use of a population-based multiobjective optimization technique such as VEGA [43] to handle each of the constraints of a single-objective optimization problem as an objective. At each generation, the population is split into $m+1$ sub-populations (m is the number of constraints), so that a fraction of the population is selected using the (unconstrained) objective function as its fitness and another fraction uses the first constraint as its fitness and so on.

For the sub-population guided by the objective function, the evaluation of such objective function for a given vector $\vec{x}$ is used directly as the fitness function (multiplied by (-1) if it is a minimization problem), with no penalties of any sort. For all the other sub-populations, the algorithm used was the following [10]:

$$
\begin{array}{lll}
\textbf{if } g_j(\vec{x}) < 0.0 & \textbf{then} & \text{fitness} = g_j(\vec{x}) \\
\textbf{else if } v \neq 0 & \textbf{then} & \text{fitness} = -v \\
\textbf{else} & & \text{fitness} = f
\end{array}
$$

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

This approach provided good results in several optimization problems, but required a relatively large number of fitness function evaluations to converge [10].

The limitations of the previously reported multiobjective optimization techniques used to handle constraints were the main motivation of this work.

4 The Proposed Approach

The main goal of this work was to develop a constraint-handling technique that did not require the use of a penalty factor that needed to be fine-tuned by the user, and that could be coupled with an evolutionary algorithm (particularly, with a genetic algorithm) to perform numerical optimization. We were also concerned with the efficiency of the technique (in terms of CPU time requirements), since we were particularly interested in using it in engineering optimization problems for which the cost of evaluating the fitness function is normally high.

Motivated by the previous work related on the use of multiobjective optimization techniques to handle constraints, we decided to develop an approach based on non-dominance to assign ranks to each individual in a population, following a similar rationale than the one used by the *Multi-Objective Genetic Algorithm* (MOGA) [22], which is a very popular (and effective) evolutionary multiobjective optimization approach.

The concept of dominance (or Pareto dominance, as it is usually called in Operations Research) is the following:

Definition 1 (Pareto Dominance): *A vector $\vec{u} = (u_1, \ldots, u_k)$ is said to* **dominate** $\vec{v} = (v_1, \ldots, v_k)$ *(denoted by $\vec{u} \preceq \vec{v}$) if and only if u is partially less than v, i.e., $\forall i \in \{1, \ldots, k\},\ u_i \leq v_i \wedge \exists i \in \{1, \ldots, k\} : u_i < v_i$.* $\square$

where $f : \Omega \subseteq \mathcal{R}^n \rightarrow \mathcal{R}$, $\Omega \neq \emptyset$.

The problem with using the concept of dominance in a direct way is its inefficiency (it would imply an $O(n^2)$ procedure, where n refers to the size of the population). This is necessary when dealing with multiobjective optimization problems, but if we intend to use this procedure to handle constraints in a single-objective optimization problem, we can reduce the complexity of the task by using some simple rules. For example, if we have a feasible individual in the population (i.e., one that does not violate any constraints), we can directly use its fitness value to compare it against the others since in this case the EA will be operating as it is intended to work: as an unconstrained optimization technique. On the other hand, if an individual violates constraints, then it will be necessary to compare it to the others. This motivated the idea of the algorithm presented here, which has 3 levels of comparison: if feasible, its fitness value is used and whenever it is compared against an unfeasible individual, it wins. If it is infeasible, then we compare its number of constraints violated first and only in case of a tie, we use the amount of constraint violation to decide who wins. Since we used stochastic universal sampling in our implementation, then we assign fitness to each individual based on these simple rules described before. It is also possible to use directly a tournament selection based on dominance (like proposed by Deb [17]), but its main drawback is that the high selection pressure generated by tournament selection will make necessary to use an additional procedure to preserve diversity in the population (e.g., niching or sharing [18]), and we wanted to avoid the introduction of extra parameters into the GA.

To understand better the way in which our procedure works, we include here its main algorithm used to assign fitness to an individual:

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

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

$$\text{rank}(\vec{x}_i) = \text{count}(\vec{x}_i) + 1 \tag{5}$$

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

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

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

$$\text{coef}(\vec{x}_i) = \sum_{k=1}^{p} g_k(\vec{x}_i) \quad \text{for all } g_k(\vec{x}_i) > 0 \tag{6}$$

- Compute fitness using the following rules:

  1. If $\vec{x}_i$ is feasible, then rank($\vec{x}_i$) = $fitness(\vec{x}_i)$, else
  2. rank($\vec{x}_i$) = $\frac{1}{\text{rank}(\vec{x}_i)}$

- Individuals are selected based on rank($\vec{x}_i$) (stochastic universal sampling is used).

- Values produced by $fitness(\vec{x}_i)$ must be normalized to ensure that the rank of feasible individuals is always higher than the rank of infeasible ones.

5 Use of Adaptive Crossover and Mutation Rates

A common problem associated with the use of genetic algorithms is that they require a certain (empirical) fine-tuning to select proper values for their parameters (i.e., population size, crossover and mutation rates, maximum number of generations, etc.). To avoid this problem, we decided to experiment with a simple self-adaptive approach in which the crossover and mutation rates were

2.15	1.89	0.43	3.14	0.27	7.93	5.11

Figure 1: An example of a real-coded GA.

considered as additional variables, ranging from 0.0 to 1.0 (defined as continuous variables). The population size and the maximum number of generations were (arbitrarily) defined by the user and kept constant during the evolutionary process.

Some simple rules were defined to decode the crossover and mutation rates to be used for each genetic operation:

Given individuals p_i and p_j to be crossed (after selection), which have c_i and c_j as their corresponding crossover rates encoded in their chromosomic strings, then use a crossover rate $c = \frac{c_i + c_j}{2}$.

For mutation, we mutate an individual p_i using m_i, which is the mutation rate encoded in its chromosomic string (after selection and crossover).

The best individual of each generation is kept intact (this is called "elitism").

6 Genetic Operators

We used a real-coded genetic algorithm [26, 34]. According to this representation, a chromosome is a vector of the form $\langle r_1, r_2, \ldots, r_m \rangle$, where $r_1, r_2, \ldots, r_m$ are real numbers whose lower and upper bounds are defined by the user. Despite the well-known fact that the binary alphabet offers the maximum number of schemata per bit of information of any coding [28], the 'implicit parallelism' property of genetic algorithms does not preclude the use of alphabets of higher cardinality.

Whereas theoreticians claim that small alphabets should be more effective than large alphabets, practitioners have shown through a considerable amount of real-world applications (particularly numerical optimization problems) that the direct use of real numbers in a chromosome works better in practice that the traditional binary representation [14, 20].

The use of real numbers in a chromosomic string (see Figure 1) has been common in other evolutionary techniques, such as evolution strategies [46] and evolutionary programming [21], where mutation is normally the primary operator. However, when dealing with GAs, there has been a strong criticism towards the use of real values for the genes of a chromosome, mainly because this higher cardinality representation will make the GA's behavior more erratic and difficult to predict. Because of this, several special operators have been designed over the years to emulate the effect of crossover and mutation over binary alphabets [20, 51, 19].

Practitioners argue that one of the main abilities of real-coded GAs is their capacity to exploit the gradualness of functions of continuous variables (where gradualness is taken to mean small changes in the variables correspond to small changes in the function).

This means that real-coded GAs can adequately deal with the "cliffs" produced when the variables used are real numbers, because a small change in the representation is mapped as a small change in the search space [20, 51].

For crossover, it was decided to use arithmetical crossover [34], which is defined as a linear combination of the two vectors representing the "parents" to be crossed: if p_1^t and p_2^t are the two parents selected for crossover at generation t, then the resulting offspring o_1^{t+1} and o_2^{t+1} will be:

$$\begin{aligned} o_1^{t+1} &= w \cdot p_2^t + (1-w) \cdot p_1^t \\ o_2^{t+1} &= w \cdot p_1^t + (1-w) \cdot p_2^t \end{aligned} \tag{7}$$

where w is a real number that, in our case, was randomly generated in the range [0..1].

It has been argued that a non-uniform mutation operator is more useful when optimizing with a GA because it allows us to search in different ways as needed (i.e., exploring wider or narrower regions) over time [34]. Due to some previous favorable experience with non-uniform mutation in the context of numerical optimization [13] it was decided to use this approach instead of the traditional uniform mutation operator.

To illustrate this operator, we will assume that at generation t, we have a string $C_t = \langle c_1, c_2, \ldots, c_l \rangle$. After randomly selecting a position along the string, in generation $t+1$, the new chromosome after mutation will be $C_{t+1} = \langle c_1, c_2, \ldots, c_k', \ldots, c_l \rangle$, where:

$$c_k' = \begin{cases} c_k + \Delta(t, u_k - c_k) & \text{if flip}(0.5) = 0 \\ c_k - \Delta(t, c_k - l_k) & \text{if flip}(0.5) = 1 \end{cases} \tag{8}$$

where the domain of c_k is $[l_k, u_k]$, and the function $flip(0.5)$ returns randomly and with equal probability one of two possible values: either zero or one. The function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases. The expression used here for the variation of the mutation step is the function originally suggested by Michalewicz [34]:

$$\Delta(t, y) = y \cdot \left(1 - r^{\left(1-\frac{t}{T}\right)^b}\right) \tag{9}$$

where r is a randomly generated real number in the range [0..1], T is the maximum number of generations, and b is a system parameter that determines the degree of dependency on the current generation number. The value adopted for the current implementation was $b = 5$, as suggested by Michalewicz [34].

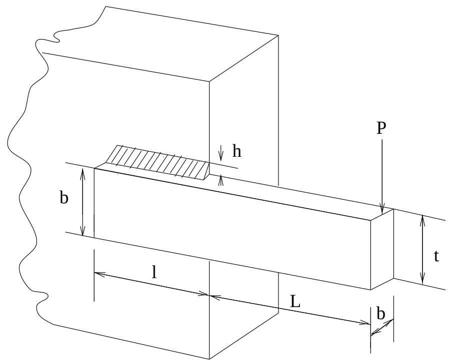

Figure 2: The welded beam used for the first example.

7 Examples

Several examples taken from the optimization literature will be used to show the way in which the proposed approach works. These examples have linear and nonlinear constraints, and have been previously solved using a variety of other techniques (both GA-based and traditional mathematical programming methods), which is useful to determine the quality of the solutions produced by the proposed approach.

7.1 Example 1 : Welded Beam Design

A welded beam is designed for minimum cost subject to constraints on shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), end deflection of the beam (δ), and side constraints [40]. There are four design variables as shown in Figure 2 [40]: h (x_1), l (x_2), t (x_3) and b (x_4).

The problem can be stated as follows:
Minimize:

$$F(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2) \tag{10}$$

Subject to:

$$g_1(\vec{x}) = \tau(\vec{x}) - \tau_{max} \leq 0 \tag{11}$$

$$g_2(\vec{x}) = \sigma(\vec{x}) - \sigma_{max} \leq 0 \tag{12}$$

$$g_3(\vec{x}) = x_1 - x_4 \leq 0 \tag{13}$$

$$g_4(\vec{x}) = 0.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5.0 \leq 0 \tag{14}$$

$$g_5(\vec{x}) = 0.125 - x_1 \leq 0 \tag{15}$$

$$g_6(\vec{x}) = \delta(\vec{x}) - \delta_{max} \leq 0 \tag{16}$$

$$g_7(\vec{x}) = P - P_c(\vec{x}) \leq 0 \tag{17}$$

where

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} \tag{18}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2}, \tau'' = \frac{MR}{J}, M = P\left(L + \frac{x_2}{2}\right) \tag{19}$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \tag{20}$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \tag{21}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2}, \delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4} \tag{22}$$

$$P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \tag{23}$$

$$P = 6000\ lb, \quad L = 14\ in, \quad E = 30 \times 10^6\ psi, \quad G = 12 \times 10^6\ psi \tag{24}$$

$$\tau_{max} = 13,600\ psi, \quad \sigma_{max} = 30,000\ psi, \quad \delta_{max} = 0.25\ in \tag{25}$$

7.2 Example 2 : Design of a Pressure Vessel

A cylindrical vessel is capped at both ends by hemispherical heads as shown in Figure 3. The objective is to minimize the total cost, including the cost of the material, forming and welding. There are four design variables: T_s (thickness of the shell), T_h (thickness of the head), R (inner radius) and L (length of the cylindrical section of the vessel, not including the head). T_s and T_h are integer multiples of 0.0625 inch, which are the available thicknesses of rolled steel plates,

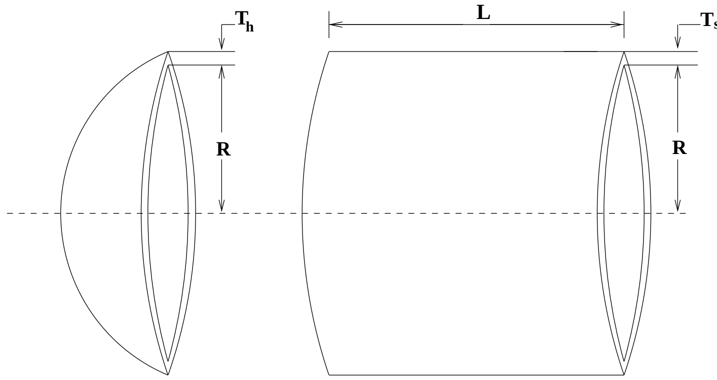

Figure 3: Center and end section of the pressure vessel used for the second example.

and R and L are continuous. Using the same notation given by Kannan and Kramer [32], the problem can be stated as follows:

Minimize :

$$F(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \tag{26}$$

Subject to :

$$g_1(\vec{x}) = -x_1 + 0.0193x_3 \leq 0 \tag{27}$$

$$g_2(\vec{x}) = -x_2 + 0.00954x_3 \leq 0 \tag{28}$$

$$g_3(\vec{x}) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1,296,000 \leq 0 \tag{29}$$

$$g_4(\vec{x}) = x_4 - 240 \leq 0 \tag{30}$$

7.3 Example 3 : Design of a 10-bar plane truss

Consider the 10-bar plane truss shown in Figure 4 [4]. The problem is to find the moment of inertia of each member of this truss, such that we minimize its weight, subject to stress and displacement constraints. The weight of the truss is given by:

$$f(x) = \sum_{j=1}^{10} \rho\, A_j\, L_j \tag{31}$$

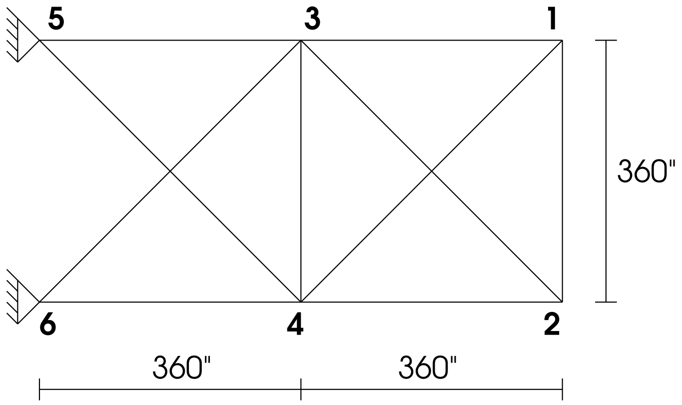

Figure 4: 10-bar plane truss used for Example No. 3.

where x is the candidate solution, A_j is the cross-sectional area of the jth member, L_j is the length of the jth member, and ρ is the weight density of the material.

The assumed data are: modulus of elasticity, $E = 1.0 \times 10^4$ ksi 68965.5 MPa), $\rho = 0.10$ lb/in^3 (2768.096 kg/m^3), and a load of 100 kips (45351.47 Kg) in the negative y-direction is applied at nodes 2 and 4. The maximum allowable stress of each member is called σ_a, and it is assumed to be ± 25 ksi (172.41 MPa). The maximum allowable displacement of each node (horizontal and vertical) is represented by u_a, and is assumed to be 2 inches (5.08 cm).

There are 10 stress constraints, and 12 displacement constraints (we can really assume only 8 displacement constraints because there are two nodes with zero displacement, but they will nevertheless be considered as additional constraints by the new approach). The moment of inertia of each element can be different, thus the problem has 10 design variables.

7.4 Example 4 : Design of a 25-bar space truss

Consider the 25-bar space truss taken from Rajeev and Khrisnamoorthy [39] shown in Figure 5. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight, the displacement of each free node, and the stress that each member has to support.

Loading conditions are given in Table 1, member groupings are given in Table 2, and node coordinates are given in Table 3. The assumed data are: modulus of elasticity, $E = 1 \times 10^4$ ksi, $\rho = 0.10$ lb/in^3; $\sigma_a = \pm 40$ ksi, $u_a = \pm 0.35$ in. This problem has 8 design variables and 86 constraints.

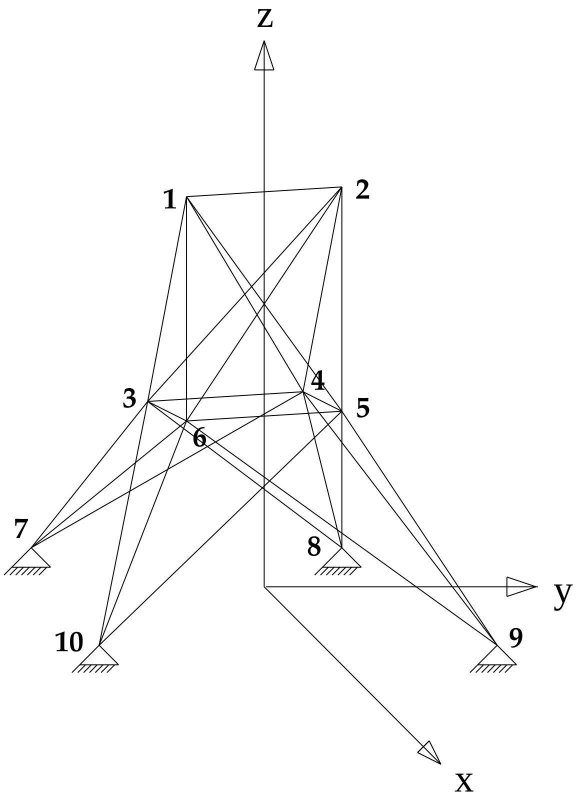

Figure 5: 25-bar space truss used for example No. 4.

Node	Fx (lbs)	Fy (lbs)	Fz (lbs)
1	1000	-10000	-10000
2	0	-10000	-10000
3	500	0	0
6	600	0	0

Table 1: Loading conditions for the 25-bar space truss shown in Figure 5.

Group Number	**Members**
1	1-2
2	1-4, 2-3, 1-5, 2-6
3	2-5, 2-4, 1-3, 1-6
4	3-6, 4-5
5	3-4, 5-6
6	3-10, 6-7, 4-9, 5-8
7	3-8, 4-7, 6-9, 5-10
8	3-7, 4-8, 5-9, 6-10

Table 2: Group membership for the 25-bar space truss shown in Figure 5.

Node	**X**	**Y**	**Z**
1	-37.50	0.00	200.00
2	37.50	0.00	200.00
3	-37.50	37.50	100.00
4	37.50	37.50	100.00
5	37.50	-37.50	100.00
6	-37.50	-37.50	100.00
7	-100.00	100.00	0.00
8	100.00	100.00	0.00
9	100.00	-100.00	0.00
10	-100.00	-100.00	0.00

Table 3: Coordinates of the joints of the 25-bar space truss shown in Figure 5.

8 Comparison of Results

8.1 Example 1

This problem was solved before by Deb [15] using a simple genetic algorithm with binary representation, and a traditional penalty function as suggested by Goldberg [25]. It has also been solved by Ragsdell and Phillips [38] using geometric programming. Ragsdell and Phillips also compared their results with those produced by the methods contained in a software package called "Opti-Sep" [47], which includes the following numerical optimization techniques: ADRANS (Gall's adaptive random search with a penalty function), APPROX (Griffith and Stewart's successive linear approximation), DAVID (Davidon-Fletcher-Powell with a penalty function), MEMGRD (Miele's memory gradient with a penalty function), SEEK1 & SEEK2 (Hooke and Jeeves with 2 different penalty functions), SIMPLX (Simplex method with a penalty function) and RANDOM (Richardson's random method).

Their results are compared against those produced by the approach proposed in this paper, which are shown in Table 4. In the case of Siddall's techniques [47], only the best solution produced by the techniques contained in "Opti-Sep" is displayed. The solution shown for the technique proposed here is the best produced after 30 runs, and using the following ranges for the design variables: $0.1 \leq x_1 \leq 2.0$, $0.1 \leq x_2 \leq 10.0$, $0.1 \leq x_3 \leq 10.0$, $0.1 \leq x_4 \leq 2.0$.

The mean from the 30 runs performed was $f(\vec{x}) = 1.919013258$, with a standard deviation of 0.053775284. The worst solution found was $f(\vec{x}) = 1.99498555$, which is better than any of the solutions produced by any of the other techniques depicted in Table 4. The population size used was 50, and the maximum number of generations was 100. It is important to point out that Deb [15] used a population size of 100 and a maximum number of generations of 50 with his GA. Therefore, the total number of fitness function evaluations is the same in both cases $(5,000)$. However, in our case the size of the search space is larger, because of the real numbers used in the chromosomic string. Deb [15] used binary strings of length 40 in his paper, with which the intrinsic size of the search space is $2^{40} \approx 1.099 \times 10^{22}$. It should be noted that the solution reported here is not the global optimum, and we have found a better solution in previous work ($f(\vec{x}) = 1.74830941$ [11]). However, that solution was found with over $900,000$ fitness function evaluations per run, whereas in the current paper, we found a solution that is less than 5% below the other one, performing only $5,000$ fitness function evaluations (about 0.6% of the number of evaluations used before). Better solutions are possible if we allow a larger number of fitness function evaluations.

8.2 Example 2

This problem was solved before by Deb [16] using GeneAS (Genetic Adaptive Search), by Kannan and Kramer using an augmented Lagrangian Multiplier approach [32], and by Sandgren [42] using a branch and bound technique. Their

Design Variables	Best solution found			
	This paper	**Deb [15]**	**Siddall [47]**	**Ragsdell [38]**
$x_1(h)$	0.1829	0.2489	0.2444	0.2455
$x_2(l)$	4.0483	6.1730	6.2189	6.1960
$x_3(t)$	9.3666	8.1789	8.2915	8.2730
$x_4(b)$	0.2059	-0.2533	0.2444	0.2455
$g_1(\vec{x})$	-408.732772	-5758.603777	-5743.502027	-5743.826517
$g_2(\vec{x})$	-2105.91421	-255.576901	-4.015209	-4.715097
$g_3(\vec{x})$	-0.023060	-0.004400	0.000000	0.000000
$g_4(\vec{x})$	-3.321528	-2.982866	-3.022561	-3.020289
$g_5(\vec{x})$	-0.057884	-0.123900	-0.119400	-0.120500
$g_6(\vec{x})$	-0.237029	-0.234160	-0.234243	-0.234208
$g_7(\vec{x})$	-160.586452	-4465.270928	-3490.469418	-3604.275002
$f(\vec{x})$	**1.82455147**	**2.43311600**	**2.38154338**	**2.38593732**

Table 4: Comparison of the results for the first example (optimal design of a welded beam).

results were compared against those produced by the approach proposed in this paper, and are shown in Table 5. The solution shown for the technique proposed here is the best produced after 30 runs, and using the following ranges for the design variables and the weights: $1 \leq x_1 \leq 99$, $1 \leq x_2 \leq 99$, $10.0 \leq x_3 \leq 200.0$, $10.0 \leq x_4 \leq 200.0$. The values for x_1 and x_2 were considered as integer (i.e., real values were rounded up to their closest integer value) multiples of 0.0625, and the values of x_3 and x_4 were considered as real numbers.

The mean from the 30 runs performed was $f(\vec{x}) = 6263.792526931$, with a standard deviation of 97.944473926. The worst solution found was $f(\vec{x}) = 6403.45006683$, which is better than any of the solutions produced by any of the other techniques depicted in Table 5. The population size used was 50, and the maximum number of generations was 1000. The parameters used by GeneAS were not available for comparison.

8.3 Example 3

This problem was used by Belegundu [4] to evaluate the following numerical optimization techniques: Feasible directions (CONMIN and OPTDYN), Pshenichny's Recursive Quadratic Programming (LINRM), Gradient Projection (GRP-UI), Exterior Penalty Function (SUMT), Multiplier Methods (M-3, M-4 and M-5).

The results reported by Belegundu [4] are compared to the current approach in Tables 6 and 7 (all the solutions presented are feasible). To solve this problem, it was necessary to add a module responsible for the analysis of the plane truss. This module uses the matrix factorization method included in Gere and Weaver [24] together with the stiffness method [24] to analyze the structure, and returns

Design Variables	Best solution found			
	This paper	**GeneAS [16]**	**Kannan [32]**	**Sandgren [42]**
$x_1(T_s)$	0.8750	0.9375	1.125	1.125
$x_2(T_h)$	0.5000	0.5000	0.625	0.625
$x_3(R)$	42.0939	48.3290	58.291	47.700
$x_4(L)$	177.0805	112.6790	43.690	117.701
$g_1(\vec{x})$	-0.000088	-0.004750	0.000016	-0.204390
$g_2(\vec{x})$	-0.035924	-0.038941	-0.068904	-0.169942
$g_3(\vec{x})$	-2156.836486	-3652.876838	-21.220104	54.226012
$g_4(\vec{x})$	-62.919507	-127.321000	-196.310000	-122.299000
$f(\vec{x})$	**6069.3267**	**6410.3811**	**7198.0428**	**8129.1036**

Table 5: Comparison of the results for the second example (optimization of a pressure vessel).

the values of the stress and displacement constraints, as well as the total weight of the structure.

The solution shown for the technique proposed here is the best produced after 30 runs. The range $0.1 \leq x \leq 999.0$ was used for the 10 design variables (moments of inertia were used as the design variables, and their square roots were found in order to obtain the cross-sectional areas of each truss member).

The mean from the 30 runs performed was $f(\vec{x}) = 5338.41193$, with a standard deviation of 73.63898. The worst solution found was $f(\vec{x}) = 5469.38852$, which is better than any of the solutions produced by any of the other techniques depicted in Tables 6 and 7. The population size used was 50, and the maximum number of generations was set to $1,000$ (a total of $50,000$ fitness function evaluations).

8.4 Example 4

This problem has been solved using many different optimization techniques: Schmit and Farshi [44] used an adaptation of the method of inscribed hyperspheres, Khan et al. [33] used an optimality criterion method, Adeli and Kamal [2] used geometric programming, Chao et al. [7] used quadratic programming, Schmit and Miura [45] used both the CONstrained function MINimization (CONMIN) and the NEW Unconstrained Sequential Minimization Technique (NEWSUMT), and Coello et al. [8] used a simple genetic algorithm with binary representation and binary tournament selection (the search space in that case was smaller than in the work reported in the current paper, because each variable was considered discrete whereas we considered them as continuous in the current paper).

The results reported by the previously mentioned authors are compared to the current approach in Tables 8 and 9 (all the solutions presented are feasible).

To solve this problem, it was necessary to add a module responsible for the

Design Variables	Best solution found			
	This paper	**CONMIN**	**OPTDYN**	**LINRM**
x_1	979.7582	639.20	664.30	21.57
x_2	0.1039	3.60	0.01	10.98
x_3	539.4210	618.40	630.70	22.08
x_4	208.7898	250.50	375.90	14.95
x_5	0.1087	0.01	0.01	0.10
x_6	0.1243	3.05	0.01	10.98
x_7	69.6764	280.80	235.90	18.91
x_8	483.3090	389.20	413.00	18.42
x_9	44.7585	440.10	430.30	18.40
x_{10}	0.1036	6.30	1.30	13.51
$f(\vec{x})$	**5153.61**	**5563.32**	**5471.48**	**6428.89**

Table 6: Comparison of results for the third example (10-bar plane truss). The value of all variables is given in in^4. Part I.

Design Variables	Best solution found				
	GRP-UI	**SUMT**	**M-3**	**M-4**	**M-5**
x_1	614.30	942.00	667.90	1000.0	667.70
x_2	17.40	5.60	9.40	139.40	8.30
x_3	614.40	1000.0	697.80	1000.0	699.40
x_4	208.80	135.90	163.10	306.40	162.60
x_5	0.01	0.01	0.01	1000.0	0.01
x_6	17.40	13.80	11.80	105.00	14.20
x_7	304.80	471.20	373.90	1000.00	375.50
x_8	370.90	467.00	367.60	1000.00	368.00
x_9	371.30	195.30	351.90	1000.00	352.20
x_{10}	27.70	10.60	19.50	1000.00	19.20
$f(\vec{x})$	**5727.05**	**5932.21**	**5719.19**	**11279.22**	**5726.08**

Table 7: Comparison of results for the third example (10-bar plane truss). The value of all variables is given in in^4. Part II.

Design Variables	Best solution found			
	This paper	**Chao [7]**	**CONMIN [45]**	**NEWSUMT [45]**
x_1	0.1303	0.0100	0.1660	0.0100
x_2	0.1201	2.0415	2.0170	1.9850
x_3	3.4834	3.0011	3.0260	2.9960
x_4	0.1102	0.0100	0.0870	0.0100
x_5	1.6583	0.0100	0.0970	0.0100
x_6	0.8373	0.6836	0.6750	0.6840
x_7	0.1172	1.6248	1.6360	1.6670
x_8	4.0900	2.6716	2.6690	2.6620
$f(\vec{x})$	**470.09**	**545.03**	**548.47**	**545.17**

Table 8: Comparison of results for the fourth example (25-bar space truss). The value of all variables is given in in^2. Part I

analysis of the space truss. This module uses the matrix factorization method included in Gere and Weaver [24] together with the stiffness method [24] to analyze the structure, and returns the values of the stress and displacement constraints, as well as the total weight of the structure.

The solution shown for the technique proposed here is the best produced after 30 runs. The range $0.1 \leq x \leq 999.00$ was used for the 8 design variables (all variables were treated as continuous).

The mean from the 30 runs performed was $f(\vec{x}) = 482.505592634$, with a standard deviation of 6.351394347. The worst solution found was $f(\vec{x}) = 493.80920116$, which is slightly better than any of the solutions produced by any of the other techniques depicted in Tables 8 and 9.

The best solution previously reported was found by Coello et al. [8] using a population size of 300 running during 100 generations. However, to find the best solution reported, 81 runs were performed (varying the crossover and mutation rates), whereas in our case, we performed only 30 runs. To allow a fair comparison, we decided to run our current GA with a population of 50 individuals during 600 generations to have the same number of total fitness function evaluations (per run) than in Coello et al. [8] (i.e., $300,000$ fitness function evaluations).

9 Discussion

There are some interesting issues about the proposed procedure that deserve to be briefly discussed. Firstly, we have to consider the different situations that could arise when randomly generating a set of individuals (vectors of real numbers in our case). We could have 3 situations:

1. **All individuals are infeasible**. This situation can arise in highly constrained search spaces (e.g., on Example 2 from this paper) and a GA is

Design Variables	Best solution found			
	Schmit [44]	Khan [33]	Adeli [2]	Coello [8]
x_1	0.010	0.010	0.010	0.100
x_2	1.964	1.755	1.9855	0.700
x_3	3.033	2.869	2.9606	3.200
x_4	0.010	0.010	0.0100	0.100
x_5	0.010	0.010	0.0100	1.400
x_6	0.670	0.845	0.8062	1.100
x_7	1.680	2.011	1.6795	0.500
x_8	2.670	2.478	2.5298	3.400
$f(\vec{x})$	**545.22**	**553.94**	**545.66**	**493.94**

Table 9: Comparison of results for the fourth example (25-bar space truss). The value of all variables is given in in^2. Part II

expected to be able to deal with it. When we face this situation using a penalty function, all individuals would be penalized. In the proposed approach, they would all be ranked according their degree of constraint violation. Conceptually, both approaches are equivalent, but in our case, the user does not need to estimate penalty functions or factors of any kind, since the information about constraint violation derived from the evolutionary process itself is used to assign fitness to each individual.

2. **All individuals are feasible**. This could happen in some cases in which the constraints are not very difficult to satisfy, and then the GA would be very efficient, because no additional computations are required to assign fitness values. The direct value of the objective function can be used (or its reciprocal value in case of a minimization problem).

3. **There is a mixture of feasible and infeasible individuals**. This is the probably the most common situation when using an GA. The main concern when using the approach proposed here is that the ranking of infeasible individuals does not assign values higher than the fitness values of the feasible individuals. To ensure that, we scale the ranking values of infeasible individuals after the procedure has finished, so that the fitness value of the infeasible individual with the best ranking is smaller than the fitness of the worst feasible individual.

One of the main highlights of this procedure is that it does not require any additional procedure to keep diversity, because the selection pressure generated by the ranking process is not excessive and therefore individuals who might be assigned an extremely low fitness with a badly designed penalty function might still have a probability of surviving with our procedure. This fact encourages diversity, while at the same time pushes the infeasible individuals towards the feasible region.

10 Conclusions

This paper has introduced a new constraint-handling approach that is based on the concept of "non-dominance" used in multiobjective optimization. The approach is intended to be used with evolutionary algorithms as a way to reduce the burden normally associated with the fine-tuning of a penalty function.

The proposed approach performed well in several test problems both in terms of the number of fitness function evaluations required and in terms of the quality of the solutions found. The results produced were compared against those generated with other (evolutionary and mathematical programming) techniques reported in the literature.

11 Future Work

We are interested in developing constraint-handling versions of the other two most popular multiobjective optimization techniques that we have not addressed yet (NPGA [29] and NSGA [48]). These techniques have certain differences with respect to MOGA. NSGA uses ranking, but removes highest ranking individuals (i.e., non-dominated vectors) and then re-classifies the rest of the population, until no individuals are left to classify. NPGA uses a tournament selection scheme based on dominance, which is more efficient than checking dominance for the whole population (only a subset is used instead). Both approaches are promising and could present interesting aspects when adapted to handle constraints.

We are also interested in developing parallel versions of these techniques, so that they can be made more efficient when dealing with highly constrained search spaces. Work in that direction is also under way.

12 Acknowledgments

The author acknowledges support from CONACyT through project number I-29870 A. Many thanks to Margarita Reyes Sierra for her help with the experiments and data processing required by this research.

References

[1] Hojjat Adeli and Nai-Tsang Cheng. Augmented Lagrangian Genetic Algorithm for Structural Optimization. *Journal of Aerospace Engineering*, 7(1):104–118, January 1994.

[2] Hojjat Adeli and Osama Kamal. Efficient Optimization of Space Trusses. *Computers and Structures*, 24(3):501–511, 1986.

[3] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation.* Institute of Physics Publishing and Oxford University Press, 1997.

[4] Ashok Dhondu Belegundu. *A Study of Mathematical Programming Methods for Structural Optimization.* PhD thesis, Department of Civil and Environmental Engineering, University of Iowa, Iowa, 1982.

[5] 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.

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

[7] N. H. Chao, S. J. Fenves, and A. W. Westerberg. Application of Reduced Quadratic Programming Technique to Optimal Structural Design. In E. Atrek, R. H. Gallagher, K. M. Radgsdell, and O. C. Zienkiewicz, editors, *New Directions in Optimum Structural Design.* John Wiley, New York, 1984.

[8] Carlos A. Coello, Michael Rudnick, and Alan D. Christiansen. Using genetic algorithms for optimal design of trusses. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, pages 88–94, New Orleans, LA, November 1994. IEEE Computer Society Press.

[9] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

[10] Carlos A. Coello Coello. Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization*, 32(3):275–308, 2000.

[11] Carlos A. Coello Coello. Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, 41(2):113–127, January 2000.

[12] Carlos A. Coello Coello and Alan D. Christiansen. A Simple Genetic Algorithm for the design of reinforced concrete beams. *Engineering with Computers*, 13(4):185–196, 1997.

[13] Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Using a new GA-based multiobjective optimization technique for the design of robot arms. *Robotica*, 16:401–414, 1998.

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

[15] Kalyanmoy Deb. Optimal Design of a Welded Beam via Genetic Algorithms. *AIAA Journal*, 29(11):2013–2015, November 1991.

[16] Kalyanmoy Deb. GeneAS: A Robust Optimal Design Technique for Mechanical Component Design. In Dipankar Dasgupta and Zbigniew Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 497–514. Springer-Verlag, Berlin, 1997.

[17] Kalyanmoy Deb. An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, 2000. (in Press).

[18] 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, June 1989. George Mason University, Morgan Kaufmann Publishers.

[19] Kalyanmoy Deb and Mayank Goyal. Optimizing Engineering Designs Using a Combined Genetic Search. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 521–528. Morgan Kauffman Publishers, San Mateo, California, July 1995.

[20] Larry J. Eshelman and J. Davis Schaffer. Real-coded genetic algorithms and interval-schemata. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[21] David B. Fogel and L. C. Stayton. On the Effectiveness of Crossover in Simulated Evolutionary Optimization. *BioSystems*, 32:171–182, 1994.

[22] 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.

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

[24] James M. Gere and William Weaver. *Analysis of Framed Structures*. D. Van Nostrand Company, Inc., 1965.

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

[26] David E. Goldberg. Real-Coded Genetic Algorithms, Virtual Alphabets and Blocking. Technical Report 90001, University of Illinois at Urbana-Champaign, Urbana, Illinois, sep 1990.

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

[28] John H. Holland. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1992.

[29] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, June 1994. IEEE Service Center.

[30] 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.

[31] 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.

[32] B. K. Kannan and S. N. Kramer. An Augmented Lagrange Multiplier Based Method for Mixed Integer Discrete Continuous Optimization and Its Applications to Mechanical Design. *Journal of Mechanical Design. Transactions of the ASME*, 116:318–320, 1994.

[33] M. R. Khan, K. D. Willmert, and W. A. Thornton. An Optimality Criterion Method for Large-Scale Structures. *AIAA Journal*, 17(7):753–761, July 1979.

[34] 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.

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

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

[37] 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.

[38] K. M. Ragsdell and D. T. Phillips. Optimal Design of a Class of Welded Structures Using Geometric Programming. *ASME Journal of Engineering for Industries*, 98(3):1021–1025, 1976. Series B.

[39] S. Rajeev and C. S. Krishnamoorthy. Discrete Optimization of Structures Using Genetic Algorithms. *Journal of Structural Engineering*, 118(5):1233–1250, May 1992.

[40] Singiresu S. Rao. *Engineering Optimization.* John Wiley and Sons, third edition, 1996.

[41] 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.

[42] E. Sandgren. Nonlinear integer and discrete programming in mechanical design. In *Proceedings of the ASME Design Technology Conference*, pages 95–105, Kissimine, Florida, 1988.

[43] 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.

[44] L. A. Schmit and B. Farshi. Some Approximation Concepts for Structural Synthesis. *AIAA Journal*, 12(5):692–699, May 1974.

[45] L. A. Schmit and H. Miura. A New Structural Analysis/Synthesis Capability—ACCESS 1. *AIAA Journal*, 14(5):661–671, May 1976.

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

[47] James N. Siddall. *Analytical Design-Making in Engineering Design.* Prentice-Hall, 1972.

[48] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, fall 1994.

[49] Patrick D. Surry and Nicholas J. Radcliffe. The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics*, 26(3), 1997.

[50] 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.

[51] Alden H. Wright. Genetic Algorithms for Real Parameter Optimization. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann Publishers, San Mateo, California, 1991.