# Optimizing Engineering Designs Using a Combined Genetic Search

**Kalyanmoy Deb and Mayank Goyal**
Mechanical Engineering Department
Indian Institute of Technology
Kanpur, UP 208 016, India
E-mail: deb@iitk.ernet.in

## Abstract

In the optimization of engineering designs, traditional search and optimization methods face at least two difficulties: (i) since each is specialized in solving a particular type of problem, one method does not work well on different types of problems (ii) most of them are designed to work on continuous search spaces. Since different optimal engineering design problems give rise to objective and constraint functions of varying degree of nonlinearity and since most engineering design problems involve mixed variables (zero-one, discrete, and continuous), designers often face difficulty in using the traditional methods. In this paper, a combined genetic search technique (GeneAS) is suggested to solve mixed-integer programming problems often encountered in engineering design activities. GeneAS uses a combination of binary-coded and real-coded GAs to handle different types of variables. In handling discrete variables, GeneAS restricts its search only to the permissible values of the variables. Thus, minimal search effort is required in converging to the optimum solution. The efficiency and ease of application of the proposed method are demonstrated by solving four different engineering design problems. Successful application of GeneAS in these problems suggests its use to other more complex engineering design problems.

## 1 INTRODUCTION

Optimization techniques are primarily used in engineering design activities to achieve a competitive design, which maximizes or minimizes a certain objective by satisfying a number of constraints. The first step in an optimal engineering design is to formulate the underlying nonlinear programming (NLP) problem by writing the mathematical functions relating to the objective and constraints. A generic NLP problem can be written as follows (Deb, 1995; Reklaitis et al., 1983):

$$\begin{array}{lll} \text{Minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_j(\boldsymbol{x}) \geq 0, & j = 1, 2, \ldots, J; \\ & h_k(\boldsymbol{x}) = 0, & k = 1, 2, \ldots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \ldots, N. \end{array} \quad (1)$$

In the above NLP problem, the variable vector $\boldsymbol{x}$ represents a set of variables $x_i$, $i = 1, 2, \ldots, N$, and $f(\boldsymbol{x})$ is the objective function. The functions $g_j(\boldsymbol{x})$ and $h_k(\boldsymbol{x})$ are inequality and equality constraints, respectively. The functional form (order of nonlinearity) of these functions varies from problem to problem. Moreover, the design variables are usually of different types—zero-one, discrete, or continuous. Thus, designers must rely on an optimization algorithm which is flexible yet efficient enough to handle most of such problems.

There are at least two difficulties in using traditional optimization algorithms to solve engineering design optimization problems:

1. Each traditional optimization algorithm is specialized to solve a particular type of problems. When faced with a different type of problem, the same algorithm may not work as well.
2. Most traditional optimization algorithms are designed to work on continuous variables. In solving problems having discrete search space (in the case of zero-one or discrete variables), these methods assume the search space to be continuous and then introduce artificial constraints to favor permissible discrete values. These fix-ups not only increase the computational complexity of the underlying problem, but the algorithm also spends a considerable amount of effort in evaluating non-feasible solutions.

In this paper, we suggest a flexible yet efficient optimization technique which uses the concept of **Gene**tic **A**daptive **S**earch (GeneAS). A mixed coding scheme is allowed to code all zero-one and discrete variables in binary strings. The continuous variables and, sometimes, discrete variables are coded directly. The proposed technique is a straightforward tripartite GA with modified crossover and mutation operators. Since a mixed coding is used, crossover and mutation are performed variable-by-variable. To cross a variable coded in a binary string, a binary crossover operator (single-point, uniform, or others) is used and to cross a variable coded directly, the simulated binary crossover (SBX) developed elsewhere (Deb and Agrawal, 1995) is used. Different mutation operators are also used based on the coding of the variable. Since continuous variables are coded directly, the algorithm is flexible and eliminates the difficulties (Hamming cliff problem, difficulty to achieve arbitrary precision) of coding a continuous variable with a binary coding. Moreover, since the SBX and the real-coded mutation operators are designed to have a *search power* similar to their binary counterparts, the overall algorithm performs better than the binary-coded GAs on mixed-variable problems.

In the remainder of the paper, we briefly discuss the difficulties of traditional optimization techniques in solving engineering design problems. Thereafter, we discuss the mixed coding scheme used in the proposed GeneAS technique and then demonstrate its efficiency by presenting a number of case studies from engineering design. GeneAS is compared with two well-known traditional methods and is found to obtain better designs in all cases. Although problems having a few variables and constraints are chosen, results are encouraging and suggest the application of GeneAS to more complex engineering design optimization problems.

## 2 TRADITIONAL OPTIMIZATION METHODS

Traditional optimization methods can be classified into two distinct groups: direct and gradient-based methods. In direct search methods, only objective function ($f(\boldsymbol{x})$) and constraint values ($g_j(\boldsymbol{x})$, $h_k(\boldsymbol{x})$) are used to guide the search strategy, whereas gradient-based methods use the first and/or second-order derivatives of the objective function and/or constraints to guide the search process. Since derivative information is not used, the direct search methods are usually slow, requiring many function evaluations for convergence. For the same reason, they can be applied to many problems without a major change of the algorithm. On the other hand, gradient-based methods quickly converge to an optimal solution, but are not efficient in nondifferentiable or discontinuous problems. In addition, there are some common difficulties with most of the traditional direct and gradient-based techniques:

- The convergence to an optimal solution depends on the chosen initial solution.
- Most algorithms tend to get *stuck* to a suboptimal solution.
- An algorithm efficient in solving one optimization problem may not be efficient in solving a different optimization problem.
- Algorithms are not efficient in handling problems having discrete variables.
- Algorithms cannot be efficiently used on a parallel machine.

Because of the nonlinearities and complex interactions among problem variables often exist in engineering design problems, the search space may have more than one optimal solutions, of which most are undesired locally optimal solutions having inferior objective function values. When solving these problems, if traditional methods get attracted to any of these locally optimal solutions, there is no escape.

Every traditional optimization algorithm is designed to solve a specific type of problems. For example, the geometric programming method is designed to solve only posynomial-type objective function and constraints. Geometric programming is efficient in solving such problems but can not be applied suitably to solve other types of functions. The conjugate direction or conjugate gradient methods have convergence proofs for solving quadratic objective functions having one optimal solution, but they are not expected to work well in problems having multiple optimal solutions. The Frank-Wolfe successive linear programming method (Reklaitis et al., 1983) works efficiently on linear-like function and constraints, but for solving nonlinear problems its performance largely depends on the chosen initial conditions. Thus, one algorithm may be best suited for one problem and may not even be applicable to a different problem. This requires designers to know a number of optimization algorithms to solve different design problems.

In most engineering designs, some problem variables are restricted to take discrete values only. This requirement often arises to meet the market conditions. For example, if the diameter of a mechanical component is a design variable and the component is likely to be procured off-the-self, the optimization algorithm cannot use any arbitrary diameter. An usual practice to tackle such problems is to assume that all variables are continuous during the optimization process. Thereafter, an available size closer to the obtained solution is recommended. But, there are major difficulties with this approach. Firstly, since infeasible values of a design variable are allowed in the optimization process, the optimization algorithm spends enormous time in computing infeasible solutions (in some cases, it may not be possible to compute an infeasible solution). This makes the search effort inefficient. Secondly, as post-optimization calculations, the nearest lower and upper available sizes are to be checked for each infeasible discrete variable. For $N$ such discrete variables, a total of $2^N$ additional solutions need to be evaluated. Thirdly, two options checked for each variable may not guarantee to form the optimal combination with respect to other variables. All these difficulties can be eliminated if only feasible values of the variables are allowed during the optimization process.

Many engineering optimization problems require the use of a simulation software involving finite element method, computational fluid mechanics approach, nonlinear equation solving, or other computationally extensive methods to compute the objective function and constraints. Because of the affordability and availability of parallel computing machines, it becomes now convenient to use parallel machines in solving complex engineering design optimization problems. Since most traditional methods use point-by-point approach, where one solution gets updated to a new solution in one iteration, the advantage of parallel machines cannot be exploited.

The above discussion suggests that traditional methods are not good candidates as efficient optimization algorithms for engineering design. In the following section, we describe the proposed GeneAS technique which can alleviate some of the above difficulties and may constitute an efficient optimization tool.

# 3 GeneAS TECHNIQUE

We present the GeneAS technique by first outlining its mixed coding scheme and then describing its operators.

## 3.1 GeneAS CODING

Design or decision variables are coded in either binary strings or directly depending on the type of the variable. A zero-one variable is coded in a one-bit (`0` or `1`) substring. A discrete variable is coded in either a binary string or directly depending on whether the total number of permissible choices for the variable is $2^k$ or not, where $k$ is an integer. A continuous variable is coded directly. This coding scheme allows a natural way to code different variables, as depicted in the following solution representing a complete design of a cantilever beam having four design variables:

$$(\texttt{1}) \quad 15 \quad 23.457 \quad (\texttt{1011})$$

The first variable represents the shape of the cross-section of the beam. There are two options—circular (a `1`) or square (a `0`). Thus, it is a zero-one variable. The second variable represents the diameter of the circular section if the first variable is a `1` or the side of the square if the first variable is a `0`. This variable takes only one of a few pre-specified values. Thus, this variable is a discrete variable coded directly. The third variable represents the length of the cantilever beam, which can take any real value. Thus, it is a continuous variable. The fourth variable is a discrete variable representing the material of the cantilever beam. This material takes one of 16 pre-specified materials. Thus, a four-bit substring is required to code this variable. Although a higher cardinality alphabet (hexadecimal, for example) can also be used, the use of minimal alphabet (binary with only two options) to code a discrete variable is better than an higher cardinality alphabet (Goldberg, 1991). Thus, the above string represents a cantilever beam made of the 12th material from a prescribed list of 16 materials having a circular cross-section with a diameter 15 mm and having a length of 23.457 mm. With the above coding, any combination of cross sectional shape and size, material specifications, and length of the cantilever beam can be represented. This flexibility in the representation of a design solution is not possible to achieve with traditional optimization methods. This flexibility makes GeneAS efficient to be used in many engineering design problems. Moreover, each design variable is allowed to take only permissible values (for example, no hexagonal shape will be tried or no unavailable diameter of a circular cross-section will be chosen).

## 3.2 GeneAS OPERATORS

Three usual operators—reproduction, crossover, and mutation—are used in GeneAS. They are discussed in the following.

### 3.2.1 Reproduction

Since the reproduction operator only depends on the fitness of solutions, a standard proportionate, ranking, or tournament selection can be used. In all runs of this paper, a binary tournament selection with replacement is used. The constraints are handled by using penalty functions having a fixed penalty parameter (Deb, 1995, Michalewicz and Schoenauer, 1996).

### 3.2.2 Crossover

A standard crossover operator cannot be efficiently used with the GeneAS coding. A crossover similar to a single-point or a uniform crossover operator used in binary-coded GAs can be followed in GeneAS coding by allowing the cross-sites to fall only on the variable boundaries. But, such a scheme will not constitute an efficient search (Wright, 1991). Thus, we devise a scheme where a variable (whether discrete or continuous) undergoing crossover can also get changed. In this scheme, two parent solutions are crossed variable-by-variable. If the variable to be crossed is represented by a binary substring, a binary crossover operator is applied. On the other hand, if the variable is coded directly (either continuous or discrete), we use a real-coded crossover. We describe the crossover operation first for a continuous variable and then for a discrete variable coded directly.

Although several real-coded crossover operators have been suggested in the past (Eshelman and Schaffer, 1993; Wright, 1991), recently a simulated binary crossover (SBX) operator has been found to be more efficient (Deb and Agrawal, 1995). The SBX operator creates children solutions based on a probability distribution, which is similar in principle as that observed in the binary crossover operators. Avoiding the details, it will suffice here to note that, on an average, the SBX operator creates solutions similar to what a binary crossover operator would create if variables were coded in sufficiently large strings. In order to use the technique to any pair of parent solutions ($x_i^{(1,t)}$ and $x_i^{(2,t)}$), a non-dimensionalized spread factor, $\beta$, is defined as follows:

$$\beta = \left| \frac{x_i^{(1,t+1)} - x_i^{(2,t+1)}}{x_i^{(1,t)} - x_i^{(2,t)}} \right|. \tag{2}$$

Here, the children solutions are represented as $x_i^{(1,t+1)}$ and $x_i^{(2,t+1)}$. The children solutions are found by using the following polynomial probability distribution:

$$P(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \text{if } 0 \le \beta \le 1; \\ 0.5(n+1)\frac{1}{\beta^{n+2}}, & \text{if } \beta > 1. \end{cases} \tag{3}$$

In order to create a child solution, a random number $u$ between 0 and 1 is first chosen. Thereafter, from the above probability distribution function, the ordinate $\bar{\beta}$ is found so that the area under the probability curve from 0 to $\bar{\beta}$ is equal to the chosen random number $u$. After obtaining $\bar{\beta}$, the children solutions are calculated as follows:

$$x_i^{(1,t+1)} = 0.5\left[(1+\bar{\beta})x_i^{(1,t)} + (1-\bar{\beta})x_i^{(2,t)}\right], \tag{4}$$

$$x_i^{(2,t+1)} = 0.5\left[(1-\bar{\beta})x_i^{(1,t)} + (1+\bar{\beta})x_i^{(2,t)}\right]. \tag{5}$$

As a good binary crossover operator (for example, single-point crossover) preserves common Holland's schemata between parent and children strings, this operator also preserves the common *interval* schemata[1] between parent and children solutions. Note that two children solutions created by this crossover operator are symmetric about the parent solutions. This is used to avoid any bias towards any of the parent solutions. Figure 1 (solid line) shows the above probability distribution with $n = 2$ for creating children solutions from two parent solutions $p_1$ and $p_2$. For example, if $p_1 = 5.0$ and $p_2 = 10.0$ and the chosen random number is $u = 0.25$, then $\bar{\beta} = 0.794$. Using this value of $\bar{\beta}$ and Equations 4 and 5, we calculate the children solutions as $x^{(1,t+1)} = 5.515$ and $x^{(2,t+1)} = 9.485$.

[1]An interval schema $(a, b)$ represents any real value in the interval $(a, b)$ (Eshelman and Schaffer, 1993).

In Equation 3, the distribution index $n$ is any nonnegative real number. A large value of $n$ gives a higher probability for creating near parent values and a small value of $n$ allows distant points to be selected as children solutions. This parameter is similar to the inverse of the *temperature* parameter used in simulated annealing studies. Ideally, this parameter should have a small value initially when a broad search is desired. As the solutions tend to converge, large values of $n$ are desired so that focussed search can be achieved. Since this requires more studies relating how this parameter should change optimally with number of generations, we use a fixed value of $n = 2$ in all runs of this paper.

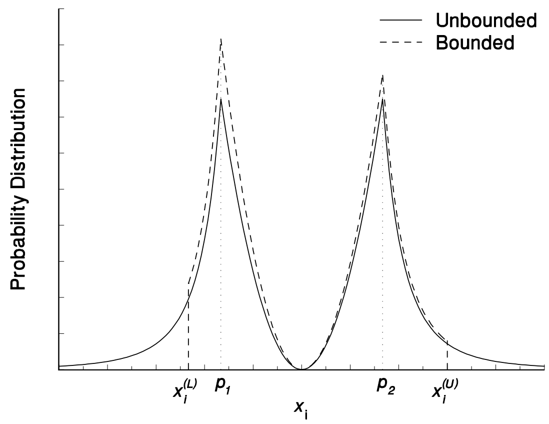


Figure 1: Probability distribution for creating children solutions from two parent solutions $p_1$ and $p_2$ for the variable $x_i$.

We now discuss the procedure for crossing two discrete variables which are coded directly. Binary-coded strings sometimes pose a difficulty for handling discrete variables having any arbitrary search space, because in a binary-coded string of length $k$ the exact number of choices for a variable must be $2^k$. This difficulty can be eliminated by using a discrete version of the SBX operator. A discrete probability distribution function of Equation 3 can be used to create permissible solutions only.

Although the above SBX procedure can create a child solution almost anywhere in the real space, a bounded SBX can also be implemented by restricting solutions within specified limits $(x_i^{(L)}, x_i^{(U)})$. The probability distribution function given in Equation 3 is multiplied by a suitable factor depending on these limits and the location of the parent solutions, so as to make a zero probability of creating any solution outside these limits (dashed line in Figure 1). For the child solution closer to parent $p_1$, this factor can be computed as $1/(1 - v_1)$, where $v_1$ is the cumulative probability of creating solutions from $x_i = -\infty$ to $x_i = x_i^{(L)}$. Similarly, a factor for the child solution closer to $p_2$ can also be calculated.

The above variable-by-variable crossover scheme allows any binary crossover operator to be simulated easily. For example, a single-point crossover can be simulated by using SBX operator on $k$-th variable (chosen at random) and by swapping all $(k+1)$-st to $N$-th variables between the two parent solutions. Here, for simplicity, we use a scheme where, on an average, 50% of the variables get operated by SBX.

### 3.2.3 Mutation

After the reproduction and crossover operators are applied, a mutation operator is used with a small mutation probability $p_m$. If the variable selected for mutation is a binary string, a random bit is changed from `1` to `0` or vice versa. For a continuous variable, the current value of the variable is changed to a neighboring value using a polynomial probability distribution having its mean at the current value and its variance as a function of the distribution index $n$ (Agrawal, 1995). For discrete variables coded directly, a neighboring permissible value is chosen with a discrete probability distribution.

## 4 FOUR ENGINEERING DESIGNS

Three different problems are borrowed from the optimization literature and solved using GeneAS. These problems were solved earlier using an augmented Lagrange (AL) method (Kannan and Kramer, 1993) and the Branch and Bound (BB) method (Sandgren, 1988). In the first method, Powell's zeroth-order method and conjugate gradient method (Reklaitis et al., 1983) are used alternatively as the unconstrained optimizer. The zero-one or discrete variables are handled by adding one additional equality constraint for each variable. According to the authors, the added constraints are nondifferentiable and cause multiple artificial 'bumps' making the search space unnecessary complex and multimodal. In all runs of GeneAS, we use a crossover probability of 0.9 and a mutation probability of 0.0 (this is deliberately set to study the effect of crossover only). In all problems, constraints are normalized and are handled using a bracket-operator penalty term. We use a penalty parameter of $r = 100$ in all problems. A run is terminated after a specific number of function evaluations have been computed.

A welded beam design problem is formulated next. The problem has different types of variables including different weld types and materials. This problem is chosen to show the difficulties of traditional approaches to solve such realistic mixed-variable problems and to show the efficacy of GeneAS to handle such problems.

### 4.1 GEAR TRAIN DESIGN

A compound gear train is to be designed to achieve a specific gear ratio between the driver and driven shafts (Figure 2). The objective of the gear train design is

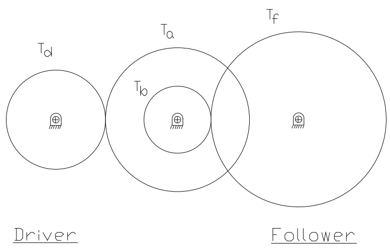


Figure 2: A compound gear train

to find the number of teeth in each of the four gears so as to minimize the error between the obtained gear ratio and a required gear ratio of 1/6.931 (Kannan and Kramer, 1993). All four variables are strictly integers. By denoting the variable vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4) = (T_d, T_b, T_a, T_f)$, we write the NLP problem:

$$\begin{array}{rl} \text{Minimize} & f(\boldsymbol{x}) = \left[\frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4}\right]^2 \\ \text{Subject to} & 12 \leq x_1, x_2, x_3, x_4 \leq 60, \\ & \text{all } x_i\text{'s are integers.} \end{array}$$

The variables are coded in the range (12, 60). We have used two different coding schemes. In Case I, the variables are coded directly allowing only integer values. In Case II, each variable can be coded in six-bit binary strings (having $2^6$ or 64 values), so that variables take values between 12 and 75 (integer values) and four constraints ($60 - x_i \geq 0$, for $i = 1, 2, 3, 4$) are added to penalize infeasible solutions. Table 1 shows the best solutions found in each case with a population of 50 solutions.

Table 1: Optimal gear train design solutions

| DESIGN | OBTAINED SOLUTION | | | |
|---|---|---|---|---|
| VARIABLES | CASE-I | CASE-II | AL | BB |
| $T_d$ | 19 | 17 | 13 | 18 |
| $T_b$ | 16 | 14 | 15 | 22 |
| $T_a$ | 49 | 33 | 33 | 45 |
| $T_f$ | 43 | 50 | 41 | 60 |
| $f$ | $2.7\times(10^{-12})$ | $1.4\times(10^{-09})$ | $2.1\times(10^{-08})$ | $5.7\times(10^{-06})$ |
| Error | 0.001% | 0.026% | 0.11% | 1.65% |

It is clear from the table that both runs with GeneAS have found much better solutions than the previously known solutions. By computing all possible gear teeth combinations ($49^4$ or about 5.76 million), it has been observed that the solution obtained by Case I is, in fact, the globally best solution (Barbosa, 1996).

## 4.2 SPRING DESIGN

A helical compression spring needs to be designed for the minimum volume. Three variables are used: The number of coils $N$, the wire diameter $d$, and the mean coil diameter $D$. Of these variables, $N$ is an integer variable, $d$ is a discrete variable having 42 non-equispaced values[2], and $D$ is a continuous variable. A typical representation of a design solution is shown below:

$$\underbrace{(\texttt{01101})}_{N}\quad \underbrace{0.1480}_{d}\quad \underbrace{1.563}_{D}$$

Denoting the variable vector $\boldsymbol{x} = (x_1, x_2, x_3) = (N, d, D)$, we write the NLP problem:

$$\begin{aligned}
\text{Min} \quad & f(\boldsymbol{x}) = 0.25\pi^2 x_2^2 x_3 (x_1 + 2) \\
\text{with} \quad & g_1(\boldsymbol{x}) = S - \frac{8KP_{\max}x_3}{\pi x_2^3} \geq 0, \\
& g_2(\boldsymbol{x}) = \ell_{\max} - \frac{P_{\max}}{k} - 1.05(x_1 + 2)x_2 \geq 0, \\
& g_3(\boldsymbol{x}) = x_2 - d_{\min} \geq 0, \\
& g_4(\boldsymbol{x}) = D_{\max} - (x_2 + x_3) \geq 0, \\
& g_5(\boldsymbol{x}) = D/d - 3 \geq 0, \\
& g_6(\boldsymbol{x}) = \delta_{pm} - \delta_p \geq 0, \\
& g_7(\boldsymbol{x}) = \ell_f - \frac{P_{\max}}{k} - 1.05(N + 2)d \geq 0, \\
& g_8(\boldsymbol{x}) = \frac{P_{\max} - P}{k} - \delta_w \geq 0, \\
& x_1 \text{ is integer, } x_2 \text{ is discrete, } x_3 \text{ is continuous.}
\end{aligned}$$

The parameters used above are as follows:

| | | |
|---|---|---|
| $K = \frac{4c-1}{4c-4} + \frac{0.615x_2}{x_3}$ | $P = 300\text{lb}$ | $D_{\max} = 3\text{in}$ |
| $k = \frac{Gx_2^4}{8x_1x_3^3}$ | $P_{\max} = 1000\text{lb}$ | $\delta_w = 1.25\text{in}$ |
| $\delta_p = \frac{P}{k}$ | $\ell_{\max} = 14\text{in}$ | $\delta_{pm} = 6\text{in}$ |
| $S = 0.189\text{Mpsi}$ | $d_{\min} = 0.2\text{in}$ | $G = 11.5\text{Mpsi}$ |

The free length $\ell_f$ is so chosen that the constraint $g_7$ becomes an equality constraint. The variable $N$ is coded in five-bit strings so that the feasible values of $N$ are integers between 1 to 32. The variable $d$ is coded as a discrete variables coded directly. The variable $D$ takes any real value. The flexibility of GeneAS allows these three different types of variables to be coded naturally. An initial population of 60 random solutions are created with a 5-bit binary string for $N$, a feasible discrete value for $d$, and a value of $D$ in the range (1.0, 30.0) inch. However, any real value for $D$ is allowed in subsequent iterations. Table 2 once again shows that the solution obtained by GeneAS in this problem has outperformed previously reported optimal solutions. The solution obtained by GeneAS is about 5% better than that reported by Sandgren (1988). When the discrete (integer) variable $N$ is coded directly, a marginally better solution has emerged: $N = 9$, $d = 0.283$, and $D = 1.224$ with $f(\boldsymbol{x}) = 2.661$.

Table 2: Optimal spring design solutions

| DESIGN VARIABLES | OBTAINED SOLUTION GeneAS | AL | BB |
|---|---|---|---|
| $N$ | 9 | 7 | 10 |
| $d$ | 0.283 | 0.283 | 0.283 |
| $D$ | 1.226 | 1.329 | 1.180 |
| $g_1$ | 713.510 | $-10,172.356$ | 5500.080 |
| $g_2$ | 8.933 | 9.543 | 8.652 |
| $g_3$ | 0.083 | 0.083 | 0.083 |
| $g_4$ | 1.491 | 1.390 | 1.537 |
| $g_5$ | 1.337 | 1.700 | 1.170 |
| $g_6$ | 5.461 | 5.400 | 5.465 |
| $g_7$ | 0.000 | 0.000 | 0.000 |
| $g_8$ | 0.009 | 0.003 | 0.003 |
| $f$ | 2.665 | $(*)$ 2.364 | 2.798 |

$(*)$ The solution in Kannan and Kramer (1993) is not feasible, as it violates constraint $g_1$.

## 4.3 HYDROSTATIC THRUST BEARING

The objective of this problem is to minimize the power loss during the operation of a hydrostatic thrust bearing subject to satisfying a number of constraints (Figure 3). A bearing has to withstand a specified load while providing an axial support. Four variables—bearing step radius $R$, recess radius $R_0$, oil viscosity $\mu$, and the flow rate $Q$—are assumed to take only real values. The details of the problem formulation is presented in Siddall (1982). There are seven constraints associated with the minimum load carrying capacity, inlet oil pressure requirements, oil temperature rise, oil film thickness, and some physical constraints. The optimal design formulation in NLP form is given be-

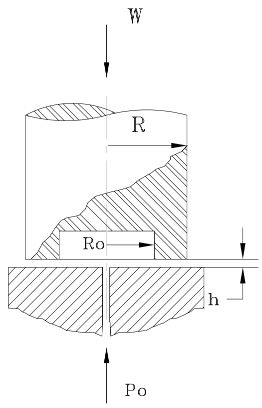


Figure 3: The hydrostatic thrust bearing

[2]Possible wire diameters (in inch) of ASTM A228 material are as follows (Kannan and Kramer, 1993): 0.0090, 0.0095, 0.0104, 0.0118, 0.0128, 0.0132, 0.0140, 0.0150, 0.0162, 0.0173, 0.0180, 0.0200, 0.0230, 0.0250, 0.0280, 0.0320, 0.0350, 0.0410, 0.0470, 0.0540, 0.0630, 0.0720, 0.0800, 0.0920, 0.1050, 0.1200, 0.1350, 0.1480, 0.1620, 0.1770, 0.1920, 0.2070, 0.2250, 0.2440, 0.2630, 0.2830, 0.3070, 0.3310, 0.3620, 0.3940, 0.4375, 0.5000.

low:

$$\begin{aligned}
\text{Minimize} \quad & \frac{QP_0}{0.7} + E_f \\
\text{Subject to} \quad & g_1(\boldsymbol{x}) = \frac{\pi P_0}{2}\frac{R^2-R_0^2}{\ln(R/R_0)} - W_s \geq 0, \\
& g_2(\boldsymbol{x}) = P_{\max} - P_0 \geq 0, \\
& g_3(\boldsymbol{x}) = \Delta T_{\max} - \Delta T \geq 0, \\
& g_4(\boldsymbol{x}) = h - h_{\min} \geq 0, \\
& g_5(\boldsymbol{x}) = R - R_0 \geq 0, \\
& g_6(\boldsymbol{x}) = 0.001 - \frac{\gamma}{gP_0}\left(\frac{Q}{2\pi Rh}\right)^2 \geq 0, \\
& g_7(\boldsymbol{x}) = 5000 - \frac{W}{\pi(R^2-R_0^2)} \geq 0.
\end{aligned}$$

GeneAS is used to solve this problem assuming that all four variables are continuous. A population size of 100 is used. The population is initialized in the following range: $1 \leq R \leq 16$, $1 \leq R_0 \leq 16$, $10^{-6} \leq \mu \leq 16(10^{-6})$, and $1 \leq Q \leq 16$. The obtained solution is shown in Table 3. This solution corresponds to a power loss of 2,161.6 ft-lb/s (3.91 hp). When binary GAs (BGA) are applied with 16 bit substrings (for each variable) mapped in the above ranges, the solution with a power loss of 2,295.1 ft-lb/s (4.15 hp) is obtained. The solution reported in Siddall (1982) obtained using Hooke and Jeeves (HJ) pattern search method corresponds to a power loss of 2,288.0 ft-lb/s (4.14 hp). Moreover, the HJ solution is found to be infeasible with our calculations, since the solution violates the first constraint as shown in Table 3. The

Table 3: Optimal bearing design solutions

| DESIGN | OBTAINED SOLUTION | | |
|---|---|---|---|
| VARIABLES | GeneAS | BGA | HJ |
| $R$ (in) | 6.778 | 7.077 | 7.155 |
| $R_0$ (in) | 6.234 | 6.549 | 6.689 |
| $\mu \times (10^{-6})$ (lb-s/in$^2$) | 6.096 | 6.619 | 8.321 |
| $Q$ (in$^3$/s) | 3.809 | 4.849 | 9.168 |
| $g_1$ | 8299.233 | 1216.140 | −11166.644 |
| $g_2$ | 177.662 | 298.830 | 402.959 |
| $g_3$ | 10.684 | 17.358 | 35.055 |
| $g_4$ | 0.001 | 0.001 | 0.001 |
| $g_5$ | 0.544 | 0.528 | 0.466 |
| $g_6$ | 0.001 | 0.001 | 0.001 |
| $g_7$ | 86.365 | 480.358 | 571.292 |
| Power (ft-lb/s) | 2161.6 | 2295.1 | 2288.0 |
| Wt (lb) | 109,298 | 102,215 | 89,833 |
| Film thk (in) | 0.00165 | 0.00189 | 0.00254 |

table shows that the bearing designed using GeneAS is much better than that obtained by other two techniques and can withstand more weight with a smaller film thickness.

### 4.4 WELDED BEAM DESIGN

A rectangular beam needs to be welded as a cantilever beam to carry a certain load. The objective of the design of the welded beam is to minimize the overall cost of the fabrication. Two different welding configurations are assumed as shown in Figure 4. A binary

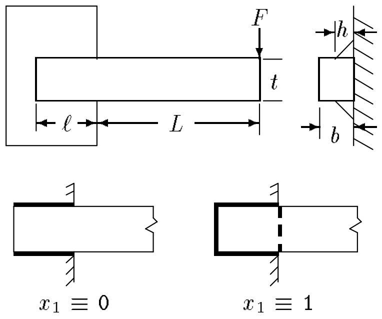


Figure 4: Welded beam problem

variable $x_1$ is assigned to code this choice of configurations (`1` is used for four-sided welding and `0` is used for two-sided welding). One of the four materials (steel, cast iron, aluminum, or brass) is to be used for the beam. A two-bit string is used to code the material as $x_2$ (the substring `00` for steel, `01` for cast iron, `10` for aluminum, and `11` for brass). Thereafter, three discrete variables (with steps of 0.0625) are used to code the thickness of the weld ($x_3 = h$), the width of the beam ($x_4 = t$), and the thickness of the beam ($x_5 = b$). Finally, the length of welded joint ($x_6 = \ell$) is used as a continuous variable. A typical GeneAS solution is shown below:

$$\underbrace{(\mathtt{0})}_{\text{weld type}} \quad \underbrace{(\mathtt{10})}_{\text{material}} \quad \underbrace{0.0625}_{h} \quad \underbrace{0.625}_{t} \quad \underbrace{0.0625}_{b} \quad \underbrace{1.563}_{\ell}$$

The NLP problem of the above welded beam problem is as follows:

$$\begin{aligned}
\text{Min} \quad & f(\boldsymbol{x}) = (1+c_1)x_3^2(x_6 + x_1x_4) + c_2x_4x_5(L + x_6) \\
\text{with} \quad & g_1(\boldsymbol{x}) = S - \sigma(\boldsymbol{x}) \geq 0, \\
& g_2(\boldsymbol{x}) = P_c - F \geq 0, \\
& g_3(\boldsymbol{x}) = \delta_{\max} - \delta(\boldsymbol{x}) \geq 0, \\
& g_4(\boldsymbol{x}) = 0.577S - \tau(\boldsymbol{x}) \geq 0, \\
& x_1 \text{ is a zero-one,} \\
& x_2,\ x_3,\ x_4, \text{ and } x_5 \text{ are discrete variables,} \\
& x_6 \text{ is a continuous variable.}
\end{aligned}$$

The stress, deflection, and buckling terms are borrowed from a similar problem formulated elsewhere (Reklaitis et al., 1983). The parameters $L = 14$ inch, $F = 6,000$ lb, and $\delta_{\max} = 0.25$ inch are used. The welding area and inertia properties for two configurations are as follows:

$$x_1 \equiv \mathtt{0}: \ A = 1.414x_3x_6, \ J = 1.414x_3x_6\left[\frac{(x_3+x_4)^2}{4} + \frac{x_6^2}{12}\right]$$

$$x_1 \equiv \mathtt{1}: \ A = 1.414x_3(x_4+x_6), \ J = 1.414x_3\frac{(x_3+x_4+x_6)^3}{12}.$$

The cost factors and the material properties are so chosen that steel becomes the most cost-effective material for this problem (Table 4). An initial random population of 50 solutions are created in the range

Table 4: Cost term and material properties

| $x_2$ | MAT. | $S$ (kpsi) | $E$ (Mpsi) | $G$ (Mpsi) | $c_1$ | $c_2$ |
|---|---|---|---|---|---|---|
| 00 | Steel | 30 | 30 | 12 | 0.1047 | 0.0481 |
| 01 | CI | 8 | 14 | 6 | 0.0489 | 0.0224 |
| 10 | Al | 5 | 10 | 4 | 0.5235 | 0.2405 |
| 11 | Brass | 8 | 16 | 6 | 0.5584 | 0.2566 |

$0.0625 \leq x_3, x_5 \leq 2$ and $2 \leq x_4, x_6 \leq 20$. However, GeneAS is allowed to create any other solution ($h$, $b$, and $t$ in steps of 0.0625 and any real value of $\ell$) in subsequent iterations. Table 5 shows the best GeneAS solution, which is a steel beam ($0.25 \times 8.25$ inch rectangular cross-section) welded on four sides ($8.25 \times 1.6849$ inch rectangle) of weld size 0.1875 inch. The above solution is close to the true optimal solution, as the constraint $g_4$ is very close to zero, meaning that the material is well utilized to have a shear strength of the weld equal to the allowable limit (In fact, the shear stress is only $9.4(10^{-4})\%$ smaller than the allowable limit). The bending stress is 1.3% smaller than the allowable limit and the critical buckling load is 6.7% higher than the applied load. If a traditional method were used to solve this problem, eight different problems, one for each combination of the weld type and material, would have to be solved independently. Then, the best of the eight solutions would have to be reported. This problem shows how easily yet efficiently GeneAS can handle mixed variables in an engineering design problem.

Table 5: Optimal welded beam design solution

| DESIGN VARIABLES | SOLN. | CONSTR./ OBJ. FUNC. | VALUE |
|---|---|---|---|
| $x_1$ (1), (0-1) | 4-sided | $g_1$ | 380.1660 |
| $x_2$ (00), discr | Steel | $g_2$ | 402.0473 |
| $x_3$ ($h$), discr | 0.1875 | $g_3$ | 0.2346 |
| $x_4$ ($t$), discr | 8.2500 | $g_4$ | 0.1621 |
| $x_5$ ($b$), discr | 0.2500 | $f$ | 1.9422 |
| $x_6$ ($\ell$), cont | 1.6849 | | |

## 5 CONCLUSIONS

Results presented in this paper show that GeneAS is a flexible yet efficient optimization technique in handling mixed variables. In order to handle discrete or integer variables, no additional constraint or no other special consideration is needed. By using a natural coding to represent the variables, GeneAS only create feasible values of the variables. Thus, this method can be used in a wide variety of problem domain. In solving three different mechanical component design problems, GeneAS has always outperformed the previously-known methods. GeneAS has also found a near-optimal solution in a welded-beam design problem having mixed variables. The flexibility and efficiency of the GeneAS technique demonstrated here suggest its immediate application to other engineering design problems. Motivated by this work, we also suggest modifying GeneAS (along the lines of multi-objective binary GA implementations (Srinivas and Deb, 1995)) to find Pareto-optimal solutions in multi-objective engineering design problems.

**Acknowledgements**

This work is funded by the Department of Science and Technology, New Delhi under grant SR/SY/E-06/93.

**References**

Agrawal, R. (1995). Simulated binary crossover for real-coded genetic algorithms (Master's thesis). Department of Mechanical Engineering: Indian Institute of Technology, Kanpur, India.

Barbosa, H. (September, 1996). Personal communication.

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

Deb, K. and Agrawal, R. (1995): Simulated binary crossover for continuous search space. *Complex Systems, 9*(2), 115–148.

Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval schemata. In D. Whitley (Ed.), *Foundations of Genetic Algorithms, II* (pp. 187–202).

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems, 5*(2), 139–168.

Kannan, B. K. and Kramer, S. N. (1995). An augmented Lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design. *Journal of Mechanical Design, 116*, 405–411.

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

Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization—Methods and Applications.* New York: Wiley.

Sandgren, E. (1988). Nonlinear integer and discrete programming in mechanical design. *Proceedings of the ASME Design Technology Conference*, Kissimee, FL, 95–105.

Siddall, J. N. (1982). *Optimal engineering design.* New York: Marcel Dekker.

Srinivas, N. and Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms, *Evolutionary Computation, 2*(3), 221–248.

Wright, A. (1991). Genetic algorithms for real parameter optimization. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms.* (pp.205–220).