CALIFORNIA INSTITUTE OF TECHNOLOGY

EARTHQUAKE ENGINEERING RESEARCH LABORATORY

OPTIMAL DESIGN OF BUILDING STRUCTURES USING GENETIC ALGORITHMS

BY

EDUARDO CHAN

REPORT NO. EERL 97-06

PASADENA, CALIFORNIA

1997

A REPORT ON RESEARCH PARTIALLY SUPPORTED BY THE
KAJIMA-CUREE JOINT RESEARCH PROGRAM AND THE
EARTHQUAKE RESEARCH AFFILIATES PROGRAM OF CALTECH
UNDER THE SUPERVISION OF JAMES L. BECK

Optimal Design of Building Structures Using Genetic Algorithms

Thesis by

Eduardo Chan

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1997

(Submitted June 10, 1997)

Acknowledgements

I would like to express my sincere appreciation to Professor James L. Beck for his support and encouragement throughout my graduate studies here at Caltech. Without him, the completion of this thesis would not have been possible. I greatly appreciate all the comments and suggestions given by my thesis committee members to make my thesis better.

I would also like to thank all the faculty and staff at Thomas building for making my stay at Caltech a very memorable one. Special gratitude is extended to my fellow graduate students David Polidori, Scott May, Michael Vanik, Anders Carlson, Mark Long, C.T. Huang and C.M. Yang for all the good times we spent together.

My deepest appreciation goes to my wife Bessie, who has constantly supported me throughout my doctoral studies. Despite being away from me for almost four years, she has demanded the minimum attention a wife would expect from her husband, which allowed to concentrate more on my research work. For her sacrifice, I would always be grateful.

Finally, I am deeply indebted to my parents for believing in my dreams. Nine years ago, they provided me with an opportunity to come the United States to pursue my college education. Completion of my doctoral studies would not be possible without the sacrifice they went through to support me through college.

Abstract

A general framework for multi-criteria optimal design is presented which is well-suited for automated design of structural systems. A systematic computer-aided optimal design decision process is developed which allows the designer to rapidly evaluate and improve a proposed design by taking into account the major factors of interest related to differeent aspects such as design, construction, and operation.

The proposed optimal design process requires the selection of the most promising choice of design parameters taken from a large design space, based on an evaluation using specified criteria. The design parameters specify a particular design, and so they relate to member sizes, structural configuration, etc. The evaluation of the design uses performance parameters which may include structural response parameters, risks due to uncertain loads and modeling errors, construction and operating costs, etc. Preference functions are used to implement the design criteria in a "soft" form. These preference functions give a measure of the degree of satisfaction of each design criterion. The overall evaluation measure for a design is built up from the individual measures for each criterion through a preference combination rule. The goal of the optimal design process is to obtain a design that has the highest overall evaluation measure - an optimization problem.

Genetic algorithms are stochastic optimization methods that are based on evolutionary theory. They provide the exploration power necessary to explore high-dimensional search spaces to seek these optimal solutions. Two special genetic algorithms, hGA and vGA, are presented here for continuous and discrete optimization problems, respectively.

The methodology is demonstrated with several examples involving the design of truss and frame systems. These examples are solved by using the proposed hGA and vGA.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The goal of structural engineers is to design structural systems according to design requirements such as the Uniform Building Code. However, in this highly competitive world, having a system that just performs the required task satisfactorily is no longer sufficient. It is essential that the design be the *best* or *optimal* based on the specified requirements. An optimal design should be a cost-effective system. To design such systems, certain analytical and numerical tools are needed. Optimal design concepts and methods provide some of the needed tools.

Existing optimal design methodologies focus on optimizing a single objective such as the cost or weight of the system subject to certain design constraints. While this is one possible way of achieving optimal designs, these approaches do not allow multiple design objectives. In addition, they are far too "rigid" as the constraints must be satisfied even though a "negligible" violation could mean a substantial improvement in the objective. Therefore, a new methodology is desirable which would allow specification of multiple design objectives about the system. Moreover, this methodology or framework should allow trade-off between different design objectives and be done in a systematic manner using digital computers.

To develop such a design framework, the following two issues have to be addressed:

1. Many design requirements are of qualitative nature. So a systematic method to characterize these requirements is needed which allows them to be traded-off

during the design process.

2. An optimal design process usually involves optimization over a large number of design parameters and traditional optimization techniques tend to perform poorly in such high-dimensional spaces.

1.2 Objectives

The overall goal of this research is to develop and implement a methodology that uses advanced computational techniques which allows the engineer to automate evaluation and improvement of a design. To achieve this, two specific objectives are desirable:

- Develop, investigate and implement a multicriterion optimal design framework.
- Apply genetic algorithms to solve the structural optimization problems formulated using the framework.

A multicriterion optimal design framework has recently been developed to address some of the issues related to automation of the design process (Beck, Papadimitriou, Chan, and Irfanoglu 1996). This framework features the use of preference functions to quantify qualitative and code requirements and a strategy to aggregate preferences of different criteria to get a single overall design evaluation measure. We will use this framework to solve optimal structural design problems.

An optimal design process then reduces to an optimization problem. To effectively solve these problems, especially in high-dimensional spaces, efficient and robust optimization techniques are required to obtain solutions. We will apply genetic algorithms to solve the consequent structural optimization problem. In particular, two special genetic algorithms are proposed to solve these optimal design problems.

1.3 Overview of This Dissertation

This thesis is organized into seven chapters. Below is a brief description of each of the remaining six chapters.

Chapter 2 is concerned with optimal design methodologies. A review of some of the existing optimal design methodologies such as optimality criteria and reliability-based optimal design is given. A brief discussion on discrete optimization in optimal design is also presented. This chapter continues with a description of a recently-developed multicriterion optimal design framework. Relations between this framework and other existing approaches, namely optimality criteria and concepts of other multicriterion optimization, are discussed.

In Chapter 3, a class of stochastic optimization methods called genetic algorithms (GA) are presented. Since these algorithms are mainly studied in computer science related fields and are relatively new to the structural engineering community, this chapter provides a basic overview of what genetic algorithms are, how they work and why they work. In addition, some issues of applying genetic algorithms are also discussed.

The discussion of genetic algorithms continues in Chapter 4 with a focus on two special classes of these algorithms: variable-length genetic algorithms and hybrid genetic algorithms. Here, it is explained why simple GAs may not work well for certain structural optimization problems. A special variable-length genetic algorithm (vGA) is proposed which addresses some of the difficulties. In addition, a hybrid genetic algorithm (hGA) is also presented which has a better convergence rate for continuous optimization problems than its simple counterparts. Examples are given which illustrate why these two proposed GAs are better.

In Chapter 5, a software application called CODA is presented. This program is a software prototype of the multicriterion framework described in Chapter 2. Functionalities of CODA, the theory behind all the computation, together with some of the implementation issues of CODA, are presented.

Chapter 6 consists of three example problems to illustrate the ideas and algorithms covered in the early chapters. The first example is a benchmark problem which has been studied extensively and provides a good basis for doing comparisons. The second example is a three-story frame building under UBC loading. A small study is conducted with different design parameters and design criteria. Several interesting

conclusions are drawn from this study. The final example is a truss tower with 72 truss members which are grouped into 16 design parameters. This is a relatively large problem compared to the other two examples. Issues concerning the applicability and efficiency of both the design framework as well as the proposed genetic algorithms are drawn.

Finally, a summary together with conclusions drawn from this thesis study is given in Chapter 7. This chapter ends with some recommendations for possible future research in areas of optimal design and genetic algorithms.

Chapter 2

Methodologies for Achieving Optimal Design

2.1 Optimal Design Concepts

Optimization is concerned with achieving the best possible solution to an objective while satisfying all specified requirements. In engineering design, the engineer strives to obtain designs that optimize cost, weight or certain other quantities. The formulation of an optimal design problem involves the identification of the following quantities: *design parameters, objective function* and *design constraints* or *design requirements.*

The first step of the formulation is to identify a set of parameters, the *design parameters*, which describe the system. Design parameters should be as independent of each other as possible. In most cases, different sets of design parameters can be used to describe the same system.

Design parameters may be continuous or discrete. Continuous design parameters have a range of values and can take on any value in the range. For instance, length of a beam may be taken as a continuous design parameter. Discrete design parameters can only take on isolated values, usually from a list of permissible ones. Member sizing of a beam selected from AISC steel sections is a good example of a discrete design parameter.

To be able to compare different designs, one must be able to distinguish one design

as being better than another. An *objective* or *cost function* provides a means of doing so. In many design problems, the objective function is the total weight of material required. A design problem can have many different objective functions but all the objective functions should be influenced by the design parameters.

Finally, every engineering system must be designed to satisfy the *design constraints* reflecting certain performance requirements or resource limitations. A design is not desirable if one or more of the design constraints are violated. All design constraints should be expressed in terms of design parameters. Constraints which impose lower or upper limits on certain quantities are called inequality constraints. A good example of an inequality constraint is the stress limit imposed on a component of a system. In some systems, equality constraints are required. A design constraint can be specified as a *design requirement* or *design criterion* although design criteria can be more general than just specifying constraints.

Once these quantities are identified, the whole design problem reduces to a numerical optimization problem such as the following:

$$\begin{aligned} &\text{Optimize} \quad f(\boldsymbol{\theta}) \\ &\text{such that} \quad g_i(\boldsymbol{\theta}) \geq 0, \ i = 1, ..., n_g \\ &\qquad\qquad h_j(\boldsymbol{\theta}) = 0, \ j = 1, ..., n_h \end{aligned}$$

where $\boldsymbol{\theta}$ denotes a vector of n design parameters, n_g and n_h are the number of inequality and equality constraints, respectively. The inequality constraints $g_i(\boldsymbol{\theta})$ and the equality constraints $h_j(\boldsymbol{\theta})$ are transformed into the form shown above.

For the case with no design constraints, the optimization problem is an unconstrained optimization problem. Otherwise, it is a constrained one. If there is only one objective function, then it is a single objective optimal design problem and most optimization techniques can be applied to the problem. For the case of two or more objectives, the problem is a multiobjective or multicriterion optimization problem. In this case, additional steps must be taken to solve such a problem.

In the next section, some of the most common optimal design methodologies are described. Since we will be focusing on multicriterion optimal design in this

dissertation, more attention will be given to the background and existing techniques in multicriterion optimal design.

2.2 Overview of Existing Optimal Design Methodologies

The following optimal design methodologies are discussed: optimality criteria methods including fully-stressed design, reliability-based approach, and multicriterion design optimization. In addition, other optimal design problems such as shape or topology optimization of engineering systems are described. This section ends with a brief overview of existing techniques for discrete optimization problems.

2.2.1 Optimality Criteria Methods

Optimality criteria methods (Haftka and Kamat 1985; Kirsch 1981) are based on the assumption that certain criteria related to the behavior of a design are satisfied at the optimum. These methods involve finding appropriate criteria for the specified design requirements and establishing an iterative procedure for finding the final optimal design. Typical design criteria involve an objective function together with constraints which are based on stresses and displacements. Other criteria can be related to buckling, nonlinear behavior, etc. A general formulation of optimality criteria for design optimization problems is given below.

Consider the following Lagrangian function:

$$L(\boldsymbol{\theta}, \Lambda) = \mathbf{f}(\boldsymbol{\theta}) + \sum_{\mathbf{i=1}}^{\mathbf{N_c}} \lambda_{\mathbf{i}} \mathbf{g_i}(\boldsymbol{\theta}) \tag{2.1}$$

where $f(\boldsymbol{\theta})$ is the objective function (usually the weight or cost of the system), $g_i(\boldsymbol{\theta})$ is the i^{th} inequality constraint among N_c of them and λ_i's are the Lagrange multipliers. Differentiate Equation 2.1 with respect to design parameters $\boldsymbol{\theta}$ and set the derivative

to zero to obtain the stationary conditions:

$$\begin{aligned} \mathbf{0} &= \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) + \sum_{i=1}^{N_c} \lambda_i \nabla_{\boldsymbol{\theta}} g_i(\boldsymbol{\theta}) \\ 0 &= -g_j + t_j^2 \qquad j = 1, ..., n_g \\ 0 &= 2\lambda_j t_j \qquad j = 1, ..., n_g \end{aligned} \tag{2.2}$$

Here, t_j is a slack variable which converts the j^{th} inequality constraint to an equality one:

$$g_j - t_j^2 = 0$$

This can be extended to include equality constraints by taking $t_j = 0$. Equation 2.2 is the necessary condition (Kuhn-Tucker criteria) for an optimal design and, therefore, provides n optimality criteria. Each Lagrange multiplier λ_i can be interpreted as a measure of sensitivity of the optimal design to the i^{th} constraint as a weighting factor that measures the importance of the i^{th} constraint to the optimal design. Note that for inequality constraints, the Lagrange multiplier λ_i is zero unless the constraint is active at the optimum.

With the conditions specified in Equation 2.2, a recursive algorithm is formulated and applied to solve the optimization problem. Usually, this algorithm involves redesign rules that modify the set of design parameters. For the i^{th} design parameter in the k^{th} iteration, the redesign rule can be written as:

$$\theta_i^{k+1} = \theta_i^k \cdot \mathcal{F}_i^k$$

where $\mathcal{F}_i^k$ is a multiplier for θ_i^k computed based on the optimality criteria.

While specification of the optimality criteria may be similar in formulation in different optimality criteria methods, the actual redesign rules and the numerical procedures employed usually differ greatly. For instance, stress criteria are often considered (see Fully Stressed Design below) and the numerical algorithms required

to compute these criteria are different than those for displacement criteria (often times with approximations specific to stress calculations to increase efficiency). In many cases, the optimal design is achieved by this approach. However, there are also cases where the obtained designs are not optimal even though they may be reasonable designs and not too far from optimal. One such situation is mentioned in the following discussion of fully stressed design.

Fully Stressed Design

The fully stressed design (FSD) technique is probably the most well-known and successful optimality criteria method in the literature. It has motivated much of the interest in optimality criteria methods. The FSD method can be applied to systems which are subject only to stress constraints.

This method is based on the following optimality criterion (Haftka and Kamat 1985):

> *For the optimum design, each member of the structure is (either) fully stressed at least under one of the design load conditions or (or) is at its minimum specified gage.*

The redesign rule of the i^{th} design parameter θ_i in the k^{th} iteration for FSD in the case of truss members can be written as:

$$\theta_i^{k+1} = \theta_i^k \cdot \frac{\sigma_i^k}{\sigma_{i,a}}$$

where σ_i^k denotes the actual stress in the members associated with design parameter θ_i in the k^{th} iteration and $\sigma_{i,a}$ denotes the allowable stress for the i^{th} design parameter.

The significance of this method is:

1. Engineering experience indicates that a good design is usually one in which each member is subjected to its allowable stress.

2. FSD can be proved to be optimal under certain conditions.

3. FSD methods are fairly efficient computationally compared to other approaches.

4. FSD is usually a good starting point for other optimal design methods.

For statically determinate structures with a single load condition, it has been shown that this FSD criterion yields optimal designs which are the minimal weight designs (Cilley 1900; Michell 1904). However, for statically indeterminate structures under multiple load conditions the optimum may not be fully stressed (Razani 1965; Schmit 1960). One reason for this is that an FSD is not unique. An indeterminate structure may have more than one FSD and there is no guarantee that an FSD will always converge to the minimum weight FSD.

Other Optimality Criteria Methods

Besides fully stressed design, there is an extensive amount of publications on other optimality criteria methods. For instance, Rozvany and Zhou (1992) introduced discretized continuum-based optimality criteria methods (DCOC) for large finite element systems with several deflection and stress constraints. For structural optimization of tall steel buildings, several optimality criteria methods have been proposed during the last few years (Soegiarso and Adeli 1996; Chan, Grierson, and Sherbourne 1995; Chan 1992) . One of the main reasons for the popularity of optimality criteria methods is that the computational effort is primarily dependent on the number of active constraints and only weakly dependent on the number of design parameters. Since most large scale systems have a large number of design parameters and far fewer design criteria, optimality criteria methods are quite efficient for solving such problems.

2.2.2 Reliability-Based Optimal Design

The optimal design methodologies described so far all assume no uncertainty in every aspect. Using these approaches, optimal designs are found which minimize the cost functions without violating any design requirements. However, such designs normally leave little redundancy compared with designs that are obtained by using more conservative approaches. Thus, such optimally-designed systems usually have higher failure

probabilities than unoptimized systems. To achieve a balance of safety and economy, reliability-based design concepts can be introduced into design optimization.

Reliability-based design optimization covers a lot of different areas: design code optimization (Moses 1989), component reliability-based optimization (Moses 1974; Moses 1990), system reliability-based optimization (Parimi and Cohn 1978; Frangopol 1987), multiobjective reliability-based optimization (Frangopol 1991; Beck et al. 1996) as well as damage and residual-oriented reliability-based optimization (Frangopol and Moses 1994). Since in this study only reliability at the component level is considered, only this topic is discussed here.

Component Reliability-Based Design Optimization

In this approach, design requirements and criteria are specified for individual elements such as beams, columns and connections of a structural system (Moses 1974). This ensures that safety requirements for individual component are satisfied. Usually, the design procedures involve achieving a certain target element reliability index which can be computed from the probability of failure of the components (Moses 1990).

There are various ways to formulate element reliability-based optimization problems. Most of them can be formulated as minimization of the total expected cost of the element:

$$C_{el} = C_{o,el} + C_{f,el}, \tag{2.3}$$

where C_{el} = total expected cost of the element over lifetime

$C_{o,el}$ = initial cost of the element

$C_{f,el}$ = expected cost of failure of element.

The term $C_{f,el}$ is usually expressed as a function of some probability of occurrence of the dominant element limit state (or collapse). Examples of $C_{f,el}$ are:

1. $C_{f,el} = C_{fd,el} P_{fd,el}$ (Moses 1974)

where $C_{fd,el}$ = cost of element failure due to occurrence of dominant limit state

$P_{fd,el}$ = probability of occurrence of the dominant limit state.

2. $C_{f,el} = C_{f1,el}P_{f1,el} + C_{f2,el}P_{f2,el}$

where $C_{f1,el}, C_{f2,el}$ = cost of element failure due to occurrence of dominant ultimate and serviceability limit states

$P_{f1,el}, P_{f2,el}$ = probability of occurrence of the dominant ultimate and serviceability limit states.

3. $C_{f,el} = \sum_{j=1}^{m} C_{fj,el}P_{fj,el},\ j = 1, 2, ..., m$

where $C_{fj,el}$ = cost of element failure due to occurrence of j^{th} limit state

$P_{fj,el}$ = probability of occurrence of the j^{th} limit state

m = total number of limit states of the element.

Besides minimizing the total expected cost, one can also minimize the probability of failure. One such formulation for an element reliability-based optimization problem is to minimize the probability of occurrence of a specified limit state $P_{fk,el}$.

2.2.3 Multicriterion Design Optimization

In all the methodologies discussed up to this point, the optimal design problems reduce to scalar-valued objective functions subject to constraints. In most cases, these functions are the weight of the system. However, in many real-life design problems, there are several conflicting and noncommensurable criteria and these single objective approaches do not apply in these cases. Multicriterion design optimization (Eschenauer, Koski, and Osyczka 1990; Stadler 1988) provides a flexible and systematic way to handle these design problems. It is also known as Pareto optimization, vector optimization or multiobjective optimization.

Using a multicriterion approach in optimal design has certain advantages. First, optimal design involving multiple criteria allows a more realistic description or modeling of design decision making since most problems in real life are normally composed

of multiple criteria (in most cases, conflicting). In addition, with multiple criteria, a wider range of alternative designs are usually available.

A general multicriterion optimization problem can be formulated as follows:

$$\begin{aligned} &\text{Optimize} \quad f(\boldsymbol{\theta}) = [f_1(\boldsymbol{\theta}), f_2(\boldsymbol{\theta}), ..., f_{N_c}(\boldsymbol{\theta})]^T \\ &\text{such that} \quad g_i(\boldsymbol{\theta}) \geq 0, \ i = 1, ..., n_g \\ &\qquad\qquad\quad h_j(\boldsymbol{\theta}) = 0, \ j = 1, ..., n_h, \end{aligned}$$

where $\boldsymbol{\theta}$ denotes a vector of n design parameters, $f(\boldsymbol{\theta})$ is the multicriterion objective function, $f_i(\boldsymbol{\theta})$ is the i^{th} criterion of the N_c individual criteria, and n_g and n_h are the number of inequality and equality constraints, $g_i(\boldsymbol{\theta})$ and $h_i(\boldsymbol{\theta})$, respectively.

Solving a multicriterion design optimization problem requires identification of the Pareto set which is the set of points in the design space that are *Pareto optimal* or *nondominated.* A feasible solution $\hat{\boldsymbol{\theta}}$ is a Pareto optimal or nondominated solution if there exists no feasible solution $\boldsymbol{\theta}$ such that

$$\begin{aligned} &\mu_i(\boldsymbol{\theta}) > \mu_i(\hat{\boldsymbol{\theta}}) \text{ for some } i\text{=}1,..,N_c, \text{ and} \\ &\mu_j(\boldsymbol{\theta}) \geq \mu_j(\hat{\boldsymbol{\theta}}) \text{ for all } j \end{aligned}$$

Simply put, a feasible solution is *Pareto optimal* if there exists no other feasible solution that will improve one criterion without causing a decrease in at least one other criterion.

Several methods exist to solve these multicriterion optimization problems such as linear weighting method (Koski 1985), constraint method (Koski and Silvennoinen 1987) and simplex method (Balachandran 1996). Generally the difficulty is not with the non-existence of Pareto optima but rather the large number of these points, which may be hard to identify and handle. Multicriterion optimization has been applied to several structural optimization problems (Stadler 1988; Leitmann 1977).

2.2.4 Other Optimal Design Problems

Up to this point, we have only considered systems that have fixed configurations. When the shape of the system is allowed to change, usually a better design can be

found. In shape optimal design problems (Budiman and Rajan 1993), the shape of the system is defined by parameters which can vary. For example, nodal coordinates of a structure can be taken as design parameters for the problem. One interesting point is that even a few design parameters relating to the structural shape can result in dramatic changes in the shape.

A more general problem than shape optimization is topology optimization. For instance, the topology of a structural system means not only where the nodes are, but also how many nodes there are and how the system is supported. The choice of member connectivity and support conditions as design parameters results in a design space that is both nonconvex and discrete. Thus, traditional gradient-based optimization cannot be applied to these problems. Integer programming and random search techniques are suitable for such problems, as well as genetic algorithms discussed later.

Recently, conceptual design has been formulated as an optimal design problem (Grierson 1997). Another emerging approach in optimal design research is multidisciplinary optimization (Kroo 1995; Braun and Kroo 1995; Wakayama and Kroo 1994). Multidisciplinary optimization is a way of finding the "best" solution or design, given an objective and a set of constraints, where both the objective and the constraints together with the design variables come from a knowledge base representing many different disciplines such as quality, cost, value, etc.

There are two main categories of multidisciplinary optimization. The first category, which is also the most recognized one, is quantitative in nature. In this category, the objective and the constraints are stated in mathematical form and the optimal solution is determined using a numerical or analytical approach to the problem. The second category is qualitative in nature. It is sometimes referred to as "experimental" or "qualitative" optimization since it comes from the experience of those doing the optimization.

2.2.5 Discrete Optimization in Optimal Design

Many design optimization methods assume that design parameters are continuous. However, often the components of systems are only available in discrete sizings and a simple roundup from the continuous values would most likely result in quite sub-optimal designs. For instance, optimal design of structural steel buildings are often performed using continuous design parameters on member sections and the continuous solutions are then rounded up to the "closest" sections which are often too conservative. Also, in some cases, rounded-up solutions may actually be in the infeasible region. Therefore, discrete optimization approaches should be employed to achieve better designs in the discrete design space.

Several discrete optimization techniques exist in the literature. One of the most popular methods is branch and bound. The solution process involves finding the continuous optimization and then going through a tree (the branches) to seek the closest discrete solution to the continuous counterpart (bounding). The main disadvantage of this technique is that it requires high computational effort.

Simulated annealing is another technique for discrete optimization. Unlike branch and bound, simulated annealing is a stochastic method. It operates by randomly perturbing the solution to generate candidate solutions that are either accepted or rejected. Infeasible candidate solutions are automatically rejected and all feasible solutions are accepted even for those with poor objective function values. The idea is that poor solutions are accepted in the earlier stages but that in the later stages, the method will converge to the global optimum. One important note is that simulated annealing normally requires a large number of objective function evaluations which can be prohibitive if the function evaluation is computationally expensive. However, this approach has a reasonable chance of finding the global optimum for some nonconvex problems (Thanedar and Vanderplaats 1992).

2.3 The Multicriterion Optimal Design Framework

One of the objectives of this research is to develop a design methodology which can assist the designer to make decisions throughout the process of designing an engineering system. In this section, a recently developed multicriterion optimal design methodology is presented (Beck, Papadimitriou, Chan, and Irfanoglu 1996; Beck, Chan, Irfanoglu, Masri, Smith, Vance, and Barroso 1996).

2.3.1 Overview of the Framework

The whole optimal design process involves making decisions related to the design of a given system. This decision making process begins with a preliminary design and then involves an iterative procedure of analysis, evaluation and revision. Figure 2.1 shows the overview of the optimal design framework.

In this framework, there are three modules: ANALYZER, EVALUATOR and REVISER. These modules are responsible for performing analysis, evaluation and revision of the current design respectively. Before we can describe each of these modules, a few basic terms need to be defined and reviewed, since the point of view taken is different to the traditional approaches to optimal structural design described earlier.

2.3.2 Terminology

For a given system, the design space consists of all the possible designs this system can have. This design space can be expressed in terms of design parameters, denoted as $\boldsymbol{\theta}$. Design parameters are quantities the decision maker uses to specify a particular design. These parameters can be very general. For instance, they can be related to the structural configuration, total material cost, component sizing, etc., and even material type.

Performance parameters, denoted as $\mathbf{q}(\boldsymbol{\theta})$, are quantities involved in the evaluation of a design. These parameters are usually engineering quantities such as the maximum

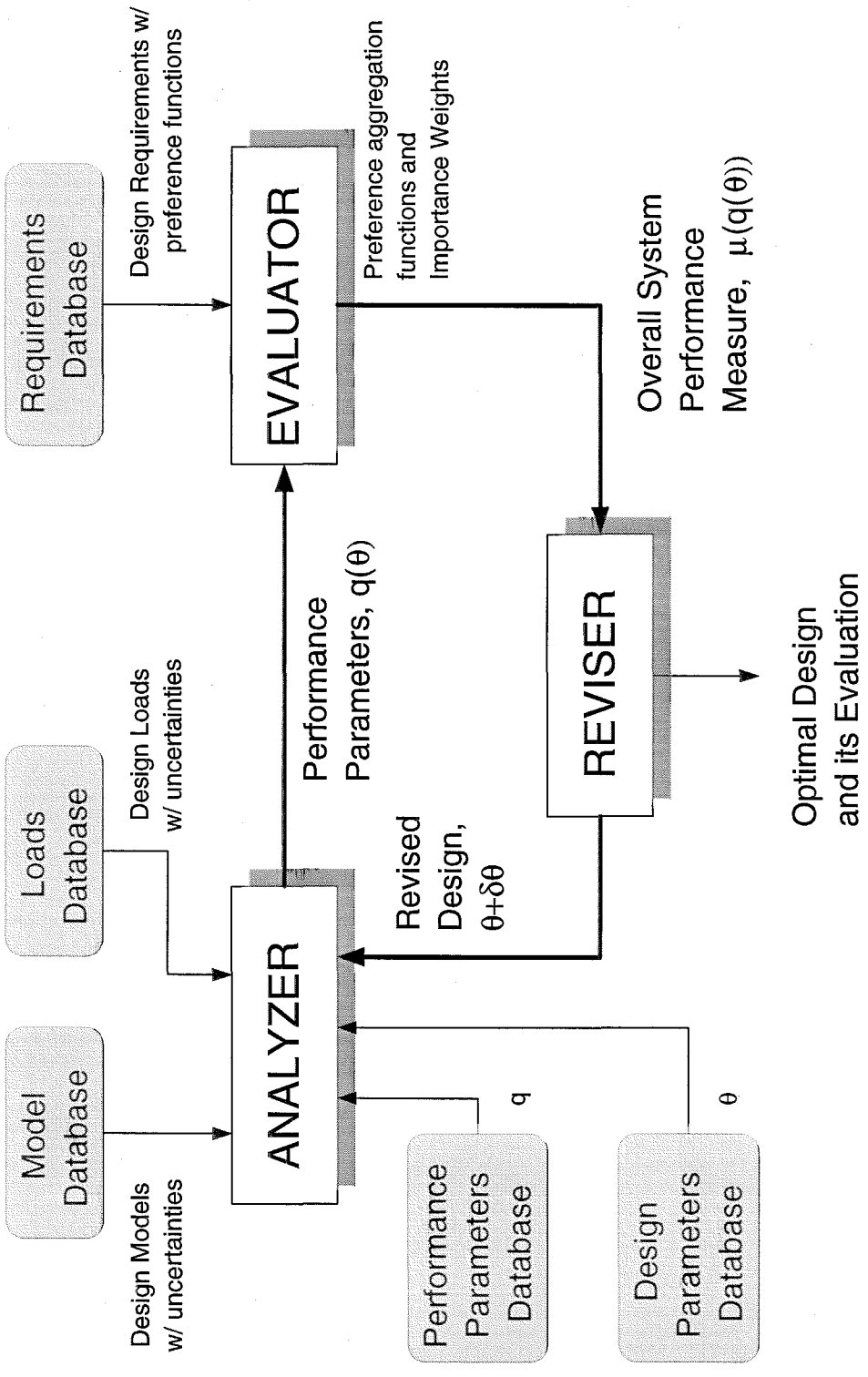


Figure 2.1: Overview of the proposed multicriterion optimal design framework

interstory drift or the peak member stresses of a structure. However, they can also include quantities such as the total material cost, liability should the current design fail, etc. Moreover, performance parameters may also include some of the design parameters. For instance, the geometrical configuration of a structural system can be specified as both design and performance parameters if there are any architectural or manufacturing constraints on the shape or sizing of the members.

Finally, a particular design can be judged by specifying a list of design criteria. These criteria are usually design requirements that need to be met for any acceptable design. Such requirements or restrictions can relate to the allowable stress in any member of a structure, the budget for the total construction cost, etc.

2.3.3 Description of the Framework Modules

As mentioned earlier, there are three modules in the framework: ANALYZER, EVALUATOR and REVISER. Each of the three modules has its unique role and responsibilities. The following is a detailed description of each of these modules and their functionalities.

ANALYZER

The role of ANALYZER is to compute performance parameter values $\mathbf{q}(\boldsymbol{\theta})$ based on the specified design parameters $\boldsymbol{\theta}$ (see Figure 2.2). Thus, ANALYZER involves different analysis techniques which depend on the performance parameters specified and the nature of the problem at hand. Examples of possible methods for ANALYZER are finite element methods for structural or mechanical systems, time history integration schemes for dynamic analyses, probabilistic analysis tools for random vibration and reliability analysis, and costing algorithms for economic calculations. For design of civil engineering structures, ANALYZER would usually be a combination of static and dynamic finite element analysis, as well as algorithms to compute material and construction costs.

It is clear that the nature of ANALYZER is highly dependent on the type of

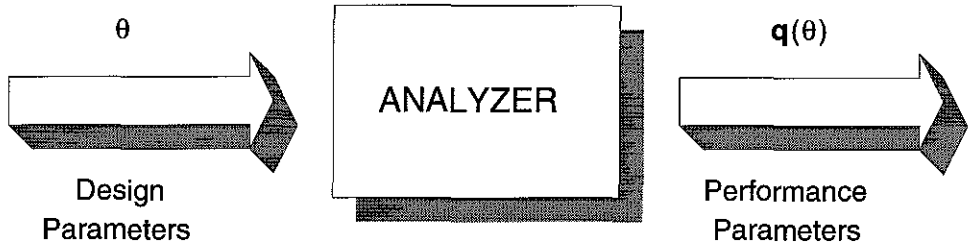


Figure 2.2: ANALYZER as a blackbox

systems to be designed and the specified performance parameters. However, the role of ANALYZER is the same for different problems: calculate performance parameters $\mathbf{q}(\boldsymbol{\theta})$ from design parameters $\boldsymbol{\theta}$.

EVALUATOR

The task of EVALUATOR is to provide an overall design evaluation measure $\mu(\boldsymbol{\theta})$ for the design specified by the current values of the design parameters $\boldsymbol{\theta}$ (see Figure 2.3). The measure $\mu(\boldsymbol{\theta})$ serves as an objective function which REVISER uses to improve the current design as well as seek the optimum.

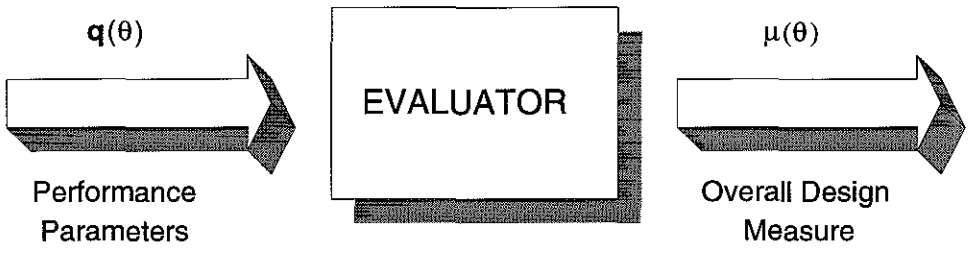


Figure 2.3: EVALUATOR as a blackbox

To evaluate a particular design, the designer may impose multiple design criteria. Under this multicriterion decision methodology, the design is first evaluated on the basis of each design criterion, one at a time, and then these numerical values are aggregated into a single design evaluation measure using certain aggregation strategies. Furthermore, since the individual design criteria cannot usually all be satisfied in an optimal fashion at the same time, trade-off is allowed among the criteria to the degree that each of the criteria is satisfied.

A preference function $\mu_i(\mathbf{q})$ is used to quantify the degree of satisfaction of i^{th} design criterion based on the values of the performance parameters $\mathbf{q}$ involved in the design criterion. Values of the preference function must lie in the unit interval [0,1]. A larger preference value for one performance parameter value compared with another implies that the first parameter value is more preferable than the other value. An extreme value $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 1$, or $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 0$, implies that the current design specified by $\boldsymbol{\theta}$ perfectly satisfies, or does not satisfy at all, the i^{th} design criterion. For example, Figure 2.4 shows a preference function for the design criterion that the maximum interstory drift not exceed some code prescribed value. In this case, those values of the maximum interstory drift which are less than 90% of the code specified drift value are most preferred, since the preference function has its greatest possible value (unity) there. On the other hand, the designer prefers least those values of the maximum interstory drift which exceed the code specified drift value, since the preference function has its least possible value (zero) there. The designer has selected a linear fall-off between these extreme preference values for those values of the maximum interstory drift which lie between 90% and 100% of the code specified drift value.

Another interpretation is to view the preference function as a membership function for the fuzzy set of "acceptable performance" as judged by the i^{th} design criterion. In this case, an extreme value $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 1$, or $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 0$, implies that on the basis of the i^{th} design criterion, the current design specified by $\boldsymbol{\theta}$ is definitely acceptable, or definitely unacceptable. Intermediate values express the degree to which the user feels the design gives "acceptable performance."

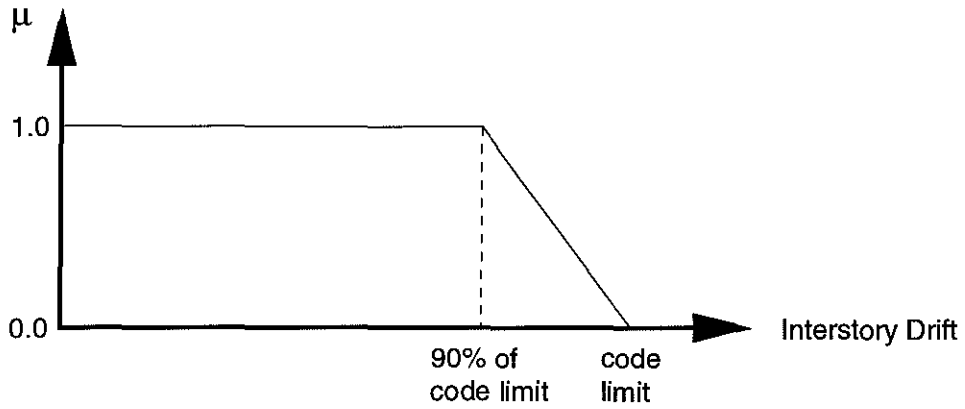


Figure 2.4: Example of one type of preference function

Any constraints directly imposed on the design parameters, such as geometrical constraints, are treated as additional design criteria. Each such criterion is expressed as a "soft" constraint through a preference function. For example, a preference function similar to the one shown in Figure 2.4 can be used to express a "soft" upper bound on a design parameter. If the designer also wishes to impose a lower bound on the parameter, then the two-sided version of the preference function shown in Figure 2.5 can be used. By treating design parameter constraints in this way, the degree to which the constraint is satisfied can be traded off against other design criteria during the optimization of the design.

The final step in the EVALUATOR methodology is to compute an overall design evaluation measure $\mu(\boldsymbol{\theta})$ on the basis of the quantitative evaluations $\mu_i(\mathbf{q}(\boldsymbol{\theta}))$, $i = 1, ..., N_c$ of the design for each of the N_c design criteria. This is done by a preference aggregation rule which must satisfy certain consistency requirements. Different aggregation rules give different design strategies for trading off the design criteria, and so they lead to different optimal designs, in general. Also, for a given aggregation rule, the user can give more influence to some design criteria than others by assigning them larger values of an importance weight. The choice of the values for these weights is subjective. The user can gain experience with respect to their selection in any design problem by investigating the influence that different values for the

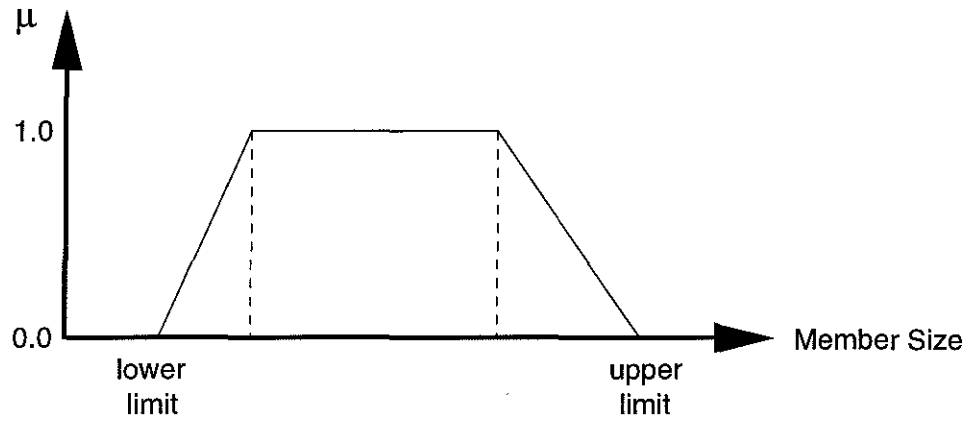


Figure 2.5: A preference function for specifying constraints on design parameters

weights have on the final optimal design and its corresponding preference values for each design criterion.

A preference aggregation rule is simply a functional relationship between the overall design evaluation measure and the individual preference values for all of the design criteria: $\mu = f(\mu_1, \mu_2, ..., \mu_{N_c})$. An optimal design for a given preference aggregation rule is therefore given by a design parameter vector $\boldsymbol{\theta}$ which maximizes:

$$\mu(\boldsymbol{\theta}) = f(\mu_1(\mathbf{q}(\boldsymbol{\theta})), \mu_2(\mathbf{q}(\boldsymbol{\theta})), ..., \mu_{N_c}(\mathbf{q}(\boldsymbol{\theta}))) \tag{2.4}$$

where it is understood, despite the notation here, that some of the preference functions μ_i may correspond to design parameter constraints and therefore these μ_i will depend directly on the design parameter values.

The following axioms of consistency are imposed on the preference aggregation rule (Otto 1992):

1. μ lies in the unit interval $[0, 1]$, with $\mu = 1$ for a perfectly acceptable design and $\mu = 0$ for a completely unacceptable design.

2. μ is a monotonically increasing continuous function of each μ_i.

3. $\mu_0 = f(\mu_0, \mu_0, ..., \mu_0)$, where μ_0 is some value between 0 and 1.

4. $\mu = 0$ if and only if $\mu_i = 0$ for some i.

Axiom 1 allows the overall design measure $\mu(\boldsymbol{\theta})$ to have the same scale, $\mu \in [0, 1]$, as the individual preference values $\mu_i(\boldsymbol{\theta})$. The continuity requirement in Axiom 2 ensures that a small change in preference in one or more of the design criteria results only in a small change in the overall design measure. In addition, monotonicity guarantees that any improvement in one or more of the criteria yields an improvement in the overall design and vice versa. Axiom 3 expresses the following argument: If all the preference values μ_i of the design criteria are equal to μ_o, then the overall design measure μ should also be μ_o since it would not be rational to give it a higher or lower preference. Finally, axiom 4 ensures that if any of the design criteria are not satisfied, i.e. $\mu_i = 0$, for some i, then the design is not acceptable ($\mu = 0$). Similarly, the design is unacceptable only if at least one of the design criteria is not satisfied.

Various aggregation rules exist which satisfy these axioms. Two such preference aggregation rules are:

- Conservative ("weakest link") strategy:

$$\mu = \min({\mu_1}^{n_1}, {\mu_2}^{n_2}, ..., {\mu_{N_c}}^{n_{N_c}}), \tag{2.5}$$

  where $n_i = w_i / \max_j w_j$, $i = 1, ..., N_c$ and w_i is a positive importance weight assigned to the i^{th} design criterion.

- Multiplicative trade-off strategy:

$$\mu = {\mu_1}^{m_1} {\mu_2}^{m_2} ... {\mu_{N_c}}^{m_{N_c}}, \tag{2.6}$$

  where $m_i = w_i / \sum_{j=1}^{N_c} w_j$, $i = 1, ..., N_c$ and w_i is a positive importance weight assigned to the i^{th} design criterion.

Handling of Stochastic Design Criteria

In order to be able to trade-off reliability of performance and cost of a design in the design process, the uncertainties in performance parameters due to the uncertainties in system models and uncertainties in loadings must be considered. These uncertainties can be the most influential factors in the design decisions. The proposed optimal design framework can be extended to treat these uncertainties. This extension was developed by Beck, Papadimitriou, Chan and Irfanoglu (1996) and is included here for completeness.

In the stochastic case, there is no longer a function $\mathbf{q}(\boldsymbol{\theta})$ relating the design parameters $\boldsymbol{\theta}$ to all the performance parameters as assumed in the earlier description of the methodology. Some of the performance parameters will be uncertain because of the uncertain loads and modeling errors. For example, one of the performance parameters q_i may be the peak interstory drift over the lifetime of the structure due to earthquakes, which is clearly a very uncertain quantity. Parametric uncertainties are conveniently modeled by random variables. The probability distribution assigned to each random variable specifies the relative plausibility of each possible value of the corresponding uncertain parameter. Similarly, uncertain continuous-valued variables are modeled by random fields or random processes. This probabilistic description of loads and/or system model necessitates the use of probability tools to calculate the uncertain performance parameters. Therefore, in this stochastic case, a probability density function $p(q_i|\boldsymbol{\theta})$ is calculated rather than the value of the performance parameter q_i.

Performance parameters such as the manufacturing cost or structural performance parameters from code-based design loads, can be treated as deterministic functions of $\boldsymbol{\theta}$. They can be interpreted within the general stochastic framework by simply viewing the corresponding probability distributions $p(q_i|\boldsymbol{\theta})$ as delta functions centered at $q_i(\boldsymbol{\theta})$.

A measure of the reliability of the design $\boldsymbol{\theta}$, as judged by the i^{th} design criterion, is the probability that this criterion is satisfied. Since the preference function $\mu_i(q_i)$

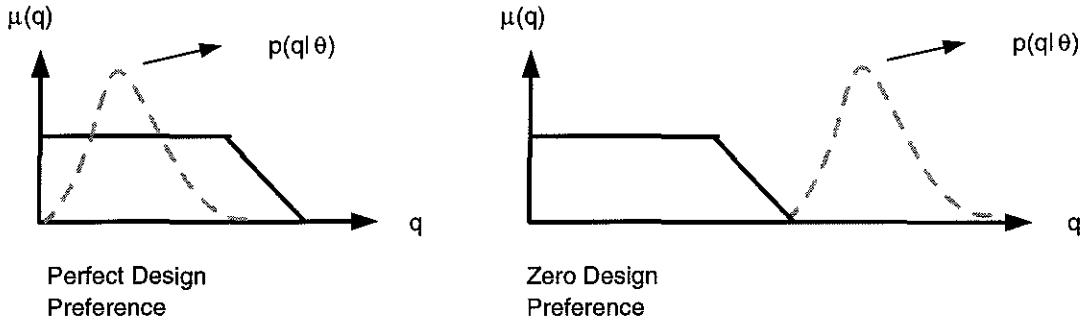


Figure 2.6: Graphical interpretation of mean preference

for the i^{th} design criterion can also be viewed as a membership function for the fuzzy set "acceptable performance" based on this criterion, the desired reliability is the probability that q_i lies in this fuzzy set:

$$\overline{\mu}_i(\boldsymbol{\theta}) = \int_0^\infty \mu_i(q_i)\ p(q_i|\boldsymbol{\theta})\ dq_i. \tag{2.7}$$

This measure is also seen to be equivalent to the mean preference value for the i^{th} design criterion. Figure 2.6 shows graphically the interpretation of the two extremes of mean preference value of a typical design criterion. In the special case of no uncertainties, for which $p(q_i|\boldsymbol{\theta})$ is taken as a delta function, $\overline{\mu}_i(\boldsymbol{\theta}) = \mu_i(q_i(\boldsymbol{\theta}))$, and so the deterministic case described earlier is recovered.

Using integration by parts, Equation (2.7) gives

$$\overline{\mu}_i(\boldsymbol{\theta}) = -\int_0^\infty \frac{d(\mu_i(q_i))}{dq_i}\ F_i(q_i \mid \boldsymbol{\theta}) dq_i \tag{2.8}$$

where

$$F_i(\widehat{q} \mid \boldsymbol{\theta}) = P(q_i \leq \widehat{q} \mid \boldsymbol{\theta}) = \int_0^{\widehat{q}} p(q_i \mid \boldsymbol{\theta})\, dq_i \tag{2.9}$$

is the classical reliability function for the performance parameter q_i given $\boldsymbol{\theta}$. Using, for example, the preference function for the peak lifetime interstory drift shown in Figure 2.4, denoting the code value by u_i and 90% of the code value by ℓ_i, Equation (2.8) yields

$$\overline{\mu}_i(\boldsymbol{\theta}) = \frac{1}{u_i - \ell_i} \int_{\ell_i}^{u_i} F_i(q_i \mid \boldsymbol{\theta}) dq_i \tag{2.10}$$

which is the average value of the classical reliability over the interval $[\ell_i, u_i]$. Clearly, a high mean preference value $\overline{\mu}_i(\boldsymbol{\theta})$ means that the design $\boldsymbol{\theta}$ has a high fuzzy reliability, or, equivalently, a high average classical reliability, as judged by the i^{th} design criterion.

To generalize the deterministic optimal design methodology described earlier, all that remains is to replace each $\mu_i(\boldsymbol{\theta})$ corresponding to a stochastic design criterion by $\overline{\mu}_i(\boldsymbol{\theta})$ in the preference aggregation rule (2.6). The evaluation of $\overline{\mu}_i(\boldsymbol{\theta})$ depends on the choice of the user-supplied preference function $\mu_i(q_i)$ for the i^{th} design criterion, and either the probability density function $p(q_i|\boldsymbol{\theta})$ or the reliability function $F_i(\widehat{q} \mid \boldsymbol{\theta})$ (see Equation 2.7, 2.8).

REVISER

Given the current design $\boldsymbol{\theta}$ and its design measure $\mu(\mathbf{q}(\boldsymbol{\theta}))$, the role of REVISER is to improve this design based on the specified design criteria (see Figure 2.7).

Similar to ANALYZER, REVISER is problem dependent although to a lesser extent. Although REVISER would be some optimization technique in most problems, the exact method to be employed varies depending on the nature of the design parameters of a problem. Examples of possible methods for REVISER are quasi-Newton

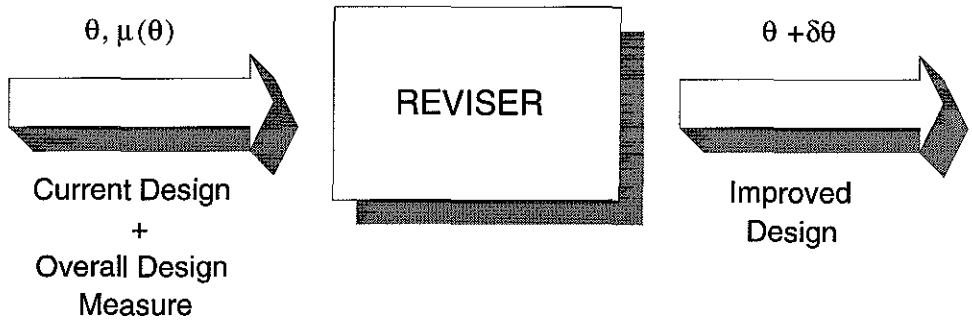


Figure 2.7: REVISER as a blackbox

methods for continuous design parameters, stochastic optimization for both continuous and discrete parameters, integer programming methods for discrete variables and combinatorial schemes for shape and topological design parameters.

2.3.4 Discussion of the Optimal Design Framework

The optimal design framework just presented provides a flexible scheme for the designer to formulate a design problem with several design criteria and be able to make design decisions allowing trade-offs among these criteria in a systematic manner. In this section, we will look at how this framework is related to existing concepts of multicriterion optimization and also how a special case of this methodology reduces to an optimality criteria problem which would yield a minimum weight design.

Relation to Pareto Optimal Set in Multicriterion Optimization

Recall that a solution or design $\hat{\boldsymbol{\theta}}$ is *nondominated* or *Pareto optimal* if there exists no feasible solution or design $\boldsymbol{\theta} \in \boldsymbol{\Theta}$, the set of all feasible designs, such that

$$\mu_i(\boldsymbol{\theta}) > \mu_i(\hat{\boldsymbol{\theta}}) \text{ for some } i=1,..,N_c, \text{ and}$$
$$\mu_j(\boldsymbol{\theta}) \geq \mu_j(\hat{\boldsymbol{\theta}}) \text{ for all } j$$

where N_c is the number of design criteria.

Suppose we find an optimal design $\hat{\boldsymbol{\theta}}$ using the proposed multicriterion optimal design framework. We have

$$\mu(\boldsymbol{\theta}) \leq \mu(\hat{\boldsymbol{\theta}}), \forall \boldsymbol{\theta} \in \boldsymbol{\Theta}. \tag{2.11}$$

The equality represents the cases when we have multiple optimal designs. We wish to show that the optimal design obtained from this framework lies in the Pareto optimal set defined by all the design criteria.

Suppose the contrary, that is, we can find a design $\boldsymbol{\theta}$ such that $\hat{\boldsymbol{\theta}}$ is dominated:

$$\mu_i(\boldsymbol{\theta}) \geq \mu_i(\hat{\boldsymbol{\theta}}),\ 1 \leq i \leq N_c$$

with inequality holding for at least one i. By the axioms of consistency imposed on the aggregation strategy discussed earlier, we have

$$\mu(\boldsymbol{\theta}) > \mu(\hat{\boldsymbol{\theta}})$$

which is a contradiction. Thus, $\hat{\boldsymbol{\theta}}$ is a nondominated or Pareto optimal solution. However, this inequality is true only if we have strict monotonicity in the aggregation strategy (see Axiom 2). The trade-off strategy satisfies this condition but the conservative strategy does not. It can be seen, however, that the result also holds for the conservative strategy if there exists only a single optimum, that is, inequality holds in Equation (2.11).

Note that this result is true regardless of the choice of importance weights of the design criteria, $\boldsymbol{w} = [w_1, ...w_{N_c}]^T$. Different values of importance weight $\boldsymbol{w}$ yield different optimal designs $\hat{\boldsymbol{\theta}}$ and hence, different Pareto optima. Thus, one can obtain a subset of the set of all Pareto optimal solutions $\mathcal{P}$ by varying the importance weights. For some convex problems, this subset may actually be the whole set of Pareto optimal solutions $\mathcal{P}$. However, for nonconvex problems, the subset is a proper subset of $\mathcal{P}$.

Relation to Optimality Criteria Design (Minimal Weight)

Consider the trade-off strategy with N_c design criteria, we have

$$\mu(\boldsymbol{\theta}) = \prod_{i=1}^{N_c} \mu_i^{n_i}(\mathbf{q}(\boldsymbol{\theta}))$$

where $n_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j}$, and w_i is the importance weights of the i^{th} design criterion. Taking the logarithm, we have

$$\ln \mu(\boldsymbol{\theta}) = \frac{1}{\sum_{j=1}^{N_c} w_j} \sum_{i=1}^{N_c} w_i \ln \mu_i(\mathbf{q}(\boldsymbol{\theta}))$$

which is a weighted average of the logarithms of the preference functions of each of the design criteria. Clearly, in maximizing μ, or $\ln \mu$, with respect to $\boldsymbol{\theta}$, the only design criteria "active" at each step are those for which $\mu_i < 1$.

At an extremum, $0 = \frac{\partial \ln \mu}{\partial \theta_k}$, so

$$0 = \sum_{i=1, \mu_i<1}^{N_c} w_i \frac{1}{\mu_i} \frac{\partial \mu_i}{\partial \theta_k}, \ \forall k. \tag{2.12}$$

Let μ_1 correspond to low steel volume and the preference function (see Figure 2.8a) is:

$$\mu_1(\boldsymbol{\theta}) = \frac{V_{max} - V(\boldsymbol{\theta})}{V_{max} - V_{min}}, \ V_{min} \leq V(\boldsymbol{\theta}) \leq V_{max}$$

where V_{min} and V_{max} are the minimum and maximum allowable steel volumes, respectively. Thus, we have

$$\frac{1}{\mu_1} \frac{\partial \mu_1}{\partial \theta_k} = -\frac{1}{V_{max} - V} \frac{\partial V}{\partial \theta_k}.$$

Suppose all other μ_i are functions of only one performance parameter $q_i(\boldsymbol{\theta})$ in the

form shown in Figure 2.8b, or mathematically,

$$\mu_i = \begin{cases} 1, & 0 \le q_i \le q_i^{(l)} \\ 1 - \frac{q_i - q_i^{(l)}}{q_i^{(u)} - q_i^{(l)}}, & q_i^{(l)} \le q_i \le q_i^{(u)} \\ 0, & q_i^{(u)} \le q_i \end{cases} .$$

Differentiate this expression and multiply with $\frac{1}{\mu_i}$ yields

$$\frac{1}{\mu_i}\frac{\partial \mu_i}{\partial \theta_k} = \begin{cases} 0, & 0 \le q_i \le q_i^{(l)} \\ -\frac{1}{q_i^{(u)} - q_i}\frac{\partial q_i}{\partial \theta_k}, & q_i^{(l)} \le q_i \le q_i^{(u)} \\ 0, & q_i^{(u)} \le q_i \end{cases} .$$

Substitute this into Equation 2.12, we have for $q_i < q_i^{(u)}$, $\forall i$:

$$0 = -w_1 \frac{1}{V_{max} - V}\frac{\partial V}{\partial \theta_k} - \sum_{i=2}^{N_c} w_i \frac{1}{q_i^{(u)} - q_i}\frac{\partial q_i}{\partial \theta_k} H(q_i - q_i^{(l)}), \ \forall k, \tag{2.13}$$

where

$$H(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$

is the Heaviside function and picks up the "active" constraints, i.e. those with $q_i \in (q_i^{(l)}, q_i^{(u)})$.

If $q_j = \theta_k$, i.e. μ_j is a soft constraint on the design parameter θ_k with preference function similar to Figure 2.8c, then the j^{th} design criterion term in the sum in Equation 2.13 for $\theta_k^{(l)} < \theta_k < \theta_k^{(u)}$ is

$$w_j \frac{1}{(\theta_k^{(u)} - \theta_k)(\theta_k - \theta_k^{(l)})} max\{H(\theta_k^{(l')} - \theta_k), H(\theta_k - \theta_k^{(u')})\},$$

which can be viewed as a penalty term which gets larger as the hard constraints $\theta_k = \theta_k^{(l)}$ or $\theta_k = \theta_k^{(u)}$ (see Figure 2.8) are approached.

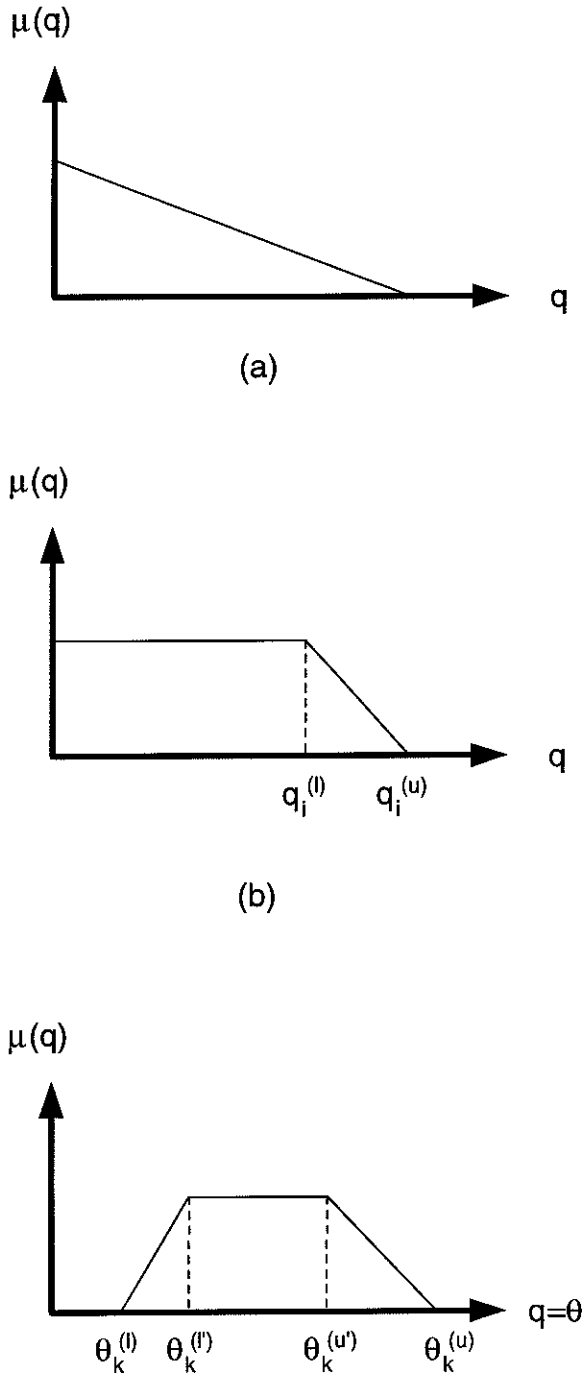


Figure 2.8: Various preference functions

Note that Equation 2.13 is equivalent to:

$$\mathbf{0} = \nabla_{\boldsymbol{\theta}} \underbrace{(w_1 \ln(V_{max} - V(\boldsymbol{\theta})))}_{\text{objective function}} + \underbrace{\sum_{active} \frac{w_i}{q_i^{(u)} - q_i} \nabla_{\boldsymbol{\theta}} q_i(\boldsymbol{\theta})}_{\text{constraints with Lagrange multipliers}}. \tag{2.14}$$

This set of equations is analogous to those which arise in minimizing total weight (or total volume) subject to the constraints $q_i \leq q_i^{(u)}$ if we identify the Lagrange multipliers λ_i with $\frac{w_i}{q_i^{(u)} - q_i}$ in Equation 2.14 and identify the "active" constraints as those for which $q_i \in (q_i^{(l)}, q_i^{(u)}]$, not just $q_i = q_i^{(u)}$. Thus, with our choice of the preference functions μ_i's, we are actually solving a problem analogous to optimality criteria approach to structural optimization.

2.4 Conclusions

In this chapter, we have looked at several existing optimal design methodologies in the literature. A recently-developed optimal design methodology is presented which allows automation of the decisions to be made in a design process. Such automation is achieved by quantifying satisfaction of each design criterion with a preference function. Using the trade-off aggregation strategy, these design criteria can be traded off with one another in a systematic manner. This optimal design framework allows easy incorporation of design criteria from different parties such as the owner, the engineer, etc.

Optimal designs obtained from this framework can be related to those obtained using other existing methodologies. For instance, these designs have been shown to lie in the Pareto optimal set of the problem. By varying the importance weights w_i of the design criteria, different Pareto optimal designs are obtained. Furthermore, with certain choices of preference function and the trade-off aggregation strategy, this method behaves very much like optimality criteria methods. This property is very useful as the engineer can compare optimal designs obtained from this framework with those obtained from optimality criteria methods.

Structural optimization problems are often solved in a continuous parameter space where each design parameter can assume any value within its specified range. However, since components required to build an engineering system normally do not come in continuous sizes, design optimization should really be done at the discrete level. In this chapter, we have mentioned some of the discrete optimization methods existing in the literature. Some of the disadvantages of these existing techniques are that they are computational inefficient, conservative, and hard to implement.

In the next few chapters, we will shift our focus to a newer class of optimization algorithms: genetic algorithms. Genetic algorithms are stochastic optimization schemes which have the potential to solve large optimization problems including discrete cases. Moreover, the implementation of genetic algorithms is quite straightforward compared to other discrete optimization techniques.

Chapter 3

Introduction To Genetic Algorithms

3.1 Introduction

Genetic algorithms (GAs) are search methods that are based on evolutionary theory which can be used to find an optimum of an objective function. They are part of a larger class of evolution-based heuristic search techniques called evolution algorithms (EAs), which consist of three main paradigms: genetic algorithms, evolution strategies (ES) and evolution programming (EP). There are also other paradigms such as classifier systems, genetic programming etc., but all these can be viewed as variants of the three main paradigms.

Genetic algorithms, which originated in the United States, are by far the most common among all the evolution algorithms. Evolution strategies are the next most common paradigm and are more popular in Europe (mainly Germany). The main differences among the different EA paradigms are the representations of variables and the choice of genetic operators. Both evolution strategies and evolution programming use mutation as the main operator for exploration in the search space while genetic algorithms emphasize crossover as their main search operator. A brief comparison among the three paradigms of EAs is given in Table 3.1. A more detailed explanation of the differences can be found in Bäck's book (1996).

In this study, we will only look at genetic algorithms. Genetic algorithms have been successfully applied to a wide range of problems ranging from the traveling salesman's problem to image recognition to machine learning. Part of this thesis

Table 3.1: Comparisons of different evolutionary algorithms

Comparison	ES	EP	GA
Representation	Real-valued	Real-valued	Binary or real-valued
Selection	Deterministic, extinctive or based on preservation	Probabilistic, extinctive	Probabilistic, based on preservation
Recombination	Discrete and intermediate, sexual	None	n-point crossover, only sexual - main operator
Mutation	Gaussian - main operator	Gaussian - only operator	Bit-inversion - background operator
Built-in Constraints Handling	Arbitrary inequality constraints	None	Simple bounds by encoding schemes
Theory	Convergence rate for special cases	Convergence rate for special cases	Schema processing theory, global convergence for elitist version

research involves applying genetic algorithms to solve discrete structural optimization problems, and, in particular, optimal design over a set of available steel sections.

3.2 A Brief History of Genetic Algorithms

Applications of simulated evolution can be dated back to the 1960s. Various biologists such as Baricelli (1957), Fraser (1960), Martin and Cockerham (1960) performed simulations of genetic systems using digital computers. Even though most of these studies were not aimed at application to search and optimization, they were not too distant from the modern notion of genetic algorithms.

In 1962, John H. Holland at University of Michigan laid out the foundation for applying genetic-like operators to artificial problems (Holland 1962a; Holland 1962b; Holland 1962c). Holland recognized the need for selection in these artificial systems and chose a population approach instead of a single point-by-point approach common in most search algorithms. However, it was not until three years later (Holland 1965) that he recognized the importance of crossover or other recombinant genetic operators such as mutation.

Between 1967 and 1975, various applications of then called genetic plans were found in theses of several students of Holland's. Bagley (1967) constructed genetic algorithms to search for parameter sets in game evaluation functions. His results indicated that his genetic algorithms were insensitive to the game nonlinearity and performed well over a wide range of environments. Around the same time, Rosenberg (1967) simulated a population of single-celled organisms under certain environments. Despite the biological emphasis of his dissertation, Rosenberg's work was important to the subsequent development of genetic algorithms because of its resemblance to optimization and root-finding. In that same period of time, Cavichhio (1970), Weinberg (1970), and Hollstien (1971) also applied genetic algorithms to pattern recognition, cell simulation and function optimization in their dissertations, respectively.

The year 1975 was an important year for genetic algorithms. Holland published

his influential book, *Adaptation in Natural and Artificial Systems* (Holland 1992). In this book, Holland explained many important aspects of genetic algorithms, and introduced the theory of schemata which allowed questions like why they work and how they perform searches in parameter space to be addressed. In that same year, De Jong completed his important and pivotal dissertation (DeJong 1975). In his studies, De Jong carefully designed a series of numerical experiments in which he considered functions which were both continuous and discrete, convex and nonconvex, unimodal and multimodal, deterministic and stochastic, etc. He also examined the effect of various control parameters of GAs such as population size, crossover and mutation probabilities, as well as different reproduction schemes (see next section for definitions of these terms). What DeJong achieved was far-reaching: he put genetic algorithms on a much firmer foundation.

Since then, a lot of research effort has been applied to both theory and applications of genetic algorithms. Many models that better describe the behavior of genetic algorithms have been proposed and many problems in various engineering fields have been solved by applying genetic algorithms. However, the field of genetic algorithms is still immature and much more research needs to be done before genetic algorithms can reach maturity and robustness.

3.3 Basic Mechanics of Genetic Algorithms

3.3.1 Terminology

The goal in any optimization problem is to find the best solution(s) to the problem. In order to apply a genetic algorithm, one must choose a suitable data structure to represent the possible solutions. Such representations can be viewed as points in the search space of all possible solutions to the optimization problem.

The data structure of genetic algorithms consists of one or more *chromosomes*. Single chromosomes are usually employed and are typically strings of binary bits, and so the term *string* is often used instead. However, genetic algorithms are not restricted

to bit-string representations. Various possible representations exist which include real numbers (Michalewicz 1994) and high level computer programs (Koza 1992). Variable length representations are also possible. A form of variable length representation known as *Messy Genetic Algorithms* (Goldberg, Korb, and Deb 1989; Goldberg, Deb, and Korb 1990) is very suitable for solving certain hard optimization problems, often referred to as GA-hard problems. Difficult problems for genetic algorithms, such as these problems, will be discussed later in this chapter.

Each string is a concatenation of a number of subcomponents called *genes*. Genes occur at different locations or *loci* of the chromosomes, and take on certain values or *alleles*. For instance, in binary-string representation, a gene is a bit, a locus is the position along the string and an allele is the value of the gene (0 or 1). In biological science, the term *genotype* refers to the overall genetic makeup of an individual and is analogous to a structure in genetic algorithms. Also, *phenotype* refers to external characteristics of an individual and is analogous to an actual parameter set such as design parameters.

Consider the following illustrative example of a GA optimization problem:

$$f(x) = x^2$$

where x = integer set of [0,31]. A common representation scheme is to transform the integer set into binary strings. In this case, there are 32 possible values of x, which requires a binary string of length 5. Thus, the string $B = 00000$ represents $x = 0$, while $B = 11111$ represents $x = 31$. Here, B is the genotype of an individual while x is the phenotype. The genotype is a point in the 5-dimensional Hamming space where the genetic algorithm searches. The phenotype is a point in the one-dimensional space of the decoded variable or the actual parameter.

To optimize a structure using genetic algorithms, some measure of quality of each individual in the search space is necessary. The *fitness function* is used for this purpose. In function optimization, the fitness function is related to the objective function. In our example, $f(x) = x^2$ is the fitness function.

3.3.2 Components of a Genetic Algorithm

A simple genetic algorithm consists of the following components:

1. representation of the parameter set of possible solutions of the optimization problem by a finite length string over some finite alphabet (usually binary)
2. initial population of strings as starting trial points
3. genetic operators and their control parameters
4. positive fitness function.

As mentioned earlier, genetic algorithms do not operate on the actual parameters of the problem. Instead, coding of these parameters as strings is necessary. In many existing optimization techniques, a single point is chosen to move in the search space and very often these methods would end up locating false peaks or local optimal points. Genetic algorithms, on the other hand, work from a population of points in the form of strings so they can simultaneously climb many peaks in a form of parallel processing. Thus, the probability that genetic algorithms locate false peaks is reduced compared to other existing methods.

Three basic operators are essential to genetic algorithms:

1. reproduction or selection
2. crossover
3. mutation.

Reproduction is a process in which individual strings are selected based on their fitness. Since the optimization goal is to maximize the objective function, strings with higher fitness should have a higher probability of contributing one or more offsprings in the next generation. Simple GAs perform *proportionate selection*, which assigns each individual string in the population a probability of selection p_s. This selection probability $p_s(i)$ of the i^{th} string in the population is simply the ratio of the string fitness $f(i)$ to the overall population fitness:

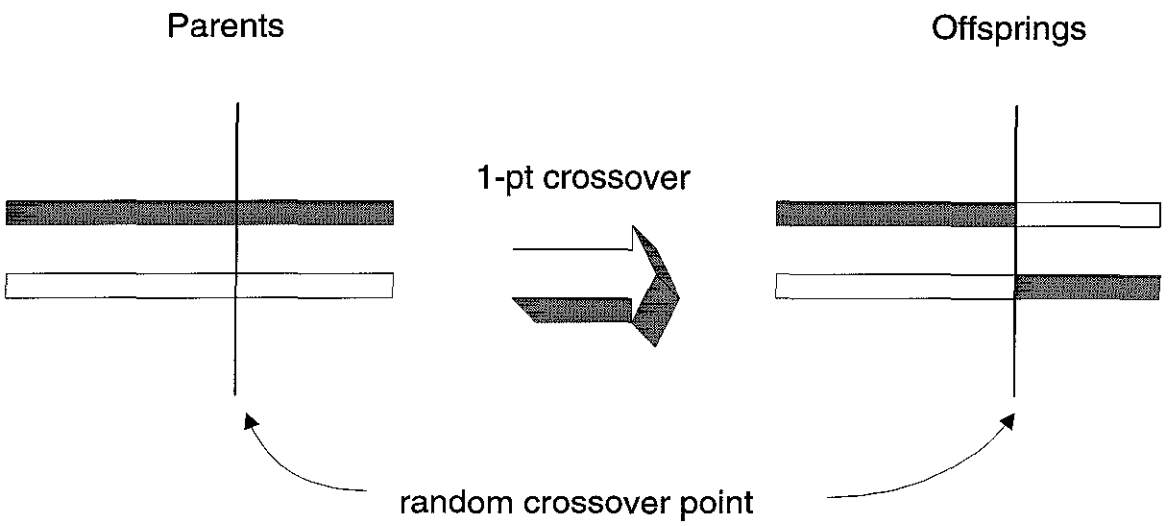


Figure 3.1: Illustration of a crossover operation

$$p_s(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}. \tag{3.1}$$

A total of n strings is selected for furthering processing according to the probability distribution based on $p_s(i)$. The simplest implementation of proportionate selection is *roulette-wheel selection* (Goldberg 1989). This selection chooses individuals by simulating n spins of a roulette wheel which has one slot for each string in the population. The size of each slot is proportional to the selection probability of the string.

After reproduction, the n selected strings undergo crossover and mutation. These two operators are the basic search mechanisms of genetic algorithms. Crossover and mutation operators create new strings from strings which have survived after the selection process. Crossover operators take two strings and generate two new individuals based on certain rules. For instance, the simple single-point crossover operator take the two parents and generates two offsprings by cutting and splicing. The cutting is performed at a randomly chosen location along the string for each parent with some crossover probability p_c, then the end parts are swapped and spliced to each initial part (see Figure 3.1).

After crossover, mutation takes place. Unlike crossover, mutation operates on one string at a time. For each string, mutation changes each element with mutation probability p_m. The typical mutation operator is binary mutation. This operator flips each bit in every string in the population with probability p_m ($p_m << 1$).

The operators described above are the simplest form of selection, crossover and mutation operators, respectively. Other alternatives exist and are usually designed for specific purposes. For instance, *tournament selection* (Goldberg and Deb 1991) and *elitist selection* (DeJong 1975) are two common alternatives to proportionate selection. *Two-point crossover* (Cavicchio 1970) and *uniform crossover* (Syswerda 1991) are alternatives to single-point crossover. For mutation, alternatives are *non-uniform mutation* (Michalewicz 1994) and *arithmetic mutation* (Bäck 1996).

Figure 3.2 shows the flowchart of a simple genetic algorithm.

3.3.3 Comparison with Existing Optimization Techniques

Simple genetic algorithms differ from other conventional optimization schemes in four major ways. Genetic algorithms are based on stochastic rules. While there are other methods that are based on simple random walks, the stochastic operators of genetic algorithms are highly exploitative (via random choice). Although using chance to achieve results may seem unusual, nature, an evolutionary process, is a very good example of apparent success of random choice.

Genetic algorithms manipulate the control variables of objective functions at the representation level (strings) to exploit similarities among well-fitted strings. Since genetic algorithms operate at the coding level, they are more difficult to be disoriented even when the function may be difficult for traditional techniques.

Genetic algorithms search using evolution of a whole population while many existing methods use only a single point. By maintaining a population of well-adapted samples, the probability that genetic algorithms will converge to global optima rather than a local optimum is increased.

Finally, genetic algorithms achieve optimal solutions by ignoring all other infor-

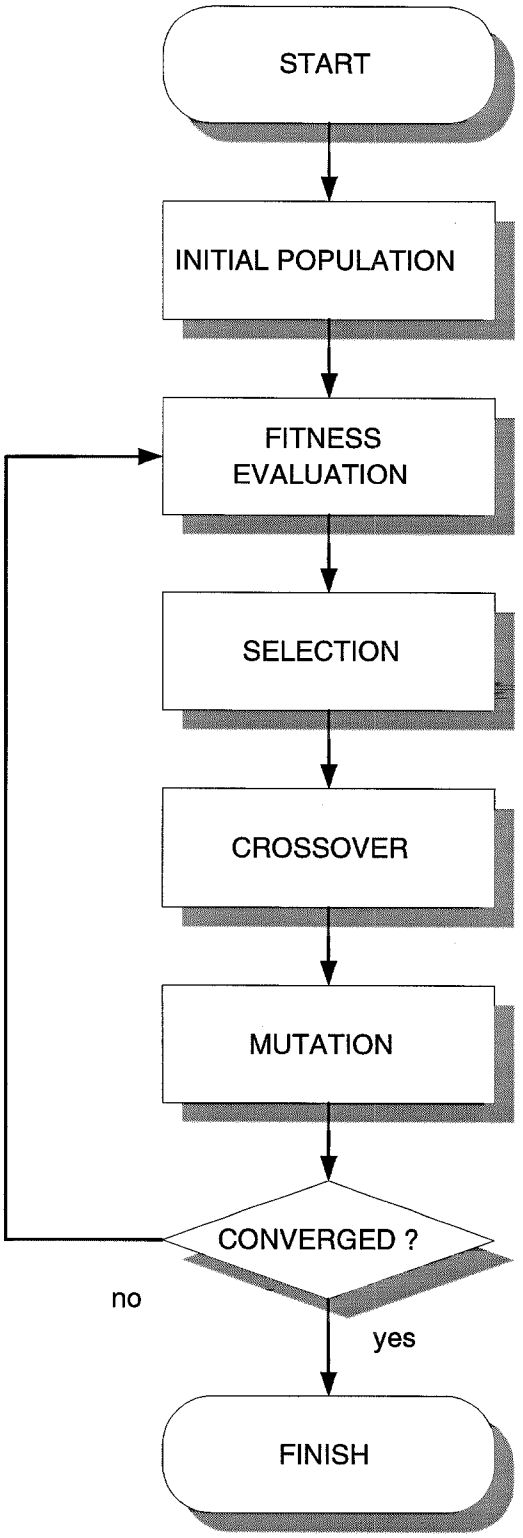


Figure 3.2: Flowchart of a simple genetic algorithm

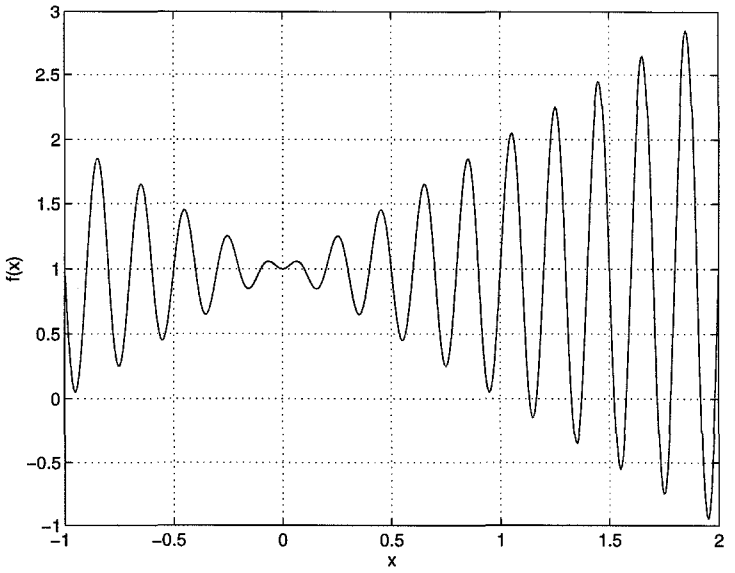


Figure 3.3: Graph of the function $f(x) = x \cdot sin(10\pi \cdot x) + 1.0$

mation except the fitness function. Other methods rely on additional information such as gradients, and in problems where such information is not available, these schemes break down. Genetic algorithms are a general method because they only use information available in any search problem, namely, the fitness or the objective function values. This means that genetic algorithms have wide applicability. On the other hand, when additional information is available, the simple GAs are not able to exploit it.

3.3.4 Illustrative Example

Consider an oscillatory function to illustrate the explorative power of genetic algorithms. We wish to find x from the range [-1,2] which maximizes the function f defined as follows:

$$f(x) = x \cdot sin(10\pi \cdot x) + 1.0.$$

Figure 3.3 shows the graphical representation of the function. Note that this func-

tion is highly oscillatory and has many local maxima. If gradient-based optimization methods are employed to maximize this function, chances are they will converge to one of the local maxima and will never get to the global maximum.

A simple genetic algorithm is used to solve this problem. Binary representation is chosen to represent the real values of the variable x. The domain of x has length 3 and binary strings of length 22 are selected to give precision of real values of x to around 10^{-6}. Thus, the strings (0000000000000000000000) and (1111111111111111111111) represent the boundaries of the domain, -1.0 and 2.0, respectively.

Snapshots of population of several generations are given in Figure 3.4. The initial population consisting of 20 strings is intentionally selected to be "far" from the global optimum. The probabilities p_c and p_m are taken to be 0.85 and 0.05, respectively. After only 10 generations, several genes approach the global optimum and by the 20th generation, several genes have found the global optimum. Note that as the population grows from generation to generation, much of the population start to converge to the neighborhood of the global optimum except a few individuals. Convergence results from the optimization run are shown in Figure 3.5.

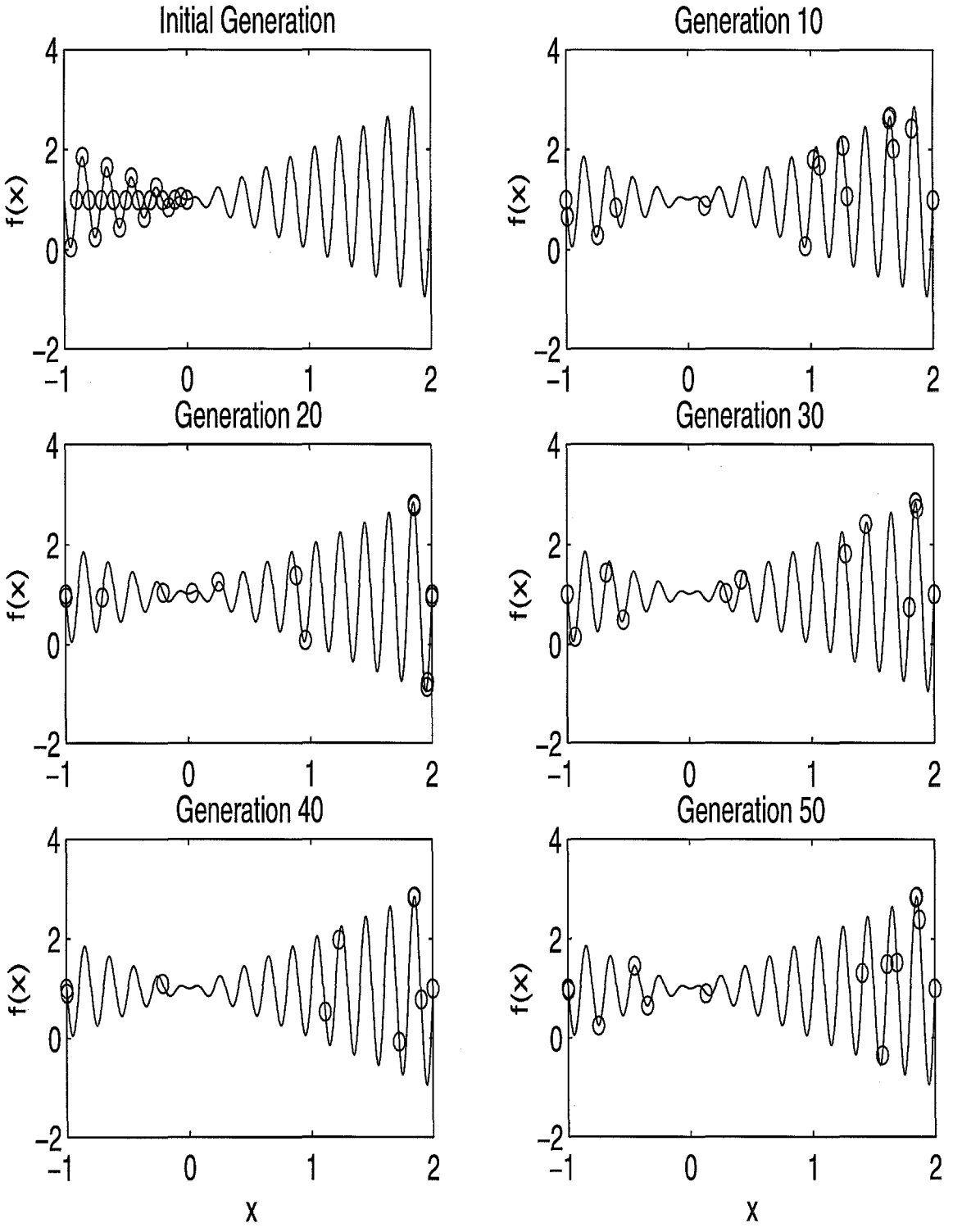


Figure 3.4: Snapshots of population of different generations in a genetic search

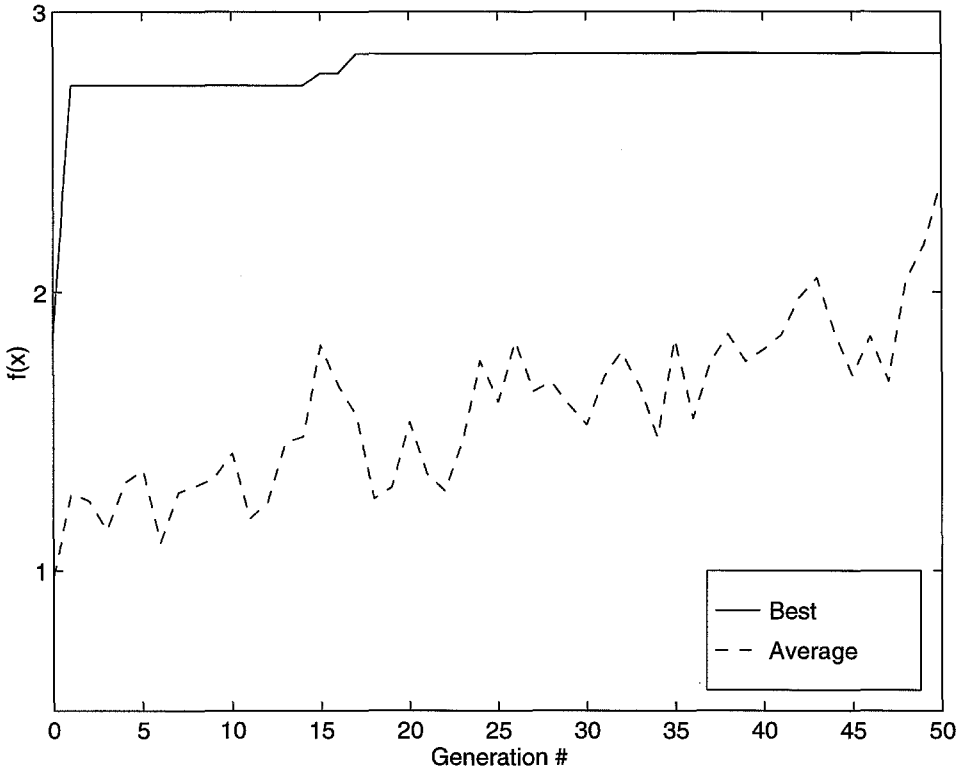


Figure 3.5: GA optimization results: best and average vs generation

3.4 Theory of Genetic Algorithms

A first exposure to the power of genetic algorithms usually leaves the impression that they simply improve by random searches via a population of strings. However, genetic algorithms efficiently explore search spaces in a different dimension than simply the string population. In this section, we examine the inner workings of a simple genetic algorithm. The notion of a schema is introduced together with some useful definitions for classification of schemata. Using schemata, the schema theorem and the building block hypothesis will be presented and questions like why GA works, and how GA seeks an optimum, will be answered.

3.4.1 Schemata

Genetic algorithms search by processing a population of strings. However, for high dimensional problems, it would be impractical, if not impossible, to search every single point in the search space. Yet, genetic algorithms have been successfully applied to large-scale problems (Unger and Moult 1993; Furuya and Haftka 1993). To explain this paradox, the notion of schema is defined.

A *schema* (Goldberg 1989; Holland 1992), denoted by H, is a similarity template which describes a subset of strings with similarities at certain string loci. It is another "string" of the same length as the strings in the population. For binary-representations, each gene in a schema takes on values of 0, 1, or *, where '*' is a wild-card or a "don't care" symbol. Each schema of length l represents the set of all strings of length l, whose corresponding loci contain bits identical to the '0' and '1' bits of the schema. For example, the schema, 1**01, represents the set of 5-bit strings {10001, 11001, 10101, 11101}. Any given string in the population can be grouped into one or more schema. For binary strings, it can be easily shown that each string of length l can be an element in 2^l schemata. The concept of schema provides a powerful and compact way to talk about all the well-defined similarities among finite-length strings over a finite alphabet.

3.4.2 Order, Defining Length and Fitness of Schemata

To describe schemata, two notations are often used in their descriptions: *order* and *defining length.* The *order* of a schema H, denoted as $o(H)$, is the number of fixed positions present in the template. In a binary alphabet, the order of a schema is simply the number of 0's and 1's. For example, the order of the schema 1**01 is 3 or symbolically, $o(1{**}01) = 3$.

The *defining length* of a schema H, denoted as $\delta(H)$, is the distance between the first and last specified genes. For the schema 1**01, the defining length $\delta(1 * *01)$ is 4. For schemata with only one specific gene such as 0****, **1** etc., the defining length is 0.

Order and defining length of schema are important notational devices for discussing and classifying string similarities. Moreover, they provide the means for analyzing the effect of reproduction and genetic operators on the population.

Finally, the *fitness* of a schema is simply defined as the average fitness of all the strings it represents.

3.4.3 Building Blocks Processing

Counting the total number of possible schemata is an enlightening process! For binary strings of length 5, there are $3^5 = 243$ different similarity templates because at each of the five loci, there are three possibilities: 0, 1 or *. So, in general for alphabets of cardinality (number of alphabet characters) k, there are $(k+1)^l$ schemata of length l. It may seem as though schemata are making the search more difficult because they increase the number of possibilities from k^l of strings to $(k+1)^l$ schemata. However, among all the possible templates, one group of them is of particular importance: the *building blocks.*

Building blocks are low-order, short defining-length and highly fit schemata (Goldberg 1989). Genetic algorithms explore the search space through successive generations. During each generation, selection, crossover and mutation take place. As described earlier, selection chooses individual strings with high fitness for further

processing. Hence, strings that are members of highly fit schemata are usually selected more frequently. For crossover operators, schemata with low defining lengths are less susceptible to disruption than those with long defining lengths. Similarly, mutation, at a low mutation probability, infrequently disrupts low order schemata. Therefore, it is easy to see that building blocks which are highly fit and low order with short defining lengths are more likely to proliferate from generation to generation. This is also presented more quantitatively in a later section. Thus, genetic algorithms perform searches through processing these useful schemata.

3.4.4 Implicit Parallelism of Genetic Algorithms

It is clear that the building block processing is essential to the success of genetic algorithms in efficiently seeking optimal solutions. The next logical question to ask is how efficiently do genetic algorithms process building blocks? Holland (1992) estimates that for a population of n strings, genetic algorithms process on the order of n^3 building blocks in each generation . Since in each generation, only n function evaluations are needed (for fitness calculations), this is a significant processing leverage that is apparently unique to genetic algorithms. Since nothing extra is needed to achieve this processing "parallelism," Holland calls this phenomenon *implicit parallelism.* For solution of large-scale problems, the existence of implicit parallelism means that a larger population has the potential to find the optimal solution(s) in polynomially faster time than a smaller population.

3.4.5 Fundamental Theorem of Genetic Algorithms

Let $m(H,t)$ be the expected number of instances of schema H present in the population at generation t. Thus, $m(H,t+1)$ represent the expected number of instances of schema H in the next generation. Assuming that proportional selection is used, we have $m(H,t+1) = m(H,t) \cdot \frac{f(H)}{\overline{f}}$, where $f(H)$ is the fitness of the schema H and $\overline{f}$ is the average population fitness.

The probability that the schema H survives crossover is greater than or equal to

the term, $1 - p_c \cdot \frac{\delta(H)}{l-1}$. The equality provides the lower bound of the probability of survival while the inequality takes into account that a disrupted schema may regain its composition through crossover with a similar schema. The probability that H survives mutation is $(1 - p_m)^{o(H)}$, which is approximately $(1 - o(H) \cdot p_m)$ for small p_m (recall that $p_m << 1$). Putting all these together, we have:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\overline{f}} \cdot (1 - p_c \cdot \frac{\delta(H)}{l-1} - o(H) \cdot p_m). \tag{3.2}$$

This expression is known as the *Schema Theorem* or the *Fundamental Theorem of Genetic Algorithms*. Although the above arguments are not a formal proof of the theorem, this mathematical expression does provide the expectation of the survival of building blocks and its implication is far-reaching and subtle (Goldberg 1989).

3.5 Issues Concerning Genetic Algorithms as Optimizers

So far in this chapter, we have discussed the basic components of genetic algorithms, how GAs work and how they explore the search space. To end this chapter, we will look at some of the issues and difficulties in applying GAs to optimization problems. We will focus on control parameters of GAs (population size, operator probabilities, etc) and importance of GA operators, representation difficulties, scaling of fitness, GA-hard and GA-deceptive problems.

3.5.1 Control Parameters and Importance of Genetic Operators

From the earlier discussions, genetic operators of a standard GA include crossover and mutation. These two operators have probabilities of p_c and p_m, respectively. Also, there is another parameter which is very important to the success of GA operation:

population size. The choice of these parameters can have significant impact on the performance. This issue was studied thoroughly by Schauffer and his colleagues (1989) and they suggested the following values:

- Population size: 20-30
- Crossover probability: 0.75-0.95
- Mutation probability: 0.005-0.01.

Traditionally, mutation in GA has always been considered as the secondary operator while crossover is the main operator that performs most of the exploration chore. However, Schauffer's studies also found that:

1. mutation plays a stronger role than previously recognized
2. importance of crossover is overrated
3. search strategy based on selection and mutation only might be fairly powerful even without crossover.

The question on whether crossover or mutation is more important has been the focus of hot arguments. Up till now, no researcher can provide good research evidence showing one way or the other although it seems that most applications of GA involve both operators.

3.5.2 Representation Difficulties

As mentioned earlier, in order to apply a GA to an optimization problem, a representation of the parameters of the objective function is required. So far, we have only considered binary strings for such a representation. The problem with binary representation is that it requires the number of possible values each parameter can assume to be some power of 2. One can use alphabets with higher cardinality to represent other cases. However, if the number of possible values of a parameter is some

arbitrary number, chances are there is no alphabet that would provide a one-to-one representation between the actual parameter values and the encoded strings.

In addition, the choice of representation can have significant impact on the performance of the genetic algorithms. In practice, to efficiently solve an optimization problem often requires some knowledge of the problem in order to come up with a suitable representation. Up to now, there is not a set of guidelines which would provide newcomers to genetic algorithms with assistance in choosing representations. These two representation issues can hinder the applicability of genetic algorithms to various engineering problems.

To get around these difficulties, Michalewicz advocates bypassing the representation and have GAs directly work on the parameter set or the phenotype. In his book (Michalewicz 1994), he provides various examples of using such an approach to solve optimization problems with real-valued parameters. Michalewicz proposes that using such a real-value representation, one can achieve at least as good, if not better, results than using the more traditional coding schemes such as binary representation.

Another approach is to pose the problem as a constrained optimization problem. To solve this problem one can choose any representation and then apply penalties to invalid individuals in the population using some penalty function. However, penalty functions can change the landscape of the search space enough to affect the optimal solution. Richardson, Palmer, Liepins and Hilliard (1989) suggest certain guidelines for using penalty functions with genetic algorithms.

3.5.3 Fitness Scaling and Tournament Selection

For genetic algorithms with small populations, control of the number of copies each string has is very important. In earlier generations, it is common to have a few extraordinary individuals in a population of less than average strings. If the proportional fitness selection ($P_s = \frac{f_i}{\sum_j f_j}$) is used, the extraordinary individuals would take over a significant proportion of the small population in a few generations and this leads to premature convergence to a possibly non-optimal solution. In addition,

there is a very different problem in later generations. Late in a GA run, the average fitness of the population may be close to the best fitness of the population. Thus, average members and the best ones get nearly the same number of copies in future generations, and the survival of the fittest necessary for improvement becomes more of a random walk among average individuals.

To address these shortcomings, fitness scaling is often used. The simplest way to scale fitness is linear scaling:

$$f' = af + b.$$

Here, the scaled fitness f' is scaled by using two coefficients a and b such that the average scaled fitness f'_{avg} equals to the average of raw fitness f_{avg} and the best individual f_{max} is scaled down to around $2f'_{avg}$. Fitness scaling helps prevent early domination of better fitness individuals, and in later generation, it also encourages more competition among the population as the best fitness is still around twice the average fitness. However, fitness scaling does not work for all populations.

Another approach is to use a selection with good selection pressure. Selection pressure is the degree to which the better individuals are favored. The higher the selection, the more the better individuals are favored. However, if the pressure is too high, there is an increased chance of premature convergence. On the other hand, if the pressure is too low, slow convergence would occur. One such selection scheme is tournament selection (Goldberg and Deb 1991). This selection scheme randomly chooses a set of s individuals from the population and picks the best for reproduction. Normally, a tournament size of $s = 2$ is used: binary tournament selection. This scheme provides a good selection pressure which does not cause premature convergence in initial generations and encourages competition in later ones.

3.5.4 GA-hard and GA-deceptive Problems

Most existing optimization techniques work well for some functions but do not perform well on other type of functions. Genetic algorithms are no different. Since it is

a stochastic approach, it is hard to predetermine which types of functions would cause difficulties for GAs. In the last 10 years or so, many researchers in the field have tried to characterize functions that are most difficult for genetic algorithms to optimize (Bethke 1980; Goldberg 1990; Goldberg 1991; Goldberg, Korb, and Deb 1989; Liepins and Vose 1991; Whitley 1991).

It is generally recognized that optimization functions that are difficult for GAs to solve (or GA-hard problems) have one or more of the following properties:

1. multiple optima
2. isolation of optima
3. misleading sub-optima that lead GAs away from the desired optima
4. presence of noise.

Strictly speaking, multimodality is not really a source of hardness since we are usually interested in only a single optimum. In addition, the concepts of *niching, crowding* and *sharing* have been applied to genetic algorithms for multimodal function optimization with some success (Goldberg and Richardson 1987). The problem of noise can be handled by choosing appropriate population size (Goldberg, Deb, and Horn 1992).

Problems which have both misleading and isolated, desirable optima are called *deceptive* or *GA-deceptive* problems and are especially hard for a GA to solve. Deceptive problems are widely studied examples in the literature (Goldberg 1987; Whitley 1991). Classic deceptive problems usually have a global optimum and another local optimum called the *deceptive optimum.* The global optimum has a small basin of attraction, while the deceptive optimum has a large basin of attraction. Moreover, the global and deceptive optima have similar fitness values.

A well-known example of such deceptive function is shown in Figure 3.6. This is a 3 bit function with a global optimal point isolated from the rest of the crowd and there are several suboptima which have function values close to the global optimum.

From the schema perspective, let's consider the fitness of the following schema: 0**, 1**, **0, **1, 00*, 11*, *00 and *11.

$$
\begin{aligned}
f(0^{**}) &= \frac{1}{4}(f(000)+f(001)+f(010)+f(011)) = \frac{1}{4}(28+26+22+0) = 19 \\
f(1^{**}) &= \frac{1}{4}(f(100)+f(101)+f(110)+f(111)) = \frac{1}{4}(14+0+0+30) = 11 \\
f(^{**}0) &= \frac{1}{4}(f(000)+f(010)+f(100)+f(110)) = \frac{1}{4}(28+22+14+0) = 16 \\
f(^{**}1) &= \frac{1}{4}(f(001)+f(011)+f(101)+f(111)) = \frac{1}{4}(14+0+0+30) = 11 \\
f(00^{*}) &= \frac{1}{2}(f(000)+f(001)) = \frac{1}{2}(28+26) = 27 \\
f(11^{*}) &= \frac{1}{2}(f(110)+f(111)) = \frac{1}{2}(0+30) = 15 \\
f(^{*}00) &= \frac{1}{2}(f(000)+f(100)) = \frac{1}{2}(28+14) = 21 \\
f(^{*}11) &= \frac{1}{2}(f(011)+f(111)) = \frac{1}{2}(0+30) = 15.
\end{aligned}
$$

It can be easily seen that the following is true:

$$
\begin{aligned}
f(1^{**}) &< f(0^{**}) \\
f(11^{*}) &< f(00^{*}) \\
f(^{**}1) &< f(^{**}0) \\
f(^{*}11) &< f(^{*}00).
\end{aligned}
$$

All schemata consistent with 111 such as 1**, **1, *11 and 11* are the ones GA needs to obtain the global optimum 111. Since their competitors which are schemata associated with 000 (0**, **0, 00*, etc.) have better fitnesses, by the schema theorem, the number of strings processed via these schemata will increase with generations. Thus, genetic algorithms will tend to converge to 000 instead of 111 because of these misleading schemata.

From the building block viewpoint, deceptive problems are those where low-order schemata are misleading, i.e. they tend to lead GAs to the deceptive optimum, just like the example. However, it should be noted that deception on its own will not

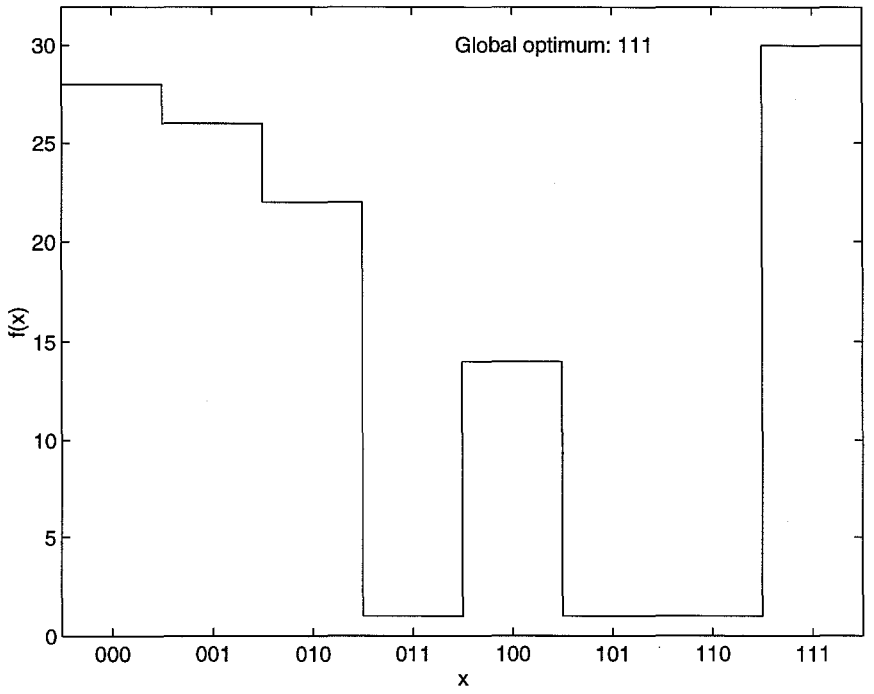


Figure 3.6: Different view of the 3-bit deceptive function

necessarily stop GAs from reaching the optimal solution. It requires both misleading schemata as well as bad *linkage* among the schemata to make a problem GA-hard. Linkage can be loosely measured by the defining length of schemata. It is a measure of how far apart the important substrings of a schema are in its representation. With some knowledge of the problem at hand, one can usually choose a certain representation or apply reordering schemes to provide tight linkage among the building blocks.

The existence of deceptive problems and the inability of simple GAs to solve these problems is an obstacle that must be overcome. In the next chapter, we will examine these deceptive problems in greater detail and we will look at one variant of GAs, variable-length genetic algorithms (Goldberg, Korb, and Deb 1989; Goldberg, Deb, and Korb 1990), which have the potential to solve these problems without requiring much prior knowledge about them.

Chapter 4

Special Classes of Genetic Algorithms

4.1 Introduction

In the last chapter, we looked at the basics of genetic algorithms, how they perform searches and why they work. In this chapter, we will focus on two special classes of genetic algorithms: *variable-length genetic algorithms* and *hybrid genetic algorithms.* Variable-length GAs have the potential to solve GA-hard problems which require certain allele combinations to be close together. Such problems may arise when applying simple GAs to structural optimization problems over available steel sections. For these problems, the use of variable-length representation is recommended and a variable-length scheme called vGA is proposed. For continuous optimization problems, hybrid GAs, which are combinations of hill-climbing methods and genetic algorithms, are quite attractive in terms of better convergence rate. A hybrid GA, denoted as hGA, is also proposed.

4.2 Variable-Length Genetic Algorithms

4.2.1 Motivation

The success of genetic algorithms depends on the growth of short, low-order and highly-fit schemata (building blocks) through successive generations to form optimal solutions. In deceptive problems, nonlinearities may prevent these building blocks

from forming optimal solutions. In addition, the genetic representation (genotype) of a problem may be such that the needed allele combinations are widely apart or loosely linked and so, genetic operators such as crossover are likely to disrupt these desirable building blocks. One way to overcome this problem is by using a tight ordering representation that codes the needed allele combination closely together to provide tight linkage. However, such a tight gene ordering in a problem requires prior knowledge about the problem which, in most cases, is not usually available. Without any knowledge of tight coding, a random coding usually results in low-order building blocks which are loosely linked.

Faced with this coding problem, researchers have come up with the idea of using a variable-length representation. Among the different variable-length schemes (Smith 1980; Shaefer 1987; Cramer 1985) existing in the literature is *messy genetic algorithm* (mGA) developed by Goldberg, Korb and Deb (1989, 1990). Using this new mGA, Goldberg and his colleagues successfully found the global solution of a high order deceptive problem to global optimality.

In this section, we will focus on variable-length genetic algorithms. The variable-length representation and the special operators associated with this coding are presented. As we will see, variable-length GAs are suitable for solving discrete optimization problems with GA-hardness and such difficulties may arise in structural optimization problems over available steel sections. A variable-length GA, denoted as vGA, is also proposed here which is based on the messy GA and is specially designed for solving the aforementioned discrete optimal design problems in structural engineering.

4.2.2 Variable Length Representation

Variable-Length Coding

A gene in a variable-length chromosome contains information of both its locus and the allele. For example, the string 10100 in simple GA can be represented by ((2 0) (1 1) (4 0) (3 1) (5 0)) or ((5 0) (3 1) (1 1) (4 0) (2 0)), where in each

duplet, the entries are the locus and the allele of the gene, respectively. Unlike its simple GA counterparts, strings in a variable-length GA can be either underspecified or overspecified. Thus, the variable-length strings ((2 0) (1 1)) and ((3 1) (1 0) (2 1) (1 1)) are acceptable strings for a 3-bit problem under this variable-length coding scheme. Here, the first string is underspecified since gene 3 is missing while the second string is overspecified because gene 1 appears twice. Note that a variable-length string can be both underspecified and overspecified at the same time. As we will soon see, this variable-length coding provides the flexibility that allows important gene combinations to stay close together even though they may be far apart in the fixed representation. However, such flexibility does not come without a price as additional effort is required to decode overspecified or underspecified strings.

Variable-Length Decoding

To evaluate the fitness of an individual, the full string is required. Since a variable-length string can be underspecified or overspecified or both, additional effort is needed to decode these strings.

Overspecification can be handled fairly easily as it requires us to choose between conflicting genes in the string. A straightforward way is to take the first instance of a gene allele using first-come-first-served rule from left to right. For example, the second string from the previous discussion, ((3 1) (1 0) (2 1) (1 1)), contains two instances of gene 1 and will be decoded as 011 since (1 0) precedes (1 1) in the string. Other decoding possibilities exist such as voting procedure and adaptive precedence (Goldberg, Korb, and Deb 1989).

For underspecification, it is necessary to fill the missing genes in an underspecified string. Different techniques can be devised for such purpose. One approach is to employ a *template* and make a given string complete by filling the missing genes of the string with the corresponding genes from the template. To illustrate how this works, consider our previous underspecified gene, ((2 0) (1 1)), and the template 000. The complete 3-bit string for our string is (1 0 0), borrowing the third gene from the template.

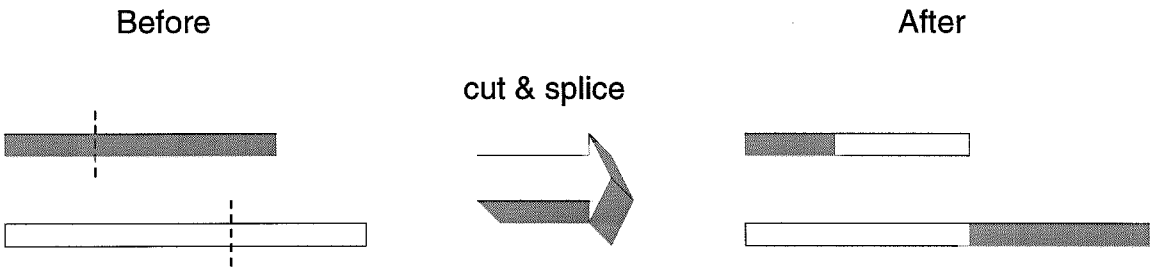


Figure 4.1: Illustration of a cut and splice operation

4.2.3 Operators of a Variable-Length GA

To handle strings of variable length, both crossover and mutation have to be modified. For crossover, a new operator, consisting of two operators, *cut* and *splice* (Goldberg, Korb, and Deb 1989) is normally used for this purpose. First, the cut operation is carried out on two randomly selected strings. Then, the splice operator combines the resulting strings to form new offsprings. Figure 4.1 illustrates a cut and splice operation.

For mutation, two operators, *allelic mutation* and *genic mutation*, are usually employed in a variable-length GA. An allelic mutation operates on allele values by flipping the bits with a specified allelic mutation probability, p_{am}. This is similiar to the mutation operator which flips bits along the fixed-length strings in simple GA described in the previous chapter. Complementary to the allelic mutation is the genic mutation, which swaps one gene with another one with a specified genic mutation probability, p_{gm}. For instance, for the string ((3 1) (1 0) (2 1) (1 1)), the genic mutation can swap the first and the third genes to form ((2 1) (1 0) (3 1) (1 1)). This reordering of genes can affect the offsprings created by the cut and splice operators.

4.2.4 Organization of a Variable-Length GA

Despite the differences in representation and the operators used, the overall flow of a variable-length genetic algorithm is very similiar to their simple counterparts. Figure 4.2 shows the overall flowchart of a variable-length GA. The only difference here is that the crossover operator in simple GA is replaced by the cut and splice operator.

4.2.5 Proposed vGA for Discrete Structural Optimization over Available Steel Sections

A variable-length genetic algorithm, denoted as vGA, is presented here. This vGA is tailored for solving discrete structural optimization problems over available wide flange sections (W-shapes). This algorithm follows many of the ideas described earlier in this chapter and the only application-specific information about vGA is: the variable-length representation of steel sections and the template used for addressing underspecification of strings.

For the representation, 256 wide flange sections are picked from the AISC manual for coding. Thus, an 8-bit string is required for each design variable. For instance, the string ((1 0) (2 0) (3 0) (4 0) (5 0) (6 0) (7 0) (8 0)) represents the smallest W section W4x13. Only 256 of the possible 297 W sections are chosen to allow an easy one-to-one mapping between the phenotype (the W-shapes) and the genotype (the variable-length strings).

For underspecification, an initial template of all zero bits is used. After every n generations, the current template is updated with information from the best strings. By doing so, any underspecified strings will get the good allele combinations from the best individuals and therefore, improve the average fitness of the population.

Recall that a variable-length GA usually employs the following three operators: selection, cut and splice, and mutation. For selection, the *binary tournament selection* scheme (Goldberg and Deb 1991) is used to avoid the necessity of function scaling and to maintain a reliable selection. In a binary tournament selection, two strings

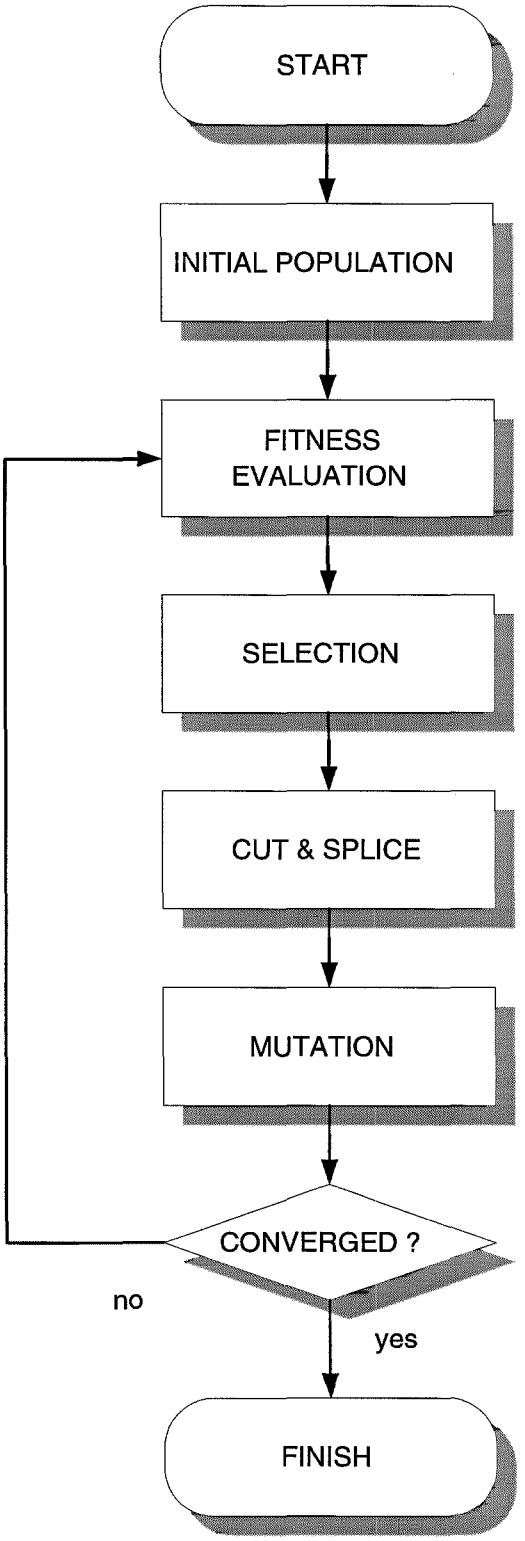


Figure 4.2: Overview of a variable-length genetic algorithm

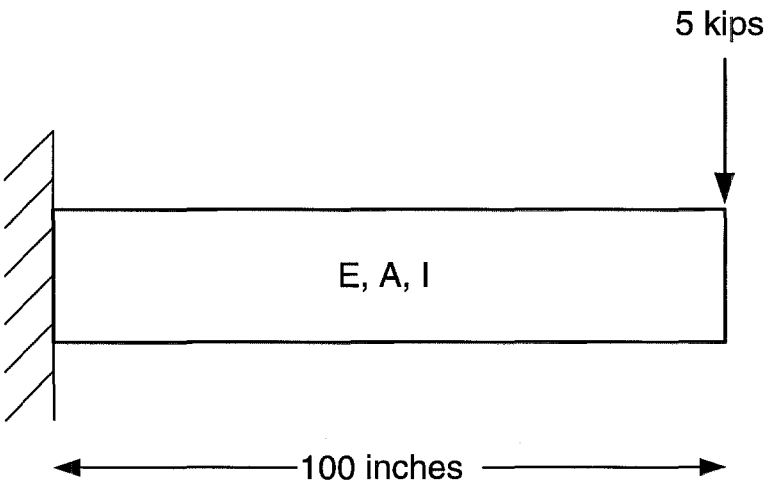


Figure 4.3: Cantilever beam for the illustrative example

are picked randomly from the population and the best of the two is selected. For a population of size n, this tournament is repeated n times to create a new population consisting of the winners of these n tournaments.

The overall flow of vGA is the same as illustrated in Figure 4.2.

4.2.6 Illustrative Numerical Example

Background

A simple cantilever beam design problem is utilized as an example to demonstrate various issues involving discrete optimal structural design over available steel sections. Consider the simple cantilever beam as illustrated in Figure 4.3. The length of the beam is 100 inches. A point load of 5 kips is applied at the far end of the beam. The objective is to obtain a design that best satisfies the following two design criteria:

1. Maximum Bending Stress: Bending stress along the beam should not be greater than 40 ksi

2. Total Steel Volume: The total steel required should be minimized.

Table 4.1: Optimization results of the illustrative example

	Simple GA		Proposed vGA	
Criteria	Value	μ	Value	μ
Stress	29.10 (ksi)	1.000	33.06 (ksi)	0.798
Volume	471 (in^3)	0.939	416 (in^3)	0.955
Overall		**0.941**		**0.949**

Final Design	Simple GA	Proposed vGA
Section	W12x16	W12x14
Area	4.71 (in^2)	4.16 (in^2)
I	103.0 (in^4)	88.6 (in^4)

The multicriterion design approach described in Chapter 2 is applied to this problem. To quantify the above design criteria, two preference functions are defined as shown in Figure 4.4. Referring first to the preference function for stress (Figure 4.4a), the perfectly acceptable stress range from 0.0 to 36.0 ksi and the upper bound is set to the specified limit of 40 ksi. The preference value of unity drops off linearly from 36.0 ksi and reaches an unacceptable value of zero when it hits 40 ksi. For steel volume, a normalized volume is used instead which is defined by:

$$V_{normalized} = \frac{V_{max} - V}{V_{max} - V_{min}}$$

where V_{max} and V_{min} are the specified maximum and minimum volumes. The preference function for steel volume is triangular in shape as shown in Figure 4.4b.

The domain of the search space is the smallest 256 wide flange sections (W sections) listed in the AISC manual. This problem is solved both with a simple genetic algorithm and also the proposed vGA. Since the search space is quite small (only 256 possible solutions), a relatively small population size of 10 is used for both schemes and an initial population is randomly generated. For comparison purpose, this same initial population is used for the two GAs. The maximum generation count is limited to 30. Optimization results are tabulated in Table 4.1.

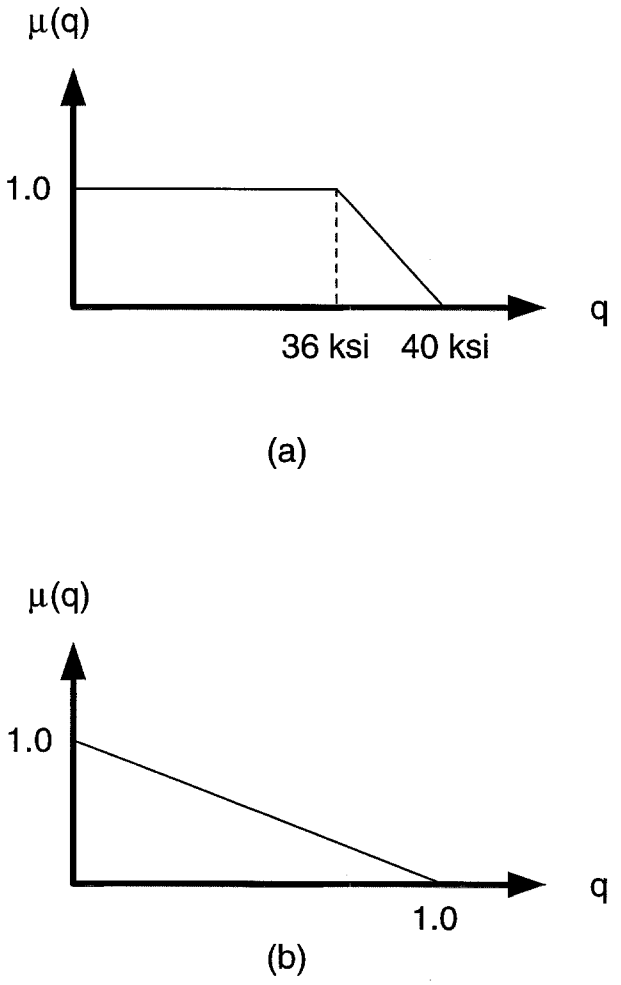


Figure 4.4: Preference functions for the illustrative example

Discussions and Comments

This optimal design problem actually possesses some of the GA-hardness described in Chapter 3. Consider the individual preference values in Figure 4.6. The plot for maximum bending stress illustrates a very interesting point. When the AISC sections are sorted by area, the corresponding moments of inertia are not monotonically increasing with the area and therefore, there are fluctuations from unacceptable to acceptable and back to unacceptable as we go from smaller sections to larger sections. Table 4.2 shows the smallest twenty W sections. Notice how moment of inertia fluctuates as the area increases. The relationship of area and moment of inertia is further illustrated in Figure 4.5. Steel volume, on the other hand, monotonically increases with area and so the preference value monotonically decreases as the section area increases. As a result, the overall preference value has the shape as shown in Figure 4.7. Note that this function looks very much like the deceptive function described in Chapter 3, which is hard for simple genetic algorithms to solve. The global optimum is isolated by a "sea" of sections which are totally unacceptable while the the majority of the relatively larger sections have reasonably acceptable preference. This property is exactly one of the causes for GA-deception. Such deception is reflected in the results of the optimization runs since the simple GA settled with one of the suboptimal designs while the vGA converged to the global optimal solution. Note that the same GA-deception characteristics will be observed if this problem is solved by sorting the members with moment of inertia instead of area.

Table 4.2: Properties of the twenty smallest AISC W-sections

Section	Area (in^2)	I (in^4)
W6X9	2.68	16.4
W8X10	2.96	30.8
W10X12	3.54	53.8
W6X12	3.55	22.1
W4X13	3.83	11.3
W8X13	3.84	39.6
W12X14	4.16	88.6
W10X15	4.41	68.9
W6X15	4.43	29.1
W8X15	4.44	48.0
W5X16	4.68	21.3
W12X16	4.71	103.0
W6X16	4.74	32.1
W10X17	4.99	81.9
W8X18	5.26	61.9
W5X19	5.54	26.2
W12X19	5.57	130.0
W10X19	5.62	96.3
W6X20	5.87	41.4
W8X21	6.16	75.3

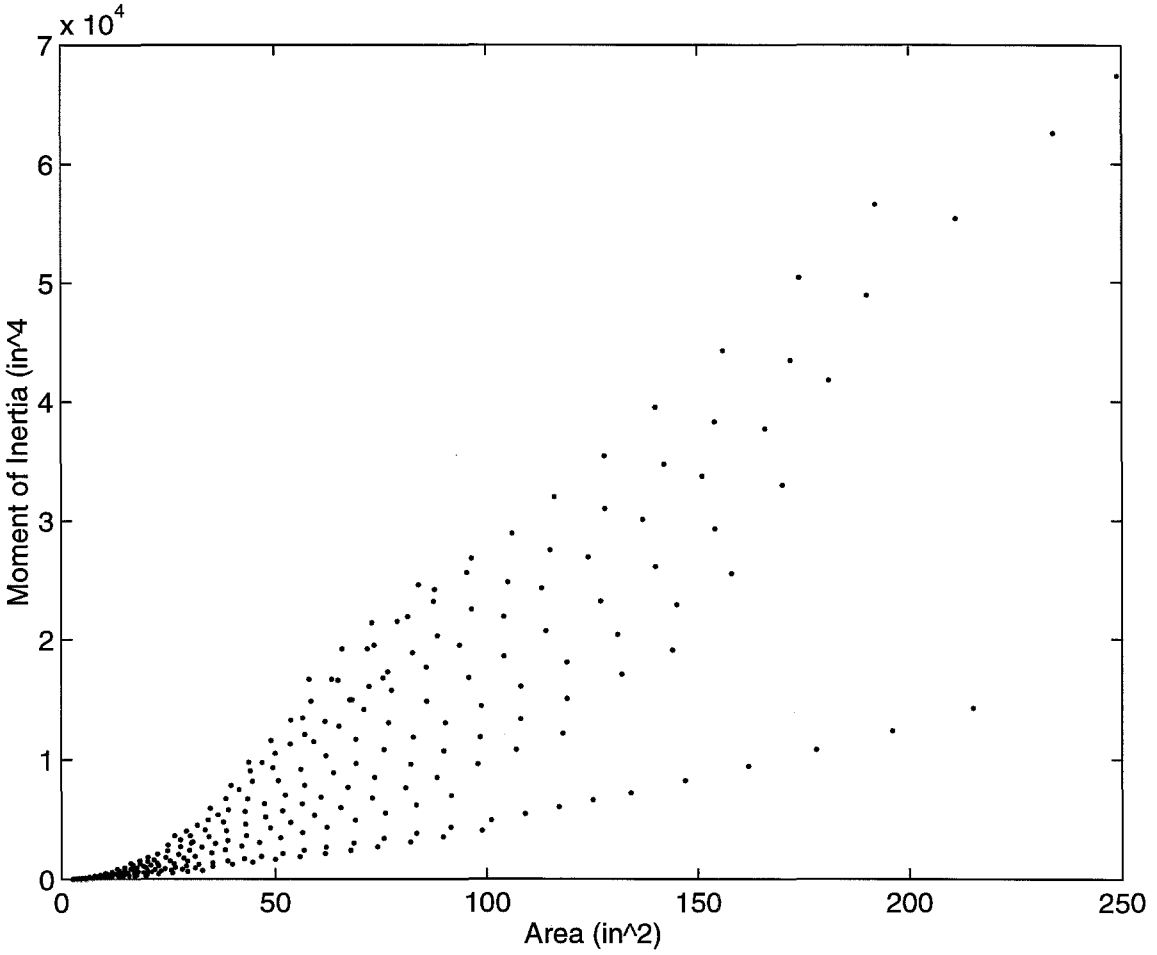


Figure 4.5: Scatter plot of AISC sections: moment of inertia vs area

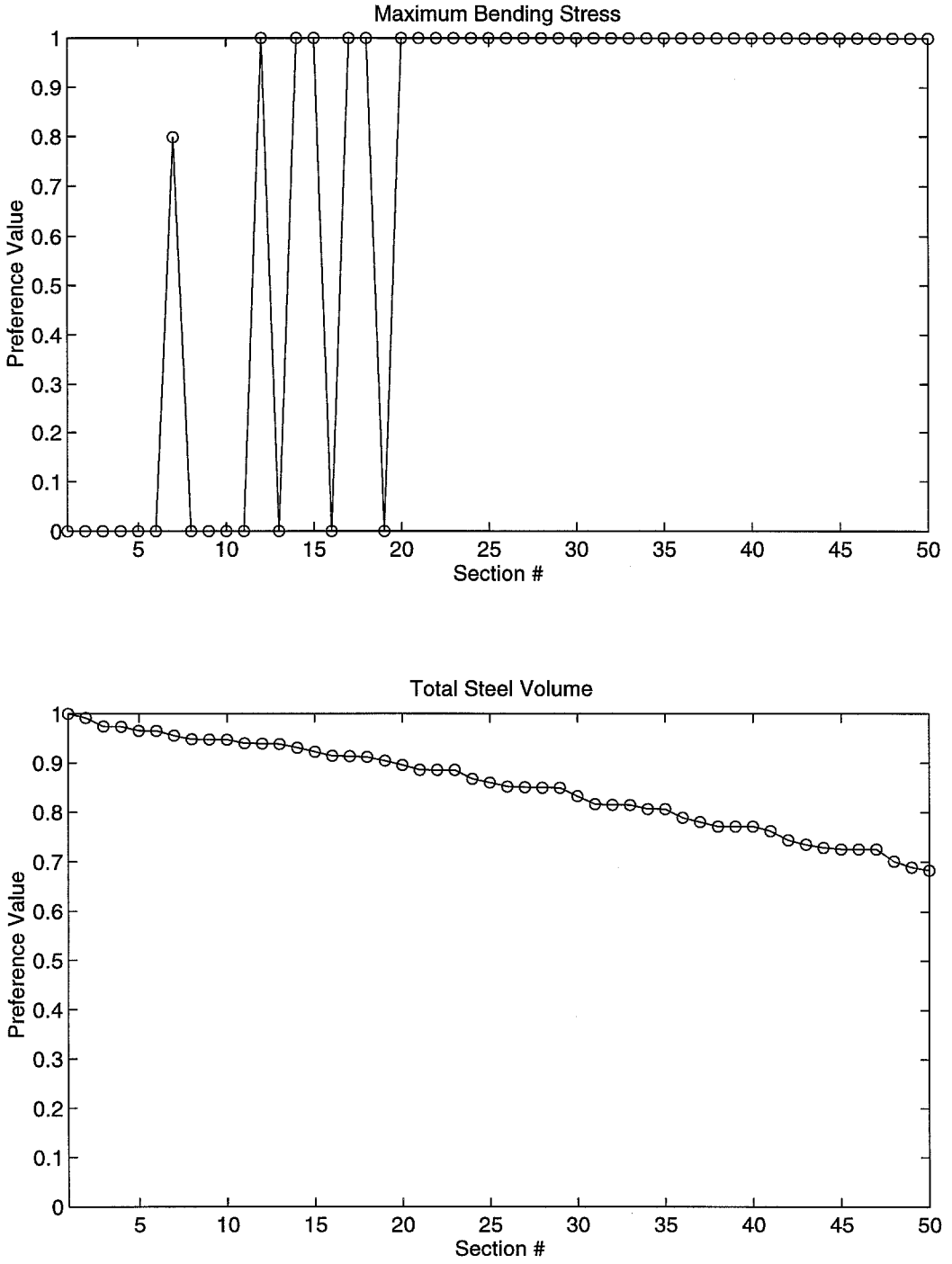


Figure 4.6: Individual preference values of stress and volume versus AISC sections (sorted by area)

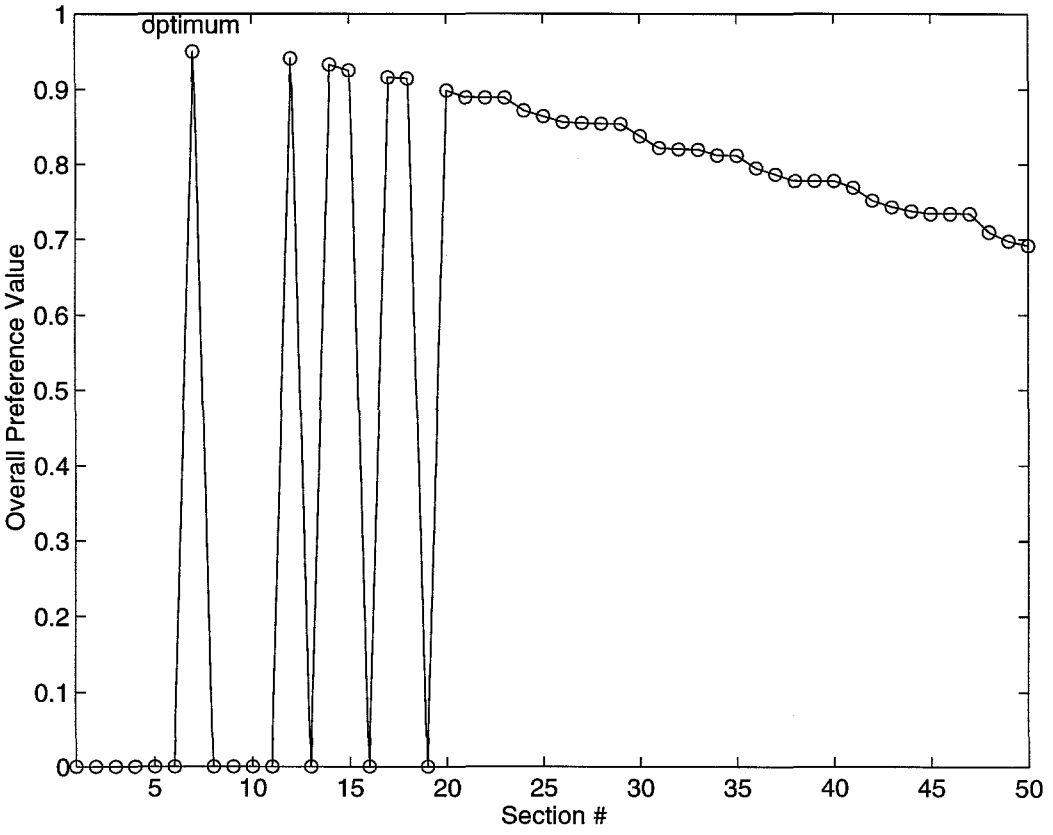


Figure 4.7: Overall preference value versus AISC sections (sorted by area)

4.3 Hybrid Genetic Algorithms for Continuous Optimization

4.3.1 Motivation

As discussed in Chapter 3, genetic algorithms are stochastic methods which offer several advantages over traditional methods for continuous variables. By using a population approach and some probabilistic rules, genetic algorithms seek an optimum through exploration of the search space. They are definitely better alternatives than traditional methods for continuous-variable optimization problems with numerous local optima. Nevertheless, GAs can suffer from slow convergence before providing an "accurate" solution primarily due to their lack of exploitation of local topological information of the problem. Moreover, the accuracy of the solutions obtained may not be very good in terms of digits of accuracy due to the stochastic nature of the method.

On the other hand, traditional methods, mostly hill-climbing, such as quasi-Newton methods, are well known to exploit the local topological information efficiently to provide an optimum in the neighborhood. Usually, a lot of local information, such as the gradient vector and Hessian (curvature) matrix, is required to achieve this high level of exploitation. If such information is not available, these methods are usually not very robust and reliable. However, if an optimum is found, the solution is usually very accurate numerically.

Obviously, there is a conflict among accuracy, reliability and computational effort when searching for the global optimal solution of a complex problem (Renders and Flasse 1996). It is generally impossible to reach an accurate and reliable solution with little computational effort. This conflict can also be viewed as a tradeoff between exploitation and exploration. Genetic algorithms, which can bypass local optima to arrive at the global optimal solution, are good at exploration but often suffer from slow convergence and a lack of exact solution due to its stochastic nature. Hill-climbing methods which are good at exploitation focus on accuracy and efficiency but lack

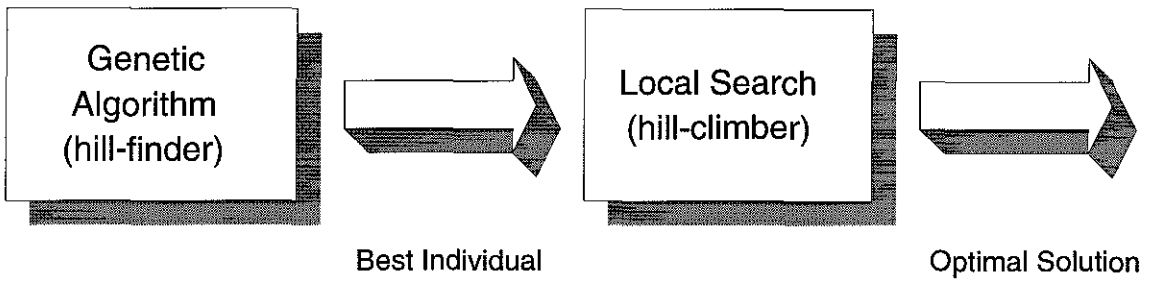


Figure 4.8: Schematic of a hybrid genetic algorithm

reliability. Thus, these two classes of optimization can complement each other in terms of their strengths and weaknesses. A combination of both the GAs and hill-climbing methods, known as hybrid genetic algorithms, provides a good balance of accuracy, reliability and efficiency.

4.3.2 Definition of a Hybrid GA

When problem-specific information is available, it is usually advantageous to consider this information during the optimization process. Since genetic algorithms do not utilize any specific information other than the objective function value, a hybridization of GAs with other schemes that can take into account this additional information is needed. Such hybridization is usually a genetic algorithm coupled with a local search scheme that utilize local topological information. In this case, the exploration power and global perspective of GAs are coupled with the hill-climbing ability of local optimization methods and result in a scheme that has the best of both worlds.

Local optimization of a continuous function is a well-studied area and numerous gradient and gradient-less methods are available for finding local optima. To develop a hybrid GA, we can simply connect some local search technique with a genetic algorithm. One simple implementation of a hybrid scheme is to feed the best solution obtained by GA into a local search technique to compute the local optimal. In a

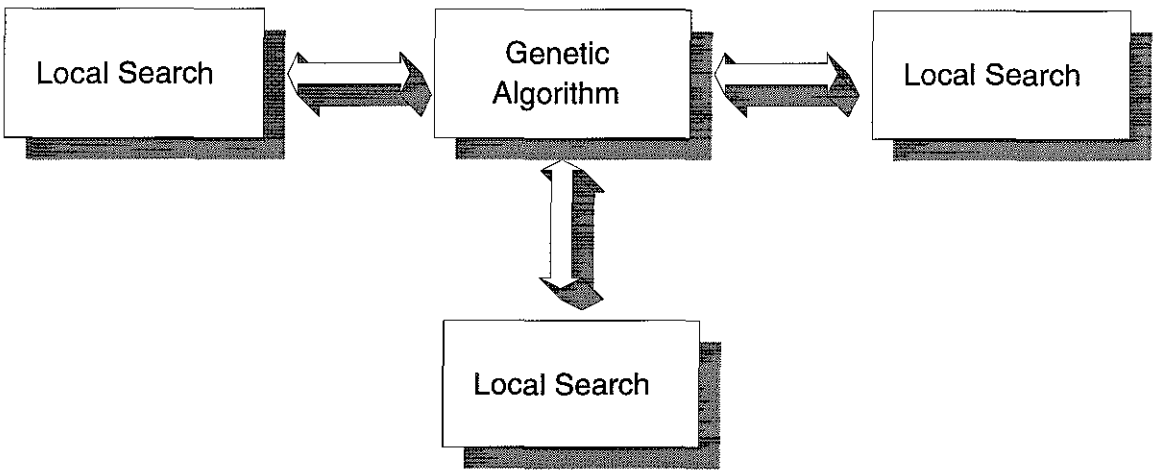


Figure 4.9: A parallel implementation of a hybrid genetic algorithm

sense, the GA finds the hills while the local search method, the hill-climber, climbs them (see Figure 4.8).

Since genetic algorithms are population-oriented, a parallel approach to hybrid GAs can be achieved in a straightforward way. Figure 4.9 depicts an implementation of a parallel hybrid GA. With numerous parallel processors, function evaluations can be carried out simultaneously for different strings within a generation. Moreover, some of the parallel processors can occasionally perform local searches on the better fitness individuals.

4.3.3 Proposed hGA

A specialized hybrid genetic algorithm, denoted as hGA, is presented here. This hGA is designed specifically for continuous optimization problems. Special characteristics of hGA, which are elaborated below, are:

1. Real-valued representation is used for coding of function variables.

2. Genetic operators such as crossover and mutation are borrowed from those of evolution strategies, which are always real-valued coded.

3. Interface with local search methods is done via an operator.

Real-valued Representation

Although binary representation is by far the most common coding scheme, one main disadvantages for using binary coding for real-valued optimization is that it requires a long binary string to achieve accuracy in the solution. Recall that for the single variable function in Chapter 3, a binary string of length 20 is required to represent a potential solution accurate to the sixth decimal place. For high dimensional problems, a much longer binary string is required and such long strings could result in weak linkage in building blocks. Moreover, significant computational effort is needed to encode and decode each binary string. Therefore, for the proposed method, a floating point or real-valued representation is employed. Another advantage for using a real-valued coding is the ease of passing potential solutions back and forth between GAs and hill-climbing techniques.

In the real-valued representation, each potential solution or chromosome is represented by a vector of real numbers. For a function of three variables, each chromosome is represented by a vector of three real values (i.e. three genes). Each gene within the chromosome is enforced to stay within its specified range.

Genetic Operators

Since a real-valued representation is used, crossover and mutation operators for binary representation have to be modified in order for them to work in a similiar way. For crossover operations, only crossover between genes are allowed. Two crossover operators are used in conjunction: simple crossover (Michalewicz 1994) and arithmetic crossover (Bäck 1996). These two operators have probabilities, p_{sc} and p_{ac}, respectively.

Simple crossover is defined very similiar to the one-point crossover for simple GAs.

If parents $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$ are crossed after the k^{th} position, where k is chosen randomly and then crossover occurs with probability p_{sc}, the resulting offsprings are $X' = (x_1, ..., x_k, y_{k+1}, ..., y_n)$ and $Y' = (y_1, ...y_k, x_{k+1}, ..., x_n)$. However, this operator may produce offspring which are infeasible. For this case, a different position is picked and crossed again. If the resulting offsprings are still infeasible, the parents are then taken as offsprings.

Arithmetric crossover can be defined as follows:

$$X' = rX + (1-r)Y \tag{4.1}$$

$$Y' = (1-r)X + rY, \tag{4.2}$$

where X, Y = parents

X', Y' = offsprings

r = random value in (0,1).

Again, this crossover occurs with probability p_{ac}. Since this operation is basically a linear combination of the two parents, the resulting offsprings are always valid for convex problems.

For mutation, a non-uniform mutation operator (Michalewicz 1994) is chosen for this method. From a chromosome $X = (x_1, ..., x_n)$, this unary operator generates an offspring $X' = (x_1, ..., x'_k, ..., x_n)$ by mutating the k^{th} gene in X, where

$$x'_k = \begin{cases} x_k + \delta(t, x_{k,ub} - x_k) \text{ if } r < 0.5 \\ x_k + \delta(t, x_k - x_{k,lb}) \text{ if } r \geq 0.5 \end{cases}.$$

Here, $x_{k,lb}, x_{k,ub}$ = lower and upper bounds of variable x_k

r = random value in (0,1)

t = generation number

$\delta(t,y)$ $= y \cdot \left(1 - r^{(1-\frac{t}{t_{max}})^b}\right)$.

The function $\delta(t,y)$ causes the algorithm to search the space more uniformly initially

(when t is small), and very locally as t approaches the maximum generation count t_{max}. This operator has a probability of p_m.

Local Search or Hill-Climbing Operator

Many hybrid schemes work by taking solutions from one method (GA) and feeding them to another method (local search). However, for the proposed method, a local search technique is used as an operator within the genetic algorithm. Because of the real-valued representation, the two methods are more integrated and chromosomes, which are vectors of real values, can be passed to and from the local search algorithm without any encoding or decoding. A quasi-Newton method, which is well-known to have second order convergence rate when it is close to the optimum, is used for the local search operator.

The local search operator works as a supplementary operator to the proposed method. Local search is performed after every m of generations, the search operator performs local searches for certain individuals, usually the best and the worst ones, in the population. These individuals are then updated with the solutions obtained from the local searches.

The Overall Picture

Incorporating all the ideas in the above discussions, the overall flowchart of the proposed hGA is summarized in Figure 4.10. Note that there are two different convergence checks in hGA. The first one is done within the local search algorithm and is not shown in the figure. This check is performed in every local search. The second convergence check is done at the GA level and is shown at the bottom of the figure.

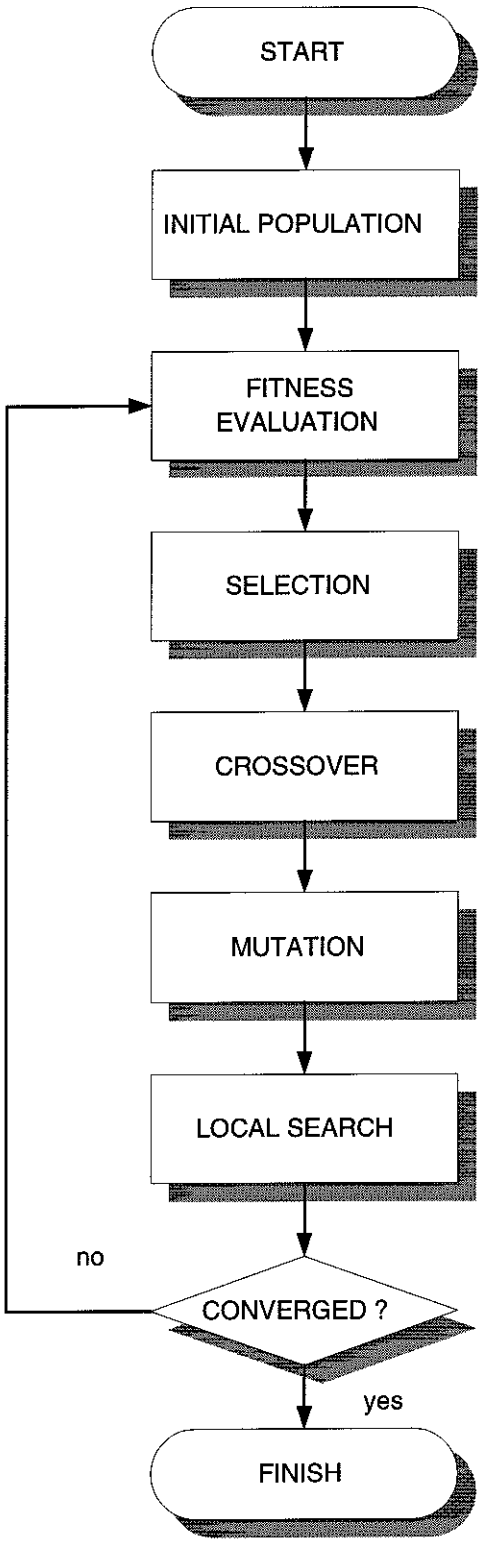


Figure 4.10: Overall flowchart of the proposed hGA

4.3.4 Numerical Example

Background

Consider a two-variable function $f(x_1, x_2)$ as illustrated in the contour plot in Figure 4.11. This function arises from model identification of a two degrees of freedom shear building. The objective function is to minimize the two-norm of the error between the modal frequencies of the predicted model and the actual model. The variables x_1 and x_2 are scaled values of the interstory stiffness for the first and second stories, respectively. Both variables x_1 and x_2 have the range of $[0, 3]$. This function has two optimal points: $[1.0, 1.0]^T$ and $[2.0, 0.5]^T$, as denoted by '*' in the figure. However, the basin of attraction of $[1.0, 1.0]^T$ is much bigger than that of $[2.0, 0.5]^T$. That is, most initial guess in the domain $[0, 3] \times [0, 3]$ will converge to the solution $[1.0, 1.0]^T$. In addition, this function has the "banana valley" characteristic, which may cause traditional hill-climbing methods to have slow convergence if not started with a good initial guess.

The function is solved by a simple genetic algorithm and the proposed hGA. The population size is set to 20 for both algorithms. The initial population is generated randomly and for comparison purposes, the same initial population is used for the two methods. The maximum generation count is limited to 30. Probabilities p_{sc}, p_{ac} and p_m are set to 0.85, 0.85 and 0.01, respectively. The convergence histories of both methods are shown in Figure 4.12. A snapshot of the initial population is given in Figure 4.13 and those of the final populations of hGA and simple GA are presented in Figure 4.14 and Figure 4.15, respectively.

Discussion and Comments

From the convergence histories, it is obvious that the proposed hGA method has a much faster convergence rate than the simple GA. The main reason why this is so is because of the presence of the local search operator in hGA. This operator is applied every five generations, and in fact, hGA found the optimal solution after the 5^{th} generation. In contrast, simple GA takes a much slower path to convergence and

in fact, at termination after the 30^{th} generation, it only found a fairly close solution $[0.98, 0.97]^T$.

Looking at the snapshots of the initial and final populations of the two methods, an interesting observation can be drawn. Notice that with an identical initial population, the majority of the final population of the simple GA (see Figure 4.15) converges to the solution $[1.0, 1.0]^T$, which is expected as the basin of attraction of this solution is much larger than that of the other solution $[2.0, 0.5]^T$. However, for the hGA, both solutions were found within the 30 generations (see Figure 4.14) although most of the strings clustered around the solution $[1.0, 1.0]^T$. In fact, they were obtained at the 10^{th} generation. Thus, with the presence of a local search operator, the proposed hGA is capable of locating multiple optimal solutions. Although this property cannot be verified mathematically, similiar results were obtained from several independent runs.

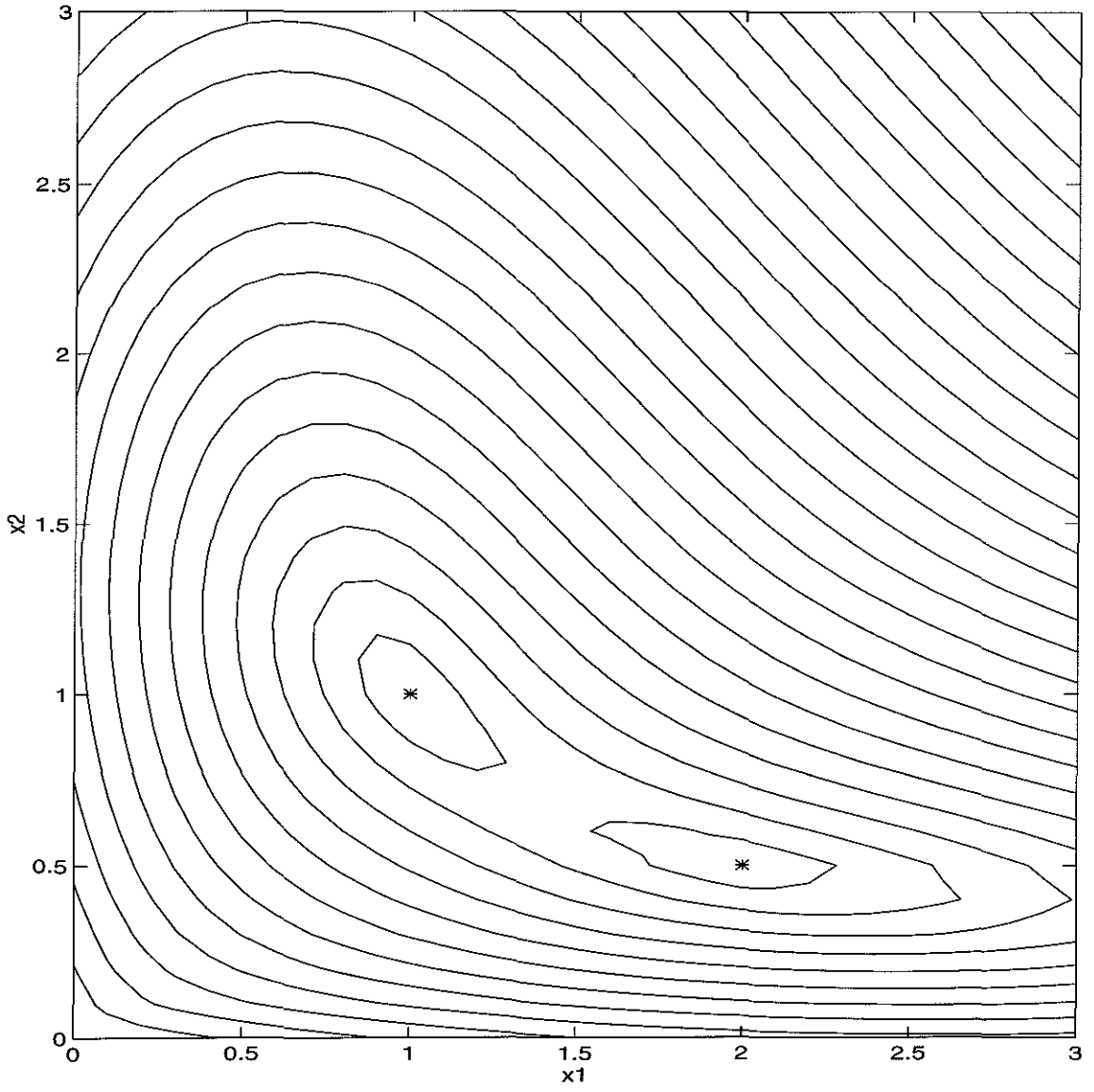


Figure 4.11: Contour plot of objective function for the numerical example

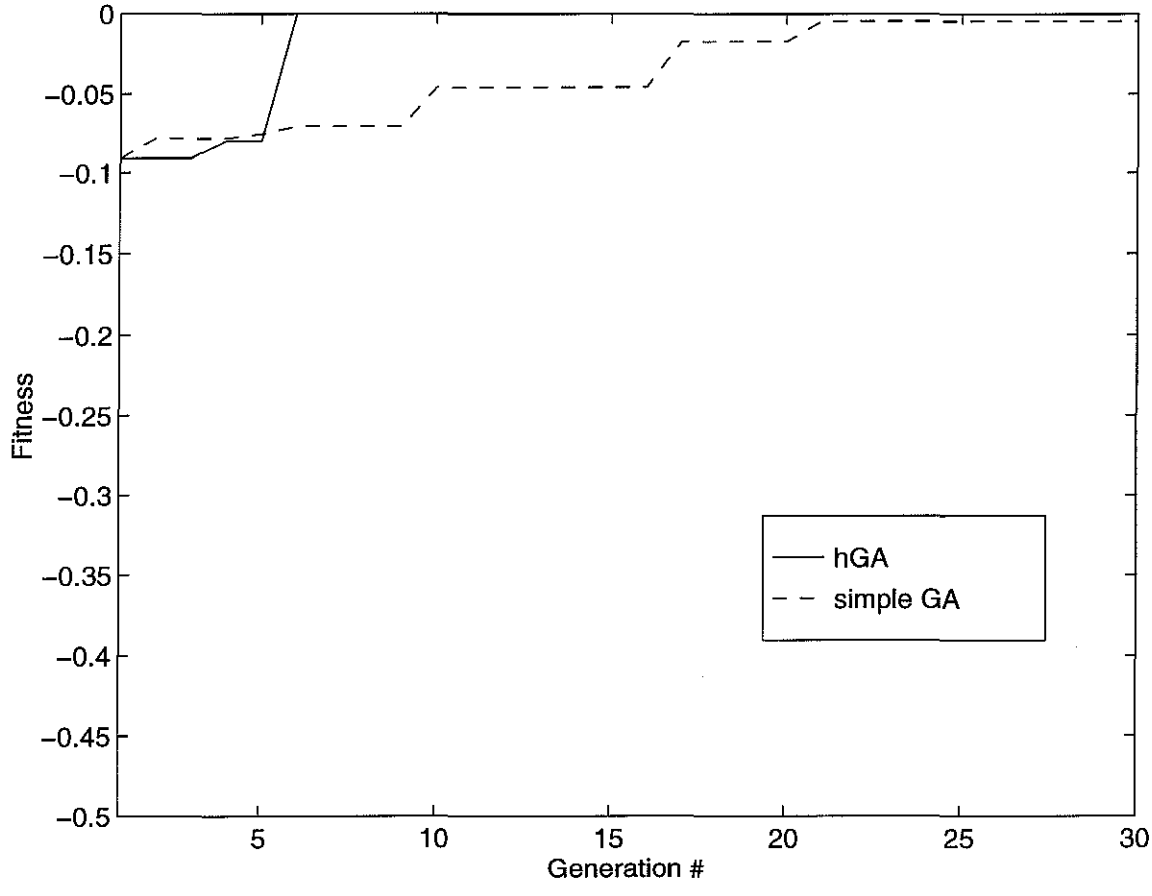


Figure 4.12: Convergence histories of simple GA and hGA

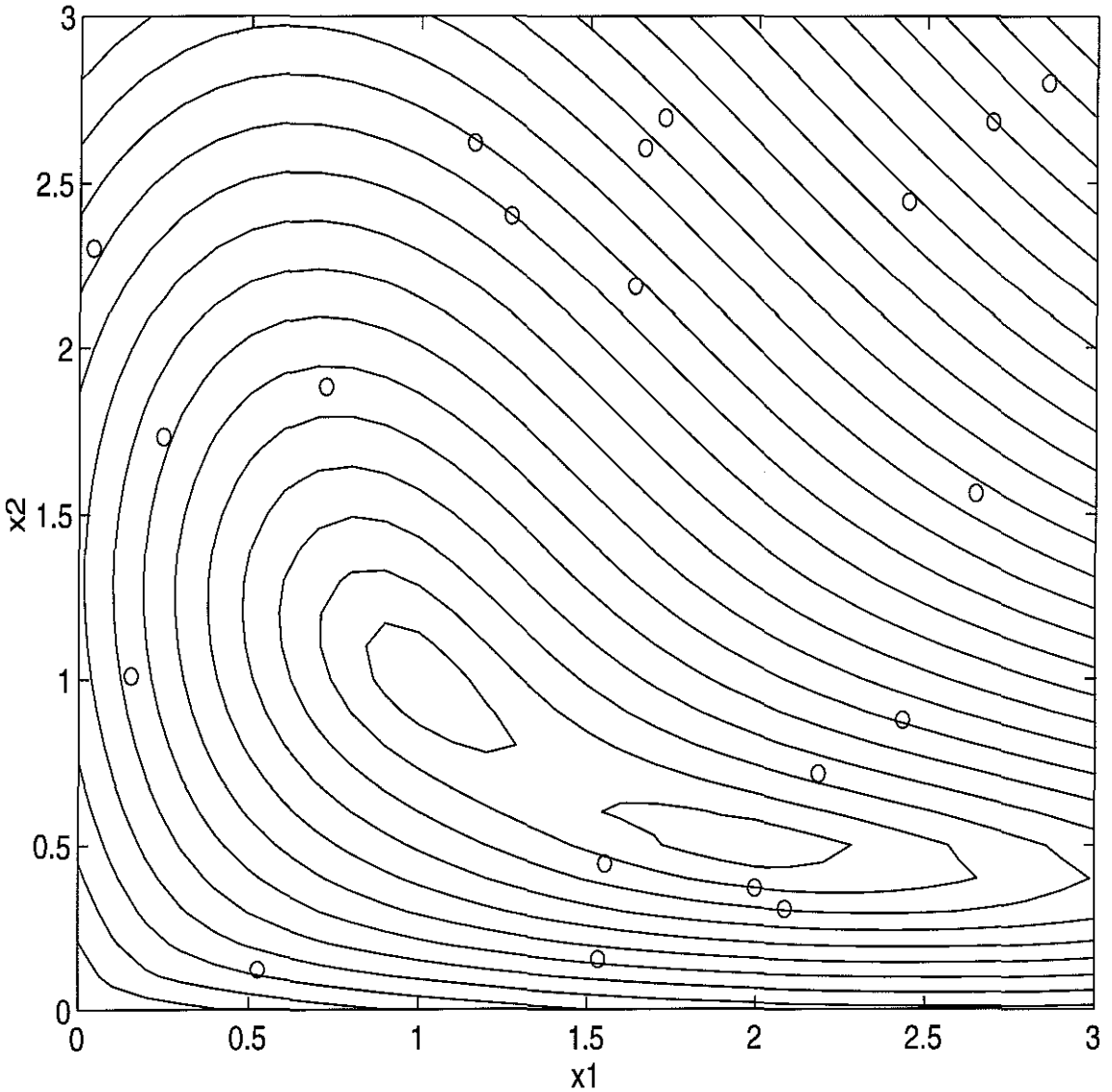


Figure 4.13: Snapshot of initial population of both methods

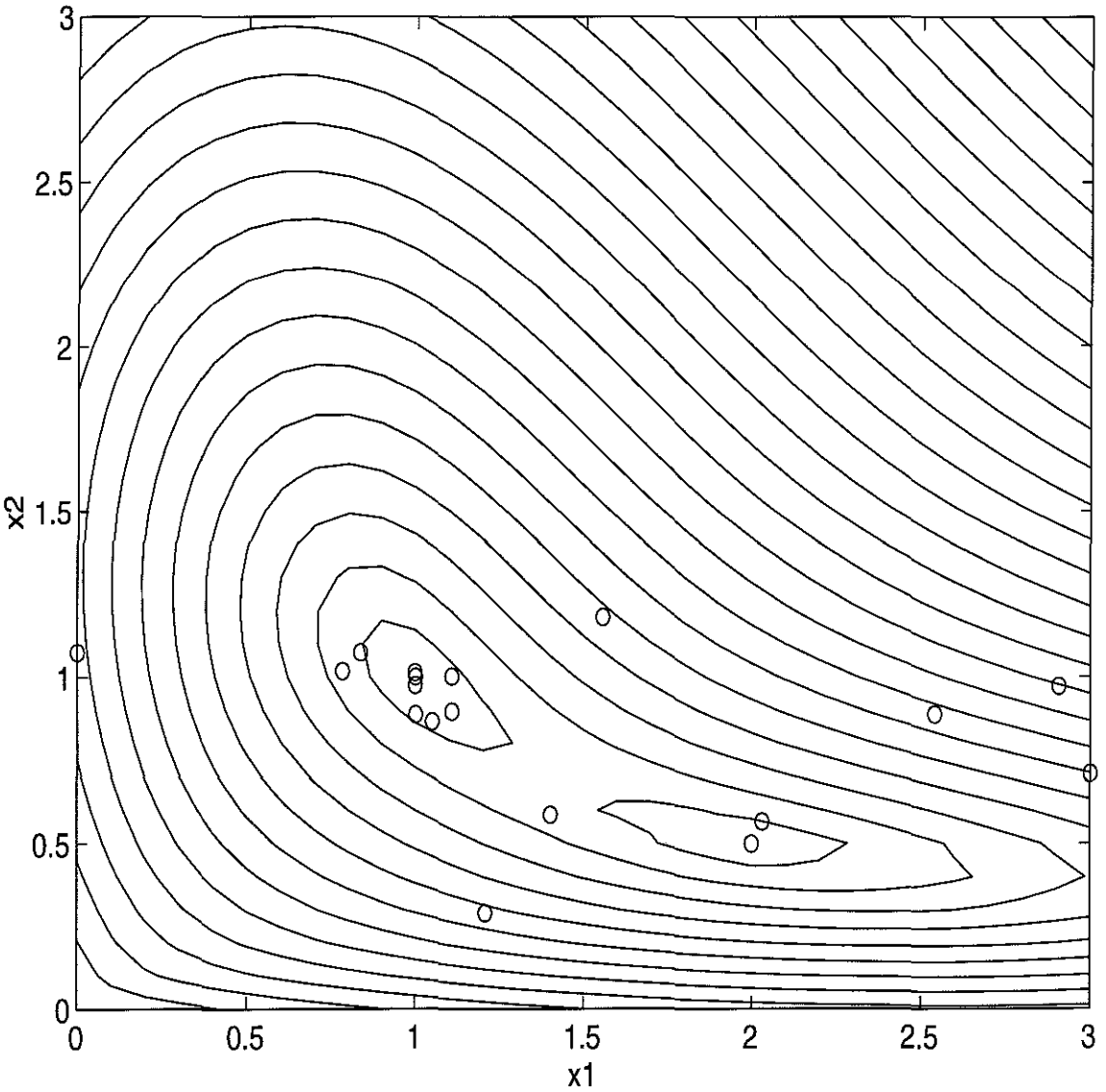


Figure 4.14: Snapshot of final population of the proposed hGA

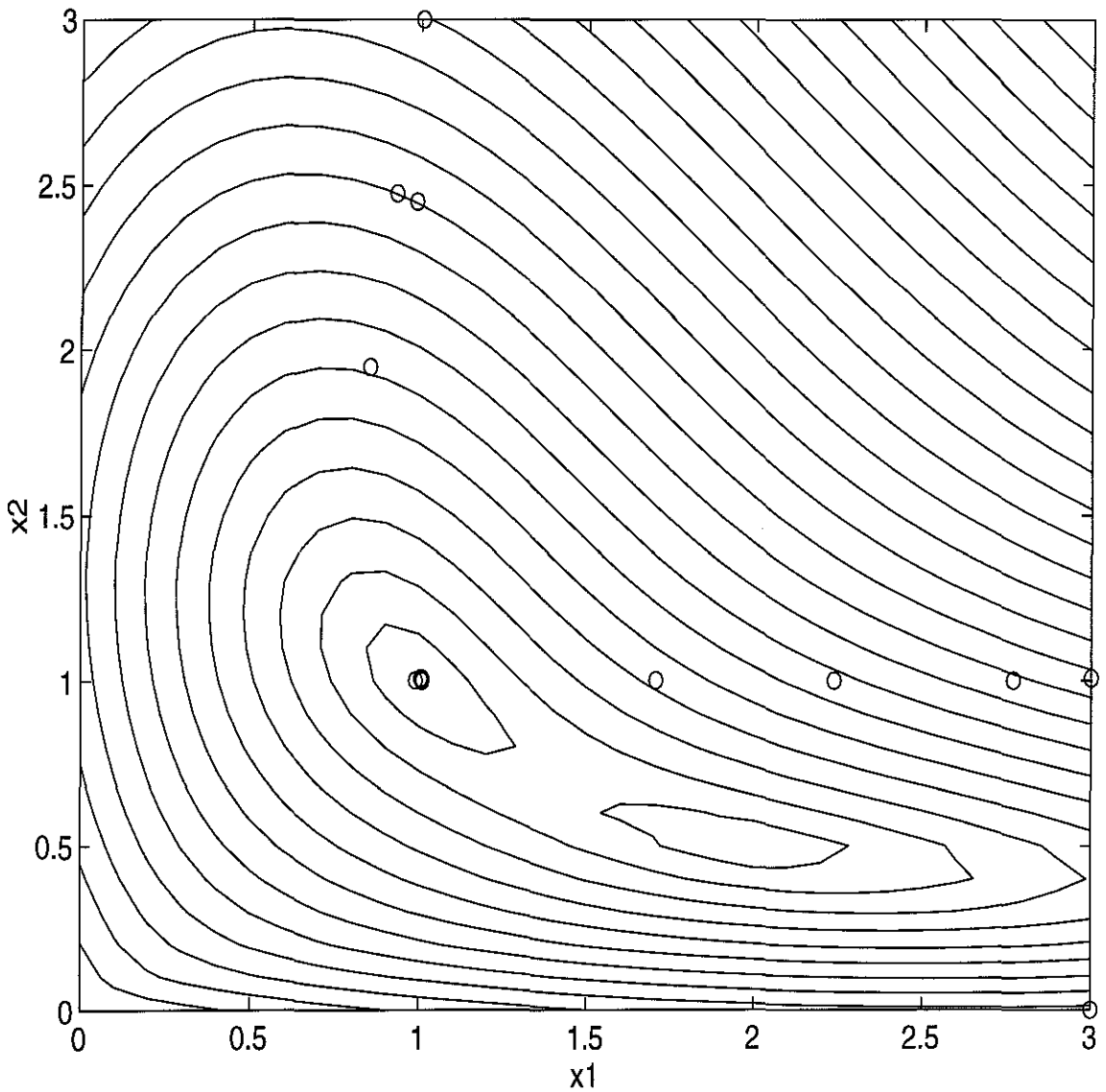


Figure 4.15: Snapshot of final population of simple GA

4.4 Conclusions

In this chapter, two special classes of genetic algorithms are presented: variable-length GA and hybrid GA. A specially-adapted variable-length GA called vGA is presented which can better handle discrete structural optimization problems over available steel sections. This method overcomes the difficulties when there is an isolated optimum, as illustrated in the example, involving optimization over AISC W-sections.

A hybrid GA called hGA is also proposed in this chapter for optimization involving continuous variables. This hybrid scheme differs from others presented in the literature in that the hill-climbing method is an operator within a GA, and not something performed after the GA has converged. This combination works very well with faster convergence and better accuracy in the solution, as illustrated in the numerical example.

Chapter 5

Software Implementation of Multicriterion Optimization with Genetic Algorithms

5.1 Introduction and Background

The multicriterion design optimization framework described in Chapter Two was investigated and implemented as part of the CUREe-Kajima project *New Computer Tools for Optimal Design Decisions in the Presence of Risk.* This project involved a collaborative research effort among Caltech, Stanford, and USC to develop an interactive computer tool that partially automates the structural analysis, evaluation and optimization process so that structural engineers can make better design decisions in the presence of uncertain risk.

A software program developed primarily at Caltech called CODA was implemented as a software prototype of the optimal design framework including the new genetic algorithms, vGA and hGA, presented in Chapter 4. CODA is a 32-bit Windows application. Hardware requirements to run the program include the Windows 95 or NT operating system, 16MB of RAM, and approximately 10MB of available disk space. The program was developed primarily using the Microsoft Visual C++ Development System for Windows Version 2.0. Much of the programming of the graphical interfaces of CODA was facilitated with the use of the *Microsoft Foundation Classes.*

CODA is actually the successor of an earlier version called SODA, which was the result of a joint development effort by the three universities involved in the CUREe-

Kajima project. However, existing commercial optimal design software was found to have the same acronym SODA. Thus, the new version of our software containing the GAs was called CODA.

In this chapter, the software prototype CODA is described in detail. The system architecture of CODA is first introduced which is followed by a description of its functionalities. The theory behind how CODA works and its implementation issues are also covered.

5.2 Overview of the CODA System

The design process in CODA begins with a preliminary design and then involves an iterative procedure of analysis, evaluation, and revision. In CODA, there is a software module devoted to each of these tasks. Figure 5.1 shows the overall architecture of CODA.

The three main modules in CODA, which have been described in Section 2.3, are:

- The ANALYZER module uses finite element analysis to compute performance parameter values based on a building configuration specified by the user and on the current values of the design parameters.

- The EVALUATOR, a module based on multicriterion decision theory, fuzzy logic and structural reliability concepts, determines an overall design evaluation measure, or level of acceptability, of the current design based on multiple performance criteria and a treatment of load uncertainties. This is done by aggregating preference values for the current design based on each of the individual design criteria, as described in Chapter 2.

- The REVISER performs revisions of the design to find an optimal design based on maximization of the overall design evaluation measure. Several optimization algorithms, both deterministic and stochastic, can be chosen, including the vGA and hGA algorithms presented in Chapter 4.

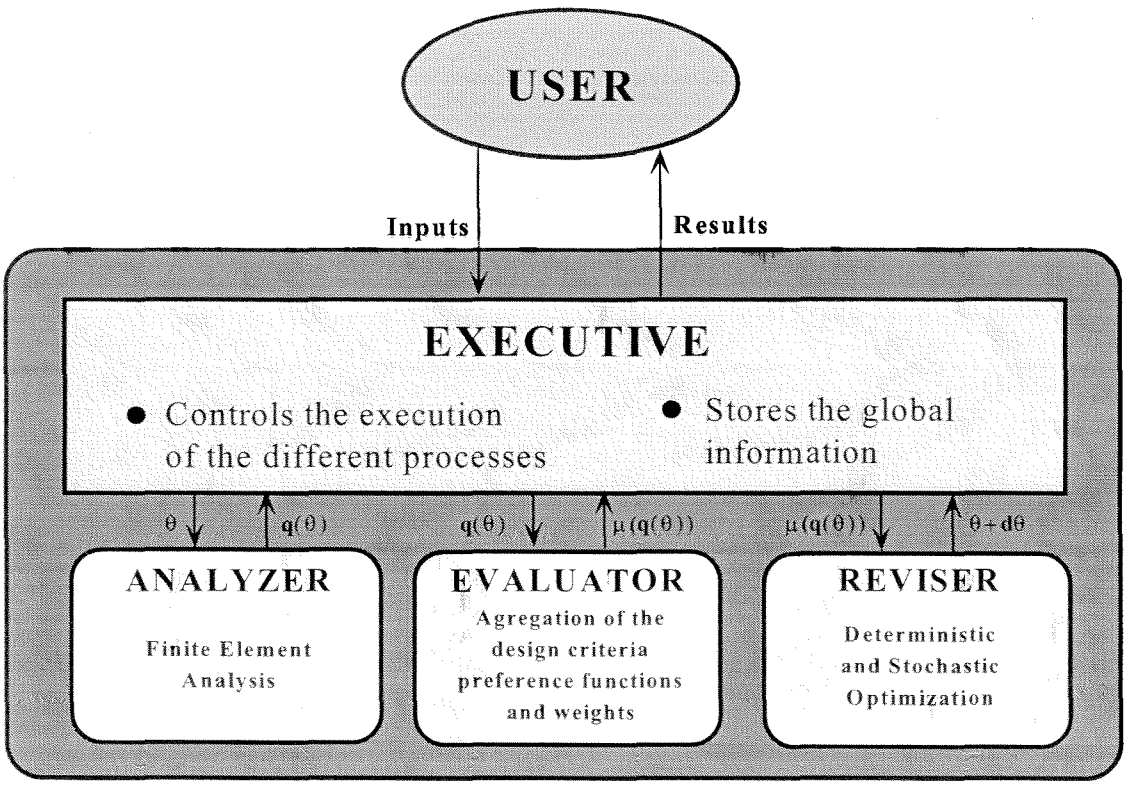


Figure 5.1: Overall system architecture of CODA

In addition, there is an EXECUTIVE module, as shown in Figure 5.1, which has a supervisory role with respect to the other modules (the ANALYZER, the EVALUATOR, and the REVISER). The EXECUTIVE module acts as an interface between these three modules and the user, assisting in the initialization of the modules, controlling the execution of the different processes, and storing the information associated with the analysis, evaluation and optimization so that it is accessible by each of the other modules. The EXECUTIVE also allows the user to view the structure under consideration in graphical form (see Figure 5.2) and to view tabular listings of the structural parameters and analysis results. This centralization of initialization, control and result presentation in the EXECUTIVE makes CODA more modular, since additional features and modifications may be made to the user interface without restructuring the entire software system.

The centralization also facilitates control and monitoring of the numerous processes involved in the execution of the program; in particular, error-checking and error-recovery can be made at each step of the analysis, evaluation or optimization, so that messages can be displayed to the user by the EXECUTIVE when problems arise and recovery from an error can be made without fatal crashing of the program.

The EXECUTIVE allows initialization of the ANALYZER by prompting the user to input the physical configuration of the initial preliminary design, including geometric information and individual member and connection information. In addition, the user must select, from a menu of possibilities, the design and performance parameters important for the design decision-making process. These design and performance parameters are combined with preference functions and weights to express the design criteria in a quantitative form. The design parameters, designated by a vector $\boldsymbol{\theta}$, are those parameters of the initial design which are selected to be varied during the search for an optimal design. In CODA, the design parameters control the geometry of the structural members (e.g., flange width or web depth). On the other hand, performance parameters, designated by a vector $\mathbf{q}$, represent quantities related to the "performance" of the design, and can take the form of conventional structural parameters (e.g., stress, deflection, etc.) or other parameters (e.g., material cost of

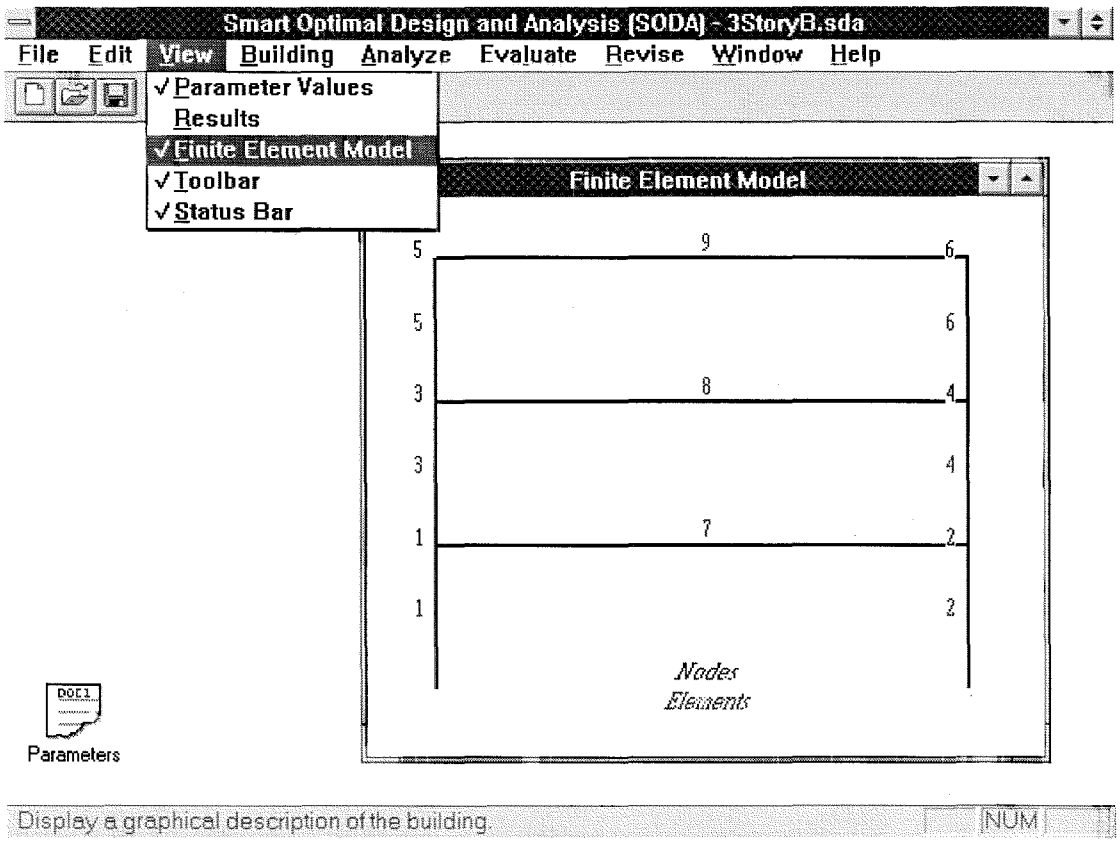


Figure 5.2: Screen dump of CODA with FEM view

the structural system).

The principal role of the ANALYZER is to calculate the performance parameters $\mathbf{q}(\boldsymbol{\theta})$ as a function of the prescribed design parameters, $\boldsymbol{\theta}$. Several types of analyses are available for computing these performance parameters (see Figure 5.3). However, in the case of uncertain loads, the probability density function $p(\mathbf{q}|\boldsymbol{\theta})$ for the corresponding uncertain performance parameters is calculated.

To evaluate the current design, the EVALUATOR requires a user-supplied preference function, μ_i, for each design criterion ($i = 1, \ldots, N_c$), which defines the preference for the various values of each design parameter or performance parameter involved in the criterion. The preference function may simply express a minimum and/or maximum (fuzzy) bound on a design quantity, or it may express a more complex design criterion. A value $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 1$ indicates perfect acceptability of the design prescribed by $\boldsymbol{\theta}$, as judged by the ith design criterion alone; whereas, $\mu_i(\mathbf{q}(\boldsymbol{\theta})) = 0$ indicates absolute unacceptability of the design. Values between 0 and 1 indicate degrees of acceptability or preference between these extreme cases. In addition, the user supplies importance factors or weights, w_i, which indicate the relative importance of the i^{th} design criterion. A large importance factor for a design criterion gives it more influence in the trade-off which occurs between the various criteria during optimization of the design, that is, it indicates that the design should be such that the corresponding preference function value is close to unity. Alternatively, if a design criterion is given a low importance factor, its associated preference function value may be close to zero without greatly affecting the overall design evaluation. All these can be specified in CODA using the dialog box shown in Figure 5.4.

The REVISER takes the overall design evaluation measure, μ, computed by the EVALUATOR from the individual preference function values, $\mu_i (i = 1, \ldots, N_c)$, and revises the design to improve it. In the optimization mode of CODA, the ANALYZER, EVALUATOR and REVISER are repeatedly called by the EXECUTIVE in order to find an optimal design. During the optimization process, a close to real time display of the progress is shown via a dialog box as shown in Figure 5.5.

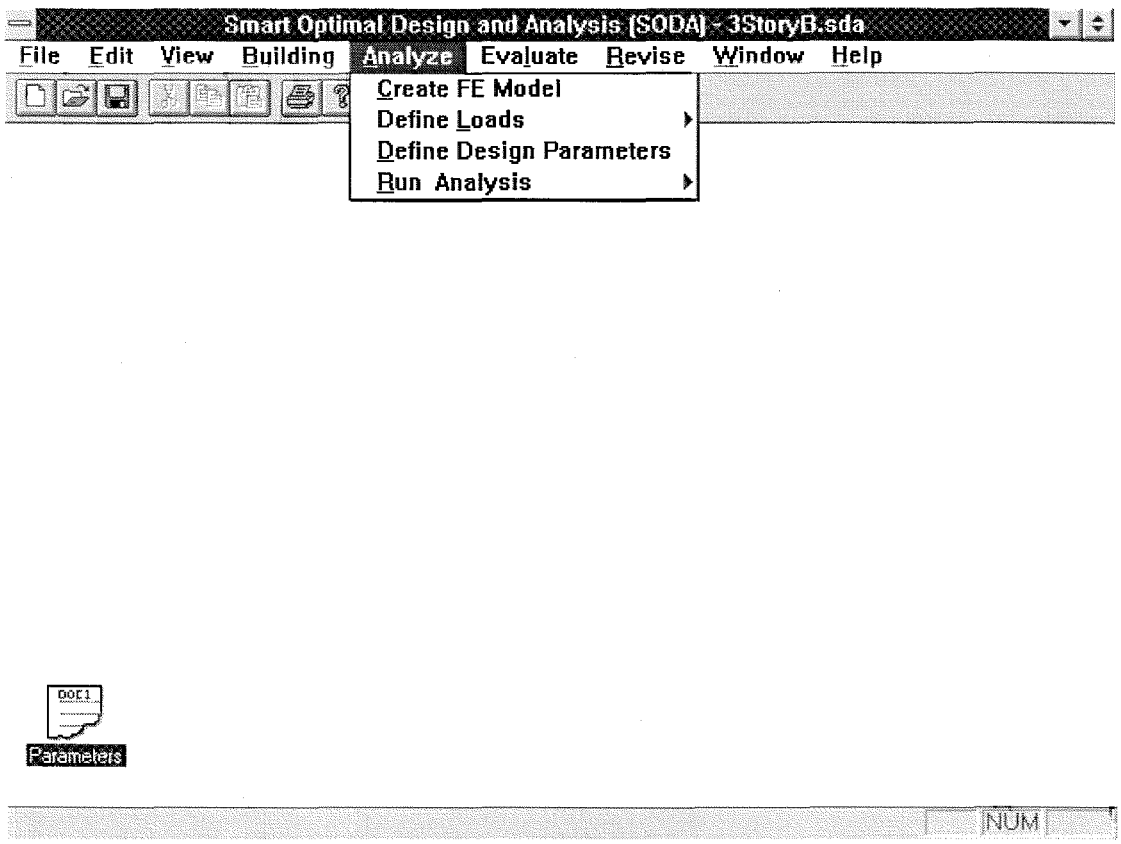


Figure 5.3: Screen dump of ANALYZER menu

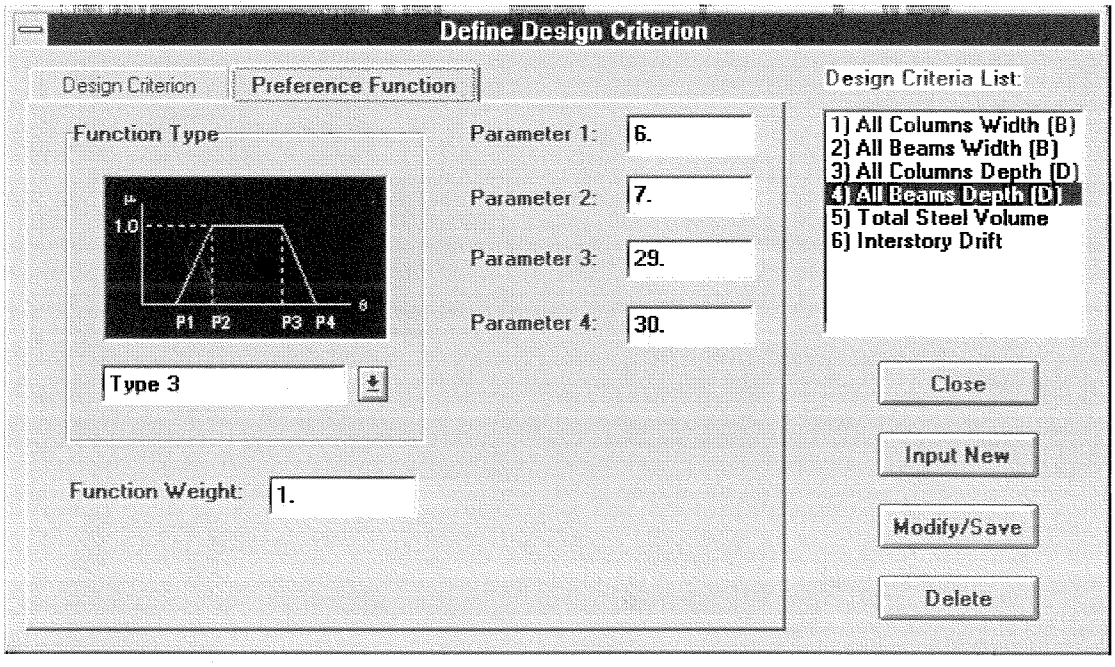


Figure 5.4: Screen dump of EVALUATOR preference function dialog

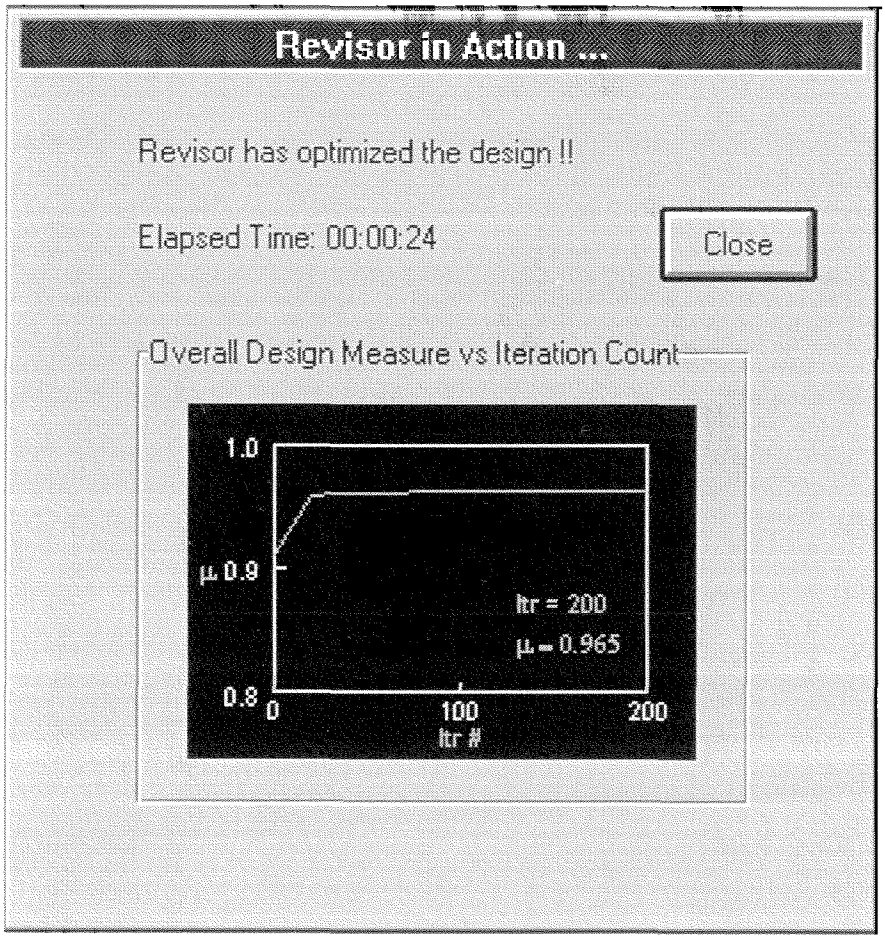


Figure 5.5: Screen dump of REVISER with optimization progress view

5.3 Functionalities

Currently, CODA is capable of handling the following types of analysis on structural systems:

- Linear static analysis with wind, gravity and earthquake loadings (equivalent static and response spectra methods) from 1994 Uniform Building Code (ICBO 1994) using finite element methods.
- Modal analysis using generalized Jacobi transformation method.
- Time history analysis using Newmark numerical integration techniques.

CODA is equipped with both the conservative and the trade-off strategies described in Chapter 2. For REVISER, optimal design of structures can be done at both continuous and discrete design parameter levels with the following algorithms:

- Continuous parameter optimization using quasi-Newton method, adaptive random search and hybrid genetic algorithm (hGA).
- Discrete parameter optimization over commercially available steel sections using varaible-length genetic algorithm (vGA).

5.4 Theory

In this section, the theory behind how CODA works is presented. The discussion here follows very closely that in the CUREe-Kajima project report (Beck, Chan, Irfanoglu, Masri, Smith, Vance, and Barroso 1996). However, some of the lengthy details are not covered here and the interested reader is referred to the report.

5.4.1 The ANALYZER

The role of the ANALYZER is to compute performance parameter values based on the specified design parameters. The performance parameters currently include steel

volume, base shear, maximum displacement, maximum drift, and member axial, shear and bending stress. Steel volume is computed simply by summing up the volume of all the members (member volume = cross-sectional area * length). The remainder of the performance parameters currently considered in CODA are computed using basic finite element methods. The current version of CODA performs linear static and dynamic finite element analyses of planar frames, including earthquake and wind analyses. The following sections discuss the theory of the various CODA analysis capabilities.

Linear Static Analysis: Gravity, Wind, and Earthquake Loads

The equation governing static deformation can be written as:

$$\boldsymbol{K}\,\boldsymbol{x} = \boldsymbol{f} \tag{5.1}$$

where: $\boldsymbol{K}$ = global stiffness matrix

$\boldsymbol{x}$ = nodal displacement vector

$\boldsymbol{f}$ = global force vector.

Two types of elements are available for finite element modeling of structural systems in CODA: 2D beam-column and 2D truss elements. The stiffness matrix for a 2D beam-column element in local coordinates is given by:

$$\boldsymbol{K}_i^l = \frac{EI}{L^3}\begin{bmatrix} \frac{AL^2}{I} & 0 & 0 & -\frac{AL^2}{I} & 0 & 0 \\ 0 & 12 & 6L & 0 & -12 & 6L \\ 0 & 6L & 4L^2 & 0 & -6L & 2L^2 \\ -\frac{AL^2}{I} & 0 & 0 & \frac{AL^2}{I} & 0 & 0 \\ 0 & -12 & -6L & 0 & 12 & -6L \\ 0 & 6L & 2L^2 & 0 & -6L & 4L^2 \end{bmatrix} \tag{5.2}$$

where: $\boldsymbol{K}_i^l$ = i^{th} beam-column element stiffness matrix in local coordinates

E = modulus of elasticity

I = moment of inertia

A = cross-sectional area

L = length.

The stiffness matrix for a 2D truss element in local coordinates is given by:

$$\boldsymbol{K}_i^l = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{5.3}$$

The stiffness matrix of the individual elements are first computed in local coordinates and then converted to global coordinates by:

$$\boldsymbol{K}_i = \boldsymbol{T}^T \boldsymbol{K}_i^l \boldsymbol{T}, \tag{5.4}$$

where: $\boldsymbol{K}_i$ = i^{th} beam-column element stiffness matrix in global coordinates

$\boldsymbol{T}$ = transformation matrix between local and global coordinates

$$\boldsymbol{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{5.5}$$

where θ is the angle between the local and global coordinate systems. The global stiffness matrix of the frame is formed using local destination arrays to combine the individual element stiffness matrices.

Once the stiffness matrix $\boldsymbol{K}$ is formed, the load vector $\boldsymbol{f}$ is computed based on loading input. Currently, gravity loading, wind loading, or earthquake loading are included based on the 1994 UBC equivalent static representation. The gravity loading is defined as load/unit area; separate values may be specified for both the roof and general dead and live loads. The dead and live loads are multiplied by the tributary area, which is the product of the beam length and one half of the out-of-plane dimension, to determine the actual loading values.

The wind load is based on the 1994 UBC prescribed loading. A wind pressure, P, is computed by:

$$P = C_q C_e q_s I_w, \tag{5.6}$$

where C_q = pressure coefficient

C_e = height, exposure, and gust factor

q_s = wind speed

I_w = wind importance factor.

The wind pressure P is multiplied by the corresponding tributary area, which is the product of the building height and one half of the out-of-plane dimension, to determine the actual wind loading values. The equivalent static lateral force procedure is based on the 1994 UBC prescribed loading pattern, where the design base shear V is estimated as follows.

$$V = \frac{ZIC}{R_w} W, \tag{5.7}$$

where Z = seismic zone factor

I = importance factor

C = base shear coefficient (function of soil type and building period)

W = seismically effective weight

R_w = structural system parameter to account for inelastic behavior.

The design base shear given in Equation (5.7) is then distributed to each floor of the building according to the floor height and weight. The distributed base shear forms the static load vector which is used to compute the building displacements, member stresses, etc.

After the dead and live loads have been specified, the next step is to solve for the nodal displacements. Equation (5.1) is solved using LU Decomposition (Strang 1988). Once the nodal displacements have been computed, additional results, including interstory drift values, base shear, and element forces and stresses, can be computed.

Pseudo-Dynamic Analysis: Response Spectra Method

The response spectra method determines seismic response based on modal superposition, calculation of modal participation factors, and the UBC response spectra for particular soil types. First, a free vibration analysis is performed on the following generalized eigenproblem:

$$\omega_i^2 \boldsymbol{M} \boldsymbol{\phi}_i = \boldsymbol{K} \boldsymbol{\phi}_i, \tag{5.8}$$

where: $\boldsymbol{M}$ = lumped mass matrix

ω_i = i^{th} circular natural frequency

$\boldsymbol{\phi}_i$ = i^{th} mode shape.

The above problem is first converted to the standard symmetric eigenvalue problem utilizing a Cholesky decomposition. The generalized Jacobi method, a classical eigensolution transformation method for symmetric eigenvalue problems, is then used to solve for the natural frequencies and modes. The procedure consists of repeated simultaneous transformation of the system stiffness, $\boldsymbol{K}$, and mass, $\boldsymbol{M}$ matrices until they are both reduced to a diagonal form (Craig 1981). Because this generalized Jacobi method is a transformation method, for a finite element model with n degrees of freedom, all n of the model's modes and frequencies are calculated. Once the

natural circular frequencies are computed, the natural periods can be determined by $T_i = 2\pi/\omega_i$.

Since the response spectra method calculations are based only on the horizontal (i.e., lateral) accelerations, only those coefficients corresponding to those degrees of freedom in the modal vectors are used in the calculation of the participation factors. Thus, the modal participation factors are calculated as follows:

$$\beta_i = \frac{\sum_{j=1}^{N} a_j M_j \phi_{ji}}{\sum_{j=1}^{N} M_j \phi_{ji}^2}, \tag{5.9}$$

where β_i = i^{th} modal participation factor

M_j = j^{th} diagonal coefficient of the frame mass matrix

ϕ_{ji} = j^{th} coefficient of the i^{th} mode shape

a_j = j^{th} element of mapping vector for picking only the horizontal degrees of freedom

N = number of degrees of freedom.

Using the specified soil type information, CODA selects one of the normalized UBC response spectra shapes. A spectral acceleration, S_{a_i}, is selected for each mode based on that mode's natural period, T_i. Story accelerations are then calculated as follows:

$$\ddot{x}_{ji} = \beta_i \phi_{ji} S_{a_i}, \tag{5.10}$$

where $\ddot{x}_{ji}$ = acceleration of the j^{th} coordinate of the i^{th} modal vector.

Utilizing the story accelerations, modal lateral forces and base shear are calculated utilizing the following:

$$F_{ji} = \ddot{x}_{ji} M_j \tag{5.11}$$

$$V_{0i} = \sum_{j=1}^{N} F_{ji} \tag{5.12}$$

where F_{ji} = inertial force for j^{th} coordinate of the i^{th} modal vector

V_{0i} = base shear for i^{th} mode.

These modal quantities are then combined by taking the square root of the sum of the squares (SRSS) over the number of modes, n, specified by the user:

$$F_j = \left(\sum_{i=1}^{n} F_{ji}^2 \right)^{1/2} \tag{5.13}$$

$$V_0 = \left(\sum_{i=1}^{n} V_{0i}^2 \right)^{1/2} \tag{5.14}$$

where F_j = inertial force for j^{th} coordinate

V_0 = base shear

n = number of modes.

If the calculated base shear for the structure is below the value input by the user, then all lateral forces are scaled so as to meet that requirement.

Dynamic Analysis: Time Domain Integration

In addition to the equivalent static earthquake analysis capability, CODA can calculate the dynamic seismic response of a linear system by using a time domain integration technique. The finite element equation of motion for support-excited structures can be written as:

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{C}\dot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}\ddot{x}_g, \tag{5.15}$$

where $\boldsymbol{M}$ = lumped mass matrix

$\boldsymbol{C}$ = damping matrix

$\ddot{\boldsymbol{x}}$ = nodal acceleration vector

$\dot{\boldsymbol{x}}$ = nodal velocity vector

$\boldsymbol{f}$ = vector mapping ground acceleration to horizontal degrees of freedom

$\ddot{x}_g$ = ground acceleration.

CODA generates a proportional viscous damping matrix by utilizing Rayleigh damping as shown below:

$$\boldsymbol{C} = a_0\boldsymbol{M} + a_1\boldsymbol{K} \tag{5.16}$$

where: a_0 = coefficient for the mass proportional term

a_1 = coefficient for the stiffness proportional term.

The required coefficients are calculated utilizing the user-specified critical damping ratio for the first two modes, which are assumed to be the same. The relationship between these factors and the critical damping ratios is:

$$\begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = 2\frac{\omega_1\omega_2}{\omega_2^2 - \omega_1^2} \begin{bmatrix} \omega_2 & -\omega_1 \\ -1/\omega_2 & 1/\omega_1 \end{bmatrix} \begin{Bmatrix} \xi_1 \\ \xi_2 \end{Bmatrix}, \tag{5.17}$$

where ω_i = circular natural frequency of the i^{th} mode

ξ_i = damping ratio of the i^{th} mode

The time domain integration procedure is performed using the full finite element model of the building. Either the average or linear acceleration methods of the

Newmark-Beta numerical integration procedures can be used to perform the analyses. Detailed development of these two methods can be found in most standard structural dynamics texts and is not included here (Craig 1981). However, it is important to realize that, while the average acceleration method is *unconditionally* stablethe time step size , the linear acceleration method is *conditionally* stable. Thus, the linear acceleration method's ability to converge to an accurate solution will depend on the specified time step size, where the condition for stability is given by:

$$\delta t / T_N \leq \sqrt{3}/\pi, \tag{5.18}$$

where δt = time step size

T_N= natural period of N^{th} mode (i.e., smallest natural period of modes participating in response).

5.4.2 EVALUATOR

The role of the EVALUATOR in CODA is to provide an overall design evaluation measure $\mu(\boldsymbol{\theta})$ for the design specified by the current values of the design parameters $\boldsymbol{\theta}$. As described in Chapter 2, the overall design measure $\mu(\boldsymbol{\theta})$ is computed from a list of design criteria specified by the user. Each of the design criteria is associated with a preference function μ_i and an importance weight w_i. Currently, preference functions in CODA are limited to function of a single design or performance parameter. The computation of $\mu(\boldsymbol{\theta})$ is done by aggregating preference values of the design criteria using an aggregation strategy. Two aggregation strategies are available in CODA: conservative strategy and multiplicative trade-off strategy.

CODA can handle both deterministic and stochastic design criteria. Several deterministic criteria are available which are based on maximum stresses, interstory drift, lateral deflection, base shear, etc. Evaluation of these deterministic criteria can be done in a straightforward manner by following the concepts described in Chapter 2. In the stochastic case, only design criteria involving uncertainties due to seismic loads are available at this point although extensions to other types of uncertain loads

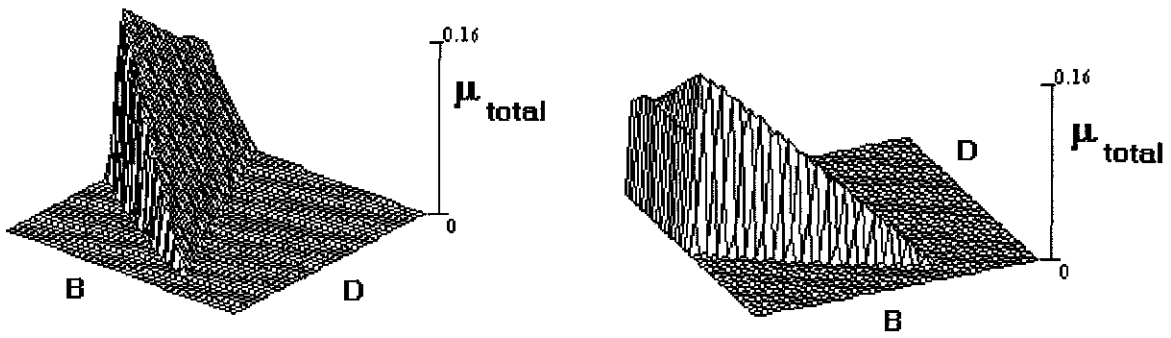


Figure 5.6: Surface plot of $\mu(\boldsymbol{\theta})$ for the conservative strategy

as well as uncertainties in models is very straightforward. Computation of preference values of these criteria is based on the theory of stochastic criteria described in Chapter 2. However, the theory behind the exact computational procedure is very involved and since seismic reliability and its related design issues are not the main thrust of this thesis, they will not be covered here. The interested reader can consult (Beck et al. 1996).

Discussion of the Aggregation Strategies

Recall that the conservative aggregation strategy can be written as:

$$\mu = \min(\mu_1{}^{n_1}, \mu_2{}^{n_2}, ..., \mu_{N_c}{}^{n_{N_c}}) \tag{5.19}$$

where $n_i = w_i / \max_j w_j$, $i = 1, ..., N_c$ and w_i is a positive importance weight assigned to the i^{th} design criterion. Consider the $\mu(\boldsymbol{\theta})$ surface plot shown in Figure 5.6 for this strategy. This surface is based on a three-story steel frame example subject to UBC wind loading. Two design parameters were chosen for this example: the flange width B and the web depth D for all the beams and columns, which are constrained to be identical I-beams.

It can be seen from Figure 5.6 that the surface is characterized by sharp edges

and the maximum value of $\mu(\boldsymbol{\theta})$ is at the intersection of these edges. The edges correspond to the equality of the preference functions for two design criteria and they are transition curves where there is a switch in which design criterion is giving the smallest preference value, and so giving the value of the overall design evaluation measure.

These sharp edges in the $\mu(\boldsymbol{\theta})$ can produce numerical difficulties in performing numerical optimization. For instance, hill climbing algorithms move from a point on the surface corresponding to the initial choice of the design parameters up a sloping face until they reach the sharp ridge. After that they are unable to efficiently move up the ridge to the peak value of the surface because of the discontinuous slope at the ridge. However, stochastic optimization schemes which do not depend on any local topological information of the surface can be applied to determine the maximum of the surface.

On the other hand, the multiplicative trade-off strategy, as given by Equation 5.20, produces a much smoother $\mu(\boldsymbol{\theta})$ surface with no sharp edges (see Figure 5.7).

$$\mu = {\mu_1}^{m_1} {\mu_2}^{m_2} \ldots {\mu_{N_c}}^{m_{N_c}}. \tag{5.20}$$

where $m_i = w_i / \sum_j w_j$.

The plotted surface is for the same example as in Figure 5.6 except that the trade-off aggregation strategy is used instead of the conservative strategy. Notice that the $\mu(\boldsymbol{\theta})$ surface in Figure 5.7 is very steep near the boundaries which are the same $\mu = 0$ as in Figure 5.6. In fact, the slope of the surface at the boundaries is theoretically infinite. These steep boundary slopes can cause hill climbing algorithms to quickly move towards the interior of the "island" if the algorithms are started near the boundary.

For the case illustrated in Figure 5.7, the importance weights w_i in Equation 5.20 were set to unity for each design criterion. A more aggressive design as far as reducing the total steel volume is concerned, can be achieved by increasing the importance weight for this design criterion. Figure 5.8 shows how the $\mu(\boldsymbol{\theta})$ surface changes when

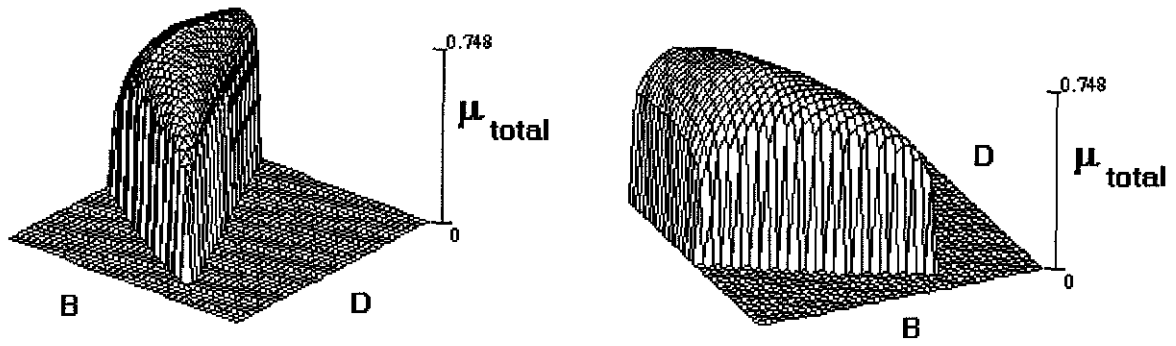


Figure 5.7: Surface plot of $\mu(\boldsymbol{\theta})$ for the trade-off strategy with all importance weight $w_i = 1$

the steel volume importance weight is increased to 10 while the other importance weights are kept at their original values of unity. A comparison of Figure 5.7 and Figure 5.8 shows that this change in importance weights pushes the peak of the surface, and hence the optimal design, towards smaller values of B and D, as desired.

5.4.3 REVISER

Given the current design $\boldsymbol{\theta}$, the role of the REVISER in CODA is to improve this design based on the specified design criteria. Basically, this involves solving the optimization problem of the function $\mu(\boldsymbol{\theta})$. This optimization problem is an unconstrained one as constraints are specified as design criteria or soft constraints with the use of preference functions. Three different optimization techniques are employed in CODA: quasi-Newton method with BFGS updating, adaptive random search method and genetic algorithms. Below are descriptions of each of these three optimization techniques.

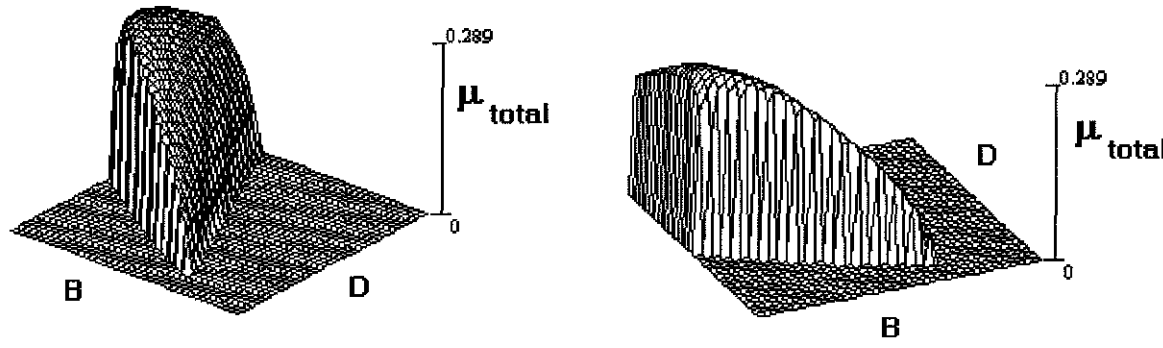


Figure 5.8: Surface plot of $\mu(\boldsymbol{\theta})$ for the trade-off strategy with steel volume importance weight $w_i = 10$ and all other $w_i = 1$

Quasi-Newton Method with BFGS updating

Given the function $f(\boldsymbol{x})$, the Newton's iterative method for finding the local optimum is given by:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{\delta}^k \tag{5.21}$$

$$\boldsymbol{\delta}^k = -[\boldsymbol{H}^k]^{-1}\nabla f(\boldsymbol{x}^k) \tag{5.22}$$

where $\boldsymbol{x}^k$ = the parameter vector $\boldsymbol{x}$ in the kth iteration

$\boldsymbol{H}^k$ = the Hessian matrix of $f(\boldsymbol{x})$, defined by $\boldsymbol{H}(\boldsymbol{x}) = [\nabla\nabla f(\boldsymbol{x})]$, in the kth iteration.

Note that this method requires both the first and second derivatives of the function f. In addition, the inversion of the Hessian matrix must be computed at each step. To improve the efficiency, an approximation to the inverse of the Hessian matrix is used instead. The Newton's method is modified as follows:

1. Start by taking $\mathcal{H}^0 = \boldsymbol{I}$, the identity matrix.

2. Compute $\boldsymbol{s}^k = -\mathcal{H}^k\nabla f(\boldsymbol{x}^k)$.

3. Compute $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \boldsymbol{s}^k$ where α is computed by minimizing f using a line search routine along the direction $\boldsymbol{s}^k$.

4. Update $\mathcal{H}^k$ to $\mathcal{H}^{k+1}$.

Here, $\mathcal{H}$ represents the approximation to the inverse of the Hessian matrix or the pseudo inverse Hessian. Several procedures exist for updating $\mathcal{H}$ such as Davidon-Fletcher-Powell (Fletcher and Powell 1963) and Broyden-Fletcher-Goldfarb-Shannon (Broyden 1970; Fletcher 1970; Goldfarb 1970; Shanno 1970), also known as DFP and BFGS updating methods. The BFGS updating scheme is chosen in CODA and is given by:

$$\mathcal{H}^{k+1} = \left\{ \mathcal{H} + \left(1 + \frac{\boldsymbol{\gamma}^T \mathcal{H} \boldsymbol{\gamma}}{\boldsymbol{\delta}^T \boldsymbol{\gamma}} \right) \frac{\boldsymbol{\delta}\boldsymbol{\delta}^T}{\boldsymbol{\delta}^T \boldsymbol{\gamma}} \left(\frac{\boldsymbol{\delta}\boldsymbol{\gamma}^T \mathcal{H} + \mathcal{H}\boldsymbol{\gamma}\boldsymbol{\delta}^T}{\boldsymbol{\delta}^T \boldsymbol{\gamma}} \right) \right\}^k \tag{5.23}$$

where $\boldsymbol{\gamma}^k = \nabla f(\boldsymbol{x}^k)$

$\boldsymbol{\delta}^k = -\mathcal{H}^k \nabla f(\boldsymbol{x}^k)$.

An overall flowchart of this quasi-Newton method is shown in Figure 5.9.

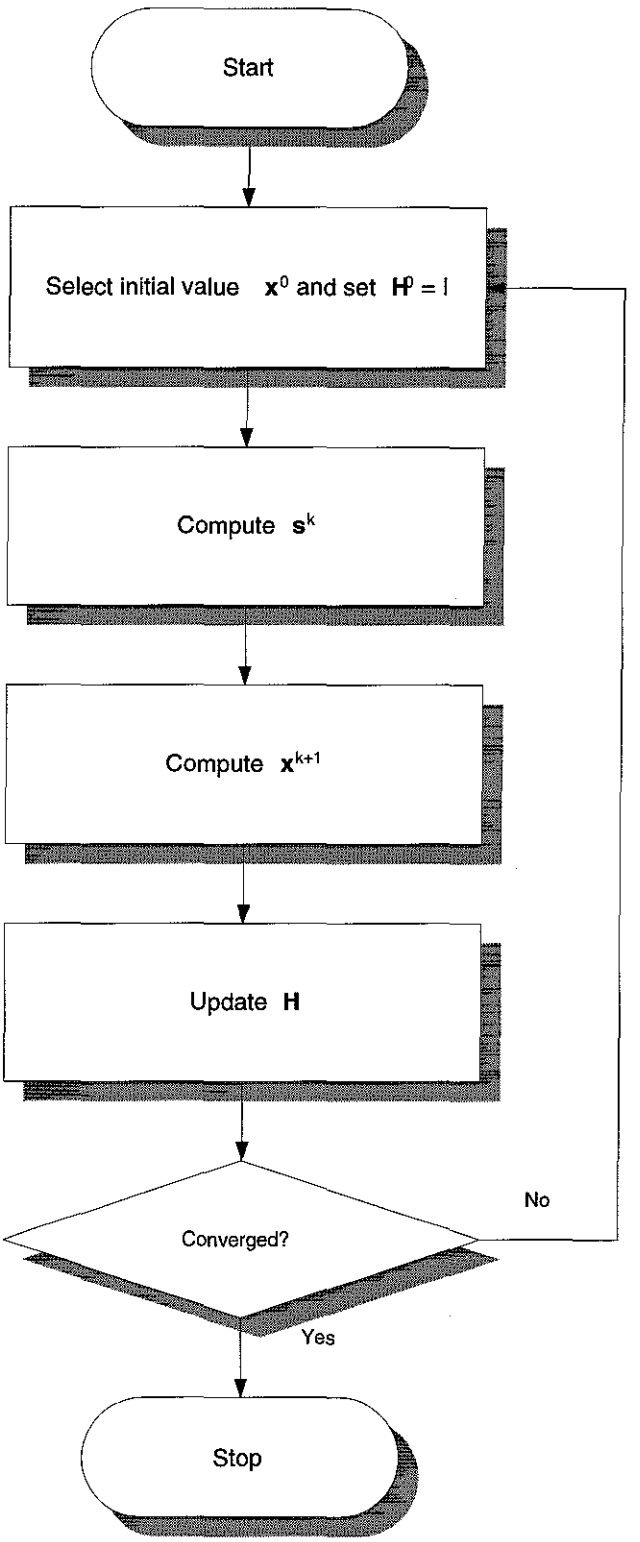


Figure 5.9: Overall flowchart of quasi-Newton method with BFGS updating

Adaptive Random Search

While quasi-Newton methods or any deterministic methods are very efficient in finding local optima for optimization problems involving a few parameters, they become computationally too demanding for high order systems with many parameters. For these problems, random search methods are more appropriate for three major reasons:

1. The speed of convergence is independent of the dimensionality of the parameter space, at least in principle.

2. The success of the method is largely independent of the degree of nonlinearity of the system.

3. The method can succeed in the presence of multiple minima.

The basic random search algorithm for the maximization of an objective function $f(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, x_2, ..., x_n)^T$ is a vector of unknown parameters, proceeds as follows:

1. An initial point $\boldsymbol{x}^0$ in the search space is chosen and $f(\boldsymbol{x}^0)$ is evaluated.

2. Trial points $\boldsymbol{x}^i \in \Omega_{\boldsymbol{x}}$, where $\Omega_{\boldsymbol{x}}$ is the given permissible region in the n-dimensional parameter space, are selected from an appropriate probability density function defined over $\Omega_{\boldsymbol{x}}$.

3. A successful point $\boldsymbol{x}^{i+1}$ is one for which $f(\boldsymbol{x}^{i+1}) < f(\boldsymbol{x}^i)$.

In general, random search algorithms exhibit convergence in probability, i.e.,

$$P\{|\boldsymbol{x}^n - \hat{\boldsymbol{x}}| > \epsilon\} \to 0 \quad for \quad n \to \infty \tag{5.24}$$

where $\hat{\boldsymbol{x}}$ = the optimum of $f(\boldsymbol{x})$

n = the iteration count.

While convergence in probability is a weak form of convergence, it is important to note that it applies in the presence of multiple minima and nondifferentiability of $f(\boldsymbol{x})$.

Rather than using the "pure random search" outlined above, most algorithms are based on a "random creep" procedure in which exploratory steps are confined to a hypersphere centered about the latest successful point $f(\boldsymbol{x}^k)$. However, convergence is highly dependent on the size of the hypersphere in relation to the local topology of the parameter space. If the steps are too small, convergence may be extremely slow; if the steps are too large, overshoot is possible since no allowance is made for variations in the nature of the criterion function surface as the search progresses toward a minimum.

In order to circumvent the slow convergence rate of conventional random creep procedures, an adaptive random search technique (Masri, Bekey, and Safford 1980) is used in CODA. This approach periodically optimizes the variance of the step-size distribution. By searching over a variance range of many decades, the algorithm finds the step-size distribution that yields the best local improvement in the objective function. The variance search is then followed by a specified number of iterations of local random search where the step-size variance remains fixed. Periodic wide-range searches are introduced to ensure that the process does not stop at a local minimum. The following list outlines the steps of adaptive random search:

1. Select initial guess $\boldsymbol{x}^0$.

2. Choose a sequence of k standard deviations, $\boldsymbol{s} = \{s_1, s_2, ..., s_k\}$, to cover as wide a range as desired.

3. Start with $\boldsymbol{x} = \boldsymbol{x}^0$ and $\sigma = s_1$, perform N function evaluations of global random search, with variance σ^2.

4. Repeat step (3) successively with $\sigma = s_2, s_3, ...s_k$.

5. Determine k^* such that random search with $\sigma = s_{k^*}$ yields best function values.

6. Perform M iterations with standard deviation s_{k^*} and store the optimal value in $\boldsymbol{x}$.

7. Repeat steps (3) to (7) until convergence tolerance is satisfied.

8. Terminate the search.

The overall flow of this algorithm is summarized in Figure 5.10.

Genetic Algorithms

The last class of optimization schemes implemented in CODA is genetic algorithms. Two different GAs are available: vGA for discrete optimization and hGA for continuous optimization. Since we have covered in detail both general concepts of genetic algorithms and also those specific to vGA and hGA in the last few chapters, the theories behind the two genetic algorithms will not be repeated here.

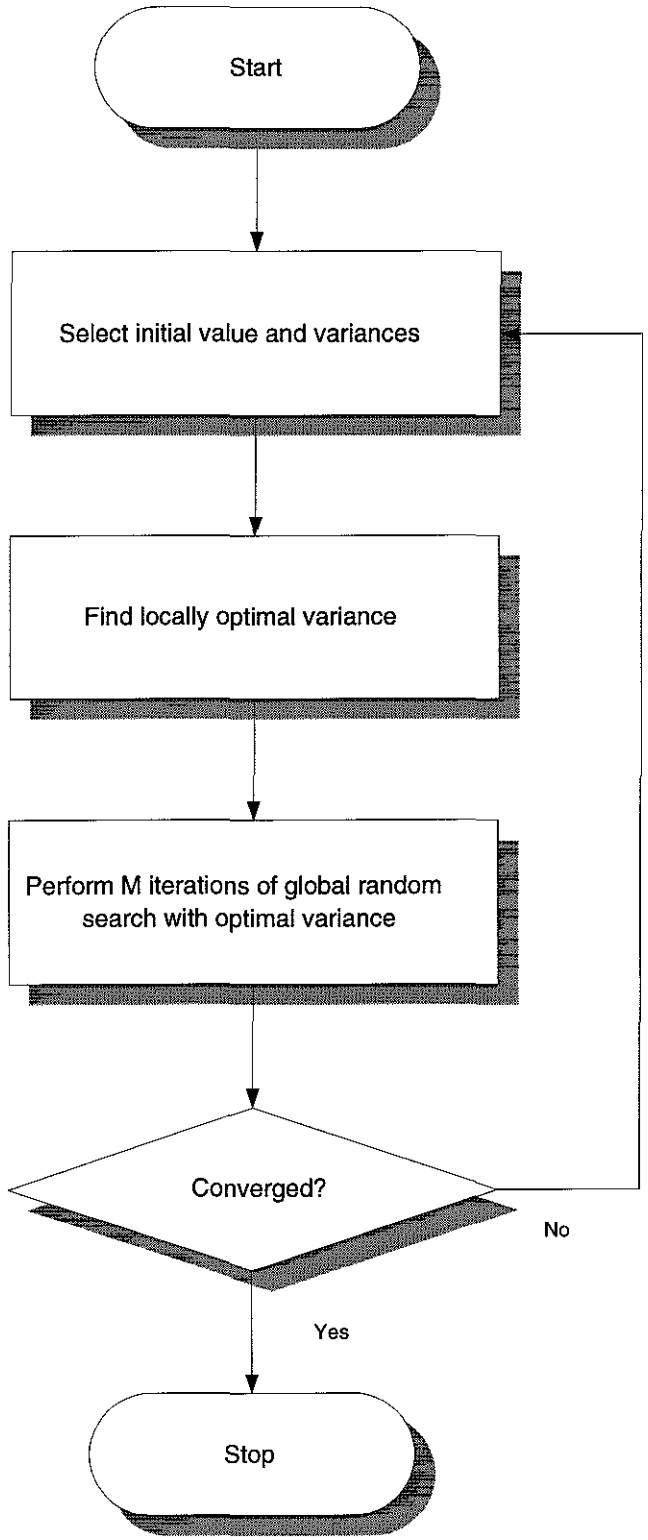


Figure 5.10: Overall flowchart of an adaptive random search algorithm

5.5 Implementation Issues

In this section, the implementation aspect of CODA is discussed. In particular, a relatively new programming approach called object-oriented programming (OOP) is introduced.

5.5.1 Object-Oriented Programming

The *CODA Optimal Design and Analysis* System was developed in C++ following an object-oriented programming style. Object-oriented programming (OOP) is a relatively new style of programming that establishes objects to represent and organize the information utilized by the program (Dym and Levitt 1991). CODA was implemented in this style to ensure that the code is modular and extendable. Modularity allows for easy addition and modification of features in the future without having to restructure the software package, which greatly enhances programming efficiency.

The object-oriented programming paradigm involves developing objects of classes to store information. A *class* may be anything from a physical object such as a *building* or an idea such as *optimization.* Each class contains certain *attributes* and *member functions. Attributes* simply refer to the characteristics or data which define the corresponding object, while *member functions* refer to the behavior or actions of the object. As an example, a class could be established to represent the *Beams* in a building. The attributes of *Beams* could include *material properties (yield strength and modulus of elasticity), location, and section properties (moment of inertia and cross-sectional area).* Member functions or behaviors associated with the *Beams* class could include *Compute Volume* and *Compute Moment of Inertia.* Numerous classes were developed specifically for CODA. Each class has an associated header file (*.h) and source file (*.cpp). The header file contains the class declaration, the object attributes, and the member function declarations. The source file (*.cpp) contains the actual code associated with each of the object member functions. Figure 5.11 shows a typical header file for a class. The classes developed for the separate modules (the ANALYZER, EVALUATOR, and REVISER) are discussed in further detail later

```
#include <iostream.h>

class Material {
    double emodulus;
    double mu;
    double Fy;

    double GetE();
    double Getmu();
    double GetFy();
};
```

Figure 5.11: A Typical Class Header File - Material Class

in this section.

The key concepts which define object-oriented programming are *abstraction*, *encapsulation*, and *inheritance* (Lippman 1991). A higher level of abstraction makes it possible to ignore many of the details associated with the problem and focus on the more important characteristics. A high level of abstraction is obtained by using classes to store the data; at first it may seem tedious to develop classes, but once created, these classes may be used to develop a more transparent and readable code. The high level of abstraction associated with OOP and classes allows the programmer to focus on the function of the program rather than on the details regarding each piece of actual data. At the very least, OOP forces the programmer to thoroughly consider and organize all aspects of the program in a modular fashion.

The association of attributes and behaviors with objects is known as *encapsulation*, another feature of object-oriented programming. By encapsulating both the attributes and behaviors of an object, the linear style of conventional computer programs may be avoided. Rather than having to sequentially call subroutines, the OOP paradigm involves "jumping" between different processes by calling the member functions of the appropriate objects. This less constraining style is more flexible and easier to maintain.

Another feature of object-oriented programming style is *inheritance.* If two classes are closely related they can share or *inherit* attributes and member functions. Class hierarchies are often drawn to show the relationships and inheritances among objects. As an example, a *Finite Element Model* class hierarchy has been developed and is shown in Figure 5.12. The finite element model contains two basic classes, *Nodes* and *Elements*, which store the necessary model information. Two different types of finite elements have been implemented: a beam-column and a truss element. In order to capture the difference between these two types of elements, the *Elements* class is then refined into a *Beam-Column Element* class and a *Truss Element* class. Beam-column and truss elements have many characteristics in common, but also have certain distinct features. As shown in Figure 5.12, common characteristics of both types of elements are inherited from the *Elements* class including locations and the associated members. Each of the element types also has certain specific characteristics, however; for example, beam-column elements have six degrees of freedom and a 6x6 stiffness matrix, while truss elements have four degrees of freedom and a 4x4 stiffness matrix. These distinct features are specified at the *Beam-Column Element* class and *Truss Element* class level. These classes also have similar behavior, such as the need to compute their respective stiffness and mass matrices. Though the existence of this behavior is defined in the *Elements* class, the specific algorithm needed for each class is specified at the *Beam-Column Element* class and *Truss Element* class level. This feature allows the user to simply call the function and allow the object to determine the algorithm needed.

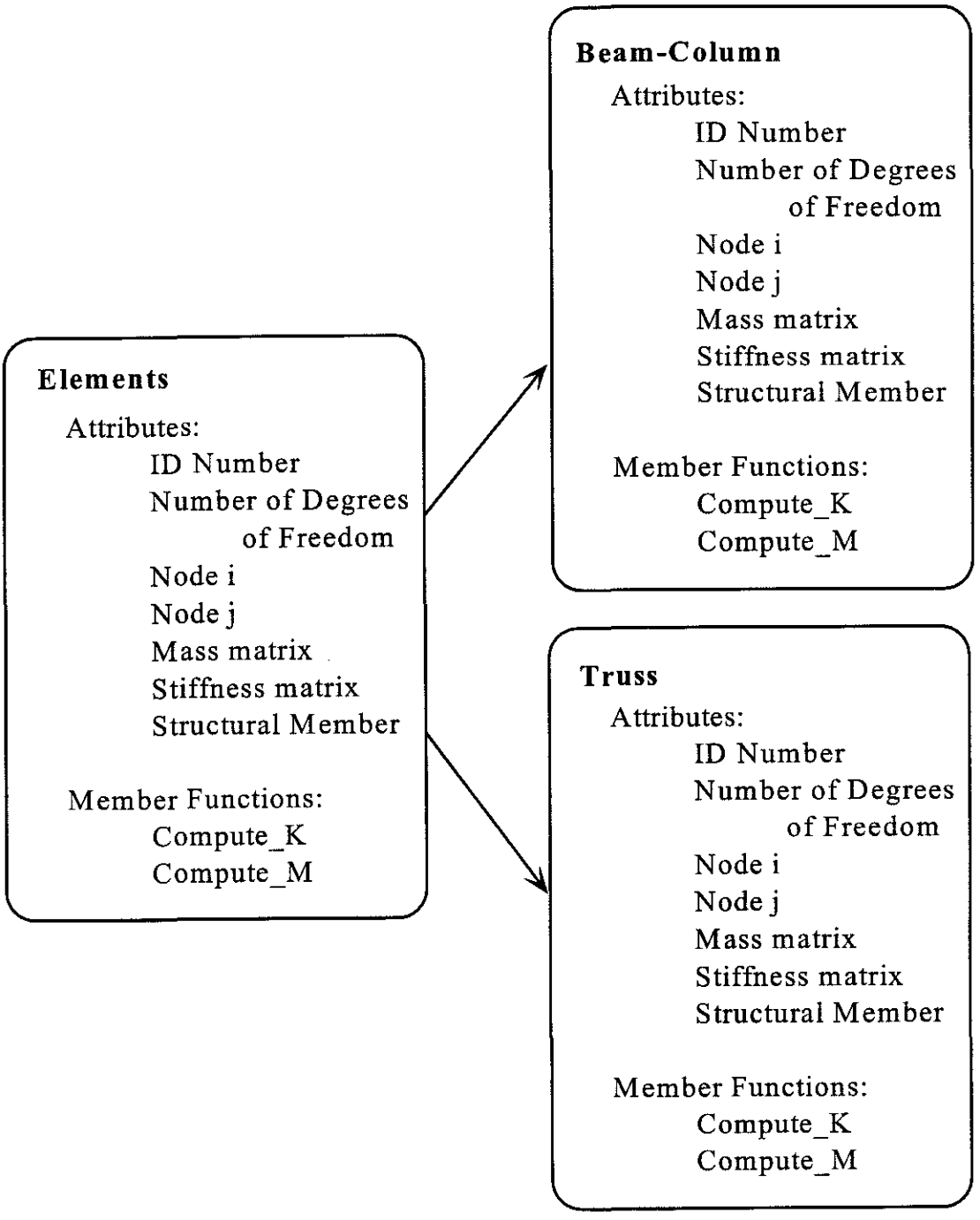


Figure 5.12: A simple finite element class hierarchy

The prototype CODA is written entirely in C++ code. The program was developed primarily using the Microsoft Visual C++ Development System for Windows Version 2.0. The classes created for the CODA Application were derived from Microsoft Foundation Classes (MFC). This class library consists of many classes which encapsulate a lot of details involved in Windows graphical user interface programming. By deriving classes from the MFC library, many features of the program including menus, dialog boxes, and graphical views, can be more easily and rapidly generated. The tools available in the Microsoft Visual C++ environment also facilitated the development of the program. The initial framework of the program was created using the *AppWizard*. Dialog boxes and other resources were then created using the *AppStudio*, while classes were created using the *VisualWorkbench* and the *ClassWizard*. The *ClassWizard* also facilitated the mapping of the variables and controls into actual pieces of code.

Numerous classes have been developed for CODA. The classes and class hierarchies associated with each of the three separate modules of the program are discussed briefly below.

5.5.2 Implementation of the ANALYZER, EVALUATOR and REVISER

The ANALYZER

The main class created for the ANALYZER is *Analysis*. The following information is included in the *Analysis* class:

- Finite Element Model
  - Elements: *Location, Associated Member, Stiffness Matrix, Element Number, Resulting Displacements, Forces, and Stresses, etc.*
  - Nodes: *Location, Associated Connection, Node Number, Nodal Loads, etc.*
- Load Vector
- Stiffness Matrix

- Mass Matrix
- Gravity, Wind and Earthquake Loads
- Time History Parameters
- Free Vibration Response
- Results

Much of the information above is actually stored as classes within the *Analysis* class. For example, separate classes have been developed to store the Element and Node information as well as the Results. The following values are stored in the *Results* class:

- Steel Volume,
- Maximum Deflection: *Value and Node where the maximum deflection occurs,*
- Maximum Drift: *Value and Story in which the maximum drift occurs,*
- Maximum Column Axial, Shear, and Bending Stress: *Values and Elements in which the maximum stresses occurs,* and
- Maximum Beam Axial, Shear, and Bending Stress: *Values and Elements in which the maximum stresses occurs.*

In addition to the characteristics listed above, the *Analysis* class also contains member functions which perform the actual finite element analysis. The following member functions or subroutines are part of the *Analysis* class:

- CreateFEModel(): *Create a Finite Element Model based on the Physical Building Description,*
- Assign_LDA(): *Assign Local Destination Array Values to each Element based on the Geometric Configuration,*
- Assemble_K(): *Assemble the Global Stiffness Matrix for the Frame,*

- Assemble_M(): *Assemble the Global Mass Matrix for the Frame from information about applied dead loads,*
- ComputeGravityLoads(): *Compute the Nodal Gravity Loads Based on the Distributed Loads Input by the user,*
- ComputeWindLoads(): *Compute the Wind Loading Based on the 1994 Uniform Building Code,*
- ComputeStaticEqLoads(): *Compute the Equivalent Static Earthquake Loads Based on the 1994 UBC Uniform Building Code,*
- ComputeResSpecEqLoads(): *Compute the Pseudo-Dynamic Earthquake Loads Based on the 1994 UBC Uniform Building Code Normalized Response Spectra Curves,*
- Solve(): *Solve for the Nodal Displacements Using LU Decomposition,* and
- ComputeResults(): *Compute the Steel Volume, Forces, Stresses, etc. in the Frame.*

Details of the ANALYZER classes and their member functions can be found in the CUREe-Kajima project report (Beck et al. 1996).

The EVALUATOR

The EVALUATOR in CODA is mainly made up of several classes derived from the *CObject* class in the Microsoft Foundation Classes (MFC). Figure 5.13 shows the hierarchy of these classes. *Theta, Performance,* and *Criteria* classes, which represent design parameters, performance parameters and design criteria, are derived from the NameObject class. *Performance* is, in turn, the base class for *ReliPerformance.* Having these classes derived from *CObject* has the advantage of inheriting capabilities such as serialization and runtime information. Besides these classes, three other classes, *DesignParmList, PerfParmList,* and *DesignCritList,* are defined which are

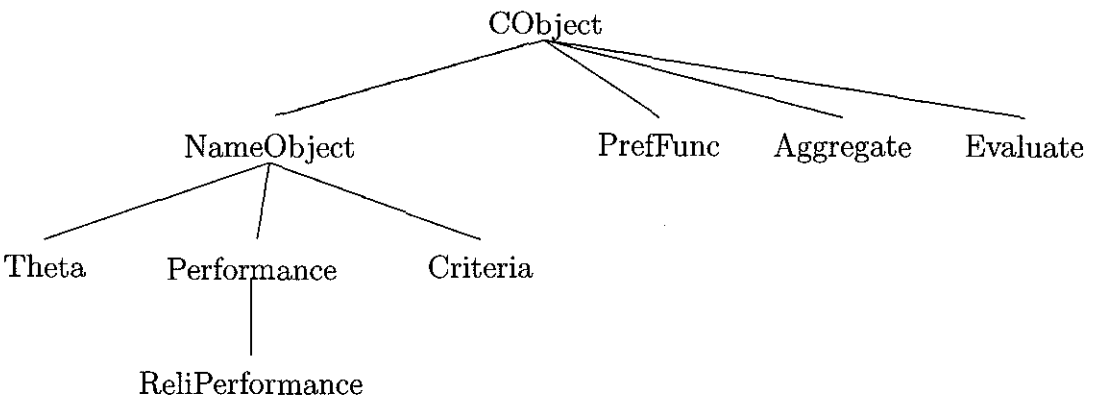


Figure 5.13: Object Hierarchy Tree for EVALUATOR

container classes for objects of *Theta, Performance, Reliperformance* and *Criteria* classes.

The intermediate derived class of *CObject* is called *NameObject* and is mainly used as the base class of other classes. Attributes such as *ID number* and *description* of the object and methods to access them are added to the class definition. These attributes and methods are common to both design and performance parameters as well as design criteria. As such, classes *Theta, Performance* and *Criteria*, are derived from *NameObject* to inherit these properties. An instance of the *NameObject* class consists of the following:

Attributes:

- ID number (*id*),
- description of the object *name*,
- initial and current values of the object,
- object type (Theta, Performance or Criteria),

Methods:

- set and get *id*,
- set and get name *name*,
- set and get initial and current values,

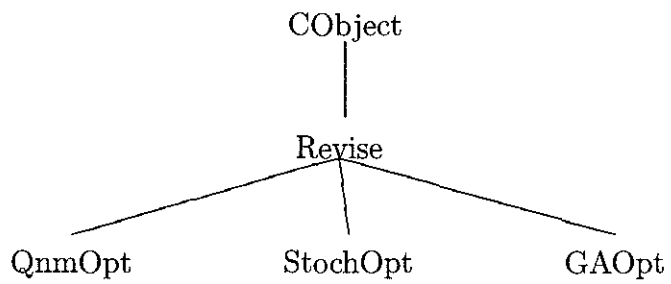


Figure 5.14: Object Hierarchy Tree for REVISER

- an assignment operator =, which allows copying one object to another, and
- serialize (save and retrieve) information of the object.

The REVISER

The REVISER module in CODA is mainly made up of four C++ classes: *Revise, QnmOpt, StochOpt* and *GAOpt*. Figure 5.14 shows the hierarchy of these classes. Similar to the *Analysis* and the *Evaluate* classes, the *Revise* class is inherited from the *CObject* class of the *Microsoft Foundation Class Library* to inherit capabilities such as serialization and runtime information. This *Revise* class is designed as a parent class for optimization from which *QnmOpt, StochOpt* and *GAOpt* classes are derived from. The *QnmOpt* class is the object-oriented encapsulation of the Quasi-Newton optimization method, while the *StochOpt* and *GAOpt* classes encapsulate the two stochastic optimization schemes: adaptive random search and genetic algorithm. Much of the communications between the optimization objects and the rest of CODA (mainly the EXECUTIVE) is encapsulated in the *Revise* class. As such, all these optimization classes contain only optimization specific coding with virtually no dependence on other parts of the code. Therefore, adapting a new optimization scheme to CODA is as simple as inheriting a class from the *Revise* class and writing member functions to perform the necessary optimization procedures. The attributes and member functions of the *Revise* class are listed in the following:

Attributes:

- number of design parameter (or dimension of the design parameter space)
- current values of $\boldsymbol{\theta}$
- domain of the design parameter space
- current overall design measure $\mu_{overall}$
- file objects for output of optimization messages and data (for debugging and result checking purposes).

Methods

- method to initialize all the attributes,
- function to communicate with the EXECUTIVE to perform one cycle of analysis and evaluation to compute new overall design measure $\mu_{overall}$,
- virtual function *Optimize()* (explained below) to call the user-selected optimization scheme: quasi-Newton, genetic algorithms or adaptive random search.

The concept of virtual functions is a powerful one in C++ and object-oriented programming. A virtual function is a class member function that is declared within a base class and redefined by a derived class, which provides a single interface to implement multiple methods. The whole idea behind using virtual functions is rather lengthy to explain and therefore, will not be covered here. However, any standard C++ reference such as (Lippman 1991) should have detailed coverage on this topic.

For the *Revise* class in the REVISER, the virtual function *Optimize* provides CODA with the capability to determine which optimization scheme to call at runtime (as selected by the user) *without* using *if-else if* statements. This approach has the benefit of allowing the addition of new optimization schemes without changing the internal structure of the REVISER module (in particular, the *Revise* class), which is an important concept of object-oriented design.

Chapter 6

Applications to Optimal Structural Design Problems

6.1 Introduction

Three structural design problems are presented in this chapter to illustrate the methodologies and algorithms discussed in the four previous chapters. The three examples we will look at are:

1. A simple ten-bar truss structure with static loads. This example serves as a benchmark problem which has been studied extensively in the literature and is utilized here for comparison between the multicriterion optimal design methodology and the minimal weight design (MWD). Also, we will compare the results obtained by the two GAs proposed for continuous and discrete optimization with some other existing techniques.

2. A planar three-story steel frame with different UBC earthquake design loads. This example is utilized for showing the different analysis capabilities of CODA. In addition, we will also study how the optimal design will differ by using different earthquake design loadings as well as specifying different design parameters.

3. A space truss tower with seventy-two members under static loads. This is a problem of reasonable size to illustrate the computational viability of both the

design methodology and the use of genetic algorithms for solving optimal design problems.

6.2 Simple Ten-Truss Structure: A Benchmark Problem

6.2.1 Problem Description

A ten-bar truss cantilever structure shown in Figure 6.1 is a classical example which is well-studied in structural optimization literature (e.g. Adeli and Kamal 1986). It is a two-dimensional truss which is subject to static loads. The structure is 720 inches in length and 360 inches in depth. It is pin-supported on the left and free on the right. The material used for the members is aluminum (E = 10^4 ksi). The loading on the structure is shown in the figure with two concentrated loads, 100 kips each, acting on the two nodes in the bottom side.

6.2.2 Problem Objective

The main objective of this example is to utilize this well-studied structure in optimal design as a benchmark problem for comparing different optimization algorithms and optimal design approaches. The design parameters for this problem are the cross-sectional areas of the truss members. There are ten design parameters which correspond to each truss member in the structure. For the multicriterion approach, the preference function for these parameters is illustrated in the first figure in Figure 6.2. Using this preference function, a soft constraint is imposed on the areas such that they must be greater than 0.1 square inches and less than 36 square inches with most preferred values lying between 0.1 and 35.9 square inches.

Performance parameters for this example are total volume, maximum deflection at the tip and maximum axial stress. The goal is to minimize the total volume while keeping the deflection and axial stress within acceptable limits. The preference

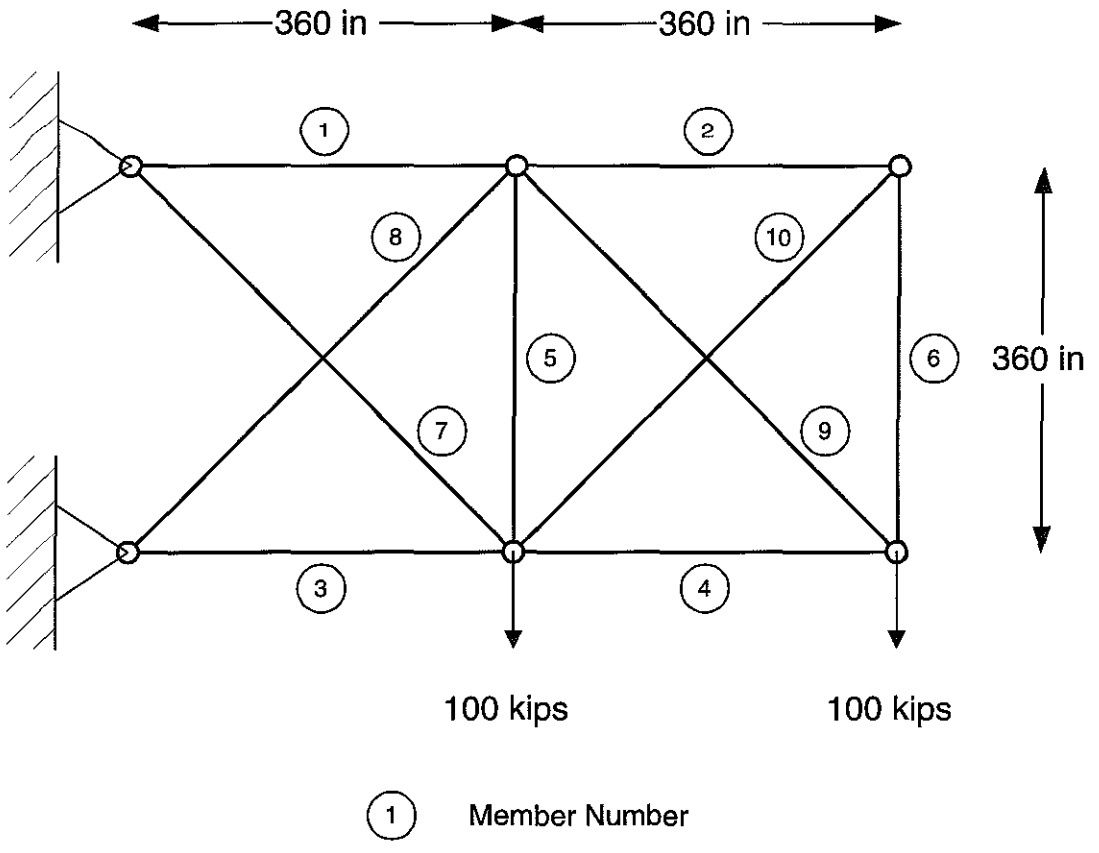


Figure 6.1: Geometry of the ten-truss structure

function for the normalized volume is linear as shown in the second figure in Figure 6.2, indicating that the preference for a design decreases linearly as volume increases. Total volume is normalized by the maximum and minimum allowable volume, which is given by the maximum and minimum permissible member areas. The axial stress in each member is required to be less than 25.0 ksi, with greatest preference $\mu = 1$ given to stresses which are less than 24.9 ksi. The preference function decreases linearly from unity to zero for axial stresses between 24.9 ksi and 25.0 ksi, and $\mu = 0$ is assigned to stresses that exceed 25.0 ksi since these are unacceptable (see third figure in Figure 6.2).

For tip deflection, the preference function is defined to be similar to that of axial stress. A preference value of unity is given to deflection values under 1.9 inches and this value decreases linearly from 1 to 0 as the deflection increases from 1.9 inches to 2.0 inches, and stays 0 for deflections greater than 2.0 inches (see last figure in Figure 6.2).

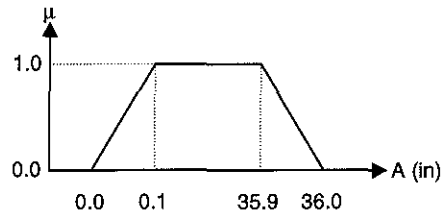
6.2.3 Cases Studied

Three cases were run based on the truss structure described above. In the first and third cases, the design parameters are taken as continuous variables, while in the second case, they are treated as discrete variables and the discrete area set is taken from the areas of AISC wide flange sections for illustrative purposes, even though the material is aluminum. The trade-off strategy is used for all three cases. Below is a description of the three cases:

1. Case 1 - Ten continuous design parameters and three design criteria, total volume, axial stress and tip deflection. All importance weights are equal and have a value of unity.

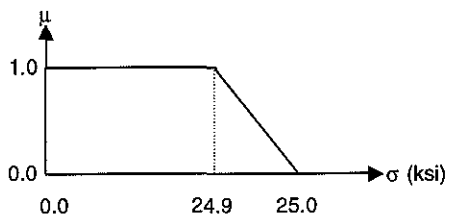
2. Case 2 - Same as Case 2 except the design parameters are treated as discrete variables with discrete area values as described.

1) Cross-sectional Area



2) Total Volume

3) Maximum Axial Stress



4) Deflection at Tip

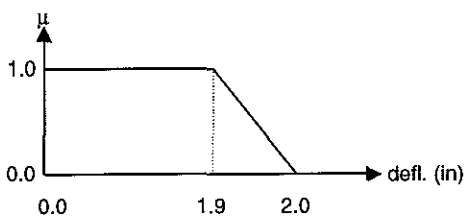


Figure 6.2: Preference functions for ten-truss structure

3. Case 3 - Minimize total volume with only constraints on member stresses and solved as continuous parameter problem by hGA.

For the continuous cases, the proposed hGA is used to find the optimal solution while the proposed vGA is used to compute the solutions for the discrete cases.

Table 6.1: Results of MWD and MCD for Case 1 of the ten-bar truss (Continuous)

	MWD		MCD	
Design Criteria	Values	μ	Values	μ
Member 1 (in^2)	30.13	1.000	30.85	1.000
Member 2 (in^2)	0.10	1.000	0.10	1.000
Member 3 (in^2)	22.93	1.000	27.37	1.000
Member 4 (in^2)	15.39	1.000	17.09	1.000
Member 5 (in^2)	0.10	1.000	0.10	1.000
Member 6 (in^2)	0.10	1.000	0.10	1.000
Member 7 (in^2)	7.42	1.000	6.99	1.000
Member 8 (in^2)	20.75	1.000	19.20	1.000
Member 9 (in^2)	21.77	1.000	24.76	1.000
Member 10 (in^2)	0.10	1.000	0.10	1.000
Total Volume (in^3)	50229	0.669	53205	0.652
Max. Axial Stress (ksi)	25.00	0.000	24.90	1.000
Tip Deflection (in)	2.00	0.000	1.90	1.000
Overall	-	**0.000**	-	**0.866**

Table 6.2: Results of MWD and MCD for Case 2 of the ten-bar truss (Discrete)

	MWD		MCD	
Design Criteria	Values	μ	Values	μ
Member 1 (in^2)	21.57	1.000	31.20	1.000
Member 2 (in^2)	10.98	1.000	3.54	1.000
Member 3 (in^2)	22.08	1.000	25.90	1.000
Member 4 (in^2)	14.95	1.000	15.60	1.000
Member 5 (in^2)	2.68	1.000	4.40	1.000
Member 6 (in^2)	10.98	1.000	4.40	1.000
Member 7 (in^2)	18.91	1.000	16.70	1.000
Member 8 (in^2)	18.42	1.000	20.80	1.000
Member 9 (in^2)	18.40	1.000	20.00	1.000
Member 10 (in^2)	13.51	1.000	5.57	1.000
Total Volume (in^3)	64289	0.576	62753	0.631
Max. Axial Stress (ksi)	10.01	1.000	8.25	1.000
Tip Deflection (in)	1.99	0.001	1.88	1.000
Overall	-	**0.071**	-	**0.855**

6.2.4 Discussion of Results

The results obtained are compared with those from Adeli (1986) and are listed in Tables 6.1 and 6.2. In this table, the results from Adeli were obtained by using the minimal weight approach and solving the problem with traditional optimization methods. These results are denoted as MWD in the tables while MCD represents the results obtained from the multicriterion optimal design methodology. For the discrete case, the structure is optimized over a list of 128 AISC wide flange sections of areas between 2.68 square inches and 35.9 square inches. The following interesting observations can be drawn from these numerical results:

1. From Table 6.1, the optimal design obtained from MWD in Case 1 has smaller steel volume than the one using the multicriterion methodology presented. This is expected since the objective of MWD is to minimize the volume while MCD is geared towards finding a design that best satisfies all the criteria. This can be easily verified by computing the overall preference for the MWD optimal design (see Table 6.1). Notice that the MWD design better satisfies the volume criterion than that of MCD, but the overall design is not acceptable according to the prescribed design criteria for MCD. It should be noted however, that the MWD optimal solution can be approached arbitrarily closely by the MCD optimal solution by taking a sufficiently large importance weight on the total volume criterion instead of an importance weight of unity used for Table 6.1 results

2. Based on the results obtained in the discrete case (Table 6.2), it is obvious that one cannot obtain an optimal discrete solution from a continuous one by simply rounding up to the nearest discrete value. In fact, simple round-up of continuous solutions often results in a discrete solution that is infeasible. Table 6.3 shows the comparison between the continuous solution, a rounded-up solution and the one obtained by vGA.

In terms of computational effort, both hGA and vGA are within an acceptable range. The number of function evaluations required to converge to the optimal so-

Table 6.3: Comparison between rounded-up and vGA discrete solutions for Case 2 of the ten-bar truss

	Continuous		Rounded-Up		vGA	
Design Criteria	Values	μ	Values	μ	Values	μ
Member 1 (in^2)	30.85	1.000	31.10	1.000	31.20	1.000
Member 2 (in^2)	0.10	1.000	2.68	1.000	3.54	1.000
Member 3 (in^2)	27.37	1.000	27.70	1.000	25.90	1.000
Member 4 (in^2)	17.09	1.000	17.10	1.000	15.60	1.000
Member 5 (in^2)	0.10	1.000	2.68	1.000	4.40	1.000
Member 6 (in^2)	0.10	1.000	2.68	1.000	4.40	1.000
Member 7 (in^2)	6.99	1.000	7.08	1.000	16.70	1.000
Member 8 (in^2)	19.20	1.000	19.70	1.000	20.80	1.000
Member 9 (in^2)	24.76	1.000	24.80	1.000	20.00	1.000
Member 10 (in^2)	0.10	1.000	2.68	1.000	5.57	1.000
Total Volume (in^3)	53205	0.652	57843	0.642	62753	0.631
Max. Axial Stress (ksi)	24.90	1.000	14.87	1.000	8.25	1.000
Tip Deflection (in)	1.90	1.000	2.00	0.000	1.88	1.000
Overall	-	**0.866**	-	**0.000**	-	**0.855**

lutions are around 1500 for continuous optimization using hGA and around 5000 for discrete optimization using vGA. The reason why vGA requires more function evaluations than hGA is that hGA uses local topological information of the search space through the quasi-Newton method. Such information is very important in speeding up the convergence rate and this information is simply not available in the discrete case. Nevertheless, the required number of function evaluations for vGA is within the same ball park as other discrete optimization techniques. Consider that for this problem, each parameter has 128 possible discrete sections, there are $128^{10} = 1.18 \times 10^{21}$ possibilities. So, 5000 trials are an infinitesimal fraction of the search space size.

For the continuous case (Case 1), another comparison was carried out between hGA and the constrained optimization algorithm that comes with the optimization toolbox in MATLAB. Table 6.4 shows the optimal designs obtained by using hGA and the constrained optimization method, denoted as COM. Notice that hGA obtained a better solution than that of COM. Since COM is a local search method, premature

Table 6.4: Comparison between hGA and COM for Case 1 of the ten-bar truss

	COM		hGA	
Design Criteria	Values	μ	Values	μ
Member 1 (in^2)	26.07	1.000	30.85	1.000
Member 2 (in^2)	0.36	1.000	0.10	1.000
Member 3 (in^2)	25.55	1.000	27.37	1.000
Member 4 (in^2)	16.54	1.000	17.09	1.000
Member 5 (in^2)	0.10	1.000	0.10	1.000
Member 6 (in^2)	0.35	1.000	0.10	1.000
Member 7 (in^2)	13.84	1.000	6.99	1.000
Member 8 (in^2)	23.85	1.000	19.20	1.000
Member 9 (in^2)	23.85	1.000	24.76	1.000
Member 10 (in^2)	0.63	1.000	0.10	1.000
Total Volume (in^3)	56479	0.627	53205	0.652
Max. Axial Stress (ksi)	10.22	1.000	24.90	1.000
Tip Deflection (in)	1.90	1.000	1.90	1.000
Overall	-	**0.856**	-	**0.866**

convergence may occur when the objective function is flat near the optimum, as is the case here. Without knowing the local topographical information, it is difficult to set the gradient tolerance for local search so that it will not converge to a suboptimum because of a flat surface. On the other hand, hGA is stochastic-based and uses gradient information only to speed up convergence but does not depend on it for a convergence check. In terms of number of function evaluations, COM requires between 1200 to 1500 evaluations depending on the starting point while hGA takes around 2000 evaluations. Although hGA requires around 500 more evaluations (about 30-40% more), the ability to possibly locate the global optimum is well worth the extra computational effort.

Finally, for Case 3, hGA was able to identify two distinct designs (see Table 6.5). Although the first design gives a local optimum, it is substantially different in design from the global optimal solution. This case illustrates the multimodal solution power of hGA.

Table 6.5: Two different designs obtained by hGA for Case 3

Member	Values (in^2)	Values (in^2)
1	4.11	7.90
2	3.89	0.10
3	11.89	8.10
4	0.11	3.90
5	0.10	0.10
6	3.89	0.10
7	11.16	5.80
8	0.15	5.51
9	0.10	3.68
10	5.51	0.14
Total Volume (in^3)	17251	14975

6.3 Three-Story Steel Frame

6.3.1 Problem Description

A three-story, single bay, steel frame is utilized as an example to demonstrate some of the capabilities of CODA. A drawing of the frame with in-plane dimensions can be seen in Figure 6.3. The out-of-plane tributary width is 120 inches, necessary for gravity load calculations. All support conditions are fixed, and beam-column connections are moment resisting. All members, both beams and columns, are wide-flange elements.

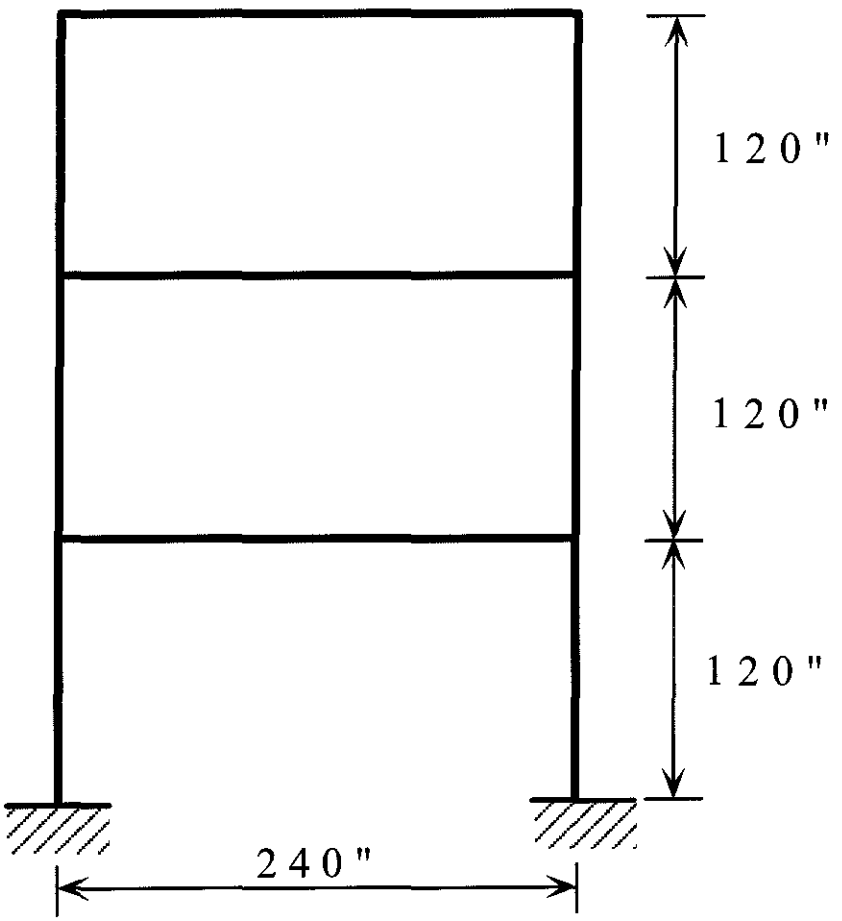


Figure 6.3: Geometry of the problem

The frame was subjected to gravity and earthquake loads. Gravity load values were taken as the defaults provided by the program; specifically, 80 lbs/ft^2 and 20 lbs/ft^2 for the roof dead and live loads respectively, and 100 lbs/ft^2 and 50 lbs/ft^2 dead and live load respectively for each floor.

The earthquake loadings were calculated using two methods available in CODA:

1. the UBC 1994 equivalent static load
2. the UBC 1994 normalized response spectra.

In the code-based analyses, the load parameters are given as follows:

- zone factor Z = 0.4 (i.e. Zone 4)
- soil type S = 1 (i.e., rocky or firm soil)
- importance factor I = 1.0
- ductility factor $R_w = 12$.

In the response spectra analysis, only the first two translational modes were considered and their damping ratios were specified as 5% of critical.

6.3.2 Problem Objective

The objective of this example is to illustrate the capabilities of CODA and the power of vGA for discrete optimization over AISC steel sections. The design parameters for this suite of examples are beam and column sections. No constraints were imposed on these parameters.

Performance parameters for the examples are total steel volume, interstory drift and member stress, where the goal is to minimize steel volume while keeping the drift and the stress within acceptable limits. The preference function for steel volume (first figure in Figure 6.4) is triangular in shape, indicating that the preference for a design decreases linearly as steel volume increases. Steel volume is normalized by the maximum and minimum allowable volume, which is given by the maximum and

minimum permissible member dimensions. Member stress is evaluated as a ratio involving allowable stress as specified in the AISC manual (AISC 1986), which includes buckling and yielding. Using the bilinear preference function (second figure in Figure 6.4), the stress ratio is perfectly acceptable if it is less than 0.9 and the preference value drops off linearly from 1.0 to 0.0 as it increases from 0.9 to 1.0. It becomes totally unacceptable ($\mu = 0.0$) when the stress ratio is greater than 1.0. Preference function for interstory drift is defined as follows: The upper bound is set to the code limit (0.04*story height/R_w for frame with a period less than 0.7sec) and equals 0.4 inches for the example here. The perfectly acceptable drift range is from 0.00 to 0.18 inches was based on nonstructural damage considerations (i.e., less than 0.0015*story height). The preference function for drift is illustrated in the last figure in Figure 6.4.

6.3.3 Cases Studied

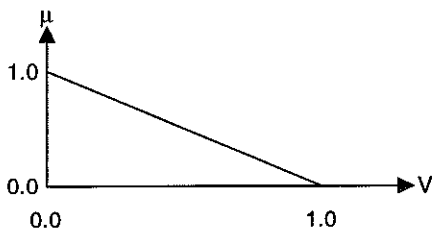
Based on the frame model and the design and performance parameters discussed above, a total of five example cases (A-E) were run. For the first four cases (A to D), a code-based optimal design was sought using equivalent static earthquake loads from the 1994 UBC. Importance weights for the design criteria or the design parameters were changed in each case to illustrate their effect on the optimal design.

Besides the cases for the equivalent static earthquake load, another case, E, was run using the 1994 UBC design spectrum. The design criteria and the associated preference functions for case E are exactly as in Case A. Cases A and E allow comparison of the results using the two available earthquake analysis tools within CODA. The cases analyzed are summarized as follows:

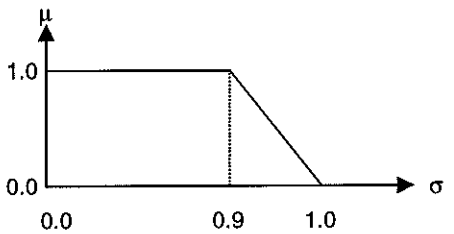
- Case A – Design parameters group all beams together and all columns together in the structure so that there are only two discrete design parameters: W-section for the beams and columns, respectively. The performance parameters are the steel volume, interstory drift and maximum stress in the members. Equal weights are placed on all design criteria.

- Case B – The importance weight of interstory drift is increased to 10.0 with all

1) Steel Volume



2) Member Stress Ratio



3) Maximum Interstory Drift

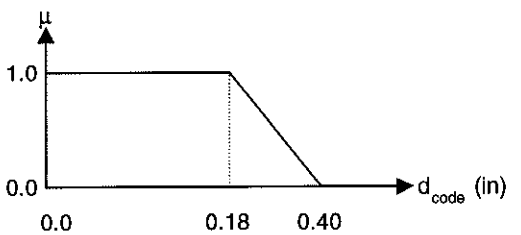


Figure 6.4: Preference functions for the 3-story frame

the others kept at 1.0 as in case A.

- Case C – Same as in case A except the total steel volume now has an importance weight of 10.0 and the others are unchanged.

- Case D – The columns and beams for each story are uncoupled, resulting in six design parameters (one for each story beam and also one for both columns in each story). The design criteria are essentially the same as in case A. All importance weights are set to unity.

- Case E – Same as Case A except a response-spectrum analysis was performed using the 1994 UBC design spectrum.

All the cases considered were solved using the vGA. In addition, Case D was also solved by a simple GA. The trade-off strategy is used for all the cases.

6.3.4 Discussion of Results

Results from the equivalent linear static earthquake analyses (Cases A through D) and that of response spectra (Case E) are presented in Table 6.3.4. Referring to this table, the following observations on the results of equivalent static analyses can be made:

- Interstory drift requirement governed the optimal design process. Total steel volume was generally increased to the point such that the drift preference value was right at the corner between constant preference and the linear drop-off in the preference function.

- Member stress requirement did not seem to have much influence on the optimal design as the maximum stress ratio is far from the value of 0.9, where the corner between constant preference and linear drop-off of preference is located.

- Beam sections of these optimal designs have much larger moments of inertia compared to those of the columns. This is explained by the fact that in most cases beams are longer than columns and their stiffnesses have a greater impact of the drift. However, this is not consistent with UBC, but the weak-beams-strong-columns constraint was not enforced. It could, however, be taken into account and the results obtained would be more consistent with UBC

- By comparing the two-parameter and six-parameter cases, it can be seen that better results, illustrated by the smaller sections, lower steel volume and the increase in overall μ, were achieved when beam and column sections were allowed to vary individually from story to story, as expected.

Comparing the results of Case A and Case E of Table 6.3.4, it can be seen that the final optimal designs for equivalent static and response spectra analyses are somewhat different. In general, the response spectra method called for larger beam sections because of the way maximum drifts are approximated from different modes conservatively (such as SRSS), resulting in a larger steel volume. Both methods resulted

Table 6.6: Results for equivalent static (Cases A-D) and response spectra (Case E)

Case	Design Criteria	Value	μ
A	All Columns	W12x16 (4.71, 103) [a]	-
	All Beams	W14x22 (6.49, 199)	-
	Total Steel Volume	8064	0.912
	Interstory Drift	0.1710	1.000
	Max. Member Stress Ratio	0.3123	1.000
	Overall	-	**0.970**
B	All Columns	W12x16 (4.71, 103)	-
	All Beams	W14x22 (6.49, 199)	-
	Total Steel Volume	8064	0.912
	Interstory Drift	0.1710	1.000
	Max. Member Stress Ratio	0.3123	1.000
	Overall	-	**0.992**
C	All Columns	W12x14 (4.16, 88.6)	-
	All Beams	W14x22 (6.49, 199)	-
	Total Steel Volume	7668	0.920
	Interstory Drift	0.1830	0.987
	Max. Member Stress Ratio	0.3624	1.000
	Overall	-	**0.932**
D	Story 1 Columns	W12x16 (4.71, 103)	-
	Story 2 Columns	W12x14 (4.16, 88.6)	-
	Story 3 Columns	W10x12 (3.54, 53.8)	-
	Story 1 Beams	W14x22 (6.49, 199)	-
	Story 2 Beams	W14x22 (6.49, 199)	-
	Story 3 Beams	W8x10 (2.96, 30.8)	-
	Total Steel Volume	6804	0.938
	Interstory Drift	0.1793	1.000
	Max. Member Stress Ratio	0.5863	1.000
	Overall	-	**0.979**
E	All Columns	W12x14 (4.16, 88.6)	-
	All Beams	W16x26 (7.68, 301)	-
	Total Steel Volume	8525	0.903
	Interstory Drift	0.1657	1.000
	Max. Member Stress Ratio	0.3317	1.000
	Overall	-	**0.966**

[a] Area (in^2) and moment of inertia (in^4), respectively.

in perfectly acceptable interstory drift values, although the actual drift values themselves are slightly different. In both cases, although interstory drift governed the optimization process, the steel volume controlled the overall design preference. The interstory drift is reduced until it just reaches a perfectly acceptable preference value of 1.00.

In Case B, the same results were obtained as those in Case A. This is expected since the drift was already perfectly acceptable in Case A, so giving more weight to drift cannot improve the design. In Case C, a more aggressive design is obtained by increasing the importance weight of steel volume. Notice that the drift is pushed beyond the corner of the preference function and results in a preference value of 0.987.

The convergence rate for Case A can be seen in Figure 6.5. From this figure, it can be seen that vGA converged to the optimal solution around the 20^{th} generation. Since a population size of 30 was used, it found the optimal solution with merely 600 function evaluations. Granted that this problem is two dimensional, the number of possibilities is still $128^2 = 16384$ and 600 trials represents only 3.6% of the whole search space. Once again, vGA displays a good convergence rate. To further illustrate its merits, Table 6.7 shows the results obtained by vGA and a simple GA for Case D. Both were carried out using population size of 30 for 100 generations. Notice that the simple GA was unable to find the optimal solution and therefore resulted in a slightly larger steel volume compared to the one obtained by vGA.

Note that in Case E, columns of the stories 1 and 2 are fairly close (within 15%). When building such a frame, it is often more convenient to select the same sections for these columns. To ensure the design is still optimal after coupling these columns, one can regroup the design parameters to reflect this and recalculate the optimal design.

Table 6.7: Results obtained by vGA and simple GA for Case D

	vGA		simple GA	
Design Criteria	Values	μ	Values	μ
Story 1 Columns	W12x16	-	W12x16	-
Story 2 Columns	W12X14	-	W12x16	-
Story 3 Columns	W10X12	-	W12x14	-
Story 1 Beams	W14X22	-	W14x22	-
Story 2 Beams	W14X22	-	W14x22	-
Story 3 Beams	W8X10	-	W10x12	-
Steel Volume	6804	0.938	7224	0.930
Interstory Drift	0.1793	1.000	0.1756	1.000
Max. Stress Ratio	0.5863	1.000	0.5427	1.000
Overall	-	**0.979**	-	**0.964**

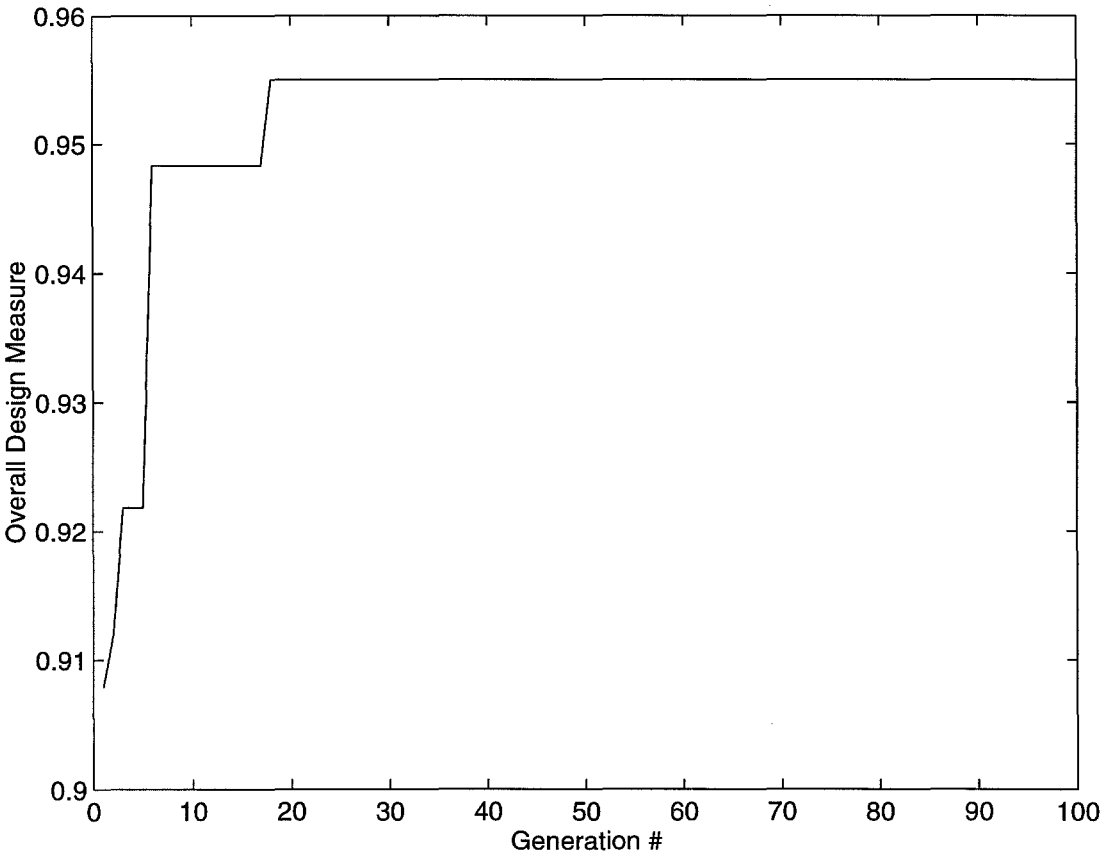


Figure 6.5: Convergence history of vGA for Case A

6.4 Three Dimension Seventy-Two Bar Truss Tower

6.4.1 Problem Description

A three-dimensional seventy-two bar truss tower, illustrated in Figure 6.6, is studied here and has been investigated by several researchers (Gellatly et al. 1971; Schmit and Miura 1976) using different techniques. This truss has four stories and is symmetric about the X and Y axes. It is 240 inches tall with equal story heights and has a square cross-section of 120 inches by 120 inches. The structure is pin-supported at the base. The members are made of aluminum (E = 10^5 ksi). The loading on this structure is as follows:

Table 6.8: Load cases of the 72-bar truss

		Load Components (kips)		
Load case	Node	X	Y	Z
A	1	5.00	5.00	-5.00
B	1	75.00	75.00	75.00

Here, node 1 is the one on the roof along the Z-axis.

6.4.2 Problem Objective

The main objective of this problem is to investigate the computational requirement of the multicriterion optimal design methodology and the application of genetic algorithms to structural optimization problems. There are 72 axial members in the structure which can be divided into four identical substructures, with each substructure representing a single story. The members are numbered according to the scheme in Figure 6.7. The numbering works as follows: starts with the four vertical members (1-4), the bracings on the four "walls" (5-12), the horizontal members (13-16) and finally, the bracings on the "ceiling" (17,18). This cycle repeats for each of the four stories to get a total of 72 members.

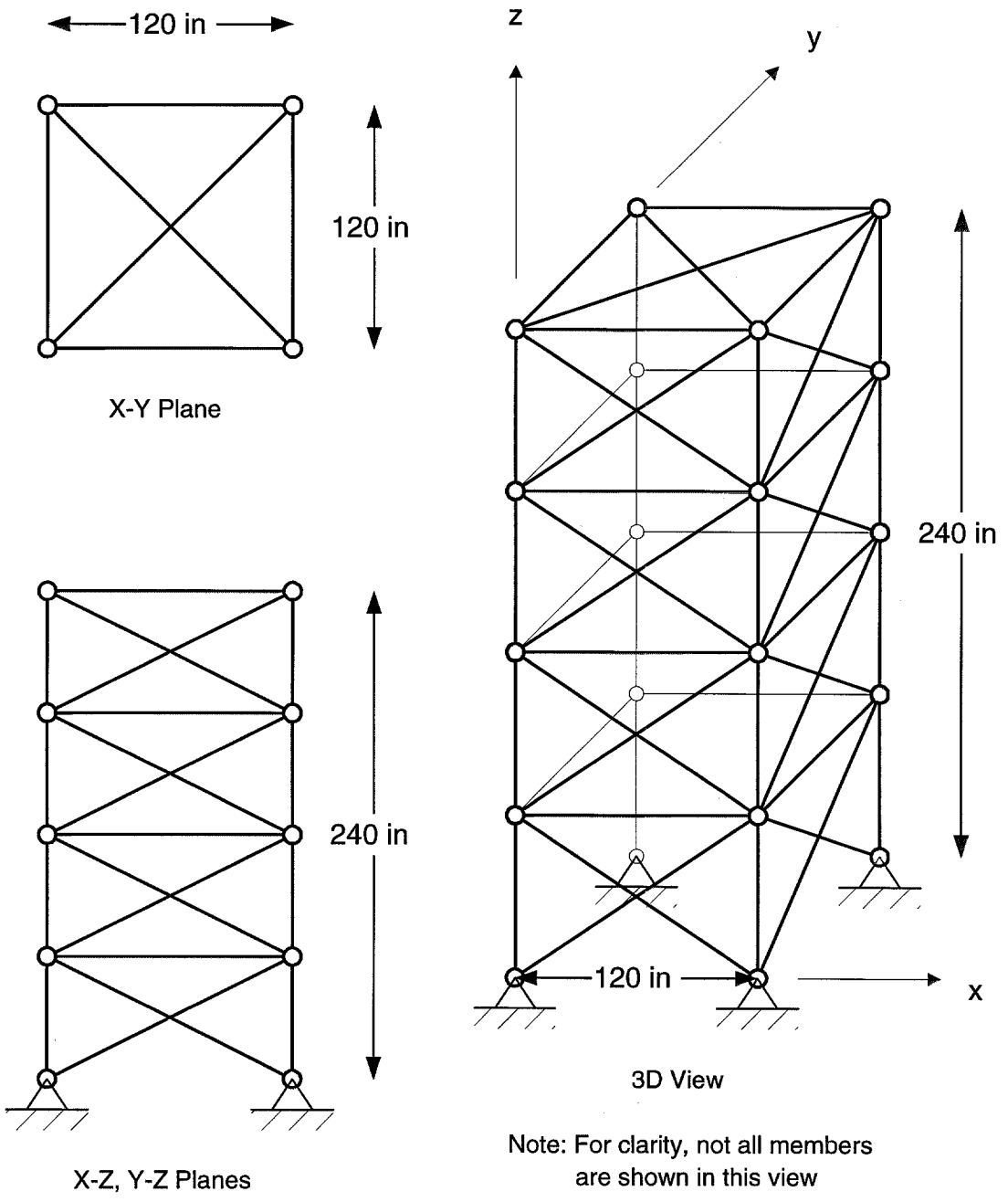


Figure 6.6: Geometry of the 72-truss structure

Design parameters for this problem are the cross-sectional areas of the members. Since there is a lot of symmetry in this structure, the members are grouped into 16 design parameters as listed in Table 6.9. The lower and upper bounds on the cross-sectional areas are 0.1 in^2 and 36.0 in^2 and these bounds are specified as "soft" constraints by using preference function as shown in the first figure in Figure 6.8.

Performance parameters for this structure are total material volume, maximum axial stress and maximum deflection at the roof. The goal is to minimize the total material volume while keeping the roof deflection and member axial stresses within acceptable limits. The preference function for total material volume is shown in the second figure in Figure 6.8, indicating that the preference of a design decrease linearly as total volume increases. The axial stress in members is required to be less than 25.0 ksi, with the greatest preference value of unity given to stresses under 24.9 ksi. For stresses between 24.9 ksi and 25.0 ksi, the preference value decreases linearly from unity to zero. A preference value of zero is assigned for stresses above 25.0 ksi (see third figure in Figure 6.8).

For roof deflection, the in-plane displacements along the X and Y directions of the four roof nodes are required to be less than 0.25 inches and perfectly acceptable if less than 0.24 inches. The preference function for this quantity is defined to be similar to that of axial stress (see last figure in Figure 6.8).

6.4.3 Cases Studied

Three cases were run based on this truss tower:

1. Case 1 - Sixteen continuous design parameters and three design criteria, total volume, axial stress and roof in-plane deflection. All importance weights are set to unity. A different preference function is used for drift in this case. For drift less than 0.25 inches, the preference value is perfectly acceptable and becomes unacceptable for drift greater than 0.251 inches. Loading Case A is used. hGA was used to find the solution.

2. Case 2 - Exactly like Case 1 except Load Case B is used and solved by hGA.

3. Case 3 - Exactly like Case 2 except the design parameters are treated as discrete variables with discrete areas taken from AISC steel sections, even though material is aluminum (as in the 10-bar truss case). vGA is used for solving this problem.

Table 6.9: Grouping of design parameters for the 72-bar truss

Design Parameter Group	Members
1	1-4
2	5-12
3	13-16
4	17-18
5	19-22
6	23-30
7	31-34
8	35-36
9	37-40
10	41-48
11	49-52
12	53-54
13	55-58
14	59-66
15	67-70
16	71-72

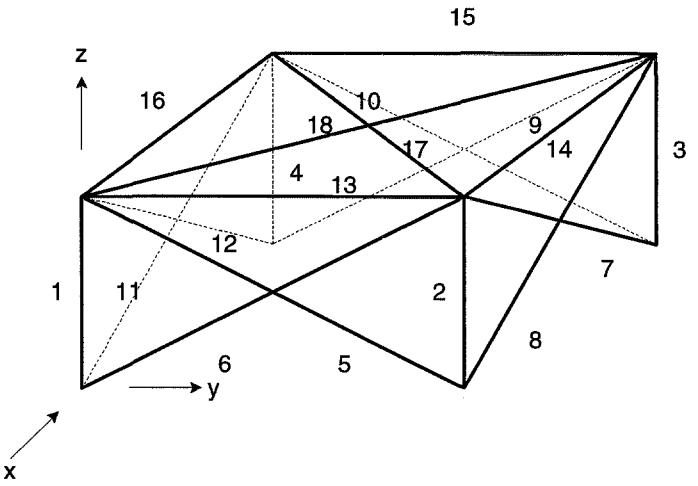
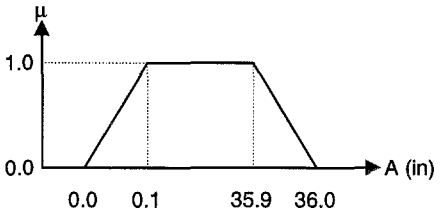
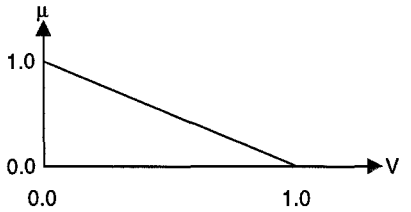


Figure 6.7: Member numbering of the 72-bar truss structure

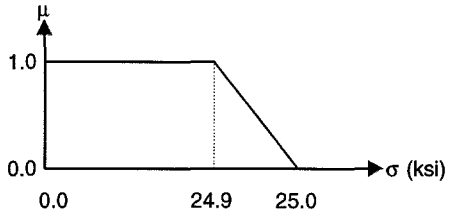
1) Cross-sectional Area



2) Total Volume



3) Maximum Axial Stress



4) Roof In-plane Deflection

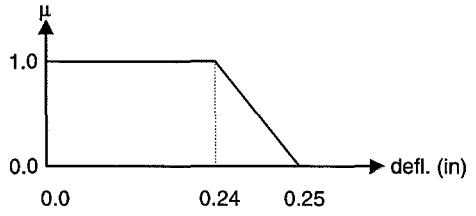


Figure 6.8: Preference functions for the 72-bar truss structure

Table 6.10: Results of Case 1 and those from Haftka and Kamat (1985)

	MCD w/hGA		MWD	
Design Criteria	Values	μ	Values	μ
Group 1 (in^2)	0.10	1.000	0.10	1.000
Group 2 (in^2)	0.52	1.000	0.52	1.000
Group 3 (in^2)	0.40	1.000	0.40	1.000
Group 4 (in^2)	0.54	1.000	0.54	1.000
Group 5 (in^2)	0.50	1.000	0.50	1.000
Group 6 (in^2)	0.50	1.000	0.51	1.000
Group 7 (in^2)	0.10	1.000	0.10	1.000
Group 8 (in^2)	0.10	1.000	0.10	1.000
Group 9 (in^2)	1.25	1.000	1.25	1.000
Group 10 (in^2)	0.50	1.000	0.50	1.000
Group 11 (in^2)	0.10	1.000	0.10	1.000
Group 12 (in^2)	0.10	1.000	0.10	1.000
Group 13 (in^2)	1.86	1.000	1.86	1.000
Group 14 (in^2)	0.50	1.000	0.50	1.000
Group 15 (in^2)	0.10	1.000	0.10	1.000
Group 16 (in^2)	0.10	1.000	0.10	1.000
Steel Volume (in^3)	3690	0.966	3696	0.966
Max. Axial Stress (ksi)	23.86	1.000	23.83	1.000
Tip Deflection (in)	0.25	1.000	0.25	1.000
Overall	-	**0.987**	-	**0.987**

Table 6.11: Case 2 and 3 solutions for the 72-bar truss

	Continuous (hGA)		Discrete (vGA)	
Design Criteria	Values	μ	Values	μ
Group 1 (in^2)	1.71	1.000	2.68	1.000
Group 2 (in^2)	7.88	1.000	7.34	1.000
Group 3 (in^2)	10.27	1.000	11.70	1.000
Group 4 (in^2)	9.23	1.000	5.54	1.000
Group 5 (in^2)	6.69	1.000	9.71	1.000
Group 6 (in^2)	7.91	1.000	11.80	1.000
Group 7 (in^2)	0.10	1.000	4.68	1.000
Group 8 (in^2)	0.95	1.000	2.68	1.000
Group 9 (in^2)	18.38	1.000	18.30	1.000
Group 10 (in^2)	7.64	1.000	10.00	1.000
Group 11 (in^2)	0.10	1.000	3.83	1.000
Group 12 (in^2)	0.10	1.000	7.06	1.000
Group 13 (in^2)	27.84	1.000	20.10	1.000
Group 14 (in^2)	8.30	1.000	8.79	1.000
Group 15 (in^2)	0.10	1.000	3.54	1.000
Group 16 (in^2)	0.10	1.000	7.34	1.000
Steel Volume (in^3)	55770	0.821	67279	0.764
Max. Axial Stress (ksi)	24.90	1.000	10.88	1.000
Tip Deflection (in)	0.24	1.000	0.24	1.000
Overall	-	**0.936**	-	**0.914**

6.4.4 Discussion of Results

Results from all three cases are listed in Tables 6.11 and 6.10. In Table 6.10, the optimum obtained by using MCD and hGA is almost identical to the results obtained by using the minimal weight approach (MWD). Because many of the areas obtained in Case 1 are much smaller than those in the discrete member database, a much greater loading was used in Cases 2 and 3. In Table 6.11, the continuous and discrete solutions are listed. Notice that again the discrete solution does not appear very similar to the continuous one, just like the previous examples.

The convergence histories of the continuous and discrete cases (2 and 3) are illustrated in Figure 6.9. Notice the convergence rate stabilized for hGA after the first 150-200 generations and vGA took about almost 300 generations to converge. A population size of 30 was used for both cases. Although 9000 function evaluations for vGA to obtain a solution seem like a lot, the total possible designs in the search space are $128^{16} = 5.19 \times 10^{33}$!

Notice that the lower story columns in Cases 1 and 2 have very small cross-sectional areas. While these tiny member areas may be required for the optimal design, such designs may not be practical. To remedy this problem, one can group these lower story columns with the upper story ones as a single design parameter or imposing a strict preference function that requires a higher minimum area. By doing so, the optimal design obtained would be one that is more consistent what we would expect.

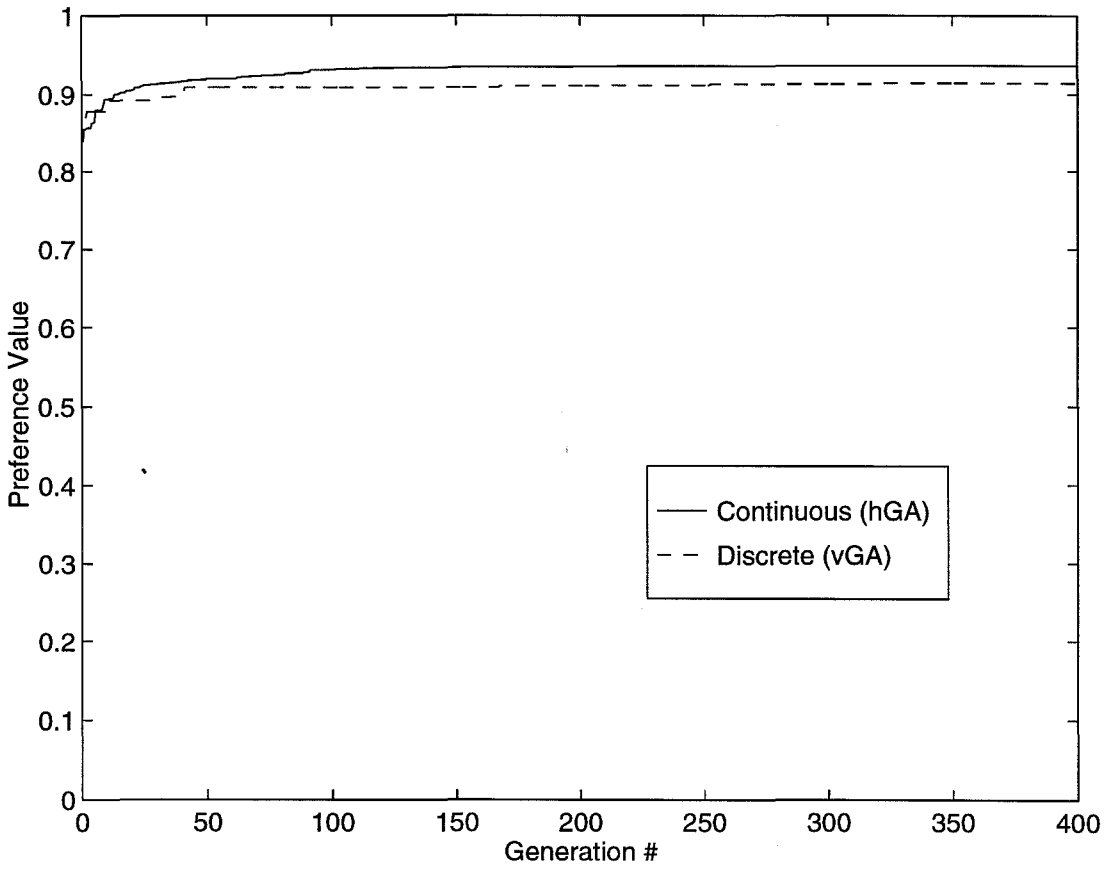


Figure 6.9: Convergence histories of continuous and discrete solutions

Chapter 7

Conclusions

7.1 Summary and Conclusions

In this work, a recently-developed multicriterion optimal design framework is presented. This framework is based upon the use of preference functions for design criteria which can be used to quantify preference or acceptability of both engineering and non-engineering quantities. The notion of design criterion and its associated preference function allows easy incorporation of different design objectives from different parties such as the owner, the engineer, etc. With trade-off aggregation strategies, design decisions involving these different design criteria can be automated in a systematic way using digital computers.

This framework has been shown to be related to other existing optimal design methodologies. In chapter 2, it was shown that for a given multicriterion design problem, each optimal design obtained using this framework lies in the Pareto optimal set of the problem. Other Pareto optimal designs for the problem can be obtained by varying the importance weights of the design criteria. Furthermore, with the choice of certain preference functions for different design criteria, it was also shown that this framework behaves very much like the fully stressed design method which is one of the most popular single objective optimal design methodologies in structural engineering. Such a relationship is further verified by the numerical results presented in Chapter 6.

Genetic algorithms are presented in this work. GAs are stochastic methods that

are based on Darwinian evolution theory. Two special classes of GAs were studied: variable-length GAs and hybrid GAs. A variable-length GA, vGA, is presented for solving optimal structural design problems over a discrete member database such as the AISC steel sections. In Chapter 4, a simple design example was shown to illustrate a possible difficulty known as GA-deception that may arise when simple genetic algorithms are applied to this problem. vGA was designed to address this difficulty. Based on the results presented for the three-story frame in Chapter 6, vGA shows its superiority over simple GAs for seeking global optimal solution in discrete search space. The numerical results also show that vGA converges quite quickly to the solution. Since discrete optimization problems are computationally challenging, especially for high dimension systems, vGA provides one viable method.

A specially-designed hybrid GA, called hGA, is also presented here. hGA is designed for continuous optimization problems. By incorporating a hill-climbing algorithm as an operator within the GA loop, a very efficient and powerful hybrid GA is obtained. It is evident from the numerical results presented in Chapter 4, hGA has a better convergence rate than simple GA for optimization problems with continuous parameters. hGA is a global optimization technique and it was able to find a global solution that a quasi-Newton method could not because of obstruction by a local optimum. Furthermore, although not rigorously proved, hGA has displayed multimodal solution capability in the two equivalent stiffness models for the two-story shear building and the two optimal designs for the 10-bar truss problem.

A software prototype of the multicriterion design framework called CODA is presented in Chapter 6. CODA is a software application for performing an optimal structural design based on 1994 UBC wind and earthquake specifications. Although the analysis capabilities of CODA are limited to only linear static and dynamic problems, it can be used to design simple plane structures and it is a good tool for graphical illustration of various aspects of the optimal design framework. Moreover, it can be used as a research tool for studying how various quantities such as different seismic ground motion models can affect the optimal design. Some results using CODA for reliability-based optimal structural design were also presented (Beck et al. 1996).

From the results of the examples presented in Chapter 6, it is clear that designs obtained from structural optimization may not be practical and sometimes, even inconsistent with standard design codes such as UBC. However, one can remedy such shortcomings by specifying design criteria that take into account factors that we normally would consider in a design procedure. For instance, by imposing the weak-beams-strong-columns requirement on the three-story frame example, the optimal design obtained would be more consistent with UBC.

7.2 Future Research

It is desirable to apply the optimal design framework presented here to multicriterion performance-based design. Since reliability theory is incorporated into the framework (Beck et al. 1996), different levels of performance can be specified with different target reliabilities. It will be both interesting and informative to compare how different are the optimal designs for various performance levels and reliabilities.

Another improvement to both the theoretical framework and CODA is to incorporate artificial intelligence techniques throughout the design process. For example, a knowledge-based expert system can be employed to remove some of the least likely candidates for an optimal solution, thus reducing the size of the search space. Future research could be focused on development of rules for performing such selections. One possible approach is to apply classifier systems which are genetics-based expert systems (Goldberg 1989).

Another interesting topic would be to develop a method to approximate the objective function of a continuous-variable optimization problems. Since an evaluation of the objective function could involve a nonlinear finite element analysis, having such a method can drastically reduce the computational requirements. One possibility is to use neural networks to approximate the computations of the ANALYZER by training them to estimate the topography of the objective function surface. Once the neutral network is trained, it can be used to quickly compute the analysis results. In addition, one can train neural networks with commercial packages such as NASTRAN

or ABACUS and then connect them with CODA. Doing so can greatly increase the computational capability of CODA. On the other hand, neural networks require a large amount of data to be trained and they are not good at extrapolating outside the region covered by the training data.

The multicriterion design framework presented here has been applied to structural optimization problems. However, this framework is a very general decision making methodology that can be applied to other problems as well. One possible application is to employ this framework for optimal design of controllers for structural systems.

Finally, genetic algorithms can be applied to other engineering problems. In this study, special genetic algorithms such as hGA and vGA have been shown to be very powerful techniques for solving continuous and discrete structural optimization problems. Other problems exist in various structural engineering fields that require optimization as well. For instance, in system identification we often have to identify multiple equivalent models which require an optimization scheme that can handle multimodal problems (Beck and Katafygiotis 1997; Katafygiotis and Beck 1997). Since hGA seems to possess the capability to solve these problems, further research would be useful to investigate the multimodal capability of hGA. Also, in system identification, we are often interested in finding optimal locations for the placement of sensors. This problem is a combinatorial optimization one and the search space for this problem rapidly grows as the number of sensors and possible locations increases. It is desirable to apply genetic algorithms to these problems.

Bibliography

Adeli, H. and O. Kamal (1986). Efficient optimization of space trusses. *Computers and Structures*, 501–511.

AISC (1986). *Manual of steel construction.* Chicago, Illinois: American Institute of Steel Constructions.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice.* Oxford: Oxford University Press.

Bagley, J. D. (1967). *The behavior of adaptive systems which employ genetic and correlation algorithms.* Ph. D. thesis, University of Michigan.

Balachandran, M. (1996). *Knowledge-based optimum design.* Topics in Engineering. (10).CMP.

Barricelli, N. A. (1957). Symbiogenetic evolution processes realized by artificial methods. *Methodos 9*, 143–182.

Beck, J., E. Chan, A. Irfanoglu, S. Masri, H. Smith, V. Vance, and L. Barroso (1996). *New computer tools for optimal design decisions in the presence of risk.* Final report on CUREe-Kajima Project, Caltech-USC-Stanford. Caltech-USC-Stanford, CA, USA.

Beck, J. and L. Katafygiotis (1997). Updating structural dynamic models and their uncertainties: statistical system identification. *ASCE Journal of Engineering Mechanics*. to appear.

Beck, J., C. Papadimitriou, E. Chan, and A. Irfanoglu (1996). Reliability-based optimal design decisions in the presence of seismic risk. In *Proceedings of 11th World Conf. on Earthquake Engineering: Paper No. 1058.* Elsevier Science Ltd.

Bethke, A. D. (1980). *Genetic algorithms as function optimizers.* Ph. D. thesis, University of Michigan.

Braun, R. and I. Kroo (1995). Development and application of the collaborative optimization architecture in a multidisciplinary design environment. In N. Alexandrov and M. Hussaini (Eds.), *Multidisciplinary design optimization: state-of-the-art.* SIAM.

Broyden, C. (1970). The convergence of a class of double rank minimization algorithms, parts I and II. *Jnl. Inst.Math. Applns. 6*, 76–90,222–231.

Budiman, J. and S. Rajan (1993). Shape optimal design methodology - the hybrid natural approach. In *Proceedings of 31rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, New York, NY, pp. 554–554.

Cavicchio, D. J. (1970). *Adaptive search using simulated evolution.* Ph. D. thesis, University of Michigan.

Chan, C. (1992). An optimality criteria algorithm for tall steel building design using commercial standard sections. *Structural Optimization 5*, 26–29.

Chan, C., D. E. Grierson, and A. Sherbourne (1995). Automatic optimal design of tall steel building frameworks. *ASCE Journal of Structural Engineering 121*(5), 838–847.

Cilley, F. (1900). The exact design of statically determinate frameworks, and exposition of its possibility, but futility. *Trans. ASCE 43*, 353–407.

Craig, R. R. (1981). *Structural Dynamics.* New York: Wiley.

Cramer, N. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of International Conference on Genetic Algorithms and Their Applications*, pp. 183–187.

DeJong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Ph. D. thesis, University of Michigan.

Dym, C. L. and R. E. Levitt (1991). *Knowledge-Based Systems in Engineering.* New York: McGraw-Hill.

Eschenauer, H., J. Koski, and A. Osyczka (Eds.) (1990). *Multicriteria design optimization - Procedures and applications*, Berlin. Springer.

Fletcher, R. (1970). A new approach to variable metric algorithms. *Computer J. 13*, 317–322.

Fletcher, R. and M. Powell (1963). A rapidly convergent descent method for minimization. *Computer J. 6*, 163–168.

Frangopol, D. (1987). Unified approach to reliability-based structural optimization. In J. Roesset (Ed.), *Dynamics of Structures*, pp. 156–167. New York: ASCE.

Frangopol, D. (1991). Multiobjective decision support spaces for optimum design of nondeterministic structural systems. In G. Apostolakis (Ed.), *Probabilistic Safety Assessment and Management*, Volume 2, pp. 977–982. Amsterdam: Elsevier.

Frangopol, D. and F. Moses (1994). Reliability-based structural optimization. In H. Adeli (Ed.), *Advances in Design Optimization*, pp. 492–570. New York: Chapman and Hall.

Fraser, A. (1960). Simulation of genetic systems by automatic digital computers. 5-linkage, dominance and epistasis. In O. Kempthorne (Ed.), *Biometrical genetics*, New York, pp. 70–83. Macmillian.

Furuya, H. and R. Haftka (1993). Genetic algorithsm for placing actuators on space structures. In S. Forrest (Ed.), *Genetic algorithms and their applications: Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, pp. 536–542. Morgan Kaufmann Publishers.

Gellatly, R., L. Berke, and W. Gibson (1971). The use of optimality criteria in automated structural design. affdel. In *Proceedings of 3rd on Matrix Methods in Structural Analysis.*

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problems. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, London., pp. 74–88. Pitman.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison Wesley.

Goldberg, D. E. (1990). A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems* (4), 445–460.

Goldberg, D. E. (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems* (5), 139–167.

Goldberg, D. E. and K. Deb (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA, pp. 69–93. Morgan Kaufmann Publishers.

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

Goldberg, D. E., K. Deb, and B. Korb (1990). Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems 4*, 415–444.

Goldberg, D. E., B. Korb, and K. Deb (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems 3*, 493–530.

Goldberg, D. E. and J. Richardson (1987). Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette (Ed.), *Genetic algorithms and thier applications: Proceedings of the Second International Conference on Genetic Algorithms*, Hillsdale, NJ, pp. 59–68. Lawrence Erlbaum Associates, Publishers.

Goldfarb, D. (1970). A family of variable metric methods derived by variational means. *Maths. Comput. 24*, 23–26.

Haftka, R. T. and M. P. Kamat (1985). *Elements of structural optimization.* Dordrecht: Martinus Nijhoff Publishers.

Holland, J. H. (1962a). Concerning efficient adaptive systems. In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein (Eds.), *Self-organizing systems*, Washington, D.C., pp. 215–230. Spartan Books.

Holland, J. H. (1962b). Information processing in adaptive systems. *Information Processing in the Nervous System, Proceedings of the International Union of Physiological Sciences 3*, 330–339.

Holland, J. H. (1962c). Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery 3*, 297–314.

Holland, J. H. (1965). Some practical aspects of adaptive systems theory. In A. Kent and E. Taulbee (Eds.), *Electronic Information Handling*, Washington, D.C., pp. 209–217. Spartan Books.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems, 2nd Edition.* Cambridge, MA: The MIT Press. First Edition, 1975.

Hollstien, R. B. (1971). *Artificial genetic adaptation in computer control systems.* Ph. D. thesis, University of Michigan.

ICBO (1994). *Uniform Building Code.* Whittier, California: International Conference of Building Officials.

Katafygiotis, L. and J. Beck (1997). Updating structural dynamic models and their uncertainties: model identifiability. *ASCE Journal of Engineering Mechanics.* to appear.

Kirsch, U. (1981). *Optimal structural design.* New York: McGraw-Hill.

Koski, J. (1985). Defectiveness of weighting method in multicriterion optimization of structures. *Communications of Applied Numerical Methods 1*, 333–336.

Koski, J. and R. Silvennoinen (1987). Multicriteria design of ceramic piston crown. *International Journal for Numerical Methods in Engineering 24*, 1101–1121.

Koza, J. (1992). *Genetic Progamming: On the programming of computers by means of natural selection.* Cambridge, MA: MIT Press.

Kroo, I. (1995). Decomposition and collaborative optimization for large-scale aerospace design programs. In N. Alexandrov and M. Hussaini (Eds.), *Multidisciplinary design optimization: state-of-the-art.* SIAM.

Leitmann, G. (1977). Some problems of scalar and vector-valued optimization in linear viscoelasticity. *Journal of Optimization Theory and Applications 23*, 93–99.

Liepins, G. and M. Vose (1991). Deceptiveness and genetic algorithm dynamics. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA, pp. 36–50. Morgan Kaufmann Publishers.

Lippman, S. (1991). *C++ Primer* (2 ed.). Reading, Massachusetts: Addison-Wesley.

Martin, F. G. and C. C. Cockerham (1960). High speed selection studies. In O. Kempthorne (Ed.), *Biometrical genetics*, London, pp. 35–45. Pergamon Press.

Masri, S., G. Bekey, and F. Safford (1980). An adaptive random search method for identification of large scale nonlinear systems. *Applied Mathematics and Computation 7*, 353–375.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs, 2nd Edition.* New York: Springer-Verlag.

Michell, A. (1904). The limits of economy of material in framed structures. *Phil. Mag 6*, 589–597.

Moses, F. (1974). Reliability of structural systems. *Journal of the Structural Division, ASCE 100*, 1813–1820.

Moses, F. (1989, 4). Calibration of bridge-strength evaluation code. *Journal of Engineering Mechanics, ASCE 115*, 1538–1554.

Moses, F. (1990). New directions and research needs in system reliability search. *Structural Safety 7*, 93–100.

Otto, K. N. (1992). *A formal Representational Theory for Engineering Design.* Ph. D. thesis, California Institute of Technology, Pasadena, California.

Parimi, S. and M. Cohn (1978, 1). Optimum solutions in probabilistic structural design. *Journal of Applied Mechanics 2*, 47–92.

Razani, R. (1965). The behaviour of the fully stressed design of structures and its relationship to minimum weight design. *AIAA J. 3*, 2262–2268.

Renders, J.-M. and S. P. Flasse (1996). Hybrid methods using genetic algorithms for global optimization. *IEEE Trans on Systems, Man and Cybernetics 2*, 243–258.

Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Genetic algorithms and their applications: Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, pp. 191–197. Morgan Kaufmann Publishers.

Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties.* Ph. D. thesis, University of Michigan.

Schaffer, J., R. Caruana, L. Eshelman, and R. Das (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer (Ed.), *Genetic algorithms and their applications: Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, CA, pp. 51–60. Morgan Kaufmann Publishers.

Schmit, L. (1960). Structural design by systematic synthesis. In *Proceedings of the 2nd ASCE Conference on Electronic Computation*, pp. 105–132.

Schmit, L. and H. Miura (1976). A new structural analysis/synthesis capability-access 1. *AIAA J. 14*, 661–671.

Shaefer, C. (1987). The argot strategy: Adaptive representation genetic optimizer technique. In *Genetic Algorithms and Their Applications: Proceedings of Second International Conference on Genetic Algorithms*, pp. 50–58.

Shanno, D. (1970). Conditioning of quasi-Newton methods for function minimization. *Maths. Comput. 24*, 647–656.

Smith, S. (1980). *A learning system based on genetic adaptive algorithms.* Ph. D. thesis, University of Pittsburgh.

Soegiarso, R. and H. Adeli (1996). Optimization of large steel truss structures using standard cross sections. *AISC Engineering Journal, 3rd Quarter*, 83–94.

Stadler, W. (1988). *Multicriteria optimization in engineering and in the sciences*, Volume 37 of *Mathematical Concepts and Methods in Science and Engineering.* New York: Plenum.

Strang, G. (1988). *Linear Algebra and its Application - 3rd Edition.* San Diego: Harcourt Brace Jovanovich, Publishers.

Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA, pp. 94–101. Morgan Kaufmann Publishers.

Thanedar, P. and G. Vanderplaats (1992). Survey of discrete variable optimization for structural design. *ASCE Journal of Structural Engineering 121*(2), 301–306.

Unger, R. and J. Moult (1993). Genetic algorithm for 3d protein folding simulations. In S. Forrest (Ed.), *Genetic algorithms and their applications: Proceedings of the Fifth International Conference on Genetic Algorithms*, Los Altos, CA, pp. 581–588. Morgan Kaufmann Publishers.

Wakayama, S. and I. Kroo (1994). Subsonic wing design using multidisciplinary optimization. In *Proceedings of the 5th AIAA/USAF/NASA/ISSMO Symposium on multidisciplinary analysis and optimization, AIAA-94-4409.*

Weinberg, R. (1970). *Computer simulation of a living cell.* Ph. D. thesis, University of Michigan.

Whitley, D. (1991). Fundamental principles of deception in genetic search. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA, pp. 221–241. Morgan Kaufmann Publishers.

Zhou, M. and G. Rozvany (1992). A new discretized optimality criteria method in structural optimization. In *Proceedings of 33rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Dallas, TX*, Washington, DC, pp. 3106–20.