Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

GENETIC ALGORITHMS IN OPTIMIZATION PROBLEMS WITH DISCRETE AND INTEGER DESIGN VARIABLES

C.-Y. LIN [a] & P. HAJELA [a]
[a] Department of Mechanical Engineering, Aeronautical Engineering & Mechanics, ensselaer Polytechnic Institute, Troy, New York, 12180, U.S.A.
Published online: 27 Apr 2007.

To cite this article: C.-Y. LIN & P. HAJELA (1992) GENETIC ALGORITHMS IN OPTIMIZATION PROBLEMS WITH DISCRETE AND INTEGER DESIGN VARIABLES, Engineering Optimization, 19:4, 309-327, DOI: 10.1080/03052159208941234

To link to this article: http://dx.doi.org/10.1080/03052159208941234

Eng. Opt., 1992, Vol. 19, pp. 309–327
Reprints available directly from the publisher
Photocopying permitted by license only

GENETIC ALGORITHMS IN OPTIMIZATION PROBLEMS WITH DISCRETE AND INTEGER DESIGN VARIABLES

C.-Y. LIN and P. HAJELA

Department of Mechanical Engineering, Aeronautical Engineering & Mechanics, Rensselaer Polytechnic Institute, Troy, New York 12180, U.S.A.

(Received 28 February 1991; in final form 2 October 1991)

The paper describes an implementation of genetic search methods in the optimal design of structural systems with a mix of continuous, integer and discrete design variables. Design variable representation schemes for such mixed variables are proposed and the performance of each is evaluated in the context of structural design problems. The approach is proposed as an alternative to the branch-and-bound techniques that are used in conjunction with nonlinear programming methods. The methodology is inherently equipped with a better chance of locating the global optimum than the conventional gradient based methods.

KEY WORDS: Structural optimization, genetic algorithms, integer variables

INTRODUCTION

Mathematical nonlinear programming algorithms have emerged as the method of choice for applications in engineering optimization problems. They provide a general approach for obtaining solutions to both single and multi-objective design problems with a mix of equality and inequality constraints. The more efficient of this class of methods are generally gradient based, and require at least the first-order derivatives of both objective and constraint functions with respect to the design variables. With this "slope-tracking" ability, gradient-based methods can easily identify a relative optimum closest to the initial guess of the optimum design. There is no guarantee of locating the global optimum if the design space is known to be nonconvex[1]. These methods are also inadequate in problems where the design space is discontinuous, as the derivatives of both the objective function and constraints may become singular across the boundary of discontinuity.

In engineering design problems, the mix of continuous, discrete, and integer design variables has been approached by treating all variables as continuous, and then rounding specific variables either up or down to the nearest integer or discrete variable. This simple rounding procedure often fails completely, resulting in either a suboptimal design, or in some cases, even generating an infeasible design[2]. The branch-and-bound strategy[3–5] has been proposed as a systematic solution to this class of problems. This approach is based on an enumeration of all feasible discrete and integer solutions to identify the optimum. A modified mixed integer and discrete programming algorithm using the branch-and-bound technique was proposed in Ref. [6]. This strategy consisted of a systematic search of continuous solutions in which the discrete and integer variables were successively forced to assume specific values. However, in doing so, the original optimization problem was undesirably expanded to a large number of sub-optimization problems.

Exhaustive search and random search methods are among the simplest and most robust strategies for automated optimum design problems. These methods can work on almost all kinds of design spaces and without any restriction on types of design variables. An improvement on the simple enumerative techniques are methods such as random walk and random walk with direction exploitation. The only drawback is that these methods often require thousands of function evaluations to achieve the optimum, even for the simplest of problems. It is hence crucial to examine alternate strategies for structural optimal design problems, which need less computational effort than is required by the enumerative search techniques, and are also not susceptible to convergence to a local optimum as exhibited by gradient-based nonlinear programming algorithms. Genetic algoriths (GAs) as proposed by Holland[7] have the potential to successfully fill this gap.

Genetic algorithms belong to a category of stochastic search techniques, where only the most promising regions of the design space are enumerated to locate the optimal design. These algorithms have their philosophical basis in Darwin's theory of survival of the fittest. Analogous to the natural process where a population of a given species adapts to a natural habitat, a population of designs is created and is then allowed to adapt to the design requirements. Designs that do not adapt in a favorable manner to the requirements are eliminated from consideration. The mechanism of adaptation borrows extensively from principles of biological evolution, in that basic characteristics of designs in one population are transferred to a population in another generation through gene transfer operators. Stated differently, design alternatives representing a population in a given generation are allowed to reproduce and cross among themselves with bias allocated to the most fit members of the population. Combination of the most favorable characteristics of the mating members of the population results in a progeny population that is more fit than the parent population. If the measure which indicates the fitness of a generation is also the desired goal of a design process, successive generations produce better values of the objective function.

The mechanics of genetic search, though simple to implement, encompass features that render the approach highly applicable to the problem of search in a non-convex/disjoint design space with a mix of continuous, discrete, and integer design variables. These desirable characteristics are largely attributed to the fact that genetic search moves from a population of designs to another population of designs; this is in contrast to the point to point search available in traditional mathematical programming methods, and therefore offers a better possibility of locating a global optimum. Furthermore, genetic algorithms work on a coding of the design variables rather than the variables themselves. This allows for an efficient treatment of integer and discrete variables. The terminology of genetic search and its principal components can be found in Refs. [8-9], and are summarized next.

GENETIC ALGORITHMS

The basic approach in genetic algorithms is to represent possible solutions to a given problem by a population of bit strings of finite lengths, and to subsequently use transformations analogous to biological reproduction and evolution to improve and vary the coded solutions. This approach is facilitated by defining a fitness function or a measure indicating the "goodness" of a member of the population in a given generation during the evolution process. For unconstrained maximization problems, the objective function could serve as the fitness function. The inverse of the objective function, or the difference between a large number and the objective function can be used as the fitness function in a minimization problem. For constrained optimal design problems, an exterior penalty function formulation can be adapted to transform a constrained optimization problem into an unconstrained one.

In biological systems, generational memory is preserved and transferred to progeny populations in the form of chromosomes. A number of chromosome strings comprises a genotype, the total genetic makeup for an organism. In order to use genetic algorithms, an artificial chromosome-like string representing an n-dimensional design has to include all information of these n design variables. One simple yet effective approach to accomplish this is to represent each design variable by a finite length binary string and then connect, head-to-tail all n binary strings into a single binary string. Various simulations of genetic evolution and adaptation are conceivable. Three principal components of the gene-transformation mechanism in this artificial evolution and adaptation simulation are reproduction, crossover, and mutation. These transformations are best described with reference to a specific optimization problem.

Consider the following function minimization problem:

$$\text{Minimize} \quad F(x_1, x_2) = x_1^3 + x_1 x_2 + x_2^2 \tag{1}$$

$$\text{Subject to:} \quad g_1 \equiv x_2^2 + x_1^2 x_2 \le 0 \tag{2}$$

$$g_2 \equiv x_1^3 - x_1 x_2 \le 0 \tag{3}$$

$$x_{i_{\min}} \le x_i \le x_{i_{\max}} \tag{4}$$

To use genetic search, the constrained minimization problem is first converted into an unconstrained problem using the exterior penalty function formulation, resulting in the following problem:

$$\text{Minimize} \quad F^* = F + \bar{P} \tag{5}$$

where $\bar{P}$ is the penalty term. Careful consideration must be given to this selection, and in the present work, the following bounding strategy was adopted. If the average fitness of feasible designs is F_{av}, then a limiter value of the penalty $\bar{L}$ is selected as, selected as,

$$\bar{L} = kF_{av} \tag{6}$$

where k is of order 2; the penalty $\bar{P}$ that is appended to an infeasible design is then obtained as follows:

$$\bar{P} = \begin{cases} G, & \text{if } G \leq \bar{L} \\ \bar{L} + \alpha(G - \bar{L}), & \text{if } G > \bar{L} \end{cases} \tag{7}$$

where

$$G = r \sum_{j=1}^{2} \langle g_j \rangle^2$$

Here r is a penalty parameter of the form encountered in the exterior penalty function approach, and $\langle g_j \rangle$ represents the violated constraints. The effect of this scaling is to prevent radical departures in the value of the penalty term from the specified $\bar{L}$ value. If $\alpha = 0.0$, penalty for all violated designs is $\bar{L}$. If instead, α is assigned a small value of the order 0.1, then the extent of constraint violation due to severely violated and less violated designs varies linearly from $\bar{L}$, albeit with a small slope. To convert this function minimization into a fitness maximization as required by GAs, the following fitness function is created,

$$f_i = F^*_{\max} - F^* \tag{8}$$

where f_i is the fitness of the ith design, and $F^*_{\max}$ is the maximum value of F^*. To obtain a bit string solution to this problem, each x_i can be converted into a binary string of 0's and 1's. For purposes of discussion, we choose a 5 digit binary number, with $x_{i_{\max}}$ and $x_{i_{\max}}$ represented by the following binary numbers.

$$x_{i_{\min}} = 00000$$

$$x_{i_{\max}} = 11111$$

A linear scaling can be introduced to convert intermediate values of the binary number into design variable values. The binary string representations for x_1 and x_2 can be placed head-to-tail to create a 10-digit number, also referred to as a schema, which represents a solution to the problem. Several such 10-digit chromosomal strings are defined to constitute a population of designs, which includes a mix of feasible and infeasible designs. The fitness f_i corresponding to each member of the population is computed before invoking the genetic transformations.

Reproduction

The simulation of genetic evolution here is contrived in that the population size is forced to remain unchanged during the evolution process; two mating parents create only two progenies but are themselves eliminated. The reproduction process is nothing more than a selection of the more fit members of the population into the mating pool, from which members are selected for crossover and mutation transformations. One approach to selecting members from the initialized population is to assign each member a probability of selection $f_i/\sum_m f_i$, where m is the total population size. A mating pool can then be created, of the same size as the initial population, but with a higher average fitness function value.

Crossover

While reproduction represents an elitist selection which retains only the most fit members of a population for mating, it does not in any way improve any single design in the population. It is the crossover transform that allows the characteristics of the designs in the population pool to be altered, with the intent of representing the best characteristics in the next generation. This is similar to transfer of genetic material in biological processes facilitated by DNA and RNA strings. Crossover is executed by selecting two mating parents, randomly choosing two sites on the genetic strings, and swapping strings of 0's and 1's between two chosen sites among the mating pair. An illustration of the crossover process between mating parents represneted by 10-digit binary strings (as in the example above) is as follows:

$$\text{Parent1} = 110\underline{0}1\underline{0}0100$$

$$\text{Parent2} = 010\underline{1}1\underline{1}0001$$

$$\text{Child1} = 1101110100$$

$$\text{Child2} = 0100100001$$

The crossover sites on the parent strings are indicated by an understrike. A probability of crossover p_c is defined to determine if crossover should be implemented.

Mutation

Mutation safeguards the genetic search process from a premature loss of valuable genetic material due to reproduction and crossover. The process of mutation is simply to choose a few members from the population pool according to the probability of mutation p_m, and to switch a 0 to 1 or vice versa at a randomly selected mutation site on each chosen string.

It is important to distinguish GAs from a variant of the random walk approach. Genetic algorithms simply use probabilistic transition rules to guide a highly exploitative search process. The extent of this exploitative search can be gauged by examining how the structure of a "chromosomal" string is changed by the genetic transformations. If an n-digit binary string is used to represent a design, than a total of 2^n variations of the design are available through the representation scheme. Furthermore, in such a scheme, not all digits have to be specifically a 0 or 1 for the string to convey meaningful information about the design. As an example, for our two variable problem, a schema 1001000010 indicates a value of x_1 close to its maximum value and x_2, closer to its minimum value. If we chose a '*' to indicate that a particular location can be either a 0 or a 1, then a schema 1***0000** conveys the same information. Note that the order of the first schema is 10, i.e., the number of specific 0's or 1's in the schema; the order of the second schema is only 5. In effect then, for an n-digit string consisting of 0, 1, and *, there are 3^n schemata that are present in 2^n unique designs. Another parameter of interest is the defining length of a schema, which is the distance between the first and last specific digits on the schema. The defining length for the first schema is 9 and that of the second is 7. This parameter is important when one considers the potential of disruption of a schema during corssover; shorter defining lengths are less likely to be disrupted.

If we denote the number of schema of type H at a given generation t by $m(H, t)$, then, due to reproduction alone, at generation $t + 1$ this number would be[10]

$$m(H, t + 1) = m(H, t)f(H)/f_{av} \tag{9}$$

where f_{av} is the average fitness of the population. Clearly, schemata with fitness higher than the average fitness of the population are increased exponentially in successive generations. Even with the disruptive influence of mutation and crossover, shorter defining length, low order schema that are also fit, increase rapidly in successive generations of the genetic evolution. Furthermore, as estimated by Holland[7], the analysis of n designs in the population is really equivalent to an evaluation of order $O(n^3)$ schemas, giving the approach a tremendous computational advantage.

The preceding sections have introduced the mechanics of genetic search and the computational power derived from an implicit parallel processing capability present in the approach. The discussion also establishes how genetic algorithms work on a coding of the design variables rather than the variables themselves, and the importance of representation or coding schemes to the overall performance of the process. While some effort has been spent in description of GA applications in structural optimization involving continuous design variables, there is limited discussion on the use of this approach in problems with a mix of continuous, discrete, and integer design variables. This is discussed in the context of engineering design problems in subsequent sections of this paper.

REPRESENTATION SCHEMES FOR DESIGN SPACE

A. Continuous design variables

An m-digit binary number representation of a continuous design variable allows for 2^m distinct variations of that design variable to be considered. If the description of a design variable is required to a precision of A_c, then the number of digits in the binary string may be estimated from the following relationship:

$$2^m \geq [(x_U - x_L)/A_c + 1] \tag{10}$$

Here, x_L and x_U are the lower and upper bounds of a continuous variable x. As an example, if one considers representing a variable x to within a precision 0.1, and the lower and upper bounds on x are 0.0 and 0.7, then $2^m \geq 8$. This gives the result that $m = 3$, and indeed the eight 3-digit combinations of 0 and 1 can be assigned to numbers $0.0, 0.1, \ldots, 0.7$. Note than an integer value of m will not yield a precision of 0.14, and a 3-digit string would still be necessary to represent the six numbers $0.0, 0.14, 0.28, \ldots, 0.7$. There would be two excessive binary representations which would be digested by distributing them to partial discrete design space as described in later sections. It is important to recognize that even when dealing with continuous variables, GAs work on a discrete representative set of those variables, *the method is therefore ideally suited for applications to problems with a mix of continuous, integer, and discrete variables.*

B. Integer design variables

Due to the discrete nature of the binary representation schemes, integer design variables can be simply regarded as continuous design variables with a fixed accuracy A_c equal to 1. If N could be found to meet $(x_U - x_L) = 2^N - 1$, a one-to-one correspondence could be readily established. In most cases, however, this is not possible, and the excessive binary strings must be assigned in an appropriate manner. There are a number of ways in which this is done, and these are described as follows.

i) *Penalty approach:* In this approach, the smallest number m which meets the inequality

$$2^m > (x_U - x_L) + 1 \tag{11}$$

is computed. Of the 2^m possible m-digit binary strings, a unique string is assigned to each of the n integer variables. The remaining $(2^m - n)$ strings are assigned to out-of-bound integers. As an example, in representing six integers between 0 and 5, the computed value of m satisfying the above inequality is 3. The remaining two binary strings are assigned to out-of-bound variables as follows,

$$[0, 1, 2, 3, 4, 5, 6^*, 7^*]$$

$$[000, 001, 010, 011, 100, 101, 110^*, 111^*]$$

where, '*' indicates an out-of-bound variable. A penalty measure is then allocated to the fitness function of a design which includes the out-of-bound value of an integer variable. While this approach yields a one-to-one correspondence between the integer variables and their binary representations, careful consideration must be given to the magnitude of penalty assigned to the fitness function due to the presence of an out-of-bound variable. Large penalties on the fitness will adversely affect the genetic search, making it difficult to distinguish between good and average design.

ii) *Excessive-Distribution Method:* In this approach, m is first computed on the basis of Eq. (11). The excessive binary representations are then assigned to integers in the admissible range, whereby one or more integers may have more than one binary representation. For the case described above, one of the two binary strings could be assigned to integer 4 and the other to integer 5. Although this method does not create any additional design constraints, it does have the effect of an uneven expansion in the design space. The affect of this partial expansion is hard to predict without *a priori* information of the objective function space. Clearly, the expansion results in a larger number of schemata in the design space, and if the schemata belong to the poorer regions of the design space, the convergence of genetic search would be adversely influenced. One method of avoiding this problem is to distribute the excess binary strings evenly along the extent of the feasible integer space. As an example, if twenty excessive binary strings are to be distributed evenly among one hundred integers, one excess binary representation can be assigned to every fifth consecutive integer.

The uneven distribution described above may be avoided by creating excess strings on purpose to obtain a similar number of binary representations for each integer. For the example described above, two integers (33%) have twice the number of binary representations (100% more) than the other four (67%) of the integers. If $m = 4$, we would have a total of 16 binary strings to be assigned to six integers. In this case, each integer could be assigned 2 strings, and four integers (67%) could be assigned one additional string (50% more) than the remaining two (33%) integers. With increasing value of m, the disparity in distribution can be removed at the price of increasing the number of schemata to be explored.

C. *Discrete design variables*

Discrete type design variables are characterized by an uneven spacing between two consecutive values. With the unique nature of genetic search wherein gradient information is not required, these variables can be handled in the same manner as continuous or integer variables. Mapping of these variables is a two-stage process. In the first stage, discrete variables are mapped to an equivalent number of integer variables. Then, techniques of mapping integer variables into binary strings described in the preceding sections are applicable with no additional manipulation. An illustration of this process is the mapping of a set of eight discrete variables into binary strings.

$$[2.4,\ 3.76,\ 5.96,\ 8.25,\ 9.37,\ 13.70,\ 20.55,\ 24.0]$$

$$[1\quad 2\quad 3\quad 4\quad 5\quad 6\quad 7\quad 8]$$

$$[000\ 001\ 010\ 011\ 100\ 101\ 110\ 111]$$

A design space with a mix of continuous, discrete, and integer variables can be represented as required in genetic search by connecting, head-to-tail, binary string equivalents of these variables as described above.

In genetic search, the average fitness of a population is increased over generations of evolution. In an unconstrained maximization, the objective function can be chosen as the fitness function. If an unconstrained minimum is the objective, the fitness must be revised to be the inverse of the objective function. As stated earlier, another alternative is to choose the maximum objective function value of all members of the population, and subtract individual objective function values from this quantity to obtain the corresponding fitness values. In constrained optimization, designs with violated constraints are considered less fit. Hence in constrained minimization, a penalty corresponding to the constraint violations is appended to the objective function much as in the exterior penalty function approach. In the present work, the traditional penalty function approach is compared to a revised formulation discussed earlier, where the latter is characterized by the imposition of an upper bound on the penalty. Without such a bound, a highly penalized infeasible design and a good feasible design would be difficult to separate, and the genetic search would deteriorate into a random search.

D. Performance Evaluation of Representation Schemes

DeJong[11] defines online and offline performance measures to gauge the efficiency of a genetic search. While online performance is an average of all evaluations of fitness and is therefore indicative of how well the entire population adapts, the offline parameter is an average of the current best evaluations of fitness. These two parameters, in addition to the average fitness of a population at each generation of evolution, were used to measure the performance of genetic search with the different representation schemes described in preceding sections.

A test problem for this purpose was chosen as follows:

$$\text{Minimize} \quad F_1(x) = x_1^2 + x_2^2 + x_3^2$$

where x_1, x_2, and x_3 were all integer variables ranging between -512 and 511. A one-to-one correspondence can be obtained by using a 10-digit binary string (2^{10} distinct representations). String lengths of 12, 14, 16, 18 and 20 were also used, yielding 4, 16, 64, 256 and 1024 binary representations per integer variable, respectively. A fixed population size of 50 was used in conjunction with probabilities of crossover and mutation set to 0.6 and 0.005, respectively. Both online and offline performance of each of these cases was obtained. The results show improvement in both performance measures with an increase in the number of representations for each admissible value of the integer variables (20-digit string performed the best). This was attributed to two possible factors:

a) More schemata are available in such larger string representations, thereby increasing the probability of having a better initial population.

b) Populations in which there is exactly one representation for each admissible value of the integer variable run the risk of losing that variable very easily during genetic transformations. Note that each 0 or 1 along the string very specifically denotes an integer variable, and is susceptible to elimination by even a simple mutation transform. These advantages must be weighed against the slower improvement in the population, a direct consequence of schema disruption with longer binary strings.

A modification of this problem was a change in the design variable upper bound to 238. For a string length of 10, there were excess binary representations that were treated by the methods discussed earlier. In the penalty approach, a penalty ($O(10^6)$) was imposed on any design containing design variable values between 239 and 511. This choice of penalty parameter is not advised; high penalties on the objective function resulting from the design variable range violations will slow the convergence of the genetic search. The magnitude of the penalty for the design variable range violation should be appropriately selected in accordance to the average magnitude of fitness functions. An arbitrary assignment of penalty parameters should be avoided. Another genetic search with an initial penalty ($O(10^4)$), and which was linearly decreased in every generation, was also used with more satisfactory results. In the excess distribution method, two schemes were implemented. In the first, the 273 excess representations were distributed evenly over the range -512 to 238; in the second, each of 273 integers between -34 and 238 was assigned one additional binary representation. Four other experiments were conducted, with string lengths of 11, 12, 13 and 14, and all excessive representations were distributed evenly for the entire range of integer variables. It can be shown that a string length of 13 yields the most even distribution with 91% of the integers assigned only 10% more representations than the remaining 9%.

Some general conclusions emerged from this numerical experimentation. Although conventional wisdom in genetic search advocates the use of smaller length strings, when working in a discrete space, it is advantageous to have more than a one-to-one correspondence between the number of binary representations and the number of admissible discrete/integer variations. If a one-to-one correspondence is chosen with only a few excess representations, then the design variable representation can be based on either the penalty concept or the excessive distribution approach. The penalty approach is highly sensitive to the choice of the penalty parameter, and is generally not recommended unless information about the magnitudes of the objective function is available. Of the excessive distribution method, the even distribution schemes perform better than the one-sided distribution methods. This conclusion would not be valid only if excessive distribution is done in those regions of the design space where the optimal point is located. When an excessive distribution method is used for a design variable, the design space corresponding to this variable will be partially expanded. With larger length binary numbers, the excessive strings can be more evenly distributed, and result in an even expansion over all variables. This approach also makes available a larger number of schemata for each design variable.

ILLUSTRATIVE EXAMPLES

Example 1

Solutions to constrained engineering optimization problems with a mix of integer, discrete, and continuous design variables were obtained using the GA approach. The first problem involved the design of a lap joint between two steel plates, in which the size, number, and arrangement of rivet pattern were considered as design variables. A typical configuration of the plates and the rivets can be seen in Figure 1. The number of rows parallel to side AB is represented by an integer variable x_1 with permissible values between 1 and 32. The number of the rivets in each row is also an integer variable x_2 allowed to assume values between 0 and 128. The diameter x_3 of all rivets is assumed to be the same, and is chosen from a commercially available set:

$$x_3(\text{mm}) = [6,8,10,12,14,16,18,20,22,24,27,30,33,36,40,45]$$

This variable is therefore of the discrete type. The objective of this optimization is to maximize the efficiency of the joint, defined as the ratio of the strength of the joint to the strength of the plate. The strength of the joint is obtained as the minimum of the shearing failure strength P_s, tension failure strength P_t, and compression or

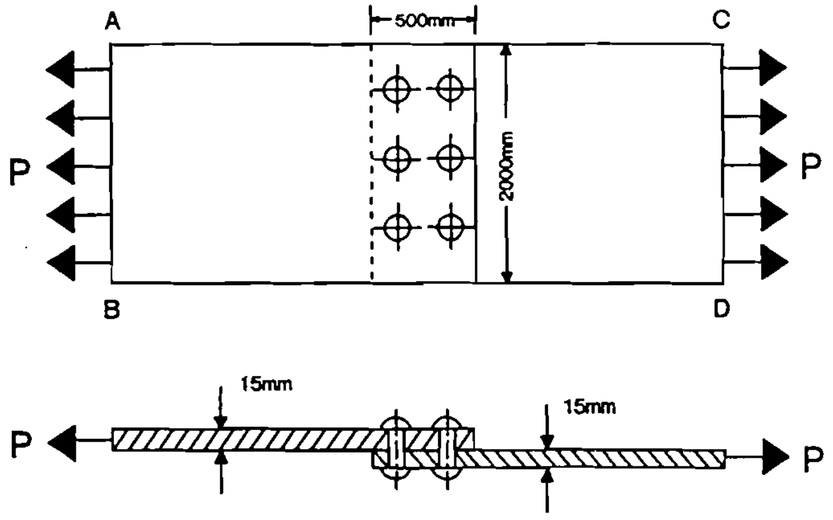

Figure 1 Geometry of riveted tap joint.

bearing failure P_b. These can be formulated as follows[12,13]:

$$\begin{aligned} P_s &= \pi x_3^2 x_1 x_2 \tau/4 \qquad \text{if} \quad x_1 < 3 \\ &= \pi x_3^2 x_1 x_2 \tau/[4(1.06 + 0.126(x_1 - 3))] \qquad \text{if} \quad x_1 \geq 3 \end{aligned} \tag{13}$$

$$P_t = (2000 - x_2 x_3) t\sigma_t \tag{14}$$

$$\begin{aligned} P_b &= t x_3 \sigma_b x_1 x_2 \qquad \text{if} \quad x_1 < 3 \\ &= t x_3 \sigma_b x_1 x_2/[(1.06 + 0.26(x_1 - 3))] \qquad \text{if} \quad x_1 \geq 3 \end{aligned} \tag{15}$$

where,

$$\tau = 80\text{MPa} \qquad \sigma_t = 90\text{MPa} \qquad \sigma_b = 120\text{MPa}$$

Stress concentrations due to the close placement of any two rivets are avoided by the imposition of the following constrains:

$$\begin{aligned} 3x_3 x_1 + 2x_3 &\leq 500 \\ 3x_3 x_2 + 2x_3 &\leq 2000 \end{aligned} \tag{16}$$

Further, the strength of the plate was specified as 2700 kN in the present analysis. For the genetic search, population sizes of 30 and 60 were considered, with two

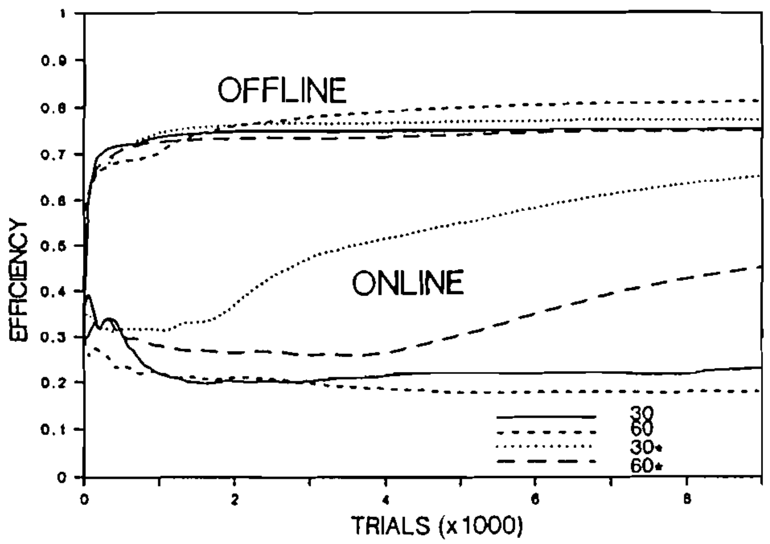

Figure 2 Iteration history of lap joint efficiency—online and offline measures. (* indicates use of bounded penalty).

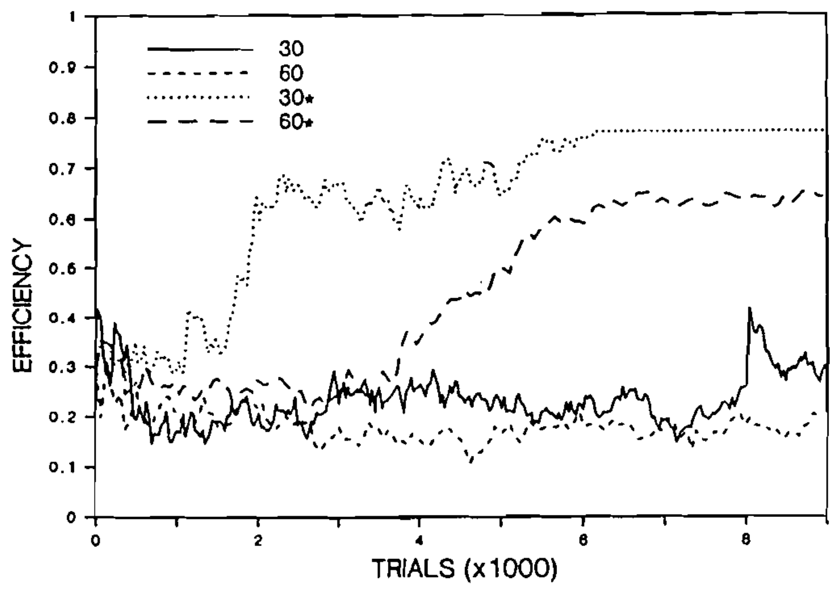

Figure 3 Iteration history of lap joint efficiency—average value. (* indicates use of bounded penalty).

distinct penalty function formulations. Design variables x_1, x_2, and x_3 were represented as binary strings of 5, 6 and 4, digits, respectively. Results of the genetic search simulation using a maximum bound on the constraint penalty are shown in Figures 2 and 3; they are marked with an asterisk to distinguish them from those obtained with an unbounded penalty formulation. A population size of 30, when used with a bounded constraint penalty, performed most favorably. An optimum design of $x_1^* = 5$, $x_2^* = 13$, and $x_3^* = 27$ mm was obtained at the 58th generation, and resulted in a joint efficiency of 82.45%. The average efficiency of the joint for both population sizes, and using no bounds on the constraint penalty, actually decreased over generations of evolution. This underscores the need for a careful selection of penalty functions in the presence of constraints.

Example 2

A second example deals with the sizing and selection of the number of Belleville springs to be stacked in series to maximize the load carrying capacity for a minimum deformation. The configurations for a series-stacked Belleville spring is shown in Figure 4. The compressive stress at the convex inner-diameter corner would be the critical consideration in such a statically loaded situation. The load deflection relation is obtained as follows[14,15],

$$P = \frac{Ef}{(1 - v^2)Ma^2} [(h - f)(h - 0.5f)t + t^3] \qquad (17)$$

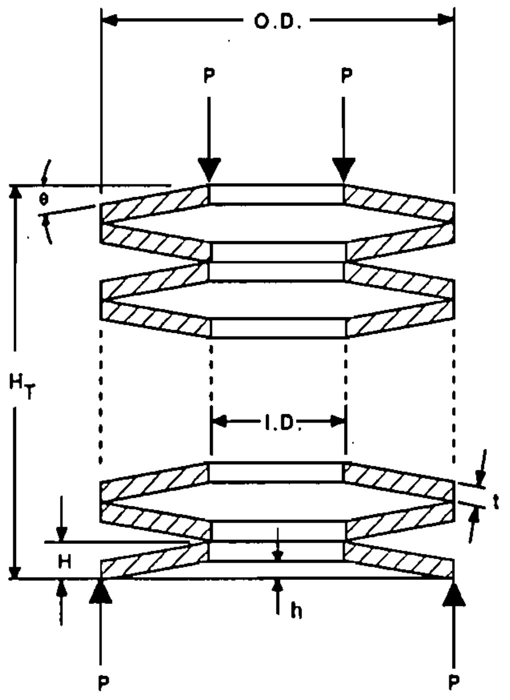

Figure 4 Series stacked Belleville springs.

where d is the deflection under load P. The expression for maximum stress is obtained as,

$$S_c = \frac{Ef}{(1 - v^2)Ma^2}[C_1(h - 0.5f) + C_2 t] \tag{18}$$

where,

$$C_1 = \frac{6}{\pi lnR}\left[\frac{R - 1}{lnR} - 1\right] \tag{19}$$

$$C_2 = \frac{6}{\pi lnR}\left[\frac{R - 1}{2}\right] \tag{20}$$

$$M = \frac{6}{\pi lnR}\left[\frac{(R - 1)^2}{R^2}\right] \tag{21}$$

and, $a = 0.5$ O.D.

$R =$ O.D./I.D.

$f =$ vertical deflection of the spring

$H = h + t \cos\theta$, the height of a single spring

$N =$ number of springs in stack

$H_T = NH$, total height of the stacked springs

In the above equations, a value of Poisson's ratio of $\nu = 0.3$ and Young's modulus $E = 2.06 \times 10^6$ N/mm^2 were used. It is important to note that the load-deflection relations are only valid for $d < 0.85$ h. Consequently, the objective of the optimization is to find the maximum load at $d = 0.8$ h, subject to other constraints. When an h/t ratio greater than 1.3 is used in a stack, the load deflection relation tends to be erratic as some springs snap through the flat position. To avoid this problem, a constraint of $h/t \leq 1.3$ was imposed. The maximum allowable working stress was set to 1400 MPA and a restriction on R to be at least 1.5 was also included.

The design variables $x_1, x_2, \ldots, x_5$ are the thickness t, outer diameter, inner diameter, number of springs N, and disk cone angle θ. The first three were treated as discrete variables selected from the following sets.

O.D.: [200, 201, 202, ..., 263] mm

I.D.: [112, 113, 114, ..., 175] mm

t: [4.8, 5.5, 6.05, 6.5, 7.1, 7.49, 9.4, 11.25] mm

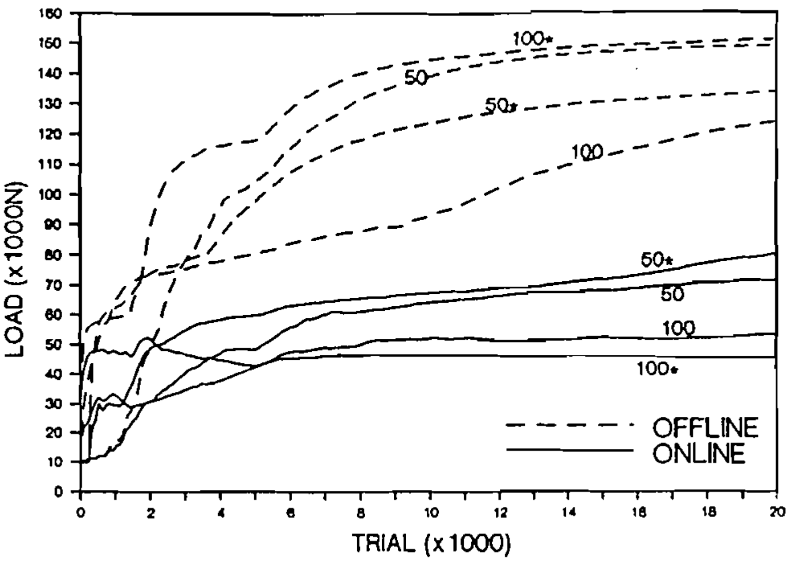

Figure 5 Iteration history of maximum spring loading—online and offline measures (* indicates use of bounded penalty).

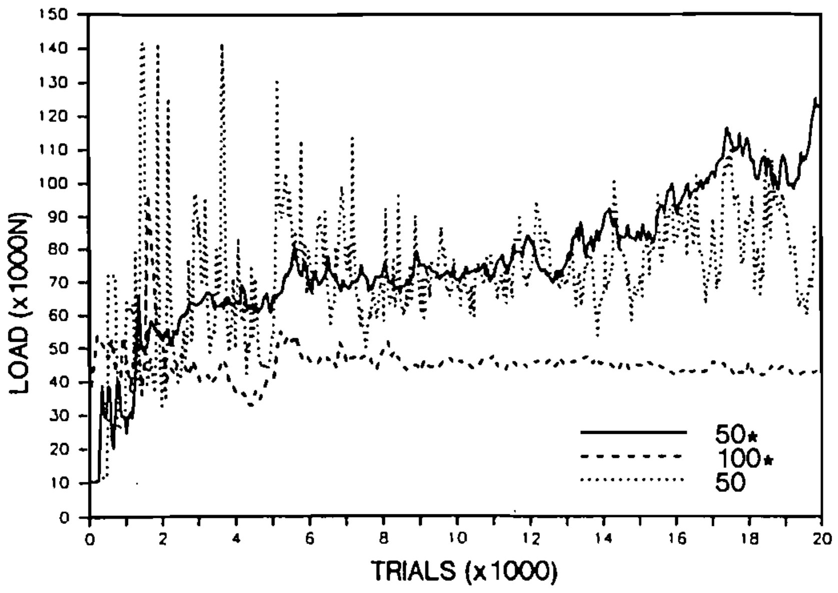

Figure 6 Iteration history of maximum spring loading—average value (* indicates use of bounded penalty).

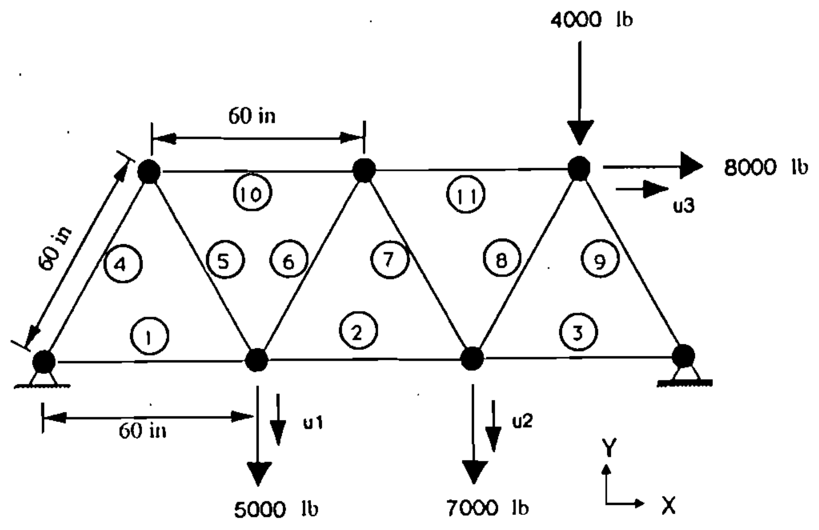

Figure 7 Eleven bar planar truss (design variables $x_1 - x_{11}$ are indicated by circled numbers).

The number of springs was an integer variable ranging from 1 to 16; the cone angle θ was a continuous variable bounded between 1 deg and 20 deg. It was represented to an accuracy of 0.1 in the binary scheme. Population sizes of 50 and 100 were considered in the numerical simulation, with constraints treated both by a general penalty and a bounded penalty formulation. Binary strings of lengths 3, 6, 6, 4 and 10 were chosen for design variables x_1 through x_5. The results obtained from the simulation are shown in Figures 5 and 6, where, as before, results corresponding to the bounded penalty formulation are denoted with an asterisk. This bounded penalty formulation yielded the most stable genetic search when working with a population size of 50. At the 317th generation, an optimal design

$$x^* = [9.4\text{ mm}, 232\text{ mm}, 130\text{ mm}, 7.8.1\text{ deg}]$$

was obtained, corresponding to a maximum load of 142,211 lbs.

Example 3

A minimum weight truss design with constraints on displacements comprised the third test problem. The structure and the loading are shown in Figure 7. The cross sectional areas of each of the eleven members of the truss are discrete variables chosen from the following available sections:

$$A_i = [0.111, 0.141, 0.196, 0.250, 0.307, 0.391, 0.442, 0.563, 0.602, 0.766, 0.785, 0.994, 1.000, 1.228, 1.266, 1.563]\text{ in}^2.$$

Material properties of steel with specific weight $\rho = 0.283$ lb/in^3 and Young's modulus $E = 30 \times 10^6$ psi were prescribed for this application. A binary string length of four was used to represent the design variables; this results in an exactly one binary string for each variable. The displacement constraints were defined as follows:

$$\begin{aligned} u_1 &\leq 0.5\text{ in} \\ u_2 &\leq 0.5\text{ in} \\ u_3 &\leq 0.5\text{ in} \end{aligned} \tag{22}$$

The traditional penalty function formulation was used with population sizes of 60 and 120; for the bounded penalty function approach, sizes of 50 and 100 were considered. These results are summarized in Figures 8 and 9, respectively. As in previous problems, the bounded penalty formulation yielded better performance. An optimal weight of 82.61 lbs was obtained with the corresponding design variable vector as follows:

$$x^* = [0.442, 0.602, 0.307, 0.442, 0.602, 0.111, 0.111, 0.602, 0.602, 0.602, 0.442]\text{ in}^2$$

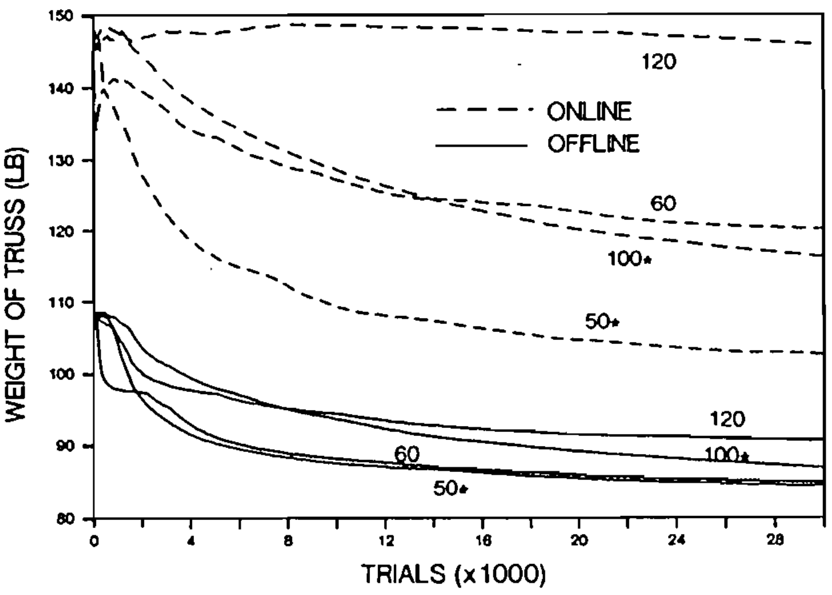

Figure 8 Iteration history of structural weight—online and offline measures (* indicates use of bounded penalty).

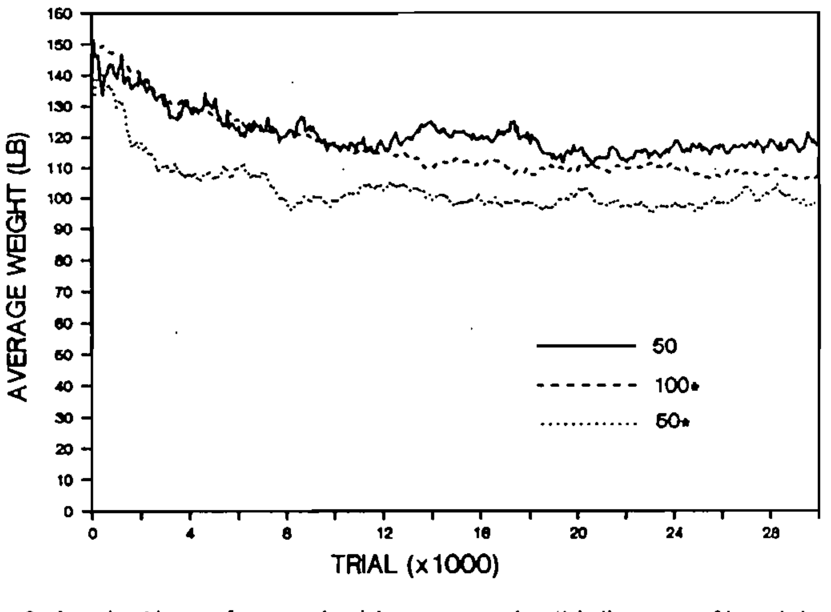

Figure 9 Iteration history of structural weight—average value. (* indicates use of bounded penalty).

CLOSING REMARKS

The present paper presents design variable representation schemes that allow for an application of genetic search methods to problems characterized by a mix of integer, discrete and continuous variables. Three distinct schemes are presented and evaluated in the context of engineering design problems. Genetic search methods would require less computational effort than the traditional branch-and-bound schemes in problems where the number of integer and discrete design variables is large. Additionally, the method has an inherently better probability of locating the global optimum than the gradient-based methods. The present study has also examined the issue of transforming constrained optimization problems into an unconstrained function maximization, as required in the genetic search approach. Preliminary findings underscore the need for a very careful evaluation of the objective function magnitude in relation to that of the penalty term. Further numerical experimentation is necessary on larger scale problems to build upon the general observations available from this study.

Acknowledgement

Support received under ARO Grant DAAL03-88-K-0013 is gratefully acknowledged.

References

1. Johnson, E. H. (1976) Optimization of Structures Undergoing Harmonic or Stochastic Excitation. *AIAA Journal*, **14**, (2), 259–261.
2. Glover, F. and Sommer, D. (1975) Pitfalls of Rounding in Discrete Management Decision Variables. *Decision Sciences*, **22**, (4), 445–460.
3. Koski, J. (1979) *Truss Optimization with Vector Criterion*. Tampere University of Technology, Publications 6, Tampere, Finland.
4. Koski, J. (1981) Multicriteria Optimization in Structural Design. *Proceedings of the International Symposium on Optimum Structural Design, 11th ONR Naval Structural Mechanics Symposium*, Tuscon, Arizona, October 19–22.
5. Osyczka, A. (1978) An Approach to Multicriterion Optimization Problems for Engineering Design. *Comp. Meths. Appl. Mech. Eng.*, **15**, 309–333.
6. Hajela, P. and Shih, C.-J. (1989) Optimal Design of Laminated Composites Using a Modified Mixed Integer and Discrete Programming Algorithm. *Computers and Structures*, **32**, (1), 213–221.
7. Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.
8. Hajela, P. (1989) Genetic Algorithms in Automated Structural Synthesis. In *Optimization and Decision Support Systems* (ed. B. H. V. Topping), Kluwer Academic Publishers.
9. Hajela, P. (1990) Genetic Search—An Approach to the Nonconvex Optimization Problem. *AIAA Journal*, **26**, (7), 1205–1210.
10. Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, Reading.
11. DeJong, K. A. (1975) Analysis of the Behavior of a Class of Genetic Adaptative Systems. *Ph.D. Thesis*, Dept. Computer and Communication Sciences, University of Michigan.
12. Granet, I. (1980) *Strength of Materials for Engineering Technology*. 2nd edition, Reston Publishing, Reston.
13. Dimarogonas, A. (1989) *Computer Aided Machine Design*, Prentice-Hall, New York.
14. Wahl, A. M. (1949) *Mechanical Springs*. Penton Publishing, Cleveland.
15. Chironis, N. P. (1961) *Spring Design and Applications*. McGraw-Hill, New York.