A Simple Genetic Algorithm for the Design of Reinforced Concrete Beams

Carlos Coello Coello[†] Alan D. Christiansen[†] Filiberto Santos Hernández[‡]

[†]*Department of Computer Science, Tulane University, New Orleans, LA 70118, USA*
[‡]*Escuela de Ingeniería Civil, UN.A.CH., Apartado Postal 61, Tuxtla Gutiérrez, Chiapas, México*

Abstract

We present an optimization model for the design of rectangular reinforced concrete beams subject to a specified set of constraints. Our model is more realistic than previously published models because it minimizes the cost of the beam based on strength design procedures, while also considering the costs of concrete, steel and shuttering. Thus our method leads to very practical designs. As there is an infinite number of possible beam dimensions and reinforcement ratios that yield the same moment of resistance, an efficient search technique is preferred over the more traditional iterative methods. We employ a simple genetic algorithm as the search engine, and we compare our results to those obtained via geometric programming. Since the adjustment of parameters in a genetic algorithm (e.g., population size, crossover and mutation rates, and maximum number of generations) is a significant problem for any application, we present our own methodology to deal with this problem. A prototype of this system is currently being tested in México, in order to evaluate its potential as a reliable design tool for real-world applications.

Keywords*:* structural optimization, genetic algorithms, design optimization, artificial intelligence.

1 Introduction

The design of reinforced concrete elements plays a very important role in México, because of its extensive use by civil engineers. The traditional design method proposes a certain solution that is then corroborated by mathematical analysis in order to verify that the problem requirements are satisfied. If such requirements are not satisfied, then a new solution is proposed by the designer based on his intuition, or some heuristics derived from his experience (See Figure 1). This process is typically of high cost in terms of time and human effort. As time is always a constraint in real design, a reasonable solution that satisfies all the requirements of the problem is usually adopted, and cost optimization is not even considered. Recently, computers have been used to help engineers automate the design process, but their use has been mainly in performing the tedious and repetitive mathematical calculations that are required, rather than in automatically generating designs.

An alternative to the traditional design method is optimal design, which consists of changing the design based on a certain "optimality condition" (See Figure 2). However, the general optimal design problem is highly nonlinear and nonconvex [1]. As a result, structural optimization problems are characterized by having multiple local optima.

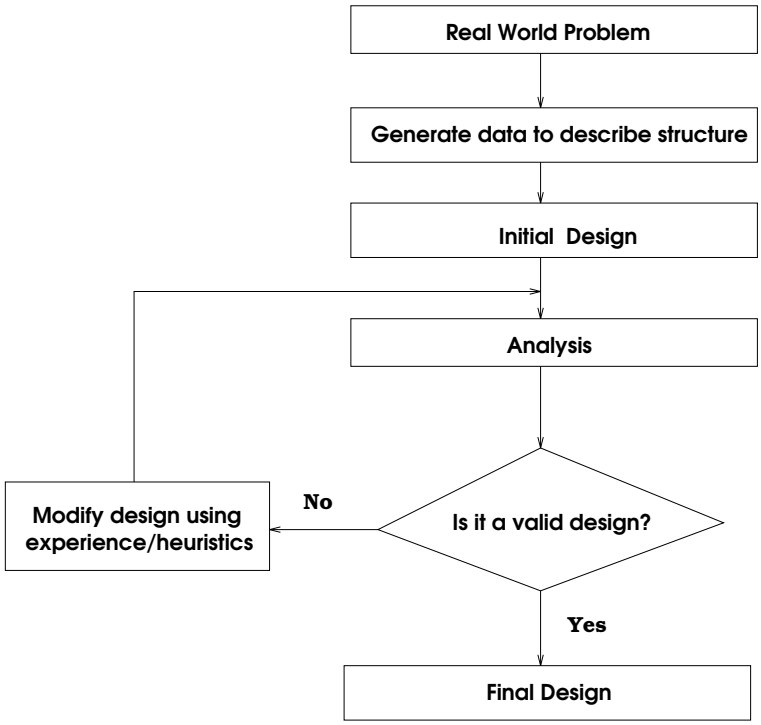

Figure 1: Traditional design process.

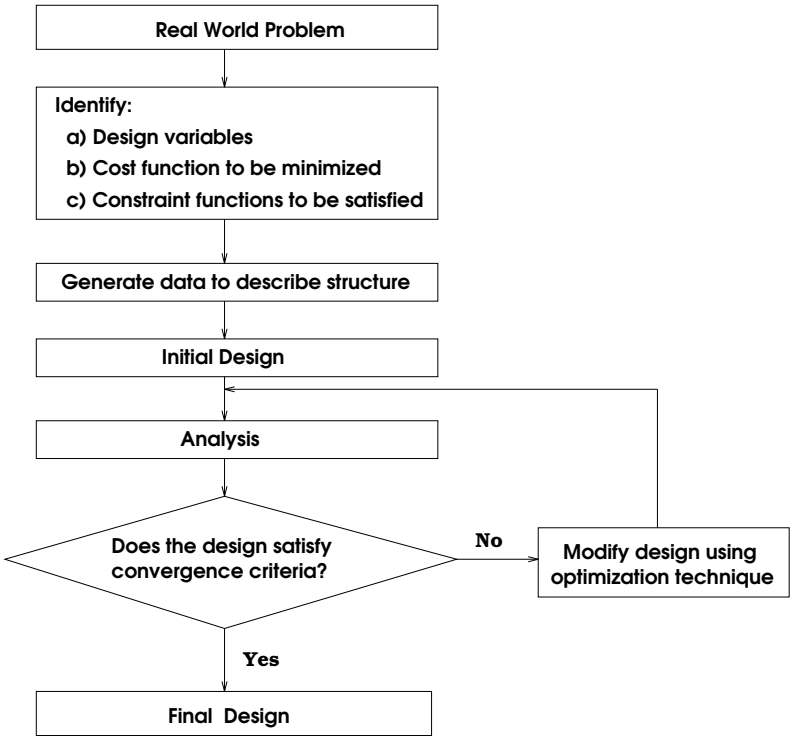

Figure 2: Optimal design process. Taken from Belegundu [1].

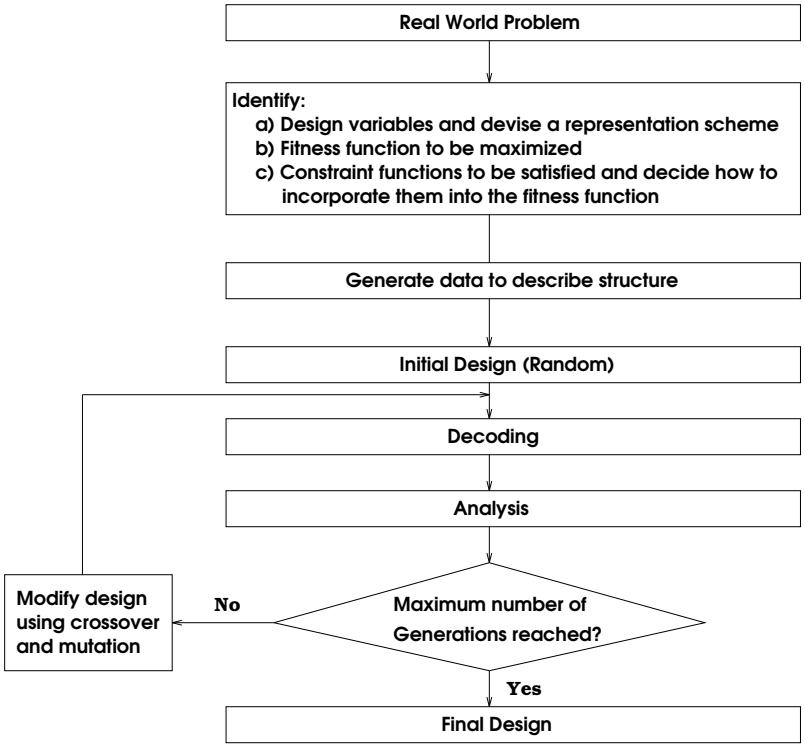

Figure 3: Optimal design process using a Genetic Algorithm.

This paper focuses on the use of an artificial intelligence (AI) technique based on the mechanics of natural selection, called the *genetic algorithm* [2] [3]. The design process based on this technique is very similar to the optimal design process (See Figure 3). The main difference is the use of a *fitness function* instead of a cost function, and the fact that the adaptation of the design is not dependent upon either the engineer (as in traditional design) or the gradient of the cost function, (as in optimal design). Even more interesting is that initial designs are randomly generated, with no human intervention, and the technique nevertheless converges to at least a reasonably good sub-optimal design in a reasonable amount of time.

The design of a reinforced concrete beam is normally an iterative process like the one shown in Figure 1, in which the engineer assumes a total weight for the beam beforehand, and a trial section is chosen. Then, the moment of resistance of this section is determined, to check its suitability against the given applied bending moment. This process is repeated until a trial section is found to be suitable. This procedure often creates a difficulty in matching the moment of resistance of the section with the total applied bending moment due to the beam's weight, which may be quite substantial in many cases. Therefore, not only is the design process of a beam slow, but it also has no economic analysis since the only concern is to find any section suitable for the given conditions. A better approach would be to find the most economical design that is also physically suitable.

In this paper, we present a model for optimal design that minimizes the cost of a rectangular reinforced concrete beam based on strength design procedures, but also considers the costs of concrete, steel and shuttering. Our model follows the one proposed by Chakrabarty [4] [5], with certain modifications (i.e., additional constraints) that make it suitable for practical applications. In the next section, we will introduce some general concepts from reinforced concrete design. Then, our model will be presented and the genetic algorithm approach will be described. Finally, we will present the results found by our model when solving some problems found in the literature, and we will discuss some of the issues that arise when using genetic algorithms in this kind of application.

2 Basic Concepts

For the purposes of this research, we adopted strength design procedures, because they have, among others, the following advantages [6]:

- Strength design better predicts the strength of a section because of the recognition of the non-linearity of the stress-strain diagram at high stress levels.
- Because the dead loads to which a structure is subjected are more certainly determined than the live loads, it is unreasonable to apply the same factor of safety to both. Considering that fact, this approach allows the use of different safety factors for them.

The basic assumptions that are taken when using strength design are the following [6]:

- Plane sections before bending remain plane after bending.
- At ultimate capacity, strain and stress are not proportional.
- Strain in the concrete is proportional to the distance from the neutral axis.
- Tensile strength of concrete is neglected in flexural computations.

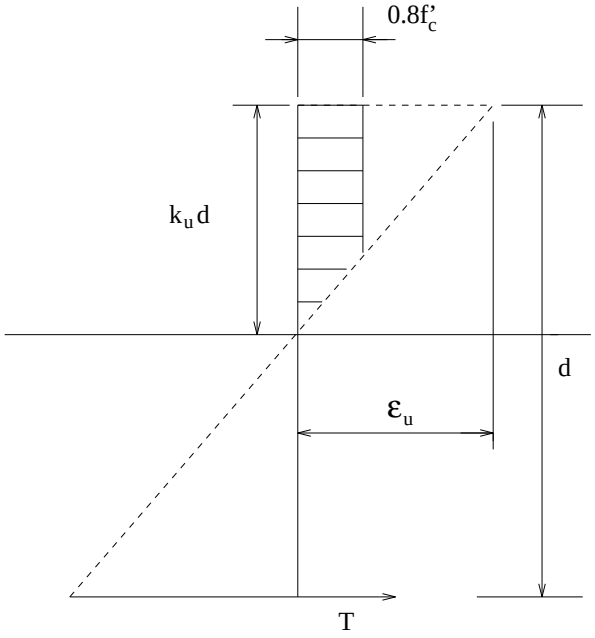

Figure 4: Trapezoid stress distribution.

- The ultimate concrete strain is 0.003.
- The modulus of elasticity of the reinforcing steel is 200,000 MPa (29,000,000 psi).
- The average compressive stress in the concrete is $0.85f_c^{'}$.
- The average tensile stress in the reinforcement does not exceed f_y.

According to this design method, if we assume a trapezoidal stress distribution as the one showed in Figure 4, the nominal moment capacity M_n of a rectangular beam with tension reinforcement only is given by [6]:

$$M_n = bd^2 f_c^{'} w(1 - 0.59w) \tag{1}$$

where b is the width of the beam, d is the distance from the extreme compressive fiber to the centroid of tension reinforcement, $f_c^{'}$ is the compressive strength of concrete, $w = (A_s f_y / bdf_c^{'})$, f_y is the yield strength of reinforcement and A_s is the area of tension reinforcement.

There is an infinite number of solutions to equation (1) that yield the same value of M_n [6]. In the traditional design process, the values of b and/or d are assumed, and the remaining parameters are calculated based on them, iterating until a suitable section is found. An obvious restriction of this approach is that only a few sections can be evaluated in this manner. Since equation (1) does not incorporate any cost parameter, there is no way of achieving a least-cost design. Therefore, we need to include certain cost parameters combined with the design parameters in our optimal design model, so that we can produce least-cost suitable designs.

3 Previous Work

The optimal design of beams was first proposed by Galileo [7], although his calculations were wrong. Apparently, the doctoral dissertation by E. J. Haug Jr. [8] (see also [9]) in 1966 was one of the first modern attempts to use a digital computer as a tool for the optimal design of this structural element. Haug reduced the non-linear optimal design problem to a Lagrange problem in the Calculus of Variations with inequality constraints. His model considered a beam made of a linearly elastic material of known density with two supports and a certain given load. The control variables were the values of cross sections at different points along the beam, and constraints on the stress, shear and deflection were imposed. Haug used an iterative method based on the generalized Newton's algorithm to solve statically determinate beams.

Venkayya [10] developed a method based on an energy criterion and a search procedure based on constraint gradient values for the design of structures subjected to static loading. His method can handle very efficiently: (a) design for multiple loading conditions, (b) stress constraints, (c) displacement constraints and (d) limits on sizes of the elements. This method also has been successfully applied to the design of trusses, frames and beams. In these cases, the weight of the structural element is the parameter to be minimized.

Karihaloo [11] presented a model to minimize the maximum deflection of a simply supported beam under a transverse concentrated load. Haug and Arora [12] used the gradient projection method to optimize the design of simply supported and clamped beams with constraints on stress, deflection, natural frequency and bounds on the design variables. Again, the weight (volume) of the beam is the parameter to be minimized.

Saouma et al. [13] developed a method for minimum cost design of simply supported, uniformly loaded, partially prestressed concrete beams. This model uses nine design variables: six geometrical dimensions, area of prestressing steel and area of tensile and compressive mild reinforcement. The imposed constraints are on the four flexural stresses, initial camber, dead and live load deflections, ultimate shear and ultimate moment capacity with respect to both the cracking moment and the applied load. This model was solved using the Penalty-Functions method coupled with Quasi-Newton unconstrained optimization techniques.

Das Gupta et al. [14] applied generalized geometric programming to the optimal design of a modular floor system, which consisted of reinforced solid concrete and voided slab units supported on steel beams. One of the most remarkable characteristics of this model is that it defines a function representing the cost of the floor system in terms of design variables, length, width and thickness of components, and other engineering cost parameters. This function is minimized subject to various constraints depending on stresses and deflections, and a dual-based algorithm was used to solve it.

Some other authors have used multiobjective optimization techniques to deal with this problem. For example, Rao [15] studied a cantilever beam with a hollow rectangular cross section and tip mass, for which he minimized its structural mass and its fatigue damage, while he maximized its natural frequency. After using many different approaches (i.e., global criterion, game theory, goal attainment, utility function, etc.), he concluded that game theory gave the best results. Also, Osyczka [16] [17] used the min-max method for the optimal design of an I-beam. More recently, Lounis and Cohn [18] considered the design of a posttensioned floor slab and a pretensioned highway bridge system for two conflicting objectives: minimum cost and minimum initial camber. They used the ε-constraint method to transform this multiobjective optimization problem into a single nonlinear optimization problem that they solved using the projected Lagrangian method.

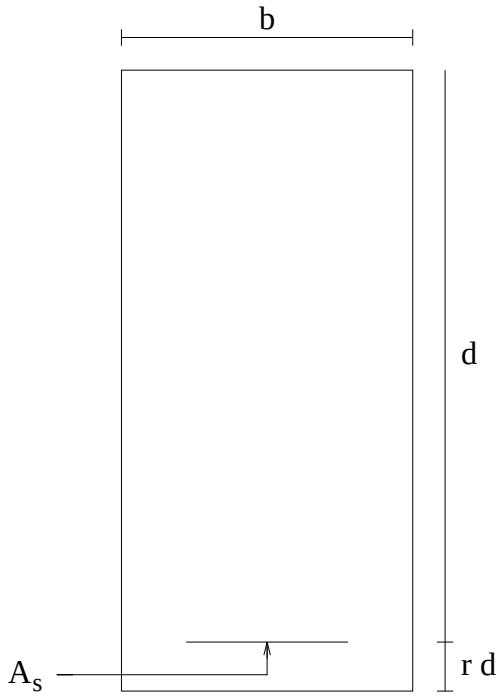

Figure 5: Schematic section of a singly reinforced rectangular beam.

Prakash et al. [19] proposed a model for optimal design of reinforced concrete sections in which the costs of steel, concrete and shuttering were included. Chakrabarty's model [4] [5] has some similarities with Prakash's model, but Chakrabarty's model is more complete and detailed. That is the main reason why we decided to use Chakrabarty's model as a basis for our implementation, although we modified it slightly to produce designs that fall into Mexico's standard regulations for reinforced concrete design. The original model leads to inconsistent designs in some cases.

4 The Optimal Design Model

A schematic section of a rectangular singly reinforced concrete beam is shown in Figure 5. The cost per unit length of the beam will be given by the following expression [5]:

$$y(x) = c_1 A_s + c_2 bd + c_3 d + c_4 b \tag{2}$$

where $y(x)$ is the cost per unit length of the beam ($\$/cm$), c_1 is the cost coefficient due to volume of tensile steel reinforcement in the beam ($\$/cm^3$), c_2 is the cost coefficient due to volume of concrete in the beam ($\$/cm^3$), c_3 is the cost coefficient due to shuttering along the vertical surfaces of the beam ($\$/cm^2$), c_4 is the cost coefficient due to shuttering along the bottom horizontal surface of the beam ($\$/cm^2$), A_s is the variable giving the area of tensile steel reinforcement as shown in Figure 5 (cm^2), d is the variable giving the depth of the beam as shown in Figure 5 (cm) and b is the variable giving the width of the beam, as shown in Figure 5 (cm).

The variables A_s, d and b not only affect the cost of a beam, but will also determine its moment of resistance. Since A_s may be calculated if we know d and b [6], we propose different values for these two variables so that the total cost of the beam is minimum, and that our section has a proper resistant moment. Then, our optimal design model is the following:

minimize: $f(x) = c_1 A_s + c_2 bd + c_3 d + c_4 b$

subject to:

$$\frac{a_1}{A_s} kbd < 1 \quad (equilibrium\ constraint) \tag{3}$$

$$\frac{a_2 + a_3 bd}{M_T} < 1$$
$$(bending\ moment\ compatibility\ constraint) \tag{4}$$

$$0.25 \leq b/d \leq 0.6$$
$$(width - height\ ratio\ constraint) \tag{5}$$

$$Q(d - a_5 kd)(f_r f_c^{'} kdb + A_s f_y) a_5 / M_T \geq 1$$
$$(acting\ moment\ constraint) \tag{6}$$

$$a_6/b < 1 \quad (minimum\ width\ constraint) \tag{7}$$

$$A_s, d, b, M_T, kd > 0 \quad (non - negativity\ constraint) \tag{8}$$

Here M_T is a variable defining the total applied bending moment including the bending moment due to self-weight of the beam; kd is a variable defining the depth of the equivalent rectangular stress block; k is a factor related to the location of the neutral axis of a cross section. Additionally, we have the following formulas:

$$c_1 = w_s \times c_s \quad (\$/cm^3) \tag{9}$$

where $w_s = 0.00785\ kg/cm^3$ (assumed value) is the unit weight of steel reinforcement, and c_s is the unit cost of steel reinforcement ($\$/kg$).

$$c_2 = (1 + r) c_c \times 10^{-6} \quad (\$/cm^3) \tag{10}$$

where c_c is the unit cost of concrete ($\$/m^3$) and r is the cover ratio.

$$c_3 = 2(1 + r) c_r \times 10^{-4} \quad (\$/cm^2) \tag{11}$$

where c_r is the unit cost of shuttering ($\$/m^2$).

$$c_4 = c_r \times 10^{-4} \quad (\$/cm^2) \tag{12}$$

$$a_1 = 0.85 f_c^{'}/f_y \tag{13}$$

where f_y is the yield strength of steel reinforcement (N/cm^2) and $f_c^{'}$ is the compressive strength of concrete (N/cm^2).

$$a_3 = D(1+r)w_c k L^2 \tag{14}$$

where $D = 1.4$ (assumed) is the load factor for dead load, $w_c = 0.0228\ N/cm^3$ is the unit weight force of concrete, k is the moment coefficient for the design section ($= 1.8$ for simply supported beam) and L is the span of the beam (cm).

$$a_4 = 1/(f_r Q f_c^{'}) \tag{15}$$

where Q is the capacity reduction factor ($= 0.90$ for flexure) and $f_r = 0.85$ (assumed) is the reduction factor of concrete. Also, a_2 is the applied bending moment ($N - cm$), $a_5 = \frac{1}{2}$ (assuming the centroid of compressive force at half the depth of equivalent rectangular stress block), and a_6 is the minimum acceptable width of the beam.

To determine M_T (total bending moment, including self-weight of the beam), we use:

$$M_T = a_2 + a_3 bd \tag{16}$$

To calculate A_s (area of reinforcement steel), we use:

$$A_s = \omega bd f_c^{'}/f_y \tag{17}$$

where $\omega = \frac{1 - \sqrt{1 - \frac{4(0.59)M_T}{0.9bd^2 f_c^{'}}}}{1.18}$

This last expression can be derived from equation (1). Finally, kd (depth of the equivalent stress block) is given by:

$$kd = A_s/(a_1 b) \tag{18}$$

5 Use of Genetic Algorithms

The *genetic algorithm* is a heuristic search technique based on the mechanics of natural selection developed by John Holland [2].

Koza [20] provides a good definition of a GA:

> The *genetic algorithm* is a highly parallel mathematical algorithm that transforms a set (*population*) of individual mathematical objects (typically fixed-length character strings patterned after chromosome strings), each with an associated *fitness* value, into a new population (i.e., the next *generation*) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (notably sexual recombination).

A genetic algorithm for a particular problem must have the following five components [21]:

1. A representation for potential solutions to the problem.
2. A way to create an initial population of potential solutions.
3. An evaluation function that plays the role of the environment, rating solutions in terms of their "fitness".
4. Genetic operators that alter the composition of children.
5. Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

Certain terminology is commonly used by the GA community:

- A *chromosome* is a data structure that holds a "string" of task parameters, or genes. This string may be stored, for example, as a binary bit-string (binary representation) or as an array of integers (floating point representation).
- A *gene* is a subsection of a chromosome that usually encodes the value of a single parameter.
- An *allele* is the value of a gene. For example, for a binary representation each gene may have an allele of 0 or 1, and for a floating point representation, each gene may have an allele from 0 to 9.
- A *schema* (plural *schemata*) is a pattern of gene values in a chromosome, which may include "do not care" states (represented by a # symbol). Thus in a binary chromosome, each schema can be specified by a string of the same length as the chromosome, with each character being one of { 0, 1, # }. A particular chromosome is said to "contain" a particular schema if it matches the scheme (e.g. chromosome 01101 matches schema #1#0#).

The basic operation of a Genetic Algorithm is illustrated in the following segment of pseudo-code [22]:

```
generate initial population, G(0);
evaluate G(0);
t:=0;
repeat
   t:=t+1;
   generate G(t) using G(t-1);
   evaluate G(t);
until a solution is found
```

First, an initial population is randomly generated. The individuals of this population will be a set of chromosomes or strings of characters (letters and/or numbers) that represent all the possible solutions to the problem. We apply a *fitness function* to each one of these chromosomes in order to measure the quality of the solution encoded by the chromosome. Knowing each chromosome's fitness, a *selection* process takes place to choose the individuals (presumably, the fittest) that will be the parents of the

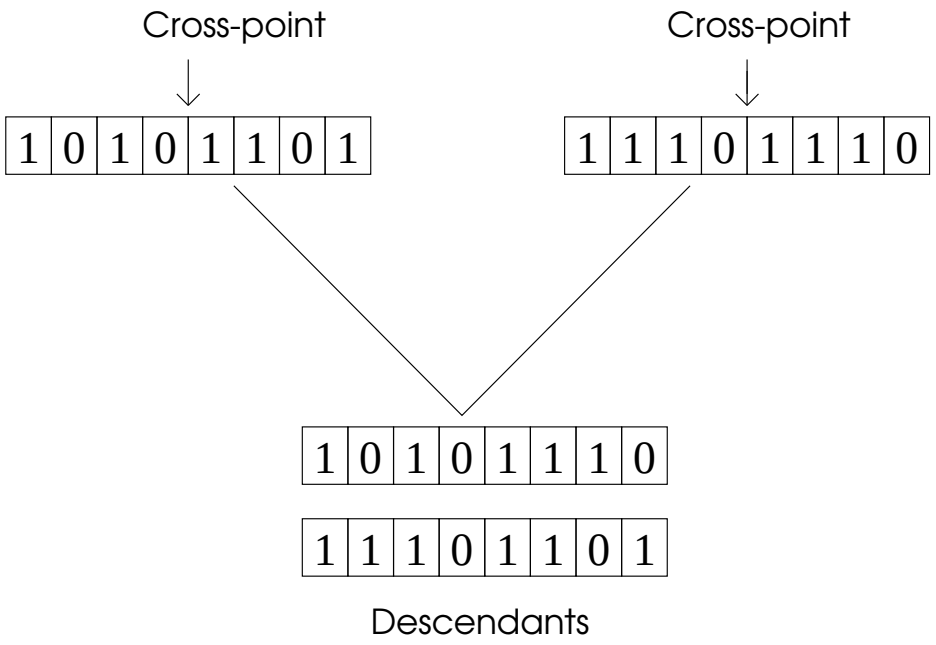

Figure 6: Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation.

following generation. There are two main selection schemes: roulette wheel selection and tournament selection.

In *roulette wheel* selection, each individual is assigned a certain probability F_i of being selected, computed according to the formula [22]:

$$F_i = \frac{f_i}{\sum_j f_j}$$

where f_i is the fitness value of each chromosome i. In this selection scheme the fittest individuals have a higher probability of being selected, but individuals with lower fitness can also eventually be selected.

In *tournament* selection, k individuals are selected in a single iteration, and the best one from this set of k elements is chosen to be a parent to produce the next generation. This process is repeated as many times as the size of the population. Large values of k increase selective pressure of this procedure [21]; a typical value accepted by many applications is $k = 2$ (binary tournament selection).

After being selected, *crossover* takes place. During this stage, the genetic material of a pair of individuals is exchanged in order to create the population of the next generation. The two main ways of performing crossover are called single-point and two-point crossover. When a *single-point* crossover scheme is used, a position of the chromosome is randomly selected as the crossover point as indicated

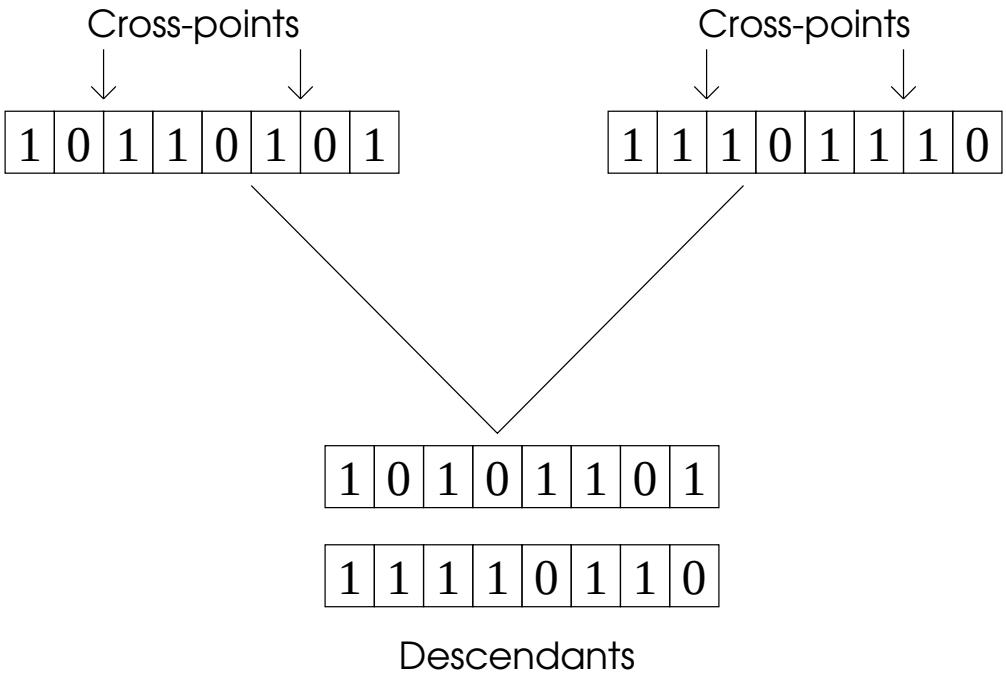

Figure 7: Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries, the parents remain intact for the next generation.

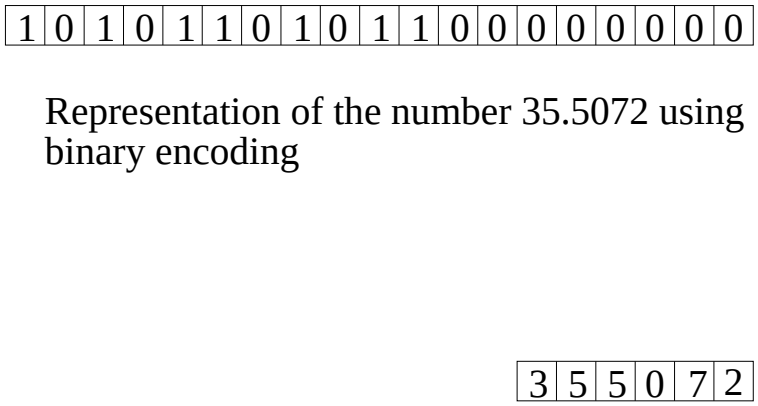

Figure 8: Representing the same number using binary and floating point encodings.

in Figure 6. When a *two-point* crossover scheme is used, two positions of the chromosome are randomly selected as indicated in Figure 7.

Mutation is another important genetic operator that randomly changes a gene of a chromosome. If we use a binary representation, a mutation changes a 0 to 1 and viceversa. This operator allows the introduction of new chromosomic material to the population and, from the theoretical perspective, it assures that—given any population—the entire search space is connected [22].

If we knew the final solution in advance it would be trivial to determine how to stop a genetic algorithm. However, as this is not normally the case, we have to use one of the two following criteria to stop the GA: either give a fixed number of generations in advance, or verify when the population has stabilized (i.e., all or most of the individuals have the same fitness).

GAs differ from traditional search techniques in several ways [22]:

- GAs do not require problem specific knowledge to carry out a search.
- GAs use stochastic operators rather than deterministic operators and appear to be robust in noisy environments.
- GAs operate on multiple partial solutions simultaneously (sometimes called implicit parallelism), gathering information from a population of search points to direct subsequent search efforts. Their ability to maintain multiple partial solutions concurrently helps make GAs less susceptible to the problems of local maxima and noise.

The traditional representation used by the genetic algorithms community is the binary scheme according to which a chromosome is a string the form $\langle b_1, b_2, \ldots, b_m \rangle$, where $b_1, b_2, \ldots, b_m$ are called *alleles* (either zeros or ones). Since the binary alphabet offers the maximum number of schemata per bit of

information of any coding [3], its use has became very popular among scientists. This coding also facilitates theoretical analysis of the technique and allows elegant genetic operators. However, since the "implicit parallelism" property of GAs does not depend on using bit strings [21] it is worthwhile to experiment with larger alphabets, and even with new genetic operators. In particular, for optimization problems in which the parameters to be adjusted are continuous, a floating point representation scheme seems a logical choice. According to this representation, a chromosome is a string of the form $\langle d_1, d_2, \ldots, d_m \rangle$, where $d_1, d_2, \ldots, d_m$ are digits (numbers between zero and nine). Consider the examples shown in Figure 8, in which the same value is represented using binary and floating point encoding.

The term "floating" may seem misleading since the position of the implied decimal point is at a fixed position, and the term "fixed point representation" seems more appropriate. However, the reason that the term "floating point" is preferred is because in this representation each variable (representing a parameter to be optimized) may have the point at any position along the string. This means that even when the point is fixed for each gene, is not necessarily fixed along the chromosome. Therefore, some variables could have a precision of 3 decimal places, while others are integers, and still they could all be represented with the same string.

Floating point representation is faster and easier to implement, and provides a higher precision than its binary counterpart, particularly in large domains, where binary strings would be prohibitively long. One of the advantages of floating point representation is that it has the property that two points close to each other in the representation space must also be close in the problem space, and vice versa [21]. This is not generally true in the binary approach, where the distance in a representation is normally defined by the number of different bit positions. Such discrepancy can, however, be reduced by using Gray coding.

To procedures to convert a binary number $\mathbf{b} = \langle b_1, b_2, \ldots, b_m \rangle$ into a Gray code number $\mathbf{g} = \langle g_1, g_2, \ldots, g_m \rangle$, where m denotes the number of bits, may be found in [21]. The Gray code representation has the property that any two points next to each other in the problem space differ by only one bit [21]. In other words, an increase of one step in the parameter value corresponds to a change of a single bit in the code. This is a well known technique used to reduce the distance of two points in the problem space, and it is argued to bring some benefit because of their adjacency property, and the small perturbation caused by many single mutations. However, the use of Gray codes did not help much in this particular application, as we will see in the next section.

To solve this optimization problem, we used the Simple Genetic Algorithm (SGA) proposed by Goldberg [3], and we experimented with several representation schemes. We have previously used a binary representation [23] and we have tried Gray coding [24] for structural optimization problems with a continuous search space like this one. For this particular application, we decided to experiment also with floating point representation.

Finally, we should mention that we used a two-point crossover and binary tournament selection in all our tests. The only operator that had to be redefined was *mutation*, which in the floating point representation selects a random number between 0 and 9. Our fitness function was given by equation (2), using a penalty function of the form $fitness = 1/(cost*(v*500+1))$ where v depends on the number of constraints violated. Whenever the design does not violate any constraint, the fitness function is just the inverse of the cost (the GA only maximizes, and we require a minimization in this case).

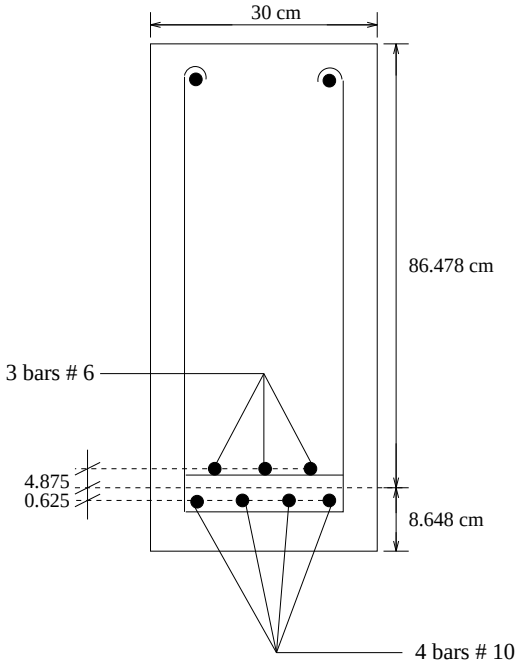

Figure 9: Optimum design of the beam of the first example.

6 Examples

The following example was taken from Everard [6]:

Design a least-cost reinforced concrete rectangular beam simply supported over a span of 10 m supporting a uniform dead load of 15 kN/m and a uniform live load of 20 kN/m. The concrete strength $f_c^{'} = 30\ MPa$ and the steel yield strength $f_y = 300\ MPa$. The unit cost of steel (CS), concrete (CC) and shuttering (CSH) are \$ $0.72/kg$, \$ $64.5/m^3$ and \$ $2.155/m^2$, respectively. Assume a cover ratio (r) of 0.10, unit weight of concrete of 2323 kg/m^3 and capacity reduction factor as 0.90.

The ultimate uniform load is

$= 1.4 \times 15 + 1.7 \times 20 = 55\ kN/m.$

The ultimate applied bending moment is

$= 55 \times 10^2/8 = 687.5\ kN, \quad m = 687.5 \times 10^5\ N - cm.$

Using this information, we can get the values of the cost coefficients and the other model constants:

$c_1 = 0.0056520$	$c_2 = 0.00007095$	$c_3 = 0.00047410$	$c_4 = 0.00021550$
$a_1 = 0.08500$	$a_2 = 68,750,000$	$a_3 = 438,233,950$	$a_4 = 0.00043573$
$a_5 = 0.50$	$a_6 = 30.00$		

Our results and their comparison with the geometric programming method used by Chakrabarty [5] are shown in Table 1 and the final design generated by the genetic algorithm is shown in Figure 9. The value of $A_s = 37.5205$ is approximated by using 4 bars # 10 (nominal cross sectional area=819 mm^2)

Parameter	Chakrabarty	GA (Binary)	GA (Gray Coding)	GA (FP)
A_s (cm^2)	37.6926	36.1893	41.5905	37.5205
d (cm)	86.0629	89.5402	78.6177	86.4776
b (cm)	30.0000	30.0162	30.0447	30.0022
M_T $(N-cm)$	$80'064,711.73$	$80'540,242.0620$	$79'111,846.5650$	$80'131,661.9160$
kd (cm)	14.7814	14.1842	16.2857	14.7128
$cost$ $(\$/cm)$	0.4435	0.4442	0.4464	0.4436

Table 1: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using binary (with and without Gray coding) and floating point representation.

and 3 bars # 6 (nominal cross sectional area=284 mm^2). This gives us a total area of 41.28 mm^2, which is greater than the design value as the ACI (American Concrete Institute) regulations recommend. The available reinforcing bar sizes and their corresponding properties according to the ASTM (American Society for Testing and Materials), are shown in Table 2 ([6]).

As we can see from these results, the floating point representation produced the best results and Gray coding the worst. The final cross-section of the beam has a total height of 95.125 (see Figure 9), which is about 1% more than Chakrabarty's design. This slight difference is due to the fact that Chakrabarty's model considers the area of reinforcement steel as a variable, even though this is a parameter that depends on the beam section, and can not take any arbitrary value. On the other hand, our costs of steel, concrete and shuttering represent 47.80%, 41.50% and 10.70% of the total cost, respectively, which corresponds almost exactly to the costs obtained by Chakrabarty. Floating point represention was used in all the further experiments, since it provided the best results overall.

An important observation should be made before showing more examples. Our model has more constraints than Chakrabarty's model, in order to make it more realistic. For example, we require the relation b/d to be between 0.25 and 0.6 which is a common practice recipe used by civil engineers. The reason for this is not purely empirical. These limits allow us to have a "reasonable" amount of reinforcement steel in our designs, so that we can guarantee a good adherence between steel and concrete, and we can provide a good control of the beam's deflection. Since Chakrabarty does not impose this constraint in his model, some of the results shown next will violate it.

First, we perform an analysis similar to that conducted by Chakrabarty, experimenting with different values of b. The results of our tests are shown in Tables 3, 4 and 5. For the case in which $b = 62.50$, Chakrabarty's model produces a design 42.98% more expensive than when $b = 30$. Our design is 63.98% more expensive. However, Chakrabarty's design violates the restriction imposed by equation (5). Therefore, in practice an engineer would prefer our design even though it is more expensive, for the reasons previously discussed. In all the remaining examples, it will always be the case that when our results are not equal to those produced by Chakrabarty's model (or almost equal, should we say, since there is always a difference in the last digit due to rounding-off errors) it is because his design is violating some constraint—usually that defined by equation (5).

Finally, we tested different values for the costs of reinforcement steel, concrete and shuttering. The results are shown in Tables 6 to 12. Again, the discrepancies between our results and those produced by Chakrabarty's method indicate some violation of the constraints imposed by our model.

ASTM standard reinforcing bars					
		Nominal dimensions—Round sections			
Bar size, designation	Weight, lb/ft	Diameter, in.	Cross-sectional area, in.^{2}	Perimeter, in.	Cross-sectional area, mm^2
# 3	0.376	0.375	0.11	1.178	71
# 4	0.668	0.500	0.20	1.571	129
# 5	1.043	0.625	0.31	1.963	200
# 6	1.502	0.750	0.44	2.356	284
# 7	2.044	0.875	0.60	2.749	387
# 8	2.670	1.000	0.79	3.142	510
# 9	3.400	1.128	1.00	3.544	645
# 10	4.303	1.270	1.27	3.990	819
# 11	5.313	1.410	1.56	4.430	1006
# 14	7.65	1.693	2.25	5.320	1452
# 18	13.60	2.257	4.00	7.090	2581

Table 2: Properties of Reinforcing Bars. Taken from Everard [6].

Parameter	**Chakrabarty**	**GA (FP)**
A_s (cm^2)	31.1267	39.5412
d (cm)	101.5494	82.7043
b (cm)	20.000	20.6825
$cost$ $(\$/cm)$	0.3725	0.3885

Table 3: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. We assume $b = 20$ cm. Notice how the constraint imposed by equation (5) is violated by Chakrabarty's design.

Parameter	**Chakrabarty**	**GA (FP)**
A_s (cm^2)	43.6017	43.7644
d (cm)	76.1499	75.9102
b (cm)	40.000	40.0042
$cost$ $(\$/cm)$	0.5073	0.5074

Table 4: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. We assume $b = 40$ cm.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	55.4435	35.7172
d (cm)	62.5974	104.3223
b (cm)	62.500	62.5010
$cost$ $(\$/cm)$	0.6341	0.7274

Table 5: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. We assume $b = 62.50$ cm. Notice how the constraint imposed by equation (5) is violated by Chakrabarty's design.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	57.0072	50.2583
d (cm)	59.8678	66.7029
b (cm)	40.000	40.0033
$cost$ $(\$/cm)$	0.3680	0.3716

Table 6: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 0.36$, $CC = 64.5$ and $CSH = 2.155$. Notice how the constraint imposed by equation (5) is violated by Chakrabarty's design.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	37.2006	37.0318
d (cm)	89.5455	90.0205
b (cm)	40.000	40.0010
$cost$ $(\$/cm)$	0.6206	0.6207

Table 7: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 1.08$, $CC = 64.5$ and $CSH = 2.155$.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	33.2691	33.2279
d (cm)	101.1724	101.3565
b (cm)	40.000	40.0001
$cost$ $(\$/cm)$	0.7198	0.7199

Table 8: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 1.44$, $CC = 64.5$ and $CSH = 2.155$.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	33.0372	35.0698
d (cm)	95.5279	95.4719
b (cm)	40.000	40.0001
$cost$ $(\$/cm)$	0.3875	0.3876

Table 9: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 0.72$, $CC = 32.25$ and $CSH = 2.155$.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	55.4240	49.9278
d (cm)	61.2698	67.0981
b (cm)	40.000	40.0050
$cost$ $(\$/cm)$	0.6987	0.7035

Table 10: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 0.72$, $CC = 129.0$ and $CSH = 2.155$. Notice how the constraint imposed by equation (5) is violated by Chakrabarty's design.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	42.3510	42.5568
d (cm)	78.3650	78.0625
b (cm)	40.000	40.0001
$cost$ $(\$/cm)$	0.4847	0.4848

Table 11: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 0.72$, $CC = 64.5$ and $CSH = 1.0775$.

Parameter	Chakrabarty	GA (FP)
A_s (cm^2)	45.9454	45.9012
d (cm)	72.4085	72.5103
b (cm)	40.000	40.0003
$cost$ $(\$/cm)$	0.5511	0.5512

Table 12: Comparison of the geometric programming approach used by Chakrabarty [4] and the GA using floating point representation. $b = 40$ cm, $CS = 0.72$, $CC = 64.5$ and $CSH = 4.31$.

7 Selecting the Parameters of the GA

One of the main problems when using GAs is how to choose the most appropriate parameter values (i.e., population size, maximum number of generations, mutation and crossover rate). This is usually a trial and error process which takes some time. One lesson derived from this research was that it is much harder to fine tune the parameters of the GA when a floating point representation scheme is used. We confronted a dilemma: the floating point representation gave the best results, but was also the hardest to deal with in terms of finding the most appropriate parameters. Obviously an optimization system will not be very useful if its outcomes are unpredictable. After much experimentation, we developed a systematic empirical process that seems to be able to generate optimal (or at least near-optimal) solutions in a very short period of time. However, we do not yet have any theoretical support of its reliability, even though the empirical evidence is quite solid. The method is the following:

- Choose a certain value for the random number seed and make it a constant.
- Choose constants for the population size and the maximum number of generations (we used 400 chromosomes and 50 generations, respectively).
- Loop the mutation and crossover rates from 0.1 to 0.9 at increments of 0.1 (this is actually a nested loop). This implies that 81 runs are necessary.
- For each run, update 2 files. One contains only the final costs, and the other has a summary that includes, besides the cost, the corresponding values of the design parameters and the mutation and crossover rates used. When the whole process ends the file with the costs is sorted in ascending order, and the smallest value is searched for in the other file, returning the corresponding design parameters as the final answer.

Since each run is completely independent from the others, we can perform this process in parallel, so that the total execution time will be practically the same required for a single run (approximately 15 seconds on a PC DX/2 running at 66 MHz and with a mathematical coprocessor).

Future Work and Conclusions

Much work remains to be done. We are considering the possibility of experimenting with other techniques for adjusting the parameters of the GA, such as fuzzy logic. Also, we are interested in doing a theoretical analysis of the search space of this optimization problem, so that we can devise some strategies to solve it more efficiently. However, so far the results seem very promising, and the system has attracted the attention of more than one engineer in our University. Although our system has been used only for academic purposes so far, we are considering the possibility of giving it commercial use.

We have been working in the use of GAs for structural optimization problems during the last two years, and now we are able to produce optimal designs of beams [25], columns [24] [26] and plane and space trusses [23] [27]. However, our final goal is to develop a complete structural optimization system that uses GAs and also incorporates the main traditional mathematical programming techniques available, together with some other powerful heuristics such as tabu search [28] [29]. This should provide a powerful tool for engineers involved in structural design, introducing considerable savings without sacrificing safety.

Acknowledgments

The authors gratefully acknowledge the work of Carlos Narcía López and Ascensión Elizalde Molina and the support of Ing. Robertony Cruz Díaz at the Escuela de Ingeniería Civil of the Universidad Autónoma de Chiapas. Without their help and dedication this paper would not have been possible.

References

1. Belegundu, A. D. (1982) *A Study of Mathematical Programming Methods for Structural Optimization.* PhD thesis, University of Iowa, Dept. of Civil and Environmental Engineering.

2. Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems.* Ann Harbor : University of Michigan Press.

3. Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, Mass. : Addison-Wesley Publishing Co.

4. Chakrabarty, B. K. (1992) A model for optimal design of reinforced concrete beam. *Journal of Structural Engineering*, 118(11):3238–3242.

5. Chakrabarty, B. K. (1992) Model for optimal design of reinforced concrete beams. *Computers and Structures*, 42(3):447–451.

6. Everard, N. J. (1993) *Theory and Problems of Reinforced Concrete Design.* Schaum's Outline Series. McGraw-Hill, third edition.

7. Galilei, G. (1950) *Dialogues Concerning Two New Sciences.* Evanston, Ill. Northwestern University Press. Originally published in 1665.

8. Haug, E. J. (1966) *Minimum Weight Design of Beams with Inequality Constraints on Stress and Deflection.* Department of mechanical engineering, Kansas State University.

9. Haug, E. J. and Kirmser, P. G. (1967) Minimum weight design of beams with inequality constraints on stress and deflection. *Journal of Applied Mechanics. Transactions of the ASME*, pages 999–1004.

10. Venkayya, V. B. (1971) Design of optimum structures. *Computers and Structures*, 1:265–309.

11. Karihaloo, B. L. (1979) Optimal design of multi-purpose tie-beams. *Journal of Optimization Theory and Applications*, 27(3):427–438.

12. Haug, E. J. and Arora, J. S. (1979) *Applied Optimal Design.* John Wiley and Sons, New York.

13. Saouma, V. E. and Murad, R. S. (1984) Partially prestressed concrete beam optimization. *Journal of Structural Engineering*, 110(3):589–604.

14. Gupta, N. C. D., Paul, H., and Yu, C. H. (1986) An application of geometric programming to structural design. *Computers and Structures*, 22(6):965–971.

15. Rao, S. S. (1984) Multiobjective optimization in structural design with uncertain parameters and stochastic processes. *AIAA Journal*, 22(11):1670–1678.

16. Osyczka, A. (1984) *Multicriterion Optimization in Engineering with FORTRAN programs.* Ellis Horwood Limited.

17. Osyczka, A. (1985) Multicriteria optimization for engineering design. In Gero, J. S., editor, *Design Optimization*, pages 193–227. Academic Press.

18. Lounis, Z. and Cohn, M. Z. (1993) Multiobjective optimization of prestressed concrete structures. *Journal of Structural Engineering*, 119(3):794–808.

19. Prakash, A., Agarwala, S. K., and Singh, K. K. (1988) Optimum design of reinforced concrete sections. *Computers and Structures*, 30(4):1009–1011.

20. Koza, J. R. (1992) *Genetic Programming. On the Programming of Computers by Means of Natural Selection.* The MIT Press.

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

22. Buckles, B. P. and Petry, F. E., editors (1992) *Genetic Algorithms.* Technology Series. IEEE Computer Society Press.

23. Coello, C. A. C. (1994) Discrete optimization of trusses using genetic algorithms. In Chen, J., Attia, F. G., and Crabtree, D. L., editors, *EXPERSYS-94. Expert Systems Applications and Artificial Intelligence*, pages 331–336, Houston, Texas. I.I.T.T. International. Technology Transfer Series.

24. Coello, C. A. and Christiansen, A. D. (1995) Using genetic algorithms for optimal design of axially loaded non-prismatic columns. Technical Report TUTR-CS-95-101, Tulane University.

25. Coello, C. A., Hernández, F. S., and Farrera, F. A. (1995) An approach to optimal design of reinforced concrete beams using genetic algorithms. In *Proceedings of the IASTED International Conference on Applied Modelling, Simulation and Optimization*, pages 141–144, Cancún, México. IASTED-ACTA Press.

26. Coello, C. A. and Christiansen, A. D. (1995) Using genetic algorithms for optimal design of axially loaded non-prismatic columns. In Pearson, D. W., Steele, N. C., and Albrecht, R. F., editors, *International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA'95*, pages 460–463, Alès, France. Ecole des Mines d'Alès, Springer-Verlag.

27. Coello, C. A. and Christiansen, A. D. (1994) Optimization of truss designs using genetic algorithms. Technical Report TUTR-CS-94-102, Tulane University.

28. Glover, F. (1989) Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206.

29. Glover, F. (1990) Tabu Search - Part II. *ORSA Journal on Computing*, 2:4–32.