TREATING CONSTRAINTS AS OBJECTIVES FOR SINGLE-OBJECTIVE EVOLUTIONARY OPTIMIZATION

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

This paper presents a new approach to handle constraints using evolutionary algorithms. The new technique treats constraints as objectives, and uses a multiobjective optimization approach to solve the re-stated single-objective optimization problem. The new approach is compared against other numerical and evolutionary optimization techniques in several engineering optimization problems with different kinds of constraints. The results obtained show that the new approach can consistently outperform the other techniques using relatively small sub-populations, and without a significant sacrifice in terms of performance.

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

1 INTRODUCTION

Even when genetic algorithms (GAs) have been quite successful in a wide range of applications [1, 2], their use in constrained optimization problems raises several issues to which a considerable amount of research has been devoted in the last few years. From these issues, one of the most important ones is how to incorporate constraints of any sort (linear, non-linear, equality or inequality) into the fitness function as to guide the search properly. Due to the nature of the problems for which the GA is more suitable, it is normally quite difficult (or even impossible) to know the shape of the search space, and therefore is not easy to produce special operators and/or to explore it efficiently, unless we severely constraint the range of applications for which such approach will be useful.

For several years, practitioners have used penalty functions to incorporate constraints (particularly inequality constraints) into the fitness function, and there have been a lot of successful applications of this approach in all engineering fields [3]. However, penalty functions have some well-known limitations [4], 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 [4].

In this paper, a new constraint-handling approach is proposed that does not require the use of a penalty function to handle equality and inequality constraints. This technique is based on a multiobjective optimization approach, and it is very suitable for parallelization.

---

*This work was performed while the author was affiliated to the Engineering Design Centre at the University of Plymouth, in the United Kingdom.

The remainder of this paper is organized as follows: first, a review of some of the most important constraint-handling techniques developed so far (in the context of numerical optimization) will be provided, then 5 engineering optimization problems will be introduced, and solved using the new approach. After that, the results produced by other (GA-based and mathematical programming) techniques will be compared with those obtained with the new method, and finally there will be some discussion of the results obtained and the expected paths of future research.

2 PREVIOUS WORK

Over the years, several approaches have been developed to handle constraints using evolutionary algorithms. Focusing only on numerical optimization, these approaches can be classified as follows [5]:

- Rejection of infeasible individuals.
- Maintaining a feasible population by special representations and genetic operators.
- Separation of objectives and constraints.
- Penalizing infeasible individuals.

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. This approach is very popular within the evolution strategies community [6], 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. However, in situations in which the initial population does not contain a single feasible solution, this approach will cause no further progress in the evolution process because all the individuals will have the same fitness value.

Davis [7] presented several specialized representations developed for particular optimization problems (e.g., parametric design of aircraft, robot trajectory generation, schedule optimization, systhesis of neural network architectures, etc.), which preserved feasibility of the individuals at all times. Another example of this approach is GENOCOP (GEnetic algorithm for Numerical Optimization for COnstrained Problems), developed by Michalewicz [8], which handles linear constraints by eliminating equalities and designing special genetic operators which guarantee to keep all chromosomes within the constrained solution space. GENOCOP assumes a feasible starting point (or feasible initial population) and since it assumes the existence of only linear constraints, it is inherently restricted to convex search spaces [5].

There are several approaches that handle constraints and objectives separately. On of them was reported by Paredis [9], and is based on a co-evolutionary model in which there are two populations: the first contains the constraints to be satisfied and the second contains potential—possibly invalid—solutions to the problem. Using an analogy with a predator-prey model, the fitness of the members of one population depends on the fitness of the members of the other population. 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.

Schoenauer and Xanthakis [10] proposed another technique called *behavioural memory* in which constraints are handled in a particular order. The algorithm is the following [10]:

- 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

  $$eval(\mathbf{X}) = g_1(\mathbf{X}) \tag{1}$$

- $j = j + 1$
- The current population is the starting point for the next phase of the evolution, minimizing the violation of the j-th constraint,

  $$eval(\mathbf{X}) = g_j(\mathbf{X}) \tag{2}$$

  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 use death penalty, because infeasible individuals are completely eliminated from the population.

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) [11].

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.

Another approach that emulates the immune system to handle constraints was proposed by Hajela and Lee [12]. The idea of this technique is to separate any feasible individuals in a population (called antigens) from those that are infeasible (called 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. Although this approach seems quite interesting and more biologically inspired, some 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.

The most common approach in the GA 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 using [5]

$$fitness_i = f_i(\mathbf{X}) \pm Q_i \tag{3}$$

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 $Q_i = 0$.

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* [13]). However, although 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 [4]. 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 [5]:

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. [4] from which the following guidelines were derived:

1. Penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints.
2. For a problem having few constraints, and few full solutions, penalties which are solely functions of the number of violated constraints are not likely to find 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 not find a solution.

Based mainly on these guidelines, several researchers have attempted to derive good techniques to build penalty functions. Homaifar, Lai and Qi [14] 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. The main drawback of this technique is the high number of parameters required [15]. For m constraints, this approach requires $m(2l+1)$ parameters in total, where l is the number of levels defined. So, if we have for example 5 constraints and 3 levels, we would need 35 parameters, which is a very high number considering the small size of the problem.

Joines and Houck [16] proposed a technique in which dynamic penalties (i.e., penalties that change over time) are used. Individuals are evaluated (at generation t) using:

$$fitness_i(\mathbf{X}) = f_i(\mathbf{X}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\mathbf{X}) \tag{4}$$

where C, α and β are constants defined by the user and m is the number of constraints. This dynamic function increases the penalty as we progress through generations. Some researchers [17] 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 this approach the quality of the solution found is very sensitive to changes in the values of the parameters. Even when a certain set of values for these parameters ($C = 0.5$, $\alpha = \beta = 2$) were found by the authors of this method to be a reasonable choice, Michalewicz [15] reported that these values produce premature convergence most of the time. 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 [15, 5].

Powell and Skolnick [18] incorporated a heuristic rule (suggested by Richardson et al. [4]) for processing infeasible solutions: evaluations of feasible solutions are mapped into the interval ($-\infty$, 1), and infeasible solutions into the interval (1, ∞). This is equivalent (for ranking and tournament selection procedures [19, 8]) to the following evaluation procedure:

$$fitness_f(\mathbf{X}) = f(\mathbf{X}) \tag{5}$$

$$fitness_u(\mathbf{X}) = f(\mathbf{X}) + r\sum_{j=1}^{m} f_j(\mathbf{X}) \tag{6}$$

In this expression, r is a constant, and

$$fitness(\mathbf{X}) = \begin{cases} fitness_f(\mathbf{X}), & \text{if X is feasible} \\ fitness_u(\mathbf{X}) + \rho(\mathbf{X}, t), & \text{otherwise} \end{cases} \tag{7}$$

$$\rho(\mathbf{X}, t) = max\{0, max\{fitness_f(\mathbf{X})\}\} - min\{fitness_u(\mathbf{X})\} \tag{8}$$

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 [18]. However, in cases where the ratio between the feasible region and the whole search space is too small, the technique will fail unless a feasible point is introduced in the initial population [11].

Michalewicz and Attia [20] considered a method based on the idea of simulated annealing [21]: the penalty coefficients are changed once in many generations (after the convergence of the algorithm to a local optima). At every iteration the algorithm considers active constraints only, and the pressure on infeasible solutions is increased due to the decreasing values of the temperature of the system.

The method of Michalewicz and Attia [20] 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 [15]:

$$fitness(\mathbf{X}) = f(\mathbf{X}) + \frac{1}{2\tau}\sum_{j\in\mathcal{A}} f_j^2(\mathbf{X}) \tag{9}$$

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 linear constraints. At each iteration, the temperature τ is decreased and the new population is created using the best solution found in the previous iteration. The process stops when a pre-defined final 'freezing' temperature τ_f is reached.

One of the main drawbacks of this 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 [21]. Also, the approach used to handle linear constraints (treated separately by this technique) is very efficient, but it requires that the user provides an initial feasible point to the algorithm.

Bean and Hadj-Alouane [22] 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:

$$fitness(\mathbf{X}) = f(\mathbf{X}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\mathbf{X}) \tag{10}$$

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

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & if\ case\#1 \\ \beta_2 \cdot \lambda(t), & if\ case\#2 \\ \lambda(t), & otherwise, \end{cases} \tag{11}$$

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.

The obvious drawback of this dynamic penalty approach 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.

Le Riche et al. [13] 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. The population is split into two cooperating groups, where individuals in each group are evaluated using either one of the two penalty parameters. The idea is to combine those 2 sub-populations into a single one, mixing then individuals which are feasible with those that are not. Linear ranking is used to decrease the selection pressure that could cause premature convergence.

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 [11] 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.

Another approach that does not seem to fit into any of the previous categories defined is tha mapping-based method proposed by Kim and Husbands [23]. This technique is a more theoretical approach that uses Riemann mappings to transform the feasible region into a shape that facilitates the search for the GA. The problem with this approach is that it has been used only for problems with low dimensionality (no more than 4 variables), in which the objective function is given in explicit (algebraic) form, and it does not seem clear how to extend the technique to handle more difficult problems.

3 MULTIOBJECTIVE OPTIMIZATION TECHNIQUES

In this approach, objectives and constraints are handled separately. 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 [24] 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).

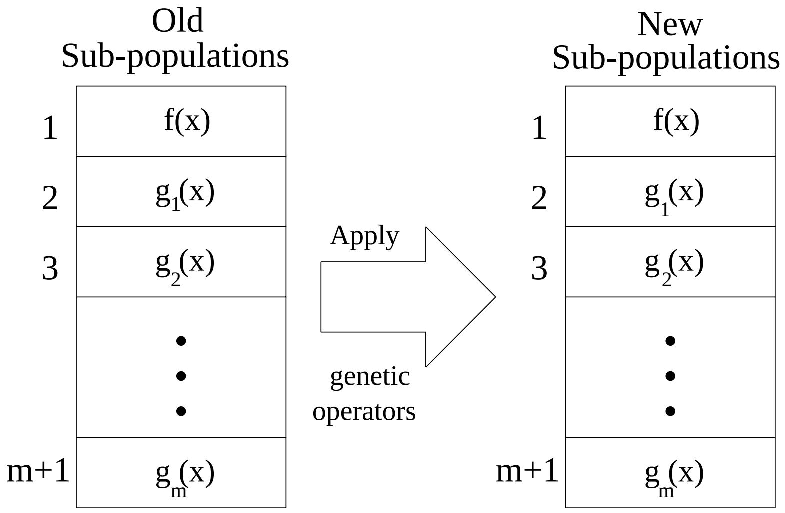

Figure 1: Graphical representation of the approach introduced in this paper.

Surry et al. [25] proposed the use of Pareto ranking [26] and VEGA [27] 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. This approach 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 [25]. It should be added that COMOGA [25] 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).

Parmee and Purchase [28] implemented a version of VEGA [27] 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 [28] 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.

3.1 Description of the new approach

The main idea behind the approach proposed in this paper is to use a population-based multiobjective optimization technique such as VEGA [27] to handle each of the constraints as an objective in the way indicated before. The technique may be better illustrated by Figure 1. 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). Although the size of each sub-population may be variable, it was decided to allocate the same size to each of them in the experiments reported in this paper, but the use of different sizes remains as an open issue that requires further research.

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. However, it is not completely obvious how to guide each of these sub-populations during the search.

For the sub-population guided by the objective function, the evaluation of such objective function for a given vector $\mathbf{X}$ (decoded from the chromosome) 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:

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

where $g_j(\mathbf{X})$ 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. However, traditionally it is necessary to define in advance either an static penalty value or a dynamic penalty function that estimates this distance from feasibility, whereas in the current approach such distance is estimated automatically by the above algorithm using the constraint violation information derived from the GA run.

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 important to clarify that the current approach does not use dominance to impose an order on the constraints based on their violation (like in the case of COMOGA [25]) which is a more expensive process (in terms of CPU time) that also requires additional parameters. In fact, the current approach does not rank individuals, but it uses instead different fitness functions for each of the sub-population allocated (whose number depends on the number of constraints) depending on the feasibility of the individuals contained within each of them. This is easier to implement, does not require special operators to preserve feasiblity (like in the case of Parmee and Purchase's approach [28]), makes unnecessary the use of a sharing function to preserve diversity (like with traditional multiobjective optimization techniques), and does not require extra parameters to control the mixture of feasible and infeasible individuals (like in the case of COMOGA [25]).

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.

Although VEGA is known to have difficulties in multiobjective optimization problems due to the fact that it tries to find individuals that excel only in one dimension regardless of the others (the so-called

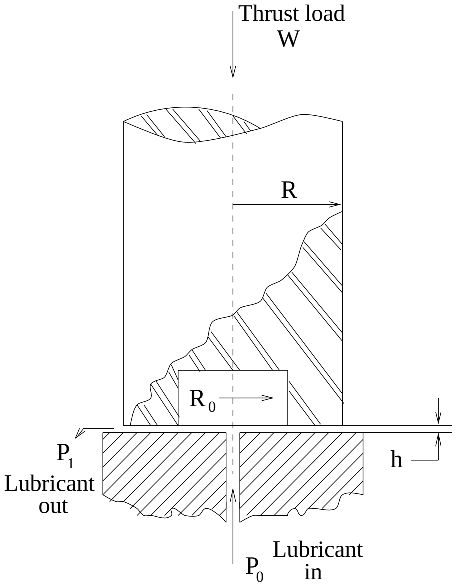

Figure 2: The hydrostatic thrust bearing used for the first example.

"middling" problem [27, 24]), that drawback turns out to be an advantage in this context, 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.

The new approach has been tested with several functions of different degrees of difficulty, and has been able to deal properly with both inequality and equality constraints, as will be seen in a further section.

4 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 new approach.

It should be mentioned that the initial goal of this work was to reproduce the quality of the results found with simple genetic algorithms (using a penalty function) that were fine-tuned to solve an specific problem using an empirical approach (normally by simple trial and error) and with traditional numerical optimization techniques. However, as will be seen later, the new technique proposed in this paper not only matched previous results, but in fact it improved them, though fairly small sub-populations were used in the experiments reported next. It should also mentioned that the ranges shown for the design variables are the same reported in the original references from where these problems were obtained.

4.1 Example 1 : Design of a hydrostatic thrust bearing

In this problem we want to minimize the power loss during the operation of a hydrostatic thrust bearing (see Figure 2) which has to withstand a specified load while providing an axial support. Four design variables are considered: R (bearing step radius), R_0 (recess radius), μ (viscosity), and Q (flow rate). The optimization problem can be stated as follows:

Minimize :

$$F(\mathbf{X}) = \frac{QP_0}{0.7} + E_f \tag{12}$$

Subject to :

$$g_1(\mathbf{X}) = \frac{\pi P_0}{2}\frac{R^2 - R_0^2}{\ln(R/R_0)} - W_s \geq 0 \tag{13}$$

$$g_2(\mathbf{X}) = P_{max} - P_0 \geq 0 \tag{14}$$

$$g_3(\mathbf{X}) = \Delta T_{max} - \Delta T \geq 0 \tag{15}$$

$$g_4(\mathbf{X}) = h - h_{min} \geq 0 \tag{16}$$

$$g_5(\mathbf{X}) = R - R_0 \geq 0 \tag{17}$$

$$g_6(\mathbf{X}) = 0.001 - \frac{\gamma}{gP_0}\left(\frac{Q}{2\pi Rh}\right) \geq 0 \tag{18}$$

$$g_7(\mathbf{X}) = 5000 - \frac{W}{\pi(R^2 - R_0^2)} \geq 0 \tag{19}$$

where the inlet pressure P_0 is defined as

$$P_0 = \frac{6\mu Q}{\pi h^3}\ln\frac{R}{R_0} \tag{20}$$

and the power loss due to friction is

$$E_f = 9336.0 Q\gamma C\Delta T \tag{21}$$

where $\gamma = 0.0307$ lb/in^3 (849.5755 kg/m^3) is the weight density of oil and $C = 0.5$ Btu/lb °F (0.5 cal/g °C) is the specific heat of oil. The temperature rise ΔT can be calculated from the following expression:

$$\Delta T = 2P_2 \tag{22}$$

where

$$P_2 = 10^{P_3} - 560 \tag{23}$$

and

$$P_3 = \frac{\log_{10}\log_{10}(8.122 \times 10^6\mu + 0.8) - C_1}{n} \tag{24}$$

C_1 and n are constants for a given oil, defined according to Table 1. For the purposes of this example, we will use SAE 20 grade oil, and therefore $C_1 = -3.55$ and $n = 10.04$. After calculating the value of E_f, we can calculate the film thickness h from the following equation:

$$h = \left(\frac{2\pi N}{60}\right)^2 \frac{2\pi\mu}{E_f}\left(\frac{R^4}{4} - \frac{R_0^4}{4}\right) \tag{25}$$

Other parameters required are: $W_s = 101000$ lb (45804.99 Kg), $P_{max} = 1000$ psi (6.89655×10^6 Pa), $\Delta T_{max} = 50$ °F (10 °C), $h_{min} = 0.001$ in (0.00254 cm), $g = 386.4$ in/seg^2 (981.456 cm/seg^2), and $N = 750$.

Oil	**C$_1$**	**n**
SAE 5	10.85	-3.91
SAE 10	10.45	-3.72
SAE 20	10.04	-3.55
SAE 30	9.88	-3.48
SAE 40	9.83	-3.46
SAE 50	9.82	-3.44

Table 1: Values of C_1 and n for various grades of oil (example 1).

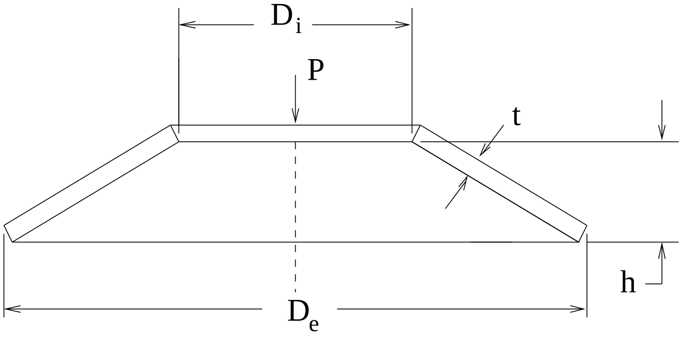

Figure 3: The Belleville spring used for the second example.

4.2 Example 2 : Design of a Belleville spring

In this case, the objective is to design a Belleville spring that has minimum weight while satisfying several constraints (see Figure 3). There are four design variables: D_e (external diameter of the spring), D_i (internal diameter of the spring), t (thickness of the spring), and h (height of the spring).

The problem can be stated as follows:

Minimize:

$$F(\mathbf{X}) = 0.07075\pi(D_e^2 - D_i^2)t \tag{26}$$

Subject to:

$$g_1(\mathbf{X}) = S - \frac{4E\delta_{max}}{(1-\mu^2)\alpha D_e^2}[\beta(h-\delta_{max}/2)+\gamma t] \geq 0 \tag{27}$$

$$g_2(\mathbf{X}) = \frac{4E\delta}{(1-\mu^2)\alpha D_e^2}[(h-\delta_{max}/2)(h-\delta_{max})t+t^3] - P_{max} \geq 0 \tag{28}$$

$$g_3(\mathbf{X}) = \delta_l - \delta_{max} \geq 0 \tag{29}$$

$$g_4(\mathbf{X}) = H - h - y \geq 0 \tag{30}$$

$$g_5(\mathbf{X}) = D_{max} - D_e \geq 0 \tag{31}$$

$$g_6(\mathbf{X}) = D_e - D_i \geq 0 \tag{32}$$

$$g_7(\mathbf{X}) = 0.3 - \frac{h}{D_e - D_i} \geq 0 \tag{33}$$

a	f(a)
≤ 1.4	1
1.5	0.85
1.6	0.77
1.7	0.71
1.8	0.66
1.9	0.63
2.0	0.60
2.1	0.58
2.2	0.56
2.3	0.55
2.4	0.53
2.5	0.52
2.6	0.51
2.7	0.51
≥ 2.8	0.50

Table 2: Values of a and f(a) for the second example.

The parameters are defined as follows: the maximum load is $P_{max} = 5400$ lb (2448.98 Kg), the maximum deflection is $\delta_{max} = 0.2$ in (0.508 cm), the allowable strength is $S = 200$ kPsi (1379.31 MPa), the modulus of elasticity $E = 30 \times 10^6$ psi (206897 MPa), Poisson's ratio for the material used is $\mu = 0.3$, the maximum limit on the overall height is $H = 2$ in (5.08 cm), and the outside diameter of the spring is $D_{max} = 12.01$ in (30.505 cm). Assuming $K = D_e/D_i$, then:

$$\alpha = \frac{6}{\pi \ln K}\left(\frac{K-1}{k}\right)^2 \tag{34}$$

$$\beta = \frac{6}{\pi \ln K}\left(\frac{K-1}{k} - 1\right) \tag{35}$$

$$\gamma = \frac{6}{\pi \ln K}\left(\frac{K-1}{2}\right) \tag{36}$$

Finally, the limiting deflection is $\delta_l = f(a)h$, where $a = h/t$, and $f(a)$ is defined according to the values given in Table 2.

4.3 Example 3 :

The third example consists of designing the combinational circuit represented by the function with 3 inputs and 1 output shown in Table 3. To solve this problem, a bidimensional matrix representation of the circuit will be encoded in each chromosomic string as shown in Figure 4 [29]. In this matrix, each element is a gate (there are 5 types of gates: AND, NOT, OR, XOR and WIRE) that receives its 2 inputs from any gate at the previous column.

More formally, we can say that the circuit will be represented as a bidimensional array of gates $S_{i,j}$, where j indicates the *level* of a gate, so that those gates closer to the inputs have lower values of j. (Level values are incremented from left to right in Figure 4). For a fixed j, the index i varies with respect to the gates that are "next" to each other in the circuit, but without being necessarily connected. It is interesting to notice that if a row-order encoding is used, the problem becomes disruptive [30], making

X	**Y**	**Z**	**F**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Table 3: Truth table for the circuit of the third example.

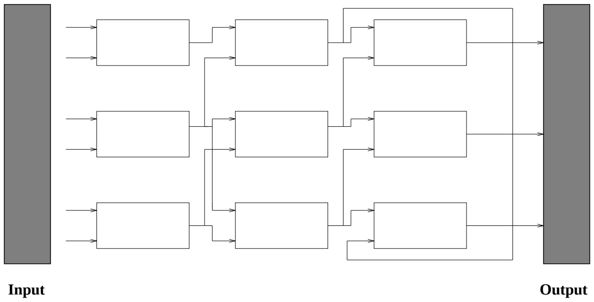

Figure 4: A gate in a two-dimensional template, gets its second input from either one of two gates in the previous column.

Input 1	Input 2	Gate Type

Figure 5: Encoding used for each of the matrix elements that represent a circuit.

it very hard for the GA. The reason is that using such an encoding, any circuit designs that are close in two-dimensional (phenotypic) space may be far apart in one-dimensional (genotypic) space, making it difficult to preserve highly fit schemas (in GA terminology, we say that the problem is deceptive [31]).

A chromosomic string will then encode the matrix shown in Figure 4 by using triplets in which the 2 first elements refer to each of the inputs used, and the third is the corresponding gate as shown in Figure 5 (only 2-input gates were used in this work).

The goal in this example was then to produce a fully functional design (i.e., one that produced all the expected outputs for any combination of inputs according to the truth table given for the problem) which maximized the number of WIREs[1].

In this example, there are $2^3 = 8$ equality constraints, each corresponding to a desired output from the truth table shown above (see Table 3).

4.4 Example 4 : Himmelblau's Nonlinear Optimization Problem

This problem was originally proposed by Himmelblau [32], and it was chosen to try the new approach because it has been used before as a benchmark for GA-based techniques that use penalties. In this problem, there are 5 design variables $(x_1, x_2, x_3, x_4, x_5)$, 6 nonlinear inequality constraints and 10 boundary conditions. The problem can be stated as follows:

$$Minimize \quad f(\mathbf{X}) = 5.3578547x_3^2 + 0.8356891x_1x_5 \tag{37}$$

Subject to:

$$g_1(\mathbf{X}) = 85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5 \tag{38}$$

$$g_2(\mathbf{X}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \tag{39}$$

$$g_3(\mathbf{X}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \tag{40}$$

$$0 \leq g_1(\mathbf{X}) \leq 92 \tag{41}$$

$$90 \leq g_2(\mathbf{X}) \leq 110 \tag{42}$$

$$20 \leq g_3(\mathbf{X}) \leq 25 \tag{43}$$

$$78 \leq x_1 \leq 102 \tag{44}$$

$$33 \leq x_2 \leq 45 \tag{45}$$

$$27 \leq x_3 \leq 45 \tag{46}$$

$$27 \leq x_4 \leq 45 \tag{47}$$

$$27 \leq x_5 \leq 45 \tag{48}$$

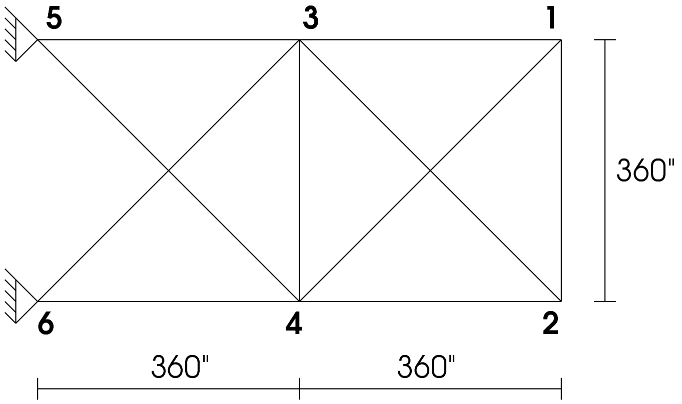

Figure 6: 10-bar plane truss used for Example No. 5.

4.5 Example 5 : Design of a 10-bar plane truss

Consider the 10-bar plane truss shown in Figure 6 [33]. The problem is to find the cross-sectional area 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:

$$f(x) = \sum_{j=1}^{10} \rho A_j L_j \tag{49}$$

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 cross-section of each element can be different, thus the problem has 10 design variables.

5 COMPARISON OF RESULTS

The genetic algorithm used for the experiments presented in this paper uses a fixed-point representation [34, 35], according to which a chromosome is a string of the form $\langle d_1, d_2, \ldots, d_m \rangle$, where $d_1, d_2, \ldots, d_m$ are digits (numbers between zero and nine). Consider the examples shown in Figure 7, in which the same value is represented using binary and fixed point encoding.

---

[1] WIRE basically indicates a null operation, or in other words, the absence of gate, and it is used just to keep regularity in the representation used by the GA that otherwise would have to use variable-length strings.

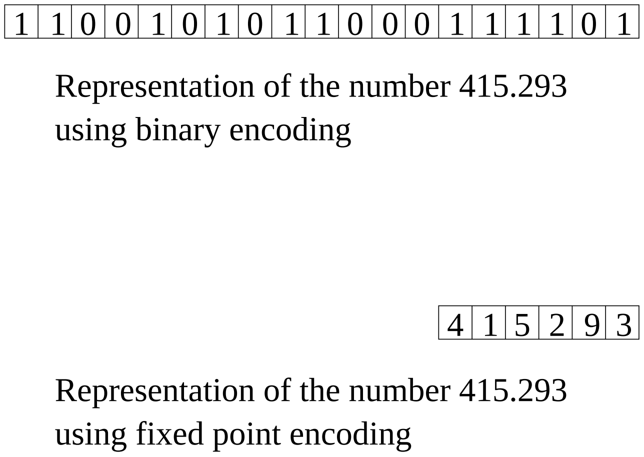
1 1 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0 1
Representation of the number 415.293
using binary encoding
4 1 5 2 9 3
Representation of the number 415.293
using fixed point encoding

Figure 7: Representing the same number using binary and fixed point encodings.

Fixed point representation is faster and easier to implement, and provides a higher precision than its binary counterpart, particularly in large domains, where binary strings would be prohibitively long. One of the advantages of fixed point representation is that it has the property that two points close to each other in the representation space must also be close in the problem space, and vice versa [8]. This is not generally true in the binary approach, where the distance in a representation is normally defined by the number of different bit positions.

For crossover, it was decided to use uniform crossover [36], which can be seen as a generalization of the more traditional one-point and two-point crossover operators [1, 8]. In this case, for each gene (i.e., string position) in the first offspring it is decided (with some probability p) which parent will contribute its value for that position. The second offspring would receive the gene from the other parent. An example of 0.5-uniform crossover can be seen in Figure 8.

Instead of using the traditional uniform mutation operator, for the purposes of this work, it was decided to use non-uniform mutation [8]. To illustrate this operator, let's assume that at generation t, we have a string $S_t = \langle s_1, s_2, \ldots, s_l \rangle$. After randomly selecting a position along the string, in generation $t+1$, the new chromosome after mutation will be $S_{t+1} = \langle s_1, s_2, \ldots, s'_k, \ldots, s_l \rangle$, where:

$$s'_k = \begin{cases} s_k + \Delta(t, 9 - s_k) & if\ flip(0.5) = 0 \\ s_k - \Delta(t, s_k) & if\ flip(0.5) = 1 \end{cases} \tag{50}$$

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 by the author is the function originally suggested by Michalewicz [8]:

$$\Delta(t, y) = y \cdot \left(1 - r^{\left(1 - \frac{t}{T}\right)^b}\right) \tag{51}$$

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

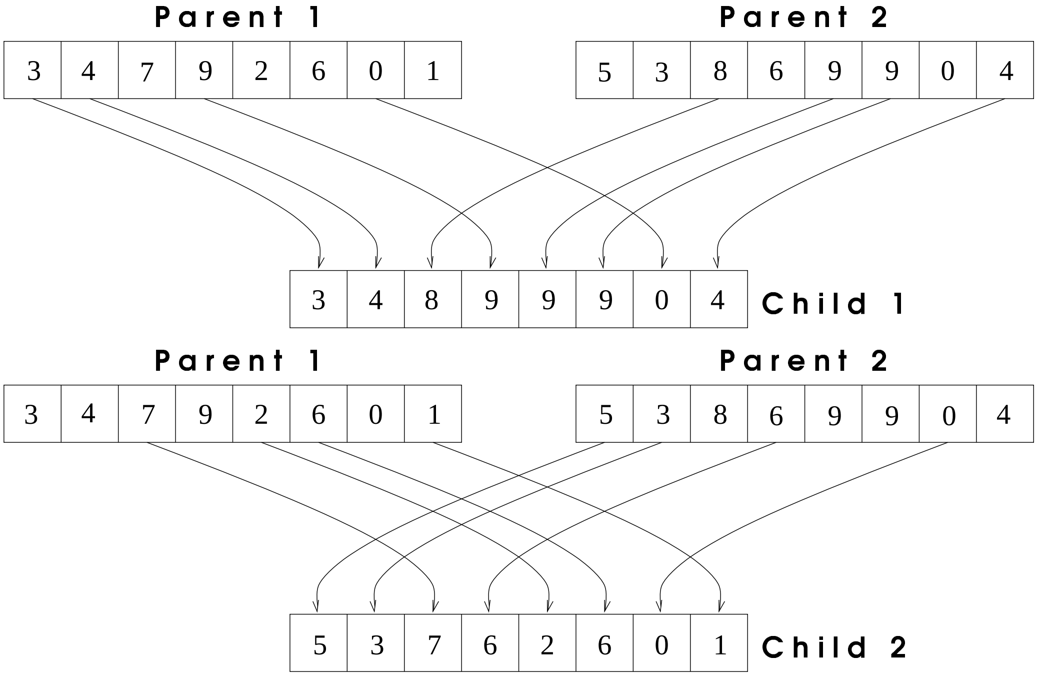

Figure 8: Use of 0.5-uniform crossover (using 50% probability) between two chromosomes. Notice how half of the genes of each parent goes to each of the two children. First, the bits to be copied from each parent are selected randomly using the probability desired, and after the first child is generated, the same values are used to generate the second child, but inverting the source of procedence of the genes.

Design	**Best solution found**			
Variables	**This paper**	**GeneAS**	**BGA**	**Siddall**
$x_1(R)$	6.271	6.778	7.077	7.155
$x_2(R_0)$	12.901	6.234	6.549	6.689
$x_3(\mu) \times 10^{-6}$	5.605	6.096	6.619	8.321
$x_4(Q)$	2.938	3.809	4.849	9.168
$g_1(\mathbf{X})$	2126.86734	8329.7681	1440.6013	-11086.7430
$g_2(\mathbf{X})$	68.0396	177.3527	297.1495	402.4493
$g_3(\mathbf{X})$	3.705191	10.684543	17.353800	35.057196
$g_4(\mathbf{X})$	0.000559	0.000652	0.000891	0.001542
$g_5(\mathbf{X})$	0.666000	0.544000	0.528000	0.466000
$g_6(\mathbf{X})$	0.000805	0.000717	0.000624	0.000144
$g_7(\mathbf{X})$	849.718683	83.618221	467.686527	563.644401
$f(\mathbf{X})$	**1950.2860**	**2161.4215**	**2296.2119**	**2288.2268**

Table 4: Comparison of results for the first example (design of a hydrostatic thrust bearing).

generation number. The value adopted for the current implementation was $b = 5$, as suggested by Michalewicz [8]. Binary tournament selection was used for all the examples presented next.

5.1 Example 1

This problem was solved before by Deb and Goyal [37] using GeneAS (Genetic Adaptive Search, which is a real-coded GA) and a traditional (binary) genetic algorithm (both approaches used a penalty function), and by Siddall [38] using ADRANS (Gall's adaptive random search with a penalty function). Their results were compared against those produced by the approach proposed in this paper, and are shown in Table 4. Notice that the solution reported by Siddall [39] is infeasible (it violates the first constraint). The solution shown for the technique proposed here is the best produced after 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 (at increments of 0.1) in a nested loop in which the mutation rate was iterated first while the crossover rate remained fixed. For example, given an initial crossover rate of 0.1, the nine values from 0.1 to 0.9 were used for the mutation rate, running a GA for each pair of parameters (crossover and mutation rates). Once the loop for the mutation rate finished, the crossover rate was incremented by 0.1 and the loop for the mutation rate was run again from 0.1 to 0.9. The process was repeated until the crossover rate reached 0.9.

The following ranges were used for the design variables: $1.000 \leq x_1 \leq 16.000$, $1.000 \leq x_2 \leq 16.000$, $1.0 \times 10^{-6} \leq x_3 \leq 16.0 \times 10^{-6}$, $1.000 \leq x_4 \leq 16.000$. The values of all variables were considered with a 3-decimal precision (the values of x_3 were considered as multiples of 1.0×10^{-6}). The total population size used was 160 (20 individuals for each of the 8 sub-populations) and the maximum number of generations was 100.

5.2 Example 2

This problem was solved before by Deb [37] using two versions of GeneAS (one that treated the variables as continuous—GeneAS I—and another that treated the variables as discrete—GeneAS II) and a traditional (binary) genetic algorithm (both approaches used a penalty function), and by Siddall [38] using APPROX (Griffith and Stewart's successive linear approximation). Their results were compared against those produced by the approach proposed in this paper, and are shown in Table 5. Notice

Design	**Best solution found**			
Variables	**This paper**	**GeneAS I**	**GeneAS II**	**Siddall**
$x_1(t)$	0.208	0.205	0.210	0.204
$x_2(h)$	0.200	0.201	0.204	0.200
$x_3(D_i)$	8.751	9.534	9.268	10.030
$x_4(D_e)$	11.067	11.627	11.499	12.010
$g_1(\mathbf{X})$	2145.4109	-10.3396	2127.2624	134.0816
$g_2(\mathbf{X})$	39.75018	2.8062	194.222554	-12.5328
$g_3(\mathbf{X})$	0.00000	0.0010	0.0040	0.0000
$g_4(\mathbf{X})$	1.592	1.5940	1.5860	1.5960
$g_5(\mathbf{X})$	0.943	0.3830	0.5110	0.0000
$g_6(\mathbf{X})$	2.316	2.0930	2.2310	1.9800
$g_7(\mathbf{X})$	0.21364	0.20397	0.20856	0.19899
$f(\mathbf{X})$	**2.121964**	**2.01807**	**2.16256**	**1.978715**

Table 5: Comparison of results for the second example (design of a Belleville spring).

Genetic Algorithm	**Human Designer**	**ESPRESSO**
$F = (X + Z)(Y \oplus YZ)$	$F = Z(X \oplus Y) + Y(X \oplus Z)$	$F = XY\overline{Z} + X\overline{Y}Z + \overline{X}YZ$
4 gates	5 gates	11 gates
2 ANDs, 1 OR, 1 XOR	2 ANDs, 1 OR, 2 XORs	6 ANDs, 2 ORs, 3 NOTs

Table 6: Comparison of results between a human designer and the approach proposed in this paper

that the solution reported by Deb and Goyal [37] for GeneAS I violates the first constraint, and the solution reported by Siddall [38] violates the second constraint (these results may vary depending on the precision used and the value of π employed in the equations, but the results shown in Table 5 were all computed using the same precision). The solution shown for the technique proposed here is the best produced after 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 in a nested loop, and the following ranges were used for the design variables: $0.010 \leq x_1 \leq 6.000$, $0.050 \leq x_2 \leq 0.500$, $5.000 \leq x_3 \leq 15.000$, $5.000 \leq x_4 \leq 15.000$. The values of all variables were considered with a 3-decimal precision. The total population size used was 160 (20 individuals for each of the 8 sub-populations) and the maximum number of generations was 150.

5.3 Example 3

This example was solved by hand using Karnaugh Maps [40] by an experienced designer. His solution required 5 gates, as shown in Table 6. Also, a computer program called ESPRESSO [41] was used to compare the results produced by the approach proposed in this paper. It should be mentioned that ESPRESSO really tries to produce a minumum representation of a circuit but using only two levels, regardless of the number of inputs that each gate receives. Therefore, it is difficult to compare its results with those found by the current technique, because ESPRESSO does not constraint itself to operate with two-input gates, nor uses the XOR gate. In the expression presented in Table 6, each of the 3-input gates used was actually split into 2-input gates to make a fair comparison with the GA, but the fact that ESPRESSO does not handle directly this type of gates obviously makes its results more complex than those produced with the GA.

Design Variables	**Best solution found**			
	This paper	**Gen**	**Homaifar**	**GRG**
x_1	78.5958	81.4900	78.0000	78.6200
x_2	33.0100	34.0900	33.0000	33.4400
x_3	27.6460	31.2400	29.9950	31.0700
x_4	45.0000	42.2000	45.0000	44.1800
x_5	45.0000	34.3700	36.7760	35.2200
$g_1(\mathbf{X})$	91.956402	90.522543	90.714681	90.520761
$g_2(\mathbf{X})$	100.545111	99.318806	98.840511	98.892933
$g_3(\mathbf{X})$	20.251919	20.060410	19.999935	20.131578
$f(\mathbf{X})$	**−30810.359**	**−30183.576**	**−30665.609**	**−30373.949**

Table 7: Comparison of results for the fourth example (Himmelblau's function).

The fitness function for this problem worked in two stages. At the beginning of the search, only validity of the circuit outputs was taken into account, and the GA was basically exploring the search space. Once a functional solution appeared, then the fitness function was modified such that any valid designs produced were rewarded for each WIRE gate that they included, so that the GA tried to find the circuit with the maximum number of gates that performed the function required. It's at this stage that the GA was actually exploiting the search space, trying to optimize the solutions found (in terms of their number of gates) as much as possible. Each of the outputs desired was treated as an equality constraint (as indicated before), and the GA used a population of 360 chromosomes (40 individuals for each of the 9 sub-populations), with a maximum number of generations of 100. The size of the matrix used was 5×5.

5.4 Example 4

This problem was originally proposed by Himmelblau [32] and solved using the Generalized Reduced Gradient method (GRG). Gen and Cheng [42] solved this problem using a genetic algorithm based on both local and global reference. The result shown in Table 7 is the best of the two reported by Gen and Cheng [42].

Homaifar, Qi, and Lai [14] solved this problem using a genetic algorithm with a population size of 400, and their results were the best previously reported in the literature for this problem (see Table 7).

The solution shown for the technique proposed here is the best produced after 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 in a nested loop, and the following ranges were used for the design variables: $78.0000 \leq x_1 \leq 102.0000$, $33.0000 \leq x_2 \leq 45.0000$, $27.0000 \leq x_3 \leq 45.0000$, $27.0000 \leq x_4 \leq 45.0000$, and $27.0000 \leq x_5 \leq 45.0000$. The values for all the variables were considered with a 4-decimal precision. The total population size used was 160 (40 individuals for each of the 4 sub-populations) and the maximum number of generations was 100.

5.5 Example 5

This problem was used by Belegundu [33] 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 [33] are compared to the current approach in Tables 8 and 9 (all

Design Variables	**Best solution found**			
	This paper	**CONMIN**	**OPTDYN**	**LINRM**
x_1	30.00	25.28	25.77	21.57
x_2	0.10	1.90	0.10	10.98
x_3	22.40	24.87	25.11	22.08
x_4	16.19	15.83	19.39	14.95
x_5	0.10	0.10	0.10	0.10
x_6	0.57	1.75	0.10	10.98
x_7	7.74	16.76	15.36	18.91
x_8	22.15	19.73	20.32	18.42
x_9	20.80	20.98	20.74	18.40
x_{10}	0.10	2.51	1.14	13.51
$f(\mathbf{X})$	**5082.76**	**5563.32**	**5471.48**	**6428.89**

Table 8: Comparison of results for the fifth example (10-bar plane truss). Part I.

Design Variables	**Best solution found**				
	GRP-UI	**SUMT**	**M-3**	**M-4**	**M-5**
x_1	24.78	30.69	25.84	31.62	25.84
x_2	4.17	2.37	3.07	11.81	2.88
x_3	24.79	31.62	26.42	31.62	26.45
x_4	14.45	11.66	12.77	17.50	12.75
x_5	0.10	0.10	0.10	31.62	0.10
x_6	4.17	3.71	3.44	10.25	3.77
x_7	17.46	21.71	19.34	31.62	19.38
x_8	19.26	20.90	19.17	31.62	19.18
x_9	19.27	13.97	18.76	31.62	18.77
x_{10}	5.26	3.26	4.42	31.62	4.38
$f(\mathbf{X})$	**5727.05**	**5932.21**	**5719.19**	**11279.22**	**5726.08**

Table 9: Comparison of results for the fifth example (10-bar plane truss). Part II.

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 [43] together with the stiffness method [43] 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 81 runs in which the crossover and mutation rates were iterated from 0.1 to 0.9 in a nested loop, and the range $0.1 \leq x \leq 299.00$ was used for the 10 design variables. The values for all the variables were considered with a 2-decimal precision. The total population size used was 230 (10 individuals for each of the 23 sub-populations) and the maximum number of generations was 100.

6 DISCUSSION

In the examples presented before, the new approach found better solutions than those previously reported in the literature by using relatively small sub-population sizes. However, the selection of an appropriate sub-population size (assuming that they are all the same) remains an issue as when using a GA with a single population.

Determining the maximum number of generations presents a similar problem, although in this case it is possible to monitor the population so that the GA is stopped when there is not enough diversity anymore (e.g., when the fitness of the best individual is very close to the mean fitness of the population).

The problems selected to illustrate the technique had different kinds of constraints so that the new algorithm could be tested under different conditions. Examples 1 and 2 for instance, are engineering design problems, with their constraints given in algebraic form. Example 4 is a numerical optimization problem that has been used several times before to test constraint handling approaches. Example 3 has equality constraints which are not given in algebraic form, but are instead derived from a module that simulates the circuit produced for all possible input combinations. Finally, example 5 does not have its constraints defined in algebraic form either, since they are derived from the module that performs the analysis of the structure.

The main drawback of the new technique 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, we could parallelize the approach, in which case a high number of sub-populations will 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). That is in fact the area of research currently being pursued by the author.

7 CONCLUSIONS

This paper has introduced a new GA-based approach that uses a multiobjective optimization technique to handle constraints, instead of using the more traditional penalty approach.

The new approach worked well in several test problems that had been previously solved using GA-based and mathematical programming techniques, producing in all cases results better than those previously reported in the literature.

The technique was able to achieve such good results with relatively small sub-populations, and without the need to use any extra parameters for the GA, although the issue of selecting the most appropriate sub-population size as well as the maximum number of generations, remains open as in the case of the simple GA [1].

8 FUTURE WORK

The first extension of this work is to develop a parallel implementation of the algorithm, so that instead of using a single population and split it according to the number of constraints, several (fairly small) sub-populations are generated, each being responsible for a single constraint or set of constraints. Some interesting issues that a parallel version of this algorithm arise are for example the migration policies required to exchange information, the consequences of restricting crossover, the effect of the topology used by the parallel architecture on the overall performance of the GA, the significance of the evolution of the small sub-populations responsible for the constraints concurrently with the evolution of a main population containing a mixed of feasible and infeasible solutions.

The introduction of a parallel version of the current algorithm may improve its overall performance, but it may also require a more detailed analysis of the parameters of the GA and the architecture used to implement it as to evaluate the robustness of the approach.

References

[1] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Co., Reading, Massachusetts.

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

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

[4] Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197. Morgan Kaufmann Publishers, George Mason University.

[5] Dasgupta, D. and Michalewicz, Z., editors (1997). *Evolutionary Algorithms in Engineering Applications.* Springer-Verlag, Berlin.

[6] Schwefel, H. P. (1981). *Numerical Optimization of Computer Models.* John Wiley and sons, Great Britain.

[7] Davis, L., editor (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, New York.

[8] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second edition.

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

[10] Schoenauer, M. and Xanthakis, S. (July 1993). Constrained GA Optimization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580. Morgan Kauffman Publishers, San Mateo, California.

[11] Michalewicz, Z., Dasgupta, D., Riche, R. L., and Schoenauer, M. (September 1996). Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering Journal*, 30(4):851–870.

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

[13] Riche, R. L., Knopf-Lenoir, C., and Haftka, R. T. (July 1995). A Segregated Genetic Algorithm for Constrained Structural Optimization. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 558–565. University of Pittsburgh, Morgan Kaufmann Publishers, San Mateo, California.

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

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

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

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

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

[19] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts.

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

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

[22] Bean, J. C. and Hadj-Alouane, A. B. (1992). A Dual Genetic Algorithm for Bounded Integer Programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

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

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

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

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

[27] Schaffer, J. D. (1985). 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.

[28] Parmee, I. C. and Purchase, G. (1994). 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. University of Plymouth, University of Plymouth, Plymouth, UK.

[29] Coello, C. A. C., Christiansen, A. D., and Aguirre, A. H. (April 1997). Automated design of combinational logic circuits using genetic algorithms. In D. G. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms ICANNGA'97*, pages 335–338. University of East Anglia, Springer-Verlag, Norwich, England.

[30] Louis, S. J. and Rawlins, G. J. (feb 1991). Using genetic algorithms to design structures. Technical Report 326, Computer Science Department, Indiana University, Bloomington, Indiana.

[31] Grefenstette, J. J. (1993). Deception Considered Harmful. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 75–91. Morgan Kaufmann, San Mateo, California.

[32] Himmelblau, D. M. (1972). *Applied Nonlinear Programming*. McGraw-Hill, New York.

[33] Belegundu, A. D. (1982). *A Study of Mathematical Programming Methods for Structural Optimization*. Dept. of civil and environmental engineering, University of Iowa, Iowa, Iowa.

[34] Coello, C. A. C., Hernández, F. S., and Farrera, F. A. (January 1997). Optimal design of reinforced concrete beams using genetic algorithms. *Expert Systems with Applications : An International Journal*, 12(1).

[35] Coello, C. A. C. and Christiansen, A. D. (1997). A simple genetic algorithm for the design of reinforced concrete beams. *Engineering with Computers*, 13(4):185–196.

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

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

[38] Siddall, J. N. (1982). *Optimal Engineering Design. Principles and Applications*. Marcel Dekker, New York.

[39] Siddall, J. N. (1972). *Analytical Design-Making in Engineering Design*. Prentice-Hall.

[40] Karnaugh, M. (November 1953). A map method for synthesis of combinational logic circuits. *Transactions of the AIEE, Communications and Electronics*, 72 (I):593–599.

[41] Brayton, R. K., Hachtel, G. D., McMullen, C. T., and Sangiovanni-Vincentelli, A. L. (1985). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers.

[42] Gen, M. and Cheng, R. (1997). *Genetic Algorithms & Engineering Design*. John Wiley & Sons, Inc, New York.

[43] Gere, J. M. and Weaver, W. (1965). *Analysis of Framed Structures*. D. Van Nostrand Company, Inc.