An Efficient Constraint Handling Method for Genetic Algorithms

Kalyanmoy Deb

Kanpur Genetic Algorithms Laboratory (KanGAL)
Department of Mechanical Engineering
Indian Institute of Technology Kanpur
Kanpur, PIN 208 016, India
E-mail: deb@iitk.ac.in

Abstract

Many real-world search and optimization problems involve inequality and/or equality constraints and are thus posed as constrained optimization problems. In trying to solve constrained optimization problems using genetic algorithms (GAs) or classical optimization methods, penalty function methods have been the most popular approach, because of their simplicity and ease of implementation. However, since the penalty function approach is generic and applicable to any type of constraint (linear or nonlinear), their performance is not always satisfactory. Thus, researchers have developed sophisticated penalty functions specific to the problem at hand and the search algorithm used for optimization. However, the most difficult aspect of the penalty function approach is to find appropriate penalty parameters needed to guide the search towards the constrained optimum. In this paper, GA's population-based approach and ability to make pair-wise comparison in tournament selection operator are exploited to devise a penalty function approach that does not require any penalty parameter. Careful comparisons among feasible and infeasible solutions are made so as to provide a search direction towards the feasible region. Once sufficient feasible solutions are found, a niching method (along with a controlled mutation operator) is used to maintain diversity among feasible solutions. This allows a real-parameter GA's crossover operator to continuously find better feasible solutions, gradually leading the search near the true optimum solution. GAs with this constraint handling approach have been tested on nine problems commonly used in the literature, including an engineering design problem. In all cases, the proposed approach has been able to repeatedly find solutions closer to the true optimum solution than that reported earlier.

1 Introduction

Many search and optimization problems in science and engineering involve a number of constraints which the optimal solution must satisfy. A constrained optimization problem is usually written as a nonlinear programming (NLP) problem of the following type:

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

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 $[x_i^l, x_i^u]$.

Constraint handling methods used in classical optimization algorithms can be classified into two groups: (i) *generic* methods that do not exploit the mathematical structure (whether linear or nonlinear) of the constraint, and (ii) *specific* methods that are only applicable to a special type of constraints. Generic methods, such as the penalty function method, the Lagrange multiplier method, and the complex search method [1,2] are popular, because each one of them can be easily applied to any problem without much change in the algorithm. But since these methods are generic, the performance of these methods in most cases is not satisfactory. However, specific methods, such as the cutting plane method, the reduced gradient method, and the gradient projection method [1,2], are applicable either to problems having convex feasible regions only or to problems having a few variables, because of increased computational burden with large number of variables.

Since genetic algorithms (GAs) are generic search methods, most applications of GAs to constraint optimization problems have used the penalty function approach of handling constraints. The penalty function approach involves a number of penalty parameters which must be set right in any problem to obtain feasible solutions. This dependency of GA's performance on penalty parameters has led researchers to devise sophisticated penalty function approaches such as multi-level penalty functions [3], dynamic penalty functions [4], and penalty functions involving temperature-based evolution of penalty parameters with repair operators [5]. All these approaches require extensive experimentation for setting up appropriate parameters needed to define the penalty function. Michalewicz [6] describes the difficulties in each method and compares the performance of these algorithms on a number of test problems. In a similar study, Michalewicz and Schoenauer [7] concluded that the static penalty function method (without any sophistication) is a more robust approach than the sophisticated methods. This is because one such sophisticated method may work well on some problems but may not work so well in another problem.

In this paper, we develop a constraint handling method based on the 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. Although at least one other constraint handling method satisfying above three criteria was suggested earlier [8] it involved penalty parameters which again must be set right for proper working of the algorithm.

In the remainder of the paper, we first show that the performance of a binary-coded GA using the static penalty function method on an engineering design problem largely depends on the chosen penalty parameter. Thereafter, we describe the proposed constraint handling method and present the performance of real-parameter GAs on nine test problems, including the same engineering design problem. The results are also compared with best-known solutions obtained using earlier GA implementations or using classical optimization methods.

2 Constraint Handling in GAs

In most applications of GAs to constrained optimization problems, the penalty function method has been used. In the penalty function method for handling inequality constraints in minimization problems, the fitness function $F(\vec{x})$ is defined as the sum of the objective function $f(\vec{x})$ and a penalty term which depends on the constraint violation $\langle g_j(\vec{x})\rangle$:

$$F(\vec{x}) = f(\vec{x}) + \sum_{j=1}^{J} R_j \langle g_j(\vec{x})\rangle^2, \tag{2}$$

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. The purpose of a penalty parameter R_j is to make the constraint violation $g_j(\vec{x})$ of the same order of magnitude as the objective function value $f(\vec{x})$. Equality constraints are usually handled by converting them into inequality constraints as follows[1] :

$$g_{k+J}(\vec{x}) \equiv \delta - |h_k(\vec{x})| \geq 0,$$

where δ is a small positive value. This increases the total number of inequality constraints to $m = J + K$ and the term J in equation 2 can then be replaced by m to include all inequality and equality constraints. Thus, there are total of m penalty parameters R_j which must be set right in a penalty function approach.

In order to reduce the number of penalty parameters, often the constraints are normalized and only one penalty parameter R is used [1]. In any case, there are two problems associated with this static penalty function approach:

(1) The optimal solution of $F(\vec{x})$ depends on penalty parameters R_j (or R). Users usually have to try different values of R_j (or R) to find what value would steer the search towards the feasible region. This requires extensive experimentation to find any reasonable solution. This problem is so severe that some researchers have used different values of R_j (or R) depending on the level of constraint violation [3], and some have used sophisticated temperature-based evolution of penalty parameters through generations [5] involving a few parameters describing the rate of evolution.

(2) The inclusion of the penalty term *distorts* the objective function [1]. For small values of R_j (or R), the distortion is small, but the optimum of $F(\vec{x})$ may not be near the true constrained optimum. On the other hand, if a large R_j (or R) is used, the optimum of $F(\vec{x})$ is closer to the true constrained optimum, but the distortion may be so severe that $F(\vec{x})$ may have artificial locally optimal solutions. This primarily happens due to interactions among multiple constraints. To avoid such locally optimal solutions, classical penalty function approach works in sequences, where in every sequence the penalty parameters are increased in steps and the current sequence of optimization begins from the optimized solution found in the previous sequence. This way a controlled search is possible and locally optimal solutions can be avoided. However, most classical methods use gradient-based search methods and usually have difficulty in solving discrete search space problems and to problems having a large number of variables. Although GAs do not use gradient information, they are not free from the distortion effect caused due to the addition of the penalty term with the objective function. However, GAs are comparatively less sensitive to distorted function landscapes due to the stochasticity in their operators.

[1] It is important to note that this transformation makes the resulting inequality constraint function non-differentiable, thereby causing difficulty to many classical search and optimization algorithms to use this transformation. In those cases, an equality constraint is converted into two inequality constraints $h_k(\vec{x}) \leq \delta$ and $h_k(\vec{x}) \geq -\delta$.

In order to investigate the effect of the penalty parameter R_j (or R) on the performance of GAs, we consider a well-studied welded beam design problem [2]. The resulting optimization problem has four design variables $\vec{x} = (h, \ell, t, b)$ and five inequality constraints:

$$\begin{aligned}
\text{Minimize } & f_w(\vec{x}) = 1.10471h^2\ell + 0.04811tb(14.0 + \ell),\\
\text{Subject to } & g_1(\vec{x}) \equiv 13,600 - \tau(\vec{x}) \geq 0,\\
& g_2(\vec{x}) \equiv 30,000 - \sigma(\vec{x}) \geq 0,\\
& g_3(\vec{x}) \equiv b - h \geq 0,\\
& g_4(\vec{x}) \equiv P_c(\vec{x}) - 6,000 \geq 0,\\
& g_5(\vec{x}) \equiv 0.25 - \delta(\vec{x}) \geq 0,\\
& 0.125 \leq h \leq 10,\\
& 0.1 \leq \ell, t, b \leq 10.
\end{aligned} \tag{3}$$

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

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

where

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

The optimized solution reported in the literature [2] is $h^* = 0.2444$, $\ell^* = 6.2187$, $t^* = 8.2915$, and $b^* = 0.2444$ with a function value equal to $f^* = 2.38116$. Binary GAs are applied on this problem in an earlier study [9] and the solution $\vec{x} = (0.2489, 6.1730, 8.1789, 0.2533)$ with $f = 2.43$ (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.

In order to get more insights on the working of GAs, we apply binary GAs with tournament selection without replacement and single-point crossover operator with

$p_c = 0.9$ on this problem. In the tournament selection, two solutions are picked at random from the population and are compared based on their fitness ($F(\vec{x})$) values. The better solution is chosen and kept in an intermediate population. This process is continued till all N population slots are filled. This operation is usually performed systematically, so the best solution in a population always get exactly two copies in the intermediate population. Each variable is coded in 10 bits, so that total string length is 40. A population size of 80 is used and GAs with 50 different initial populations are run. GAs are run till 500 generations. All constraints are normalized (for example, the first constraint is normalized as $1 - \tau(\vec{x})/13600 \geq 0$, and so on) and a single penalty parameter R is used. Table .1 shows the performance of binary GAs for different penalty parameter values.

- - - - - Table 1 here - - - - -

For each case, the best, median [2], and worst values of 50 optimized objective function values are also shown in the table. With $R = 1$, although three out of 50 runs have found a solution within 10% of the best-known solution, 13 GA runs have not been able to find a single feasible solution in 40,080 function evaluations. This happens because with small R there is not much pressure for the solutions to become feasible. With large penalty parameters, the pressure for solutions to become feasible is more and all 50 runs found feasible solutions. However, because of larger emphasis of solutions to become feasible, when a particular solution becomes feasible it has a large selective advantage over other solutions (which are infeasible) in the population. If new and different feasible solutions are not created, GAs would overemphasize this sole feasible solution and soon prematurely converge near this solution. This has exactly happened in GA runs with larger R values, where the best solution obtained is, in most cases, more than 50% away (in terms of function values) from the true constrained optimum.

Similar experiences have been reported by other researchers in applying GAs with penalty function approach to constrained optimization problems. Thus, if penalty function method is to be used, the user usually have to take many runs or 'adjust' the penalty parameters to get a solution within an acceptable limit. In a later section, we shall revisit this welded beam design problem and show how the proposed constrained handling method finds solutions very close to the true optimum reliably and without the need of using any penalty parameter.

Michalewicz [6] and later Michalewicz and Schoenauer [7] have discussed different constraint handling methods used in GAs. They have classified most of the evolutionary constraint handling methods into five categories: (1) methods based on preserving feasibility of solutions, (2) methods based on penalty functions, (3) methods making distinction between feasible and infeasible solutions, (4) methods based on decoders, and (5) hybrid methods. The methods under the first category

[2] The optimized objective function values (of 50 runs) are arranged in ascending order and the 25th value in the list is called the median optimized function value.

explicitly use the knowledge of the structure of the constraints and use a search operator that maintains the feasibility of solutions. Second class of methods uses penalty functions of various kinds, including dynamic penalty approaches where penalty parameter are adapted dynamically over time. The third class of constraint handling methods uses different search operators for handling infeasible and feasible solutions. The fourth class of methods uses an indirect representation scheme which carries instructions for constructing a feasible solution. In the fifth category, evolutionary methods are combined with heuristic rules or classical constrained search methods. Michalewicz and Schoenauer [7] have compared different algorithms on a number of test problems and observed that each method works well on some classes of problems whereas does not work well on other problems. Owing to this inconsistency in the performance of different methods, they suggested to use the static penalty function method, similar to that given in equation 2. Recently, a two-phase evolutionary programming (EP) method is developed [10]. In the first phase, a standard EP technique with a number of strategy parameters which were evolved during the optimization process was used. With the solution obtained in the first phase, a neural network method was used in the second phase to improve the solution. The performance of the second phase depends on how close a solution to the true optimal solution is found in the first phase. The approach involves too many different procedures with many control parameters and it is unclear which procedure and parameter settings are important. Moreover, out of the six test problems used in the study, five were two-variable problems having at most two constraints. It is unclear how this rather highly sophisticated method will scale up its performance to more complex problems.

In the following section, we present a different yet simple penalty function approach which does not require any penalty parameter, thereby making the approach applicable to a wide variety of constrained optimization problems.

3 Proposed Constraint Handling Method

The proposed method belongs to both second and third categories of constraint handling methods described by Michalewicz and Schoenauer [7]. Although a penalty term is added to the objective function to penalize infeasible solutions, the method differs from the way the penalty term is defined in conventional methods and in earlier GA implementations.

The method proposes to use a tournament selection operator, where two solutions are compared at a time, and the following criteria are always enforced [11]:

(1) Any feasible solution is preferred to any infeasible solution,
(2) Among two feasible solutions, the one having better objective function value is preferred,

(3) Among two infeasible solutions, the one having smaller constraint violation is preferred.

Although there exist a number of other implementations [6,8,12] where criteria similar to the above 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.

Recall that penalty parameters are needed to make the constraint violation values of the same order as the objective function value. In the proposed method, penalty parameters are not needed because in any of the above three scenarios, solutions are never compared in terms of both objective function and constraint violation information. Of the three tournament cases mentioned above, in the first case, neither objective function value nor the constraint violation information is used, simply the feasible solution is preferred. In the second case, solutions are compared in terms of objective function values alone and in the third case, solutions are compared in terms of the constraint violation information alone. Moreover, the idea of comparing infeasible solutions only in terms of constraint violation has a practical implication. In order to evaluate any solution (say a particular solution of the welded beam problem discussed earlier), it is a usual practice to first check the feasibility of the solution. 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 the 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 these arguments, we devise the following fitness function, where infeasible solutions are compared based on only their constraint violation:

$$F(\vec{x}) = \begin{cases} f(\vec{x}), & \text{if } g_j(\vec{x}) \geq 0, \quad \forall\, j = 1, 2, \ldots, m, \\ f_{\max} + \sum_{j=1}^{m} \langle g_j(\vec{x}) \rangle, & \text{otherwise.} \end{cases} \tag{4}$$

The parameter $f_{\max}$ is the objective function value of the worst feasible solution in the population. Thus, the fitness of an infeasible solution not only depends on the amount of constraint violation, but also on the population of solutions at hand. However, the fitness of a feasible solution is always fixed and is equal to its objective function value.

We shall first illustrate this constraint handling technique on a single-variable constrained minimization problem and later show its effect on contours of a two-dimensional problem. In Figure .1, the fitness function $F(\vec{x})$ (thick line in infeasible region and dashed line in feasible region) are shown. The unconstrained minimum solution is not feasible here. It is important to note that $F(\vec{x}) = f(\vec{x})$ in the feasible region and there is a gradual [3] increase in fitness for infeasible solutions away from the constraint boundary. Under the tournament selection operator mentioned earlier, there will be selective pressure for infeasible solutions to come closer and inside the feasible region. The figure also shows how the fitness value of six population members (shown by solid bullets) will be evaluated. It is interesting to note how the fitness of infeasible solutions depends on the worst feasible solution. If no feasible solution exists in a population, $f_{\max}$ is set to zero.

- - - - - Figure 1 here - - - - -

It is important to reiterate that since solutions are not compared in terms of both objective function value and constraint violation information, there is no need of any explicit penalty parameter in the proposed method. This is a major advantage of the proposed method over earlier penalty function implementations using GAs. However, to avoid any bias from any particular constraint, all constraints are normalized (a usual practice in constrained optimization [1]) 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 in every iteration and a pair-wise comparison of solutions is possible using the tournament selection operator. For the same reason, such schemes cannot be used with classical point-by-point search and optimization methods.

The proposed constraint handling technique is better illustrated in Figures .2 and .3, where fitness function is shown by drawing contours of the following NLP problem:

$$\begin{aligned} \text{Minimize } & f(x,y) = (x-0.8)^2 + (y-0.3)^2, \\ \text{Subject to } & g_1(x,y) \equiv 1 - [(x-0.2)^2 + (y-0.5)^2]/0.16 \geq 0, \\ & g_2(x,y) \equiv [(x+0.5)^2 + (y-0.5)^2]/0.81 - 1 \geq 0. \end{aligned} \tag{5}$$

The contours have higher function values as they move out of the point $(x,y) \equiv (0.8, 0.3)$. Figure .2 shows the contour plot of the objective function $f(x,y)$ and the crescent shaped (non-convex) feasible region formed by $g_1(x,y)$ and $g_2(x,y)$ constraint functions. Assuming that the worst feasible solution in a population lie at $(0.35, 0.85)$ (the point marked by a 'o' in the figure), the corresponding $f_{\max} = 0.505$. Figure .3 shows the contour plot of the fitness function $F(x,y)$ (calculated using equation 4). It is interesting to note that the contours do not get changed inside

[3] Although, in some cases, it is apparent that the above strategy may face trouble where constraint violations may not increase monotonically from the constraint boundary inside the infeasible region [13], this may not be a problem to GAs. Since the above strategy guarantees that the fitness of any feasible solution is better than fitness of all infeasible solutions in a population, once a feasible solution is found, such nonlinearity in constraint violations may not matter much. However, this needs a closer look which we plan to investigate in a future study.

the feasible region, whereas they become parallel to the constraint surface outside the feasible region. Thus, when most solutions in a population are infeasible, the search forces solutions to come closer to feasible region. Once sufficient solutions exist inside the feasible region, the search gets directed by the effect of the objective function alone. In the case of multiple disconnected feasible regions, the fitness function has a number of such attractors, one corresponding to each feasible region. When solutions come inside feasible regions, the selection operator mainly works with the true objective function value and helps to focus the search in the correct (global) feasible region.

- - - - - Figure 2 here - - - - -

- - - - - Figure 3 here - - - - -

We have realized that the proposed method is somewhat similar to Powell and Skolnick's [8] method, which involves penalty parameters. Thus, like other penalty function approaches, Powell and Skolnick 's (PS) method is also sensitive to penalty parameters. Moreover, the PS method may sometime create artificial local optima, as discussed in the following. Consider the same single-variable function shown in Figure .1. The calculation procedure of the fitness function in PS method is illustrated in Figure .4.

- - - - - Figure 4 here - - - - -

The major difference between the PS method and the proposed method is that in the PS method the objective function value is considered in calculating the fitness of infeasible solutions. In the PS method, the penalized function value[4] $f(\vec{x}) + R\sum_j \langle g_j(\vec{x})\rangle$ is raised by an amount λ (shown in the figure) to make the fitness of the best infeasible solution equal to the fitness of the worst feasible solution. Figure .4 shows that, in certain situations, the resulting fitness function (shown by a long dashed line) may have an *artificial* minimum in the infeasible region. When the feasible region is narrow, there may not be many feasible solutions present in a population. In such a case, GAs with this constraint handling method may get trapped into this artificial local optimum. It is worth mentioning that the effect of this artificial local optimum can get reduced if a large enough penalty parameter R is used. This dependency of a constraint handling method on the penalty parameter is not desirable (and the meaning of 'large penalty parameter' is subjective to the problem at hand) and has often led researchers to rerun an optimization algorithm with different values of penalty parameters.

[4] In Powell and Skolnick's study, the square of constraint violation was used. Although, this changes relative importance of constraint violation with respect to the objective function value in Powell and Skolnick's method, it does not matter in the proposed approach, because of the use of tournament selection.

The results of the welded beam design problem presented in Section 2 are all achieved with binary GAs, where all variables are coded in binary strings. 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`$*$...$*$) may be considered to be more important than schemata specifying discrete regions in the search space (such as ($*$`1`$*$`10`$*$`...`$*$), in general. 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 single-point crossover operator used in binary GAs may not always be able to create feasible children solutions from two feasible parent solutions. Moreover, in most cases, such problems have feasible region which is a tiny fraction of the entire search space. Thus, once feasible parent solutions are found, a controlled crossover operator is desired in order to (hopefully) create children solutions which are also feasible.

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 eliminate the above two difficulties associated with binary coding and single-point crossover. In this paper, we use real-coded GAs with simulated binary crossover (SBX) operator [14] and a parameter-based mutation operator [15], for this purpose. SBX operator is particularly suitable here, because the spread of children solutions around parent solutions can be controlled using a distribution index η_c (see Appendix A). With this operator any arbitrary contiguous region can be searched, provided there is enough diversity maintained among the feasible parent solutions. Let us illustrate this aspect with the help of Figure .3. Note that the constrained optimum is at the lower half of the crescent-shaped feasible region (on $g_1(x,y)$ constraint). Although a population may contain solutions representing both the lower and the upper half of the feasible region, solutions in the lower half are more important, although the representative solutions in the lower half may have inferior objective function values compared to those in the upper half. In such cases, the representative solutions of the lower half must be restored in the population, in the hope of finding better solutions by the action of the crossover operator. Thus, maintaining diversity among feasible solutions is an important task, which will allow a crossover operator to constantly find better feasible solutions.

There are a number of ways diversity can be maintained in a population. Among them, niching methods [16] and use of mutation [17] are popular ones. In this paper, we use either or both of the above methods of maintaining diversity among the feasible solutions. A simple niching 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. If a specific number (n_f) of feasible solutions are checked and none is found to qualify within the critical distance, the i-th solution is declared as winner. The normalized Euclidean distance is calculated as follows:

$$d_{ij} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(\frac{x_k^{(i)} - x_k^{(j)}}{x_k^u - x_k^l}\right)^2}. \tag{6}$$

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

3.2 Evolutionary Strategies versus Real-coded GAs

Evolutionary strategies (ESs) are evolutionary optimization methods which work on floating-point numbers directly [18,19]. The main difference in the working principles of an ES and a real-coded GA is that in ES mutation operator is the main search operator. ES also uses a block truncation selection operator, which is different from the tournament selection operator. Moreover, an ES uses two different populations (parent and children populations) with children population size about an order of magnitude larger than that of the parent population size. It is highlighted earlier that the population approach and the ability to compare solutions pairwise are two essential features of the proposed constraint handling method. Although an ES uses a population approach, it usually does not make a pairwise comparison of solutions. Although a tournament selection scheme can be introduced in an ES, it remains an open question as to how such an ES will work in general.

Moreover, there exists a plethora of other implementations of GAs such as multimodal GAs, multi-objective GAs, and others, which have been successfully implemented with real-coded GAs [20]. We believe that the constraint handling strategy suggested in this study can also be easily incorporated along with various other kinds of existing real-coded GAs.

Thus, for the sake of simplicity in implementation, we have tested the constraint handling strategy with real-coded GAs, instead with an ES framework. We are currently working on implementing the proposed constraint handling method with an ES framework and results comparing real-coded GAs and ESs will be reported at a later date.

4 Results

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

In all problems, we run GAs 50 times from different initial populations. Fixing the correct population size in a problem is an important factor for proper working of a GA. Previous population sizing considerations [21,22] based on schema processing suggested that the population size should increase with the problem size. Although the correct population size should also depend on the underlying signal-to-noise in a problem, here we follow a simple procedure of calculating the population size: $N = 10n$, 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. When binary-coded GAs are used, the single-point crossover operator is used. When real-coded GAs are used, simulated binary crossover (SBX) is used [14]. The SBX procedure is described briefly in Appendix A. When mutation is used, the bit-wise mutation operator is used for binary GAs and a parameter-based mutation is used for real-coded GAs. This procedure is also described in Appendix A. Wherever niching is used, we have used $\bar{d} = 0.1$ and $n_f = 0.25N$.

4.1 Test Problem 1

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

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

The unconstrained objective function $f_1(x_1, x_2)$ has a minimum solution at (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 $x^* = (2.246826, 2.381865)$ 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 first constraint, as shown in Figure .5.

Niching and mutation operators are not used here. We have run GAs till 50 generations. Powell and Skolnick's [8] constraint handling method (PS) is implemented with the real-coded GAs and with tournament selection and the SBX operator. With a penalty parameter $R = 1$ for both constraints, the performance of GAs is tabulated in Table .2. The table shows that 11 out of 50 runs cannot find a single feasible solution with Powell and Skolnick's method with $R = 1$, whereas the proposed method (TS-R) finds a feasible solution every time. Moreover, 58% runs have found a solution within 1% of the true optimum solution. The dependency of PS method on the penalty parameter R is also clear from the table.

- - - - - Table 2 here - - - - -

In order to investigate the performance of the binary GA on this problem, binary GAs with the proposed constraint handling method (TS-B) is applied next. Each variable is coded in 20 bits. Binary GAs find solutions within 1% and 50% of the optimum solution in only 2 and 13 out of 50 runs, respectively. Although, all 50 GA runs are able to find feasible solutions, the performance (best 13.59658, median 37.90495, and worst 244.11616) is not as good as that of real-coded GAs.

In runs where Powell and Skolnick's (PS) method did not find a feasible solution, GAs have converged to an artificially created minimum solution in the infeasible region. We show the proceedings of one such run in Figure .5 with $R = 1$. 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, a real-coded GA 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 at (3,2) (albeit infeasible). The reason for such suboptimal convergence is discussed earlier in Figure .4. When an identical real-coded GA but with the proposed constraint handling strategy (TS-R) is applied to the identical initial populations of 50 solutions (rest all parameter settings are also the same as in the Powell and Skolnick's case), the GA distributes well its population around and inside the feasible region (Figure .6) after 10 generations. Finally, GAs converge near to the true optimum solution at $\vec{x} = (2.243636, 2.342702)$ with a function value equal to 13.66464 (within 0.54% of the true optimum solution).

- - - - - Figure 5 here - - - - -

- - - - - Figure 6 here - - - - -

The number of feasible solutions found in each generation in all 50 runs are noted and their average is plotted in Figure .7. In the initial generation, there are not many feasible solutions (about 0.7%). Thereafter, the number of feasible solutions increase rapidly for both binary and real-coded GAs with the proposed constraint handling scheme. At around generation 25, more than 90% population members are feasible, whereas GAs with Powell and Skolnick's constraint handling strategy the initial rate of feasible solution discovery is also slower and GAs have found less than 50% of their population members in the feasible region.

- - - - - Figure 7 here - - - - -

Although binary GAs have found slightly more solutions in the feasible region that that found by real-coded GAs in this problem, Figure .8 shows that the average Euclidean distance among feasible solutions for the binary GAs is smaller than that for the real-coded GAs. This means that real-coded GAs is able to spread solutions better, thereby allowing their search operators to find better solutions. This is the reason why real-coded GAs has performed better than binary GAs. In the following, we compare these GAs to a more complicated test problem.

- - - - - Figure 8 here - - - - -

4.2 Test Problem 2

This problem is a minimization problem with five variables and 38 inequality constraints [23,24]:

$$\begin{aligned}
\text{Minimize } & f_2(\vec{x}) = 0.1365 - 5.843(10^{-7})y_{17} + 1.17(10^{-4})y_{14} + 2.358(10^{-5})y_{13} \\
& \quad +1.502(10^{-6})y_{16} + 0.0321y_{12} + 0.004324y_5 \\
& \quad +1.0(10^{-4})c_{15}/c_{16} + 37.48y_2/c_{12}, \\
\text{Subject to } & g_1(\vec{x}) \equiv 1.5x_2 - x_3 \geq 0, \\
& g_2(\vec{x}) \equiv y_1(\vec{x}) - 213.1 \geq 0, \\
& g_3(\vec{x}) \equiv 405.23 - y_1(\vec{x}) \geq 0, \\
& g_{j+2}(\vec{x}) \equiv y_j(\vec{x}) - a_j \geq 0, \quad j = 2, \ldots, 17, \\
& g_{j+18}(\vec{x}) \equiv b_j(\vec{x}) - y_j(\vec{x}) \geq 0, \quad j = 2, \ldots, 17, \\
& g_{36}(\vec{x}) \equiv y_4(\vec{x}) - 0.28/0.72y_5(\vec{x}) \geq 0, \\
& g_{37}(\vec{x}) \equiv 21 - 3496.0y_2(\vec{x})/c_{12}(\vec{x}) \geq 0, \\
& g_{38}(\vec{x}) \equiv 62212.0/c_{17}(\vec{x}) - 110.6 - y_1(\vec{x}) \geq 0, \\
& 704.4148 \leq x_1 \leq 906.3855, \quad 68.6 \leq x_2 \leq 288.88, \\
& 0 \leq x_3 \leq 134.75, \quad 193 \leq x_4 \leq 287.0966, \\
& 25 \leq x_5 \leq 84.1988.
\end{aligned} \tag{8}$$

The terms $y_j(\vec{x})$ and $c_j(\vec{x})$, and parameters a_j and b_j are given in Appendix B. The best solution reported in [23] and in [24] is

$$\vec{x}^* = (705.1803,\ 68.60005,\ 102.90001,\ 282.324999,\ 37.5850413),$$
$$f_2^* = -1.90513.$$

At this solution, none of the 38 constraints is active (an inequality constraint is active at any solution if the constraint violation is zero at that solution). Thus, this solution lies inside the feasible region [5] . This function is particularly chosen to test the proposed constraint handling method on a problem having a large number of constraints.

Table .3 shows the performance of real-coded GAs with the proposed constraint handling scheme with a population size 10×5 or 50. Powell and Skolnick's (PS) constraint handling method depends on the the penalty parameter used. For a large penalty parameter, PS method is similar in performance to the proposed method (TS-R). However, for small penalty parameter values, PS method does not perform well. The proposed method of this study (TS) does not require any penalty parameter. The performance of GAs with the proposed method (TS-R) improves with niching and further with the mutation operator. With mutation, all 50 runs have found solutions better than the best solution reported earlier.

- - - - - Table 3 here - - - - -

However, binary GAs with the proposed scheme (TS-B) cannot find feasible solutions in 9 runs and the best run found a solution within about 13% of the best-known solution. Six runs have found feasible solutions having an objective function value more than 150% of that of the best-known solution. The best, median, and worst function values are -1.66316, -1.20484, and -0.73044, respectively.

Figure .9 shows the average of the total normalized Euclidean distance of all feasible solutions in each iteration. It is clear that with the presence of niching, the average Euclidean distance of feasible solutions increases, meaning that there is more diversity present among the feasible solutions. With the introduction of mutation, this diversity further increases and GAs perform the best. Once again, this figure shows that real-coded GAs with PS ($R = 1$) and binary GAs with the proposed scheme have not been able to find and distribute solutions well in the feasible region.

- - - - - Figure 9 here - - - - -

It is also interesting to note that the best solutions obtained with real-coded GAs (TS-R) is *better* than that reported in [23,24]. The solution here is

$$\vec{x} = (707.337769,\ 68.600273,\ 102.900146,\ 282.024841,\ 84.198792),$$

[5] However, we shall see later in this section that this solution is not the true optimal solution. The solution obtained in this study is better than this solution and makes 5 of 38 constraints active.

$f_2 = -1.91460,$

which is about 0.5% better in the objective function value than that reported earlier. The main difference between this solution and that reported earlier is in the value of x_5. At this solution, five constraints (g_1, g_2, g_{34}, g_{35}, and g_{38}) are active with constraint values less than 10^{-3}. The ratio of the best $f_2(\vec{x})$ obtained in a GA generation and the best-known $f_2(\vec{x})$ (that is, $f_2^* = -1.90513$) is calculated for all 50 runs and their average is plotted in Figure .10 for different GA implementations.

- - - - - Figure 10 here - - - - -

Since, f_2^* is negative, for any suboptimal solution, the ratio $f(\vec{x})/f_2^*$ would be smaller than one. When this ratio is close to one, it is clear that the best-known solution x^* is found. The figure shows how real-coded GAs with the Powell and Skolnick's (PS) constraint handling method with $R = 1$ get stuck at suboptimal solutions. The average value of $f(\vec{x})$ where GAs converge in 50 runs is even less than 20% of f_2^*. However, real-coded GAs with the proposed constraint handling scheme finds this ratio greater than 0.8. This ratio further increases to more than 0.9 with niching alone. The figure also shows that for GAs with niching and mutation the ratio is little better than 1.0, indicating that better solutions than that reported earlier have been obtained in this study.

Because of the dependency of the performance of Powell and Skolnick's (PS) method on the penalty parameter, we do not apply this method in the subsequent test problems and only present the results for GAs with the proposed constraint handling method. Since binary GAs with the proposed constraint handling scheme also do not perform well on both the above constrained optimization problems (mainly due to its inability to maintain diverse solutions in the feasible region), we also do not apply binary GAs to subsequent test problems.

4.3 Test Problem 3

The problem is a minimization problem having 13 variables and nine inequality constraints [6]:

$$\begin{aligned}
\text{Minimize } & f_3(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i, \\
\text{Subject to } & g_1(\vec{x}) \equiv 2x_1 + 2x_2 + x_{10} + x_{11} \leq 10, \\
& g_2(\vec{x}) \equiv 2x_1 + 2x_3 + x_{10} + x_{12} \leq 10, \\
& g_3(\vec{x}) \equiv 2x_2 + 2x_3 + x_{11} + x_{12} \leq 10, \\
& g_4(\vec{x}) \equiv -8x_1 + x_{10} \leq 0, \\
& g_5(\vec{x}) \equiv -8x_2 + x_{11} \leq 0, \\
& g_6(\vec{x}) \equiv -8x_3 + x_{12} \leq 0, \\
& g_7(\vec{x}) \equiv -2x_4 - x_5 + x_{10} \leq 0, \\
& g_8(\vec{x}) \equiv -2x_6 - x_7 + x_{11} \leq 0, \\
& g_9(\vec{x}) \equiv -2x_8 - x_9 + x_{12} \leq 0, \\
& 0 \leq x_i \leq 1, \quad i = 1, \ldots, 9, \\
& 0 \leq x_i \leq 100, \quad i = 10, 11, 12, \\
& 0 \leq x_{13} \leq 1.
\end{aligned} \tag{9}$$

The optimal solution to this problem is

$$\vec{x}^* = (1,1,1,1,1,1,1,1,1,3,3,3,1), \qquad f_3^* = -15.$$

At this optimal solution, six constraints (all except g_4, g_5, and g_6) are active. This is a relatively easy problem with the objective function and constraints being linear or quadratic. Michalewicz [6] reported that all constraint handling methods used to solve this problem have found the optimal solution. Not surprisingly, all methods tried here have also found the true optimal solution many times, as depicted in Table .4. However, it is important to note that here no effort has been spent to exploit the structure of the constraints, whereas in the other study [6] special closed operators (in addition to standard GA operators) are applied on linear constraints to satisfy them. Although a similar approach can also be used with the proposed method, we do not consider the special cases here (because such operators can only be used to a special class of constraints), instead present a generic strategy for solving constraint optimization problems.

- - - - - Table 4 here - - - - -

GA parameters are set as before. Since there are 13 variables, a population size of (10×13) or 130 is used. With the presence of niching, the performance of GAs becomes better and 38 out of 50 runs have found solutions within 1% from the true optimum. With the presence of niching and mutation, the performance of GAs is even better.

Average normalized Euclidean distance of feasible solutions are plotted in Figure .11 and average ratio of the best fitness obtained by GAs to the best-known objective function value f_3^* is plotted in Figure .12. Figures show how diversity among feasible solutions is restored in GAs with niching and mutation. The latter figure also shows the suboptimal convergence of GAs without niching in some runs.

- - - - - Figure 11 here - - - - -

- - - - - Figure 12 here - - - - -

4.4 Test Problem 4

This problem has eight variables and six inequality constraints [6]:

$$\begin{aligned}
\text{Minimize } & f_4(\vec{x}) = x_1 + x_2 + x_3 \\
\text{Subject to } & g_1(\vec{x}) \equiv 1 - 0.0025(x_4 + x_6) \geq 0, \\
& g_2(\vec{x}) \equiv 1 - 0.0025(x_5 + x_7 - x_4) \geq 0, \\
& g_3(\vec{x}) \equiv 1 - 0.01(x_8 - x_5) \geq 0, \\
& g_4(\vec{x}) \equiv x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0, \\
& g_5(\vec{x}) \equiv x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0, \\
& g_6(\vec{x}) \equiv x_3 x_8 - x_3 x_5 + 2500 x_5 - 1250000 \geq 0, \\
& 100 \leq x_1 \leq 10000, \\
& 1000 \leq x_2, x_3 \leq 10000, \\
& 10 \leq x_i \leq 1000, \quad i = 4, \ldots, 8.
\end{aligned} \tag{10}$$

The optimum solution is

$$\begin{aligned}
\vec{x}^* = & (579.3167,\ 1359.943,\ 5110.071,\ 182.0174,\ 295.5985,\ 217.9799, \\
& \quad 286.4162,\ 395.5979), \\
f_4^* = & 7049.330923.
\end{aligned}$$

All six constraints are active at this solution.

Table .5 shows the performance of GAs with different constraint handling methods. Michalewicz [6] experienced that this problem is difficult to solve. Out of seven methods tried in that study, three found solutions somewhat closer to the true optimum. The best solution obtained by any method used in that study had an objective function value equal to 7377.976, which is about 4.66% worse than the true optimal objective function value. A population size of 70 was used and floating-point GAs with a number specialized crossover and mutation operators were run for 5,000 generations, totaling 350,070 function evaluations. As mentioned earlier, in this study, we have used a different real-coded GA with SBX operator and we have consistently found solutions very close to the true optimum with 80,080 function evaluations (population size 80, maximum generations 1,000). However, the best solution obtained by GAs with niching 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. Thus, GAs with the proposed constraint handling method has been able to find better solutions than that found by any method used in [6]. Moreover, the median solution found in GAs with niching and mutation is even better than the best solution found in [13].

- - - - - Table 5 here - - - - -

Figures .13 and .14 show the effect of niching on the average Euclidean distance among feasible solutions and the average proportion of feasible solutions in the population of 50 GA runs. The former figure shows that niching helps to maintain diversity in the population. When mutation operator is added, the diversity among feasible solutions is better and is maintained for longer generations. The latter figure shows that initially no solution was feasible. With generations, more number of feasible solutions are continuously found. Since niching helps to maintain diversity in feasible solutions, more feasible solutions are also found with generations.

- - - - - Figure 13 here - - - - -

- - - - - Figure 14 here - - - - -

4.5 Test Problem 5

This problem has seven variables and four nonlinear constraints [6]:

$$
\begin{aligned}
\text{Minimize } & f_5(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11.0)^2 \\
& \qquad +10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7, \\
\text{Subject to } & g_1(\vec{x}) \equiv 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0, \\
& g_2(\vec{x}) \equiv 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0, \\
& g_3(\vec{x}) \equiv 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0, \\
& g_4(\vec{x}) \equiv -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0, \\
& -10 \leq x_i \leq 10, \quad i = 1, \ldots, 7.
\end{aligned}
\tag{11}
$$

The optimal solution is

$$
\begin{aligned}
\vec{x}^* = & (2.330499,\ 1.951372,\ -0.4775414,\ 4.365726,\ -0.6244870, \\
& 1.038131,\ 1.594227), \\
f_5^* = & 680.6300573.
\end{aligned}
$$

At this solution, constraints g_1 and g_4 are active. Michalewicz [6] reported that the feasible region for this problem occupies only about 0.5% of the search space.

Table .6 presents the performance of GAs with the proposed constraint handling method with a population size of 10×7 or 70. In this problem also, niching seems to have done better. In the first case, when GAs are run without niching and mutation, all GA runs get stuck to a solution closer to the true optimum solution at around 577 generations. Thus, increasing the generation number to 5,000 does not alter GA's performance. However, when niching is introduced among feasible solutions, diversity of solutions is maintained and GAs with SBX operator can find better solutions. For space restrictions, we do not present generation-wise plots for this and subsequent test problems.

- - - - - Table 6 here - - - - -

The best result reported in [6] is with penalty function approach in which the penalty parameters are changed with generation. With a total of 350,070 function evaluations, the best, median, and worst objective function values of 10 runs were 680.642, 680.718, and 680.955, respectively. Table .6 shows that 50 GA runs with the proposed constrained handling method have found best, median, and worst solutions as 680.634, 680.642, 680.651, respectively with an identical number of function evaluations. These solutions are much closer to the true optimum solution than that found by the best algorithm in [6].

4.6 Test Problem 6

This problem has five variables and six inequality constraints [23,7]:

$$\begin{aligned}
\text{Minimize } & f_6(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141, \\
\text{Subject to } & g_1(\vec{x}) \equiv 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \geq 0, \\
& g_2(\vec{x}) \equiv 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \leq 92, \\
& g_3(\vec{x}) \equiv 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \geq 90, \\
& g_4(\vec{x}) \equiv 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \leq 110, \\
& g_5(\vec{x}) \equiv 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \geq 20, \\
& g_6(\vec{x}) \equiv 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 25, \\
& 78 \leq x_1 \leq 102, \\
& 33 \leq x_2 \leq 45, \\
& 27 \leq x_i \leq 45, \quad i = 3, 4, 5.
\end{aligned} \tag{12}$$

The best-known optimum solution [23] is

$$\vec{x}^* = (78.0, 33.0, 29.995, 45.0, 36.776), \quad f_6^* = -30,665.5.$$

At this solution, constraints g_2 and g_5 are active. The best-known GA solution to this problem obtained elsewhere [3] using a multi-level penalty function method is

$$\vec{x}^{\text{GA}} = (80.49, 35.07, 32.05, 40.33, 33.34), \quad f_6^{\text{GA}} = -30,005.7,$$

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

Table .7 presents the performance of GAs with the proposed constraint handling method with a population size 10×5 or 50. Once again, it is found that the presence of niching improves the performance of GAs. When GAs are run longer, the solution improves in the presence of niching. GAs without niching and mutation could not improve the solution much with more generations, but GAs with niching continuously improve the solution with generations. The presence of niching and mutation finds the best solution. The important aspect is that 47 of 50 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 [3], solved using binary GAs with a multi-level penalty function method.

- - - - - Table 7 here - - - - -

4.7 Test Problem 7

This problem has five variables and three equality constraints [6]:

$$\begin{aligned} \text{Minimize } & f_7(\vec{x}) = \exp(x_1x_2x_3x_4x_5), \\ \text{Subject to } & h_1(\vec{x}) \equiv x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10, \\ & h_2(\vec{x}) \equiv x_2x_3 - 5x_4x_5 = 0, \\ & h_3(\vec{x}) \equiv x_1^3 + x_2^3 = -1, \\ & -2.3 \le x_i \le 2.3, \quad i = 1,2, \\ & -3.2 \le x_i \le 3.2, \quad i = 3,4,5. \end{aligned} \tag{13}$$

The optimal solution to this problem is as follows:

$$\begin{aligned} & \vec{x}^* = (-1.717143,\ 1.595709,\ 1.827247,\ -0.7636413,\ -0.7636450), \\ & f_7^* = 0.053950. \end{aligned}$$

Equality constraints are handled by converting them as inequality constraints as $\delta - |h_k(\vec{x})| \ge 0$ for all k, as mentioned earlier. In this problem, δ is set to 10^{-3}, in order to allow some room for the search algorithm to work on. Table .8 shows the performance of GAs with a maximum of 350,050 function evaluations (population size 50, maximum generations 7,000). Although niching alone could not improve performance much, along with mutation 19 out of 50 runs have found a solution within 1% of the optimal objective function value.

- - - - - Table 8 here - - - - -

4.8 Test Problem 8

This problem has 10 variables and eight constraints [6]:

$$
\begin{aligned}
\text{Minimize } f_8(\vec{x}) &= x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\
&\quad +4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 \\
&\quad +7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45, \\
\text{Subject to } & g_1(\vec{x}) \equiv 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0, \\
& g_2(\vec{x}) \equiv -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0, \\
& g_3(\vec{x}) \equiv 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0, \\
& g_4(\vec{x}) \equiv -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0, \\
& g_5(\vec{x}) \equiv -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0, \\
& g_6(\vec{x}) \equiv -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0, \\
& g_7(\vec{x}) \equiv -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0, \\
& g_8(\vec{x}) \equiv 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0, \\
& -10 \leq x_1 \leq 10, \quad i = 1, \ldots, 10.
\end{aligned}
\tag{14}
$$

The optimum solution to this problem is as follows:

$$
\begin{aligned}
\vec{x}^* = (&2.171996,\ 2.363683,\ 8.773926,\ 5.095984,\ 0.9906548,\ 1.430574, \\
&1.321644,\ 9.828726,\ 8.280092,\ 8.375927), \quad f_8^* = 24.3062091.
\end{aligned}
$$

The first six constraints are active at this solution.

Table .9 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 niching 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 [6] 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.

- - - - - Table 9 here - - - - -

4.9 Welded beam design problem revisited

We shall now apply the proposed method to solve the welded beam design problem discussed earlier. GA parameter values same as that used earlier are also used here. Table .10 presents the performance of GAs with a population size 80. Real-coded GAs without niching is good enough to find a solution within 2.6% of the best objective function value. However, with the introduction of niching, 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% (to be exact) 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 user would like to use such an efficient yet robust optimization algorithm.

- - - - - Table 10 here - - - - -

When binary GAs (each variable is coded in 10 bits) with (or without) niching are applied, no solution within 50% of the best-known solution is found. With niching on, the best, median, and worst objective function values of optimized solutions are found to be 3.82098, 8.89996, and 14.29893, respectively. Clearly, the real-coded GA implementation with SBX operator is better able to find near-optimum solutions than the binary GAs.

Figure .15 shows the performance of various GAs in terms of finding a solution closer to the true optimum solution. Average ratio of the best objective function value obtained by GAs to the best-known objective function value of 50 GA runs is plotted with generation number. The figure shows that binary GAs prematurely converge to suboptimal solutions, whereas real-coded GAs with niching (and with or without mutation) find solutions very close to the true optimal solution.

- - - - - Figure 15 here - - - - -

4.10 Summary of Results

Here, we summarize the best GA results obtained in this paper (Table .11) and compare that with the best reported results in earlier studies. It is found here that the reported solution (marked with a '$*$') of test problem 2 is not the true optimum. The solution obtained here is better than this previously-known best solution. In all problems marked by a '#', a better solution than that obtained by a previous GA implementation is obtained. In all other cases, the best solution of this study matches that of the previous GA studies.

- - - - - Table 11 here - - - - -

For test problems 3 to 8, 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. In some test problems, the worst GA solution (albeit a few isolated cases) has an objective function value away from the true optimal solution. This is because reasonable values (but fixed for all problems) of GA parameter values are used in this study. With a parametric study of important GA parameters (for example, population size (here, 10 times the number of variables is used), η_c for SBX operator (here, 1 is used), η_m for mutation operator (here, a linear variation from 1 to 100 is used)), the overall performance of GAs and the worst GA solution can both be improved.

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.

5 Conclusions

The major difficulty in handling constraints using penalty function methods in GAs and in classical optimization methods has been to set appropriate values 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. The need of a penalty parameter arises in order to maintain the objective function value and the constraint violation values of the same order. In the proposed method, solutions are never compared in terms of both objective function value and constraint violation information. Thus, penalty parameters are not needed in the proposed approach. Infeasible solutions are penalized in a way so as to provide a search direction towards the feasible region and when adequate feasible solutions are found a niching scheme is used to maintain diversity. This aids GA's crossover operator to find better and better solutions with generation. All these have been possible mainly because of the population approach of GAs and ability to have pair-wise comparison of solutions using the tournament selection operator. It is important to note that the proposed constraint handling approach is not suitable for classical point-by-point search methods. Thus, GAs or other evolutionary computations methods have a niche over classical methods to handle constraints with the proposed approach.

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. On one test problem, a solution better than that reported as the optimal solution earlier is also found.

It has also been observed that since all problems used in this study are defined in the real space and the feasible regions are usually of arbitrary shape (convex or concave), the use of real-coded GAs with a controlled search operator are more suited than binary GAs in finding feasible children solutions from feasible parent solutions. In this respect, the use of real-coded GAs with SBX and a parameter-based mutation operator have been found to be useful. It would be worthwhile to investigate how the proposed constraint handling method would perform with binary GAs to problems having discrete variables.

All problem-independent GA parameters are used in this study. In all test problems, reasonable values for these GA parameters are used. It would be worthwhile to do a parametric study of important GA parameters to improve the performance of GAs even further.

The results on the limited test problems studied here are interesting and show promise for a reliable and efficient constrained optimization task through GAs.

Acknowledgments

The author greatly appreciates the programming help provided by couple of his students: Samir Agrawal and Priya Rawat. Comments made by Zbigniew Michalewicz on an earlier version of the paper are highly appreciated. Some portions of this study have been performed during the author's visit to the University of Dortmund, Germany, for which the author acknowledges the support from Alexander von Humboldt Foundation.

References

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

[2] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering optimization methods and applications* (Wiley, New York, 1983).

[3] A. Homaifar, S. H.-V. Lai, X. Qi, Constrained optimization via genetic algorithms. *Simulation* **62**/4 (1994) 242–254.

[4] J. A. Joines and C. R. Houck, On the use of nonstationary penalty functions to solve nonlinear constrained optimization problems with GAs. in: Z. Michalewicz, ed., *Proceedings of the International Conference on Evolutionary Computation* (IEEE Press, Piscataway, 1994) 579–584.

[5] Z. Michalewicz and N. Attia, Evolutionary optimization of constrained problems. in: A. V. Sebald and L. J. Fogel, eds., *Proceedings of the Third Annual Conference on Evolutionary Programming.* (World Scientific, Singapore, 1994) 98–108.

[6] Z. Michalewicz, Genetic algorithms, numerical optimization, and constraints, in: L. Eshelman, ed., *Proceedings of the Sixth International Conference on Genetic Algorithms* (Morgan Kauffman, San Mateo, 1995) 151–158.

[7] Z. Michalewicz and M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, *Evolutionary Computation* **4**/1 (1996) 1–32.

[8] D. Powell and M. M. Skolnick, Using genetic algorithms in engineering design optimization with nonlinear constraints. in: S. Forrest, ed., *Proceedings of the Fifth International Conference on Genetic Algorithms* (Morgan Kauffman, San Mateo, 1993) 424–430.

[9] K. Deb, Optimal design of a welded beam structure via genetic algorithms, *AIAA Journal* **29**/11 (1991) 2013–2015.

[10] J-H. Kim and H. Myung, Evolutionary programming techniques for constraint optimization problems. *IEEE Transcations on Evolutionary Computation* **1**/2 (1997) 129–140.

[11] D. E. Goldberg, Personal communication (September 1992).

[12] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms* (Morgan Kauffman, San Mateo, 1989) 191–197.

[13] Z. Michalewicz, Personal communication (June 1998).

[14] K. Deb, and R. B. Agrawal, Simulated binary crossover for continuous search space, *Complex Systems* **9** (1995) 115–148.

[15] K. Deb and M. Goyal, A combined genetic adaptive search (GeneAS) for engineering design, *Computer Science and Informatics* **26**/4 (1996) 30–45.

[16] K. Deb, and D. E. Goldberg, An investigation of niche and species formation in genetic function optimization, in: J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms* (Morgan Kauffman, San mateo, 1989) 42–50.

[17] D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning (Addison-Wesley, Reading, 1989).

[18] I. Rechenberg, Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution (Fromman-Holzboog Verlag, Stuttgart, 1973).

[19] H.-P. Schwefel, Numerical Optimization of Computer Models (Wiley, New York, 1983).

[20] K. Deb and A. Kumar, Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems, *Complex Systems* **9**/6 (1995) 431–454.

[21] D. E. Goldberg, K. Deb, and J. H. Clark, Genetic algorithms, noise, and the sizing of populations *Complex Systems* **6** (1992) 333–362.

[22] G. Harik, E. Cantu-Paz, D. E. Goldberg, and B. L. Miller, The gambler's ruin problem, genetic algorithms, and the sizing of populations, in: T. Bäck, Z. Michalewicz, and X. Yao, eds., *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation* (IEEE Press, Piscataway, 1997) 7–12.

[23] D. M. Himmelblau, Applied nonlinear programming (McGraw-Hill, New York, 1972).

[24] W. Hock and K. Schittkowski, Test examples for nonlinear programming code. *Lecture Notes on Economics and Mathematical Systems* **187** (Springer-Verlag, Berlin, 1981).

Appendix

A Simulated Binary Crossover and Parameter-based Mutation

The development of simulated binary crossover operator (SBX) and parameter-based mutation operator for handling floating point numbers were performed in earlier studies [14,15]. Here, we simply present the procedures for calculating children solutions from parent solutions under crossover and mutation operators.

A.1 Simulated Binary Crossover (SBX) Operator

The procedure of computing children solutions $y^{(1)}$ and $y^{(2)}$ from two parent solutions $x^{(1)}$ and $x^{(2)}$ are as follows:

(1) Create a random number u between 0 and 1.
(2) Find a parameter $\bar{\beta}$ using a polynomial probability distribution, developed in [14] from a schema processing point of view, as follows:

$$\bar{\beta} = \begin{cases} (2u)^{\frac{1}{\eta_c+1}}, & \text{if } u \le 0.5, \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise,} \end{cases} \tag{A.1}$$

where 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:

$$y^{(1)} = 0.5\left[(x^{(1)} + x^{(2)}) - \bar{\beta}|x^{(2)} - x^{(1)}|\right],$$
$$y^{(2)} = 0.5\left[(x^{(1)} + x^{(2)}) + \bar{\beta}|x^{(2)} - x^{(1)}|\right].$$

The above procedure is used for variables where no lower and upper bounds are specified. Thus, the children solutions can lie anywhere in the real space [$-\infty$, ∞] with varying probability. For calculating the children solutions where lower and upper bounds (x^l and x^u) of a variable are specified, equation A.1 needs to be changed as follows:

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

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

$$\beta = 1 + \frac{2}{y^{(2)} - y^{(1)}} \min[(x^{(1)} - x^l),\ (x^u - x^{(2)})].$$

It is assumed here that $x^{(1)} < x^{(2)}$. A simple modification to the above equation can be made for $x^{(1)} > x^{(2)}$. The above procedure allows a zero probability of creating any children solution outside the prescribed range [x^l, x^u]. It is intuitive that equation A.2 reduces to equation A.1 for $x^l = -\infty$ and $x^u = \infty$.

For handling multiple variables, each variable is chosen with a probability 0.5 in this study and the above SBX operator is applied variable-by-variable. This way about half of the variables get crossed over under the SBX oparator. SBX operator can also be applied once on a line joining the two parents. In all simulation results here, we have used $\eta_c = 1$.

A.2 Parameter-based Mutation Operator

A polynomial probability distribution is used to create a solution y in the vicinity of a parent solution x [15]. The following procedure is used for variables where lower and upper boundaries are not specified:

(1) Create a random number u between 0 and 1.
(2) Calculate the parameter $\bar{\delta}$ as follows:

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{\eta_m+1}} - 1, & \text{if } u \leq 0.5, \\ 1 - [2(1-u)]^{\frac{1}{\eta_m+1}}, & \text{otherwise,} \end{cases} \tag{A.3}$$

where η_m is the distribution index for mutation and takes any non-negative value.
(3) Calculate the mutated child as follows:

$$y = x + \bar{\delta}\Delta_{\max},$$

where $\Delta_{\max}$ is the maximum perturbance allowed in the parent solution.

For variables where lower and upper boundaries (x^l and x^u) are specified, above equation may be changed as follows:

$$\bar{\delta} = \begin{cases} [2u + (1-2u)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1, & \text{if } u \leq 0.5, \\ 1 - [2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}}, & \text{otherwise,} \end{cases} \tag{A.4}$$

where $\delta = \min[(x - x^l),\ (x^u - x)]/(x^u - x^l)$. This ensures that no solution would be created outside the range [x^l, x^u]. In this case, we set $\Delta_{\max} = x^u - x^l$. Equation A.4 reduces to equation A.3 for $x^l = -\infty$ and $x^u = \infty$.

Using above equations, we can calculate the expected normalized perturbance ($(y - x)/(x^u - x^l)$) of the mutated solutions in both positive and negative sides separately. We observe that this value is $O(1/\eta_m)$. Thus, in order to get a mutation effect of 1% perturbance in solutions, we should set $\eta_m \approx 100$. In all our simulations wherever mutation is used, 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. This setting of the mutation operator is arbitrarily chosen and has found to have worked well in all problems tried in this paper. No effort is spent in tuning these parameters for obtaining better results.

B Terms and Parameters Used in Test Function 2

The following terms are required to compute the objective function and constraints for the test problem 2 [23,24]:

$$\begin{aligned}
y_1(\vec{x}) &= x_1 + x_2 + 41.6,\\
c_1(\vec{x}) &= 0.024x_4 - 4.62,\\
y_2(\vec{x}) &= 12.5/c_1(\vec{x}) + 12.0,\\
c_2(\vec{x}) &= 0.0003535x_1x_1 + 0.5311x_1 + 0.08705y_2(\vec{x})x_1,\\
c_3(\vec{x}) &= 0.052x_1 + 78.0 + 0.002377y_2(\vec{x})x_1,\\
y_3(\vec{x}) &= c_2(\vec{x})/c_3(\vec{x}),\\
y_4(\vec{x}) &= 19.0y_3(\vec{x}),\\
c_4(\vec{x}) &= 0.04782(x_1 - y_3(\vec{x})) + 0.1956(x_1 - y_3(\vec{x}))^2/x_2 + 0.6376y_4(\vec{x}) + 1.594y_3(\vec{x}),\\
c_5(\vec{x}) &= 100.0x_2,\\
c_6(\vec{x}) &= x_1 - y_3(\vec{x}) - y_4(\vec{x}),\\
c_7(\vec{x}) &= 0.95 - c_4(\vec{x})/c_5(\vec{x}),\\
y_5(\vec{x}) &= c_6(\vec{x})c_7(\vec{x}),\\
y_6(\vec{x}) &= x_1 - y_5(\vec{x}) - y_4(\vec{x}) - y_3(\vec{x}),\\
c_8(\vec{x}) &= 0.995(y_4(\vec{x}) + y_5(\vec{x})),
\end{aligned}$$

$$
\begin{aligned}
y_7(\vec{x}) &= c_8(\vec{x})/y_1(\vec{x}),\\
y_8(\vec{x}) &= c_8(\vec{x})/3798.0,\\
c_9(\vec{x}) &= y_7(\vec{x}) - 0.0663 y_7(\vec{x})/y_8(\vec{x}) - 0.3153,\\
y_9(\vec{x}) &= 96.82/c_9(\vec{x}) + 0.321 y_1(\vec{x}),\\
y_{10}(\vec{x}) &= 1.29 y_5(\vec{x}) + 1.258 y_4(\vec{x}) + 2.29 y_3(\vec{x}) + 1.71 y_6(\vec{x}),\\
y_{11}(\vec{x}) &= 1.71 x_1 - 0.452 y_4(\vec{x}) + 0.58 y_3(\vec{x}),\\
c_{10}(\vec{x}) &= 12.3/752.3,\\
c_{11}(\vec{x}) &= 1.75 y_2(\vec{x}) 0.995 x_1,\\
c_{12}(\vec{x}) &= 0.995 y_{10}(\vec{x}) + 1998.0,\\
y_{12}(\vec{x}) &= c_{10}(\vec{x}) x_1 + c_{11}(\vec{x})/c_{12}(\vec{x}),\\
y_{13}(\vec{x}) &= c_{12}(\vec{x}) - 1.75 y_2(\vec{x}),\\
y_{14}(\vec{x}) &= 3623.0 + 64.4 x_2 + 58.4 x_3 + 146312.0/(y_9(\vec{x}) + x_5),\\
c_{13}(\vec{x}) &= 0.995 y_{10}(\vec{x}) + 60.8 x_2 + 48.0 x_4 - 0.1121 y_{14}(\vec{x}) - 5095.0,\\
y_{15}(\vec{x}) &= y_{13}(\vec{x})/c_{13}(\vec{x}),\\
y_{16}(\vec{x}) &= 148000.0 - 331000.0 y_{15}(\vec{x}) + 40 y_{13}(\vec{x}) - 61.0 y_{15}(\vec{x}) y_{13}(\vec{x}),\\
c_{14}(\vec{x}) &= 2324.0 y_{10}(\vec{x}) - 28740000.0 y_2(\vec{x}),\\
y_{17}(\vec{x}) &= 14130000.0 - 1328.0 y_{10}(\vec{x}) - 531.0 y_{11}(\vec{x}) + c_{14}(\vec{x})/c_{12}(\vec{x}),\\
c_{15}(\vec{x}) &= y_{13}(\vec{x})/y_{15}(\vec{x}) - y_{13}(\vec{x})/0.52,\\
c_{16}(\vec{x}) &= 1.104 - 0.72 y_{15}(\vec{x}),\\
c_{17}(\vec{x}) &= y_9(\vec{x}) + x_5.
\end{aligned}
$$

The values of $a[i]$ and $b[i]$ for $i = 1, \ldots, 18$ are as follows:

a[i] = {0, 0, 17.505, 11.275, 214.228, 7.458,
0.961, 1.612, 0.146, 107.99, 922.693,
926.832, 18.766, 1072.163, 8961.448,
0.063, 71084.33, 2802713.0},

b[i] = {0, 0, 1053.6667, 35.03, 665.585,
584.463, 265.916,7.046, 0.222, 273.366,
1286.105, 1444.046, 537.141, 3247.039,
26844.086, 0.386, 140000.0, 12146108.0}.

Table .1

Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using binary GAs with different penalty parameter values on the welded beam design problem.

Table .2

Number of runs (out of 50 runs) converged within ϵ% of the optimum solution for real-coded GAs with two constraint handling techniques—Powell and Skolnick's method (PS) with different R values and the proposed method (TS-R)—on test problem 1.

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 (TS-R) and using Powell and Skolnick's method (PS-a) with different penalty parameters $R = 10^a$ on test problem 2.

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 3.

Table .5

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.

Table .6

Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using GAs with the proposed constraint handling scheme on test problem 5.

Table .7

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 6.

Table .8

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 7.

Table .9

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 8.

Table .10

Number of runs (out of 50 runs) converged within ϵ% of the best-known solution using binary GAs (TS-B) and real-coded GAs (TS-R) with the proposed constraint handling scheme on the welded beam design problem.

Table .11

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

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

Fig. .2. Contour plot of the objective function $f(x, y)$ and the feasible search space are shown. Contours are plotted at $f(x, y)$ values 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, and 1.

Fig. .3. Contour plot of the fitness function $F(x, y)$ at a particular generation is shown. Contours are plotted at $F(x, y)$ values 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1, 2, 3, and 4.

Fig. .4. Powell and Skolnick's constraint handling scheme is illustrated. Six solid circles are solutions in a GA population.

Fig. .5. Population history at initial generation (marked with open circles), at generation 10 (marked with '$\times$') and at generation 50 (marked with open boxes) using Powell and Skolnick's method ($R = 1$) on test problem 1. The population converges to a wrong, infeasible solution.

Fig. .6. Population history at initial generation (marked with open circles), at generation 10 (marked with '$\times$') and at generation 50 (marked with open boxes) using the proposed scheme on test problem 1. The population converges to a solution very close to the true constrained optimum solution on a constraint boundary.

Fig. .7. Comparison of the proposed (TS) and Powell and Skolnick's (PS) methods for constraint handling in terms of average number of feasible solutions found in 50 GA runs on test problem 1.

Fig. .8. Comparison of the proposed (TS) and Powell and Skolnick's (PS) methods for constraint handling in terms of average normalized Euclidean distance among feasible solutions in 50 GA runs on test problem 1.

Fig. .9. Average normalized Euclidean distance of feasible solutions versus generation number on test problem 2.

Fig. .10. Average ratio of the best $f(\vec{x})$ found by GAs to f_2^* is plotted versus generation number on test problem 2.

Fig. .11. Average normalized Euclidean distance of feasible solutions for different real-coded GAs with the proposed constraint handling scheme is plotted versus generation number on test problem 3.

Fig. .12. Average $f(\vec{x})/f_3^*$ obtained by different real-coded GAs with the proposed constraint handling scheme is plotted versus generation number on test problem 3.

Fig. .13. Average Euclidean distance of feasible solutions in 50 runs of real-coded GAs with the proposed constraint handling scheme on test problem 4.

Fig. .14. Average proportion of feasible solutions in the population obtained by 50 runs of real-coded GAs with the proposed constraint handling scheme on test problem 4.

Fig. .15. Average $f(\vec{x})/f_w^*$ obtained by different GAs with the proposed constraint handling scheme is plotted versus generation number on the welded beam design problem.