A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms

Kalyanmoy Deb and Samir Agrawal
Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering,
Indian Institute of Technology Kanpur, PIN 208 016, India
E-mail: {deb,samira}@iitk.ac.in

Abstract

Most applications of genetic algorithms (GAs) in handling constraints use a straightforward penalty function method. Such techniques involve penalty parameters which must be set right in order for GAs to work. Although many researchers use adaptive variation of penalty parameters and penalty functions, the general conclusion is that these variations are specific to a problem and cannot be generalized. In this paper, we propose a niched-penalty approach which does not require any penalty parameter. The penalty function creates a selective pressure towards the feasible region and a niching maintains diversity among feasible solutions for the genetic recombination operator to find new feasible solutions. The approach is only applicable to population-based approaches, thereby giving GAs (or other evolutionary algorithms) a niche in exploiting this penalty-parameter-less penalty approach. Simulation results on a number of constrained optimization problems suggest the efficacy of the proposed method.

1 Introduction

Real-world search and optimization problems are written as nonlinear programming (NLP) problem of the following kind [2, 16]:

$$\begin{array}{lll} \text{Minimize} & f(\vec{x}) & \\ \text{Subject to} & g_j(\vec{x}) \geq 0, & j = 1, \ldots, J, \\ & h_k(\vec{x}) = 0, & k = 1, \ldots, K, \\ & x_i^l \leq x_i \leq x_i^u, & i = 1, \ldots, n. \end{array} \quad (1)$$

In the above NLP problem, there are n variables (that is, $\vec{x}$ is a vector of size n), J greater-than-equal-to type inequality constraints, and K equality constraints. The function $f(\vec{x})$ is the objective function, $g_j(\vec{x})$ is the j-th inequality constraints, and $h_k(\vec{x})$ is the k-th equality constraints. The i-th variable varies in the range $[x_i^l, x_i^u]$. A solution $\vec{x}$ that satisfies all the above equality and inequality constraints and above variable bounds is called a *feasible* solution. Other solutions are called *infeasible* solutions.

Most classical search and optimization methods handle above NLP problems by using a penalty function, where infeasible solutions are penalized depending on the amount of constraint violation. Although there exist a number of other specialized constraint handling techniques which are applicable to only a particular type of constraints [2, 16], the following simple procedure is generic and most popular [2]:

$$P(\vec{x}, \vec{R}, \vec{r}) = f(\vec{x}) + \sum_{j=1}^{J} R_j \langle g_j(\vec{x}) \rangle^2 + r_k \sum_k [h_k(\vec{x})]^2, \quad (2)$$

where $\langle\ \rangle$ denotes the absolute value of the operand, if the operand is negative and returns a value zero, otherwise. The parameter R_j is the penalty parameter of the j-th inequality constraint and r_k is the penalty parameter for k-th equality constraint.

The above penalized function $P(\vec{x}, \vec{R}, \vec{r})$ makes the constrained NLP problem of equation 1 into a unconstrained minimization problem. Optimization of $P(\vec{x}, \vec{R}, \vec{r})$ largely depends on the penalty parameter $\vec{R}$ and $\vec{r}$. The optimal solution of $P(\vec{x}, \vec{R}, \vec{r})$ is close to the true constrained optimal solution of equation 1 for very large values of $\vec{R}$ and $\vec{r}$. But using large values of $\vec{R}$ and $\vec{r}$ does not emphasize the objective function $f(\vec{x})$ and thus most effort is spent in finding feasible solutions. Classical gradient-based methods suffer from computation of meaningless gradients for large $\vec{R}$ and $\vec{r}$ values [2]. Although GAs do not use gradients, a different scenario happens. Since too much emphasis is given to feasible solutions, GAs usually prematurely converge around the feasible solutions found early on in the simulation. In most GA simulations using the above penalty function approach, researchers usually experiment to find a correct combination of penalty parameters $\vec{R}$ and $\vec{r}$. However, in order to reduce the number of penalty parameters, often the constraints are normalized and only one penalty parameter R is used [2]:

$$P(\vec{x}, R) = f(\vec{x}) + R \left[\sum_j \langle \bar{g}_j(\vec{x}) \rangle^2 + \sum_k [\bar{h}_k(\vec{x})]^2 \right], \quad (3)$$

Even in this case, fixing a correct penalty parameter R is an essential element of a successful simulation.

The importance of the penalty parameter in obtaining any reasonable solution, leave alone an optimal solution, is evident from a plethora of research in designing

penalty parameters for a problem. Homaifar et al. [10] designed a multi-level penalty function, depending on the level of constraint violations. Joines and Hauck [11] used a dynamic penalty parameter, which is varied with generation. Michalewicz and Attia [13] updated penalty parameters based on a temperature-based evolution used in simulated annealing techniques. Michalewicz and Schoenauer [14] suggested specific recombination operators which preserve feasibility of solutions for some specific type of constraints. All these research suggests that constraint handling techniques used in GAs are still problem-specific and one technique may work in few problems, but may not work as well in other problems.

In this paper, we develop a constraint handling method based on penalty function approach which does not require any penalty parameter. The pair-wise comparison used in tournament selection is exploited to make sure that (i) when two feasible solutions are compared the one with better objective function value is chosen, (ii) when one feasible and one infeasible solutions are compared, the feasible solution is chosen, and (iii) when two infeasible solutions are compared, the one with smaller constraint violation is chosen. This approach is only applicable to population-based search methods such as GAs or other evolutionary computation methods.

In the remainder of the paper, we present the performance of GAs with the proposed constraint handling method on five test problems, including one engineering design problem. The results are also compared with the best-known solutions obtained using earlier GA implementations or using classical methods.

2 Proposed Constraint Handling Method

Constraints are handled by using a suitable fitness function which depends on the current population. Solutions in a population are assigned a fitness so that feasible solutions are emphasized more than infeasible solutions. The proposed method uses tournament selection operator where two solutions are chosen from the population and one is chosen. The following three criteria are satisfied during the selection operator:

1. Any feasible solution wins over any infeasible solution,
2. Two feasible solutions are compared only based on their objective function values.
3. Two infeasible solutions are compared based on the amount of constraint violations.

Although there exist a number of implementations [12, 15, 17] where these criteria are imposed in their constraint handling approaches, all of these implementations used different measures of constraint violations which still needed a penalty parameter for each constraint.

In the proposed method, we choose a constraint violation measure from a practical standpoint. In order to evaluate a solution, any designer will first check if the solution is feasible. If the solution is infeasible (that is, at least one constraint is violated), the designer will never bother to compute its objective function value (such as cost of the design). It does not make sense to compute the objective function value of an infeasible solution, because the solution simply cannot be implemented in practice. Motivated by this argument, we device the following penalty term where infeasible solutions are compared based on only their constraint violation values:

$$F(\vec{x}) = \begin{cases} f(\vec{x}), & \text{if } g_j(\vec{x}) \geq 0,\ \forall j \in J, \\ f_{\max} + \sum_{j=1}^{J} \langle g_j(\vec{x}) \rangle, & \text{otherwise.} \end{cases} \tag{4}$$

The parameter $f_{\max}$ is the maximum function value of all feasible solutions in the population. Let us illustrate this constraint handling technique on a single-variable constrained minimization problem, shown in Figure 1. In the figure, the unconstrained minimum solution is not

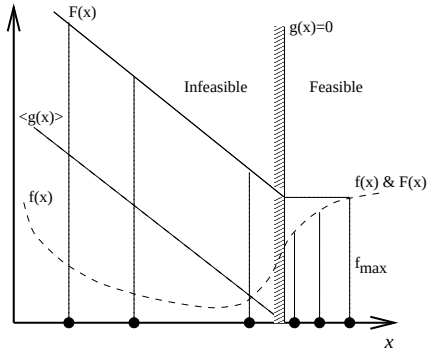

Fig. 1: The proposed constraint handling scheme is illustrated. Six solid circles are solutions in a GA population.

feasible. The objective function $f(\vec{x})$, constraint violation $\langle g(\vec{x}) \rangle$, and the fitness function $F(\vec{x})$ are shown in the figure. It is important to note that $F(\vec{x}) = f(\vec{x})$ in the feasible region. When tournament selection operator is applied to a such a fitness function $F(\vec{x})$, all three criteria mentioned above will be satisfied and there will be selective pressure towards the feasible region. The figure also shows how the fitness value of six arbitrary solutions will be calculated. Thus, under this constraint handling scheme, the fitness value of an infeasible solution may change from one generation to another, but the fitness value of a feasible solution will always be the same. Since the above constraint handling method assigns a hierarchy to infeasible solutions and tournament

selection does not depend on the exact fitness values, instead their relative difference is important, any arbitrary penalty parameter will work the same. In fact, there is no need of any explicit penalty parameter. This is a major advantage of the proposed method over earlier penalty function implementations. However, to avoid any bias from any particular constraint, all constraints are normalized (a usual practice in constrained optimization [2]) and equation 4 is used. It is important to note that such a constraint handling scheme without the need of a penalty parameter is possible because GAs use a population of solutions and pair-wise comparison of solutions is possible using the tournament selection. For the same reason, such schemes cannot be used with classical point-by-point search and optimization methods.

The above constraint handling method seems interesting and eliminate the need of any penalty parameter. But, there are two other aspects that must also be used to qualify this method as an efficient constraint handling methods.

2.1 Use real-coded GAs

It is intuitive that the feasible region in constrained optimization problems may be of any shape (convex or concave and connected or disjointed). In real-parameter constrained optimization using GAs, schemata specifying contiguous regions in the search space (such as (`110`$*\ldots*$) may be considered to be generally more important than schemata specifying discrete regions in the search space (such as ($*$`1`$*$`10`$*\ldots*$). In a binary GA under a single-point crossover operator, all common schemata corresponding to both parent strings are preserved in both children strings. Since, any arbitrary contiguous region in the search space cannot be represented by a single Holland's schema and since the feasible search space can usually be of any arbitrary shape, it is expected that the crossover operator used in binary GAs may not always be able to create feasible children solutions from two feasible parent solutions.

However, the floating-point representation of variables in a GA and a search operator that respects contiguous regions in the search space may be able to perform better than binary GAs in constrained optimization problems with continuous search space. In this paper, we use real-coded GAs with simulated binary crossover (SBX) operator and a parameter-based mutation operator [3], for this purpose. We present procedures of these operators in the Appendix. SBX operator is particularly suitable here, because the spread of children solutions around parent solutions can be controlled using a distribution index η_c. A large value of η_c allows only near-parent solutions to be created, whereas a small value of η_c allows distant solutions to be created. Another aspect of this crossover operator is that it is adaptive, allowing any solution to be created in the beginning and allowing to have a more focussed search when the population is converging.

2.2 Use a Niching Technique

With real-coded GAs having SBX operator any arbitrary contiguous region can be searched, provided there are enough diversity maintained among the feasible parent solutions. There are a number of ways diversity can be maintained in a population. Among them, sharing methods [4] and use of mutation are popular ones. In this paper, we use either or both of the above methods of maintaining diversity among the feasible solutions. A simple sharing strategy is implemented in the tournament selection operator. When comparing two feasible solutions (i and j), a normalized Euclidean distance d_{ij} is measured between them. If this distance is smaller than a critical distance $\bar{d}$, the solutions are compared with their objective function values. Otherwise, they are not compared and another solution j is checked. The normalized Euclidean distance is calculated as follows:

$$d_{ij} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(\frac{x_k^{(i)} - x_k^{(j)}}{x_k^u - x_k^l}\right)^2}. \qquad (5)$$

This way, the solutions that are far away from each other are not compared to each other and diversity among feasible solutions is maintained.

3 Results

In this section, we apply GAs with the proposed constraint handling method to five different constrained optimization problems that have been studied in the literature.

In all problems, we run GAs 50 times from different initial populations. Fixing a correct population size in a GA is as important as anything else. Taking the cue from previous studies on population sizing [6, 7], we use a population size which linearly varies with the number of variables (we use $N = 10n$, where n is the number of variables in a problem). In all problems, we use binary tournament selection operator without replacement. We use a crossover probability of 0.9. SBX operator with a distribution index of $\eta_c = 1$. In the real-coded mutation, the amount of perturbation in a variable can be controlled by fixing a parameter η_m [5]. The probability of mutation is varied with generation number so that initially only one variable is expected to get mutated for each solution with larger perturbance and at the specified maxi-

mum generation, all variables are mutated with a smaller perturbance. For sharing operator, we use $\bar{d} = 0.1$.

3.1 Test Problem 1

To investigate the efficacy of the proposed constraint handling method, we first choose a two-dimensional constrained minimization problem:

$$\begin{array}{ll} \text{Minimize} & f_1(\vec{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \\ \text{Subject to} & g_1(\vec{x}) \equiv 4.84 - x_1^2 - (x_2 - 2.5)^2 \geq 0, \\ & g_2(\vec{x}) \equiv (x_1 - 0.05)^2 + (x_2 - 2.5)^2 - 4.84 \geq 0, \\ & 0 \leq x_1 \leq 6, 0 \leq x_2 \leq 6. \end{array} \tag{6}$$

The unconstrained objective function $f_1(\vec{x})$ has a minimum solution at $\vec{x} = (3, 2)$ with a function value equal to zero. However, due to the presence of constraints, this solution is no more feasible and the constrained optimum solution is $\vec{x}^* = (2.246826, 2.381865)$ with a function value equal to $f_1^* = 13.59085$. The feasible region is a narrow crescent-shaped region (approximately 0.7% of the total search space) with the optimum solution lying on the second constraint, as shown in Figure 3.

Sharing and mutation operators are not used here. We have run GAs till 50 generations. Table 1 shows the consequence of 50 runs of three different GAs. First, a binary-coded GA is used with single-point crossover operator. Variables are coded in 20 bits each. The proposed constraint handling method is used. The table presents the number of times (out of 50 runs) a GA finds a solution within ϵ% of the best-known optimal function value. Different values of ϵ is used, as shown in the table.

Since the feasible space is non-convex, binary-coded GAs with single-point crossover cannot find a near-optimal solution many times. Whereas, real-coded GAs with the proposed constraint handling method find a solution within 1% of true optimal solution in 29 of 50 GA runs. When a similar constraint handling technique [15] (termed as 'PS' in the table) is used with real-coded GAs, the performance is not as good as the proposed method. In 11 of 50 GA runs, no infeasible solution was found.

To show the working of the proposed constrained handling technique (compared to PS technique), we show the population histories in generations 0, 10, and 50 in Figures 2 and 3. The initial population of 50 random solutions show that initially solutions exist all over the search space (no solution is feasible in the initial population). After 10 generations, GAs with Powell and Skolnick's constraint handling strategy (with $R = 1$) could not drive the solutions towards the narrow feasible region. Instead, the solutions get stuck at a solution $\vec{x} = (2.891103, 2.11839)$ with a function value equal to 0.41708, which is closer to the unconstrained minimum (albeit infeasible) solution at $\vec{x} = (3, 2)$. When a real-coded GA with the proposed constraint handling strategy (TS-R) is applied on the identical initial populations of 50 solutions (rest all parameter settings are also the same as that used in the Powell and Skolnick's algorithm), the GA distributes well its population around and inside the feasible region (Figure 2) after 10 generations. Finally, GAs converge near the true optimum solution at (2.243636, 2.342702) with a function value equal to 13.66464 (within 0.54% of the true optimum solution).

3.2 Test Problem 2

This problem has eight variables and six inequality constraints [12]:

$$\begin{array}{ll} \text{Minimize} & f_2(\vec{x}) = x_1 + x_2 + x_3 \\ \text{Subject to} & g_1(\vec{x}) \equiv 1 - 0.0025(x_4 + x_6) \geq 0, \\ & g_2(\vec{x}) \equiv 1 - 0.0025(x_5 + x_7 - x_4) \geq 0, \\ & g_3(\vec{x}) \equiv 1 - 0.01(x_8 - x_5) \geq 0, \\ & g_4(\vec{x}) \equiv x_1 x_6 - 833.33252 x_4 - 100 x_1 \\ & \qquad\qquad +83333.333 \geq 0, \\ & g_5(\vec{x}) \equiv x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0, \\ & g_6(\vec{x}) \equiv x_3 x_8 - x_3 x_5 + 2500 x_5 - 1250000 \geq 0, \\ & 100 \leq x_1 \leq 10000, \\ & 1000 \leq x_2, x_3 \leq 10000, \\ & 10 \leq x_i \leq 1000, \quad i = 4, \ldots, 8. \end{array} \tag{7}$$

The optimum solution is

$$\begin{array}{rcl} \vec{x}^* & = & (579.3167,\ 1359.943,\ 5110.071,\ 182.0174, \\ & & 295.5985,\ 217.9799,\ 286.4162,\ 395.5979), \\ f_2^* & = & 7049.330923. \end{array}$$

All six constraints are active at this solution.

Table 2 shows the performance of GAs with different constraint handling methods. Michalewicz [12] experienced that this problem is difficult to solve. The best solution obtained by any method used in that study (with approximately 350,000 function evaluations) had an objective function value equal to 7377.976, which is about 4.66% worse than the true optimal objective function value. The proposed approach consistently finds solutions very close to the true optimum with only 80,080 function evaluations (population size 80, maximum generations 1,000). However, the best solution obtained by GAs with sharing and mutation and with a maximum of 320,080 function evaluations (population size 80, maximum generations 4,000) has a function value equal to 7060.221, which is only about 0.15% more than the true optimal objective function value. This shows the efficacy of the proposed approach on this rather complex constrained optimization problem.

Table 1: Number of runs (out of 50 runs) converged within ϵ% of the optimum solution for GAs with two constraint handling techniques—proposed method with binary GAs (TS-B) and with real-coded GAs (TS-R) and Powell and Skolnick's method [15] (PS) with $R = 1$ on test problem 1.

Method	ϵ $\leq$ 1%	$\leq$ 2%	$\leq$ 5%	$\leq$ 10%	$\leq$ 20%	$\leq$ 50%	$>$ 50%	Infeasible	Best	Median	Worst
TS-B	2	2	5	6	9	13	37	0	13.59658	37.90495	244.11616
TS-R	29	31	31	32	33	39	11	0	13.59085	13.61673	117.02971
PS ($R = 1$)	17	19	20	20	24	32	7	11	13.59108	16.35284	172.81369

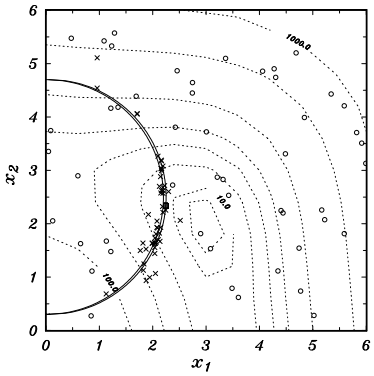

Fig. 2: Population history at initial generation (marked with open circles), at generation 10 (marked with 'x') and at generation 50 (marked with open boxes) using the proposed scheme. The population converges to a solution very close to the true constrained optimum solution on a constraint boundary.

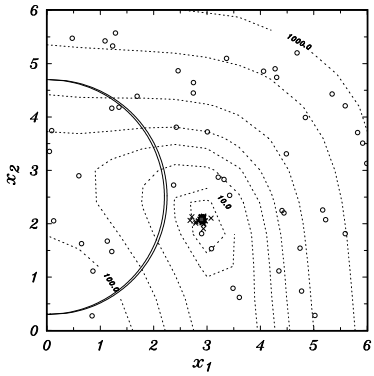

Fig. 3: Population history at initial generation (marked with open circles), at generation 10 (marked with 'x') and at generation 50 (marked with open boxes) using Powell and Skolnick's [15] method ($R = 1$). The population converges to a wrong, infeasible solution.

3.3 Test Problem 3

This problem has five variables and six inequality constraints [8, 14]:

$$
\begin{aligned}
\text{Minimize} \quad & f_3(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 \\
& \quad +37.293239x_1 - 40792.141, \\
\text{Subject to} \quad & g_1(\vec{x}) \equiv 85.334407 + 0.0056858x_2x_5 \\
& \quad +0.0006262x_1x_4 - 0.0022053x_3x_5 \geq 0, \\
& g_2(\vec{x}) \equiv 85.334407 + 0.0056858x_2x_5 \\
& \quad +0.0006262x_1x_4 - 0.0022053x_3x_5 \leq 92, \\
& g_3(\vec{x}) \equiv 80.51249 + 0.0071317x_2x_5 \\
& \quad +0.0029955x_1x_2 + 0.0021813x_3^2 \geq 90, \\
& g_4(\vec{x}) \equiv 80.51249 + 0.0071317x_2x_5 \\
& \quad +0.0029955x_1x_2 + 0.0021813x_3^2 \leq 110, \\
& g_5(\vec{x}) \equiv 9.300961 + 0.0047026x_3x_5 \\
& \quad +0.0012547x_1x_3 + 0.0019085x_3x_4 \geq 20, \\
& g_6(\vec{x}) \equiv 9.300961 + 0.0047026x_3x_5 \\
& \quad +0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 25, \\
& 78 \leq x_1 \leq 102, \\
& 33 \leq x_2 \leq 45, \\
& 27 \leq x_i \leq 45, \quad i = 3, 4, 5.
\end{aligned}
\tag{8}
$$

The best-known optimum solution [8] is

$$
\begin{aligned}
\vec{x}^* &= (78.0, 33.0, 29.995, 45.0, 36.776), \\
f_3^* &= -30,665.5.
\end{aligned}
$$

At this solution, constraints g_2 and g_5 are active. The best-known GA solution to this problem obtained elsewhere [10] using a multi-level penalty function method is

$$
\begin{aligned}
\vec{x}^{\mathrm{GA}} &= (80.49, 35.07, 32.05, 40.33, 33.34), \\
f_3^{\mathrm{GA}} &= -30,005.7,
\end{aligned}
$$

which is about 2.15% worse than the best-known optimum solution.

Table 3 presents the performance of GAs with the proposed constraint handling method with a population size 10×5 or 50. The presence of sharing and mutation produces the best performance. The best solution is as follows:

$$\vec{x} = (78, 33, 29.995, 45, 36.776).$$

Table 2: Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using real-coded GAs with the proposed constraint handling scheme on test problem 2.

Mutation	Sharing	ϵ $\leq 1\%$	$\leq 2\%$	$\leq 5\%$	$\leq 10\%$	$\leq 20\%$	$\leq 50\%$	$> 50\%$	Infeasible	Best	Median	Worst
Maximum generation = 1,000												
No	No	2	3	8	14	29	47	3	0	7063.377	8319.211	13738.276
No	Yes	3	5	7	14	29	47	2	1	7065.742	8274.830	10925.165
Maximum generation = 4,000												
Yes	Yes	17	23	33	36	42	50	0	0	7060.221	7220.026	10230.834

Table 3: Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using real-coded GAs with the proposed constraint handling scheme on test problem 3.

Mutation	Sharing	ϵ $\leq 1\%$	$\leq 2\%$	$\leq 5\%$	$\leq 10\%$	$\leq 20\%$	$\leq 50\%$	Best	Median	Worst
No	No	18	34	50	50	50	50	-30614.814	-30196.404	-29606.596
No	Yes	28	44	50	50	50	50	-30651.865	-30376.906	-29913.635
Yes	Yes	47	48	50	50	50	50	-30665.537	-30665.535	-29846.654

As many as 47 or 50 GA runs have found solutions within 1% of the best-known solution. It is also interesting to note that all GAs used here have found solutions better than that reported earlier [10], solved using binary GAs with a multi-level penalty function method.

3.4 Test Problem 4

This problem has 10 variables and eight constraints [12]:

$$\begin{aligned}
\text{Minimize} \quad & f_4(\vec{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 \\
& +(x_3-10)^2 + 4(x_4-5)^2 + (x_5-3)^2 \\
& +2(x_6-1)^2 + 5x_7^2 + 7(x_8-11)^2 \\
& +2(x_9-10)^2 + (x_{10}-7)^2 + 45, \\
\text{Subject to} \quad & g_1(\vec{x}) \equiv 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0, \\
& g_2(\vec{x}) \equiv -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0, \\
& g_3(\vec{x}) \equiv 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0, \\
& g_4(\vec{x}) \equiv -3(x_1-2)^2 - 4(x_2-3)^2 - 2x_3^2 \\
& \quad +7x_4 + 120 \geq 0, \\
& g_5(\vec{x}) \equiv -5x_1^2 - 8x_2 - (x_3-6)^2 \\
& \quad +2x_4 + 40 \geq 0, \\
& g_6(\vec{x}) \equiv -x_1^2 - 2(x_2-2)^2 + 2x_1x_2 \\
& \quad -14x_5 + 6x_6 \geq 0, \\
& g_7(\vec{x}) \equiv -0.5(x_1-8)^2 - 2(x_2-4)^2 - 3x_5^2 \\
& \quad +x_6 + 30 \geq 0, \\
& g_8(\vec{x}) \equiv 3x_1 - 6x_2 - 12(x_9-8)^2 + 7x_{10} \geq 0, \\
& -10 \leq x_1 \leq 10, \quad i = 1, \ldots, 10.
\end{aligned} \tag{9}$$

The optimum solution to this problem is as follows:

$$\begin{aligned}
\vec{x}^* = \; & (2.171996,\ 2.363683,\ 8.773926,\ 5.095984, \\
& 0.9906548,\ 1.430574,\ 1.321644,\ 9.828726, \\
& 8.280092,\ 8.375927), \quad f_4^* = 24.3062091.
\end{aligned}$$

The first six constraints are active at this solution.

Table 4 shows the performance of GAs with the proposed constraint handling scheme with a population size 10×10 or 100. In this problem, GAs with and without sharing performed equally well. However, GA's performance improves drastically with mutation, which provided the necessary diversity among the feasible solutions. This problem was also solved by Michalewicz [12] by different constraint handling techniques. The best reported method had its best, median, and worst objective function values as 24.690, 29.258, and 36.060, respectively, in 350,070 function evaluations. This was achieved with a multi-level penalty function approach. With a similar maximum number of function evaluations, GAs with the proposed constraint handling method have found better solutions (best: 24.372, median: 24.409, and worst: 25.075). The best solution is within 0.27% of the optimal objective function value. Most interestingly, 41 out of 50 runs have found a solution having objective function value within 1% (or $f(\vec{x})$ smaller than 24.549) of the optimal objective function value.

3.5 Welded beam design

Next, we apply the proposed method to solve a welded beam design problem, which has been attempted to solve by using a number of classical optimization methods

Table 4: Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using real-coded GAs with the proposed constraint handling scheme on test problem 4.

Mutation	Sharing	ϵ $\leq 1\%$	$\leq 2\%$	$\leq 5\%$	$\leq 10\%$	$\leq 20\%$	$\leq 50\%$	$> 50\%$	Infeasible	Best	Median	Worst
Maximum generation = 1,000												
No	No	0	0	8	16	28	47	3	0	24.81711	27.85520	42.47685
No	Yes	0	0	9	25	36	45	5	0	24.87747	26.73401	50.40042
Maximum generation = 3,500												
Yes	Yes	41	41	50	50	50	50	0	0	24.37248	24.40940	25.07530

[16] and by using GAs [1].

The objective of the design is to minimize the fabrication cost of a beam which is required to be welded in a column. There are four design variables $\vec{x} = (h, \ell, t, b)$ and five constraints involving stress, deflection and bucking restrictions. The resulting NLP problem is shown below:

$$\begin{array}{ll} \text{Minimize} & f_w(\vec{x}) = 1.10471h^2\ell + 0.04811tb(14.0+\ell), \\ \text{Subject to} & g_1(\vec{x}) \equiv 13,600 - \tau(\vec{x}) \geq 0, \\ & g_2(\vec{x}) \equiv 30,000 - \sigma(\vec{x}) \geq 0, \\ & g_3(\vec{x}) \equiv b - h \geq 0, \\ & g_4(\vec{x}) \equiv P_c(\vec{x}) - 6,000 \geq 0, \\ & g_5(\vec{x}) \equiv 0.25 - \delta(\vec{x}) \geq 0, \\ & 0.125 \leq h \leq 10, \\ & 0.1 \leq \ell, t, b \leq 10. \end{array} \tag{10}$$

The terms $\tau(\vec{x})$, $\sigma(\vec{x})$, $P_c(\vec{x})$, and $\delta(\vec{x})$ are given below:

$$\begin{aligned} \tau(\vec{x}) &= \sqrt{\tau'^2 + \tau''^2 + \ell\tau'\tau''/\sqrt{0.25(\ell^2+(h+t)^2)}}, \\ \sigma(\vec{x}) &= \frac{504,000}{t^2b}, \\ P_c(\vec{x}) &= 64,746.022(1-0.0282346t)tb^3, \\ \delta(\vec{x}) &= \frac{2.1952}{t^3b}, \end{aligned}$$

where

$$\begin{aligned} \tau' &= \frac{6,000}{\sqrt{2}h\ell}, \\ \tau'' &= \frac{6,000(14+0.5\ell)\sqrt{0.25(\ell^2+(h+t)^2)}}{2\{0.707h\ell(\ell^2/12+0.25(h+t)^2)\}}. \end{aligned}$$

The optimized solution reported in the literature [16] is $h^* = 0.2444$, $\ell^* = 6.2187$, $t^* = 8.2915$, and $b^* = 0.2444$ with a function value equal to $f_w^* = 2.38116$. Binary GAs are applied on this problem in an earlier study [1] and the solution $\vec{x} = (0.2489, 6.1730, 8.1789, 0.2533)$ with $f_w = 2.43$ (within 2% of the above best solution) was obtained with a population size of 100. However, it was observed that the performance of GAs largely dependent on the chosen penalty parameter values.

We use the proposed constraint handling technique here. Table 5 presents the performance of GAs with a population size 80. Real-parameter GAs without sharing is good enough to find a solution within 2.6% of the optimal objective function value. However, with the introduction of sharing, 28 runs out of 50 runs have found a solution within 1% of the optimal objective function value and this has been achieved with only a maximum of 40,080 function evaluations. When more number of function evaluations are allowed, real GAs with the proposed constraint handling technique and mutation operator perform much better—all 50 runs have found a solution within 0.1% of the true optimal objective function value. This means that with the proposed GAs, one run is enough to find a satisfactory solution close to the true optimal solution. In handling such complex constrained optimization problems, any designer would like to use such an efficient yet robust optimization algorithm.

3.6 Summary of Results

Here, we summarize the best GA results obtained in this paper and compare that with the best reported results in earlier studies. For test problems 2 and 4, earlier methods recorded the best, median, and worst values for 10 GA runs only. However, the corresponding values for GAs with the proposed method have been presented for 50 runs.

It is clear that in most cases the proposed constraint handling strategy has performed with more *efficiency* (in terms of getting closer to the best-known solution) and with more *robustness* (in terms of more number of successful GA runs finding solutions close to the best-known solution) than previous methods. Since solutions are compared either with objective function values or with the amount of constraint violation, penalty coefficients are not needed in the proposed approach.

Table 5: Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using real-coded GAs (TS-R) with the proposed constraint handling scheme.

Method	Mutation	Sharing	ϵ $\leq 1\%$	$\leq 2\%$	$\leq 5\%$	$\leq 10\%$	$\leq 20\%$	$\leq 50\%$	$> 50\%$	Best	Median	Worst
Maximum generations = 500												
TS-R	No	No	0	0	1	4	8	16	34	2.44271	3.83412	7.44425
TS-R	No	Yes	28	36	44	48	50	50	0	2.38119	2.39289	2.64583
Maximum generations = 4,000												
TS-R	No	Yes	28	37	44	48	50	50	0	2.38119	2.39203	2.64583
TS-R	Yes	Yes	50	50	50	50	50	50	0	2.38145	2.38263	2.38355

Table 6 Summary of results of this study. A '–' indicates that information is not available.

Prob No.	True optimum	Best-known GA Best	Median	Worst	Results of this study Best	Median	Worst
1	13.59085	–	–	–	13.59085	13.65413	117.02971
2	7049.331	7485.667	8271.292	8752.412	7060.221	7220.026	10230.834
3	−30665.5	−30005.7	–	–	−30665.537	−30665.535	−29846.654
4	24.306	24.690	29.258	36.060	24.372	24.409	25.075
Weld	2.381	2.430	–	–	2.381	2.383	2.384

4 Conclusions

The major difficulty in handling constraints using penalty function methods in GAs and in classical optimization methods has been to set an appropriate value for penalty parameters. This often requires users to experiment with different values of penalty parameters. In this paper, we have developed a constraint handling method for GAs which does not require any penalty parameter. Infeasible solutions are penalized in a way so as to provide a search direction towards the feasible region. This has been possible mainly because of the population approach of GAs and ability to have pair-wise comparison of solutions using the tournament selection operator. On a number of test problems including an engineering design problem, GAs with the proposed constraint handling method have repeatedly found solutions closer to the true optimal solutions than earlier GAs. The results of this study are interesting and show promise for a reliable and efficient constrained optimization task through GAs.

[1] Deb, K.: Optimal design of a welded beam structure via genetic algorithms, AIAA Journal, vol. 29, no. 11, pp. 2013–2015, 1991.

[2] Deb, K.: Optimization for engineering design: Algorithms and examples. New Delhi: Prentice-Hall 1995.

[3] Deb, K., Agrawal, R. B.: Simulated binary crossover for continuous search space. Complex Systems, vol. 9, pp. 115–148, 1995.

[4] Deb, K., Goldberg, D. E.: An investigation of niche and species formation in genetic function optimization, Proc. Third International Conference on Genetic Algorithms, Washington D. C., pp. 42–50, 1989.

[5] Deb, K., Goyal, M.: A combined genetic adaptive search (GeneAS) for engineering design, Computer Science and Informatics, vol. 26, no. 4, pp. 30–45, 1996.

[6] Goldberg, D. E., Deb, K., Clark, J. H.: Genetic algorithms, noise, and the sizing of populations, Complex Systems, vol. 6, pp. 333–362, 1992.

[7] Harik, G., Cantu-Paz, E., Goldberg, D. E., Miller, B. L.: The gambler's ruin problem, genetic algorithms, and the sizing of populations, Proc. 1997 IEEE International Conference on Evolutionary Computation, Orlando, FL, pp. 7–12, 1997.

[8] Himmelblau, D. M.: Applied nonlinear programming, New York: McGraw-Hill, 1972.

[9] Hock, W., Schittkowski, K.: Test examples for nonlinear programming code, Lecture Notes on Economics and Mathematical Systems, 187, Berlin: Springer-Verlag, 1980.

[10] Homaifar, A., Lai, S. H.-Y., Qi., X.: Constrained optimization via genetic algorithms, Simulation, vol. 62, no. 4, 242–254, 1994.

[11] Joines, J. A., Houck, C. R.: On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs, Proc International Conference on Evolutionary Computation, Orlando, FL, pp. 579–584, 1994.

[12] Michalewicz, Z.: Genetic algorithms, numerical optimization, and constraints, Proc. Sixth International Conference on Genetic Algorithms, Carnegie-Mellon, PA, pp. 151–158, 1995.

[13] Michalewicz, Z., Attia, N.: Evolutionary optimization of constrained problems, Proc. Third Annual Conference on Evolutionary Programming, pp. 98–108, 1994.

[14] Michalewicz, Z., Schoenauer, M.: Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation, vol. 4, no. 1, pp. 1–32, 1996.

[15] Powell, D., Skolnick, M. M.: Using genetic algorithms in engineering design optimization with nonlinear constraints, Proc. Fifth International Conference on Genetic Algorithms, Urbana, IL, pp. 424–430, 1993.

[16] Reklaitis, G. V., Ravindran, A., Ragsdell, K. M.: Engineering optimization methods and applications, New York: Wiley, 1983.

[17] Richardson, J. T., Palmer, M. R., Liepins, G., Hilliard, M.: Some guidelines for genetic algorithms with penalty functions, Proc. Third International Conference on Genetic Algorithms, Washington D. C., pp. 191–197, 1989.

Appendix

A Simulated Binary Crossover (SBX) and Parameter-based Mutation

The procedure of computing children solutions c_1 and c_2 from two parent solutions y_1 and y_2 under SBX operator is as follows:

1. Create a random number u between 0 and 1.
2. Find a parameter β_q using a polynomial probability distribution, developed in [3] from a schema processing point of view, as follows:

$$\beta_q = \begin{cases} (u\alpha)^{\frac{1}{\eta_c+1}}, & \text{if } u \le \frac{1}{\alpha}, \\ \left(\frac{1}{2-u\alpha}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise}, \end{cases} \tag{11}$$

where $\alpha = 2 - \beta^{-(\eta_c+1)}$ and β is calculated as follows:

$$\beta = 1 + \frac{2}{y_2 - y_1}\min[(y_1 - y_l),\ (y_u - y_2)].$$

Here, the parameter y is assumed to vary in $[y_l, y_u]$. The parameter η_c is the distribution index for SBX and can take any non-negative value. A small value of η_c allows solutions far away from parents to be created as children solutions and a large value restricts only near-parent solutions to be created as children solutions.

3. The children solutions are then calculated as follows:

$$\begin{aligned} c_1 &= 0.5\left[(y_1 + y_2) - \beta_q|y_2 - y_1|\right], \\ c_2 &= 0.5\left[(y_1 + y_2) + \beta_q|y_2 - y_1|\right]. \end{aligned}$$

It is assumed here that $y_1 < y_2$. A simple modification to the above equation can be made for $y_1 > y_2$. For handling multiple variables, each variable is chosen with a probability 0.5 and the above SBX operator is applied variable-by-variable. In all simulation results here, we have used $\eta_c = 1$.

A polynomial probability distribution is used to create a solution c in the vicinity of a parent solution y under the mutation operator [5]. The following procedure is used for a parameter $y \in [y_l, y_u]$:

1. Create a random number u between 0 and 1.
2. Calculate the parameter δ_q as follows:

$$\delta_q = \begin{cases} \left[2u + (1-2u)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1, \\ \quad \text{if } u \le 0.5, \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}}, \\ \quad \text{otherwise}, \end{cases} \tag{12}$$

where $\delta = \min[(y - y_l),\ (y_u - y)]/(y_u - y_l)$. The parameter η_m is the distribution index for mutation and takes any non-negative value.

3. Calculate the mutated child as follows:

$$c = y + \delta_q(y_u - y_l),$$

Using above equations, we can calculate the expected normalized perturbance ($(c - y)/(y_u - y_l)$) of the mutated solutions in both positive and negative sides separately. We observe that this value is $O(1/\eta_m)$. In order to get a mutation effect of 1% perturbance in solutions, we set $\eta_m = 100 + t$ and the probability of mutation is changed as follows:

$$p_m = \frac{1}{n} + \frac{t}{t_{\max}}\left(1 - \frac{1}{n}\right),$$

where t and $t_{\max}$ are current generation number and the maximum number of generations allowed, respectively. Thus, in the initial generation, we mutate on an average one variable ($p_m = 1/n$) with an expected 1% perturbance and as generations proceed, we mutate more variables with lesser expected perturbance.