An Improved Genetic Algorithm for the Optimization of Composite Structures

by

Vladimir B. Gantovnik

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Engineering Mechanics

Approved:

Professor Zafer Gürdal, Chairman/Advisor

Professor Layne T. Watson

Professor Liviu Librescu

Professor Mahendra P. Singh

Professor Levon V. Asryan

August, 2005
Blacksburg, Virginia

Keywords: Composite Structures, Genetic Algorithm Optimization, Multivariate Approximation, Binary Tree Memory

ABSTRACT

An Improved Genetic Algorithm for the Optimization of Composite Structures

by

Vladimir B. Gantovnik

There are many diverse applications that are mathematically modelled in terms of mixed discrete-continuous variables. The optimization of these models is typically difficult due to their combinatorial nature and potential existence of multiple local minima in the search space. Genetic algorithms (GAs) are powerful tools for solving such problems. GAs do not require gradient or Hessian information. However, to reach an optimal solution with a high degree of confidence, they typically require a large number of analyses during the optimization search. Performance of these methods is even more of an issue for problems that include continuous variables.

The work here enhances the efficiency and accuracy of the GA with memory using multivariate approximations of the objective and constraint functions individually instead of direct approximations of the overall fitness function. The primary motivation for the proposed improvements is the nature of the fitness function in constrained engineering design optimization problems. Since GAs are algorithms for unconstrained optimization, constraints are typically incorporated into the problem formulation by augmenting the objective function of the original problem with penalty terms associated with individual constraint violations. The resulting fitness function is usually highly nonlinear and nondifferentiable, which makes the multivariate approximation highly inaccurate unless a large number of exact function evaluations are performed. Since the individual response functions in many engineering problems are mostly smooth functions of the continuous variables (although

they can be highly nonlinear), high quality approximations to individual functions can be constructed without requiring a large number of function evaluations. The proposed modification improves the efficiency of the memory constructed in terms of the continuous variables. The dissertation presents the algorithmic implementation of the proposed memory scheme and demonstrates the efficiency of the proposed multivariate approximation procedure for the weight optimization of a segmented open cross section composite beam subjected to axial tension load. Results are generated to demonstrate the advantages of the proposed improvements to a standard genetic algorithm.

Dedication

The author wishes to dedicate this work to the memory of his grandfather, Vladimir Alexandrovich Pticyn (1926-1996), who was a mechanical engineer.

Acknowledgments

The author wishes to express his sincere appreciation to his advisor, Dr. Zafer Gürdal, for his guidance and support during the course of this research. He is also grateful to Dr. Layne Watson, Dr. Liviu Librescu, Dr. Mahendra P. Singh, Dr. Eric Johnson, and Dr. Levon Asryan for their time spent serving on the advisory committee.

This research was supported by the Air Force Office of Scientific Research under Grant F49620-02-1-0090, and the National Science Foundation under Grant DMS-9625968. This support is gratefully acknowledged.

The author would also like to express his deepest appreciation to his family for their encouragement, patience, and moral support.

TABLE OF CONTENTS

List of Figures **ix**

List of Tables **xi**

1 Introduction **1**

1.1 Optimization of Composite Structures by Genetic Algorithms 1

2 Optimization Methods and Algorithms **6**

2.1 Optimization Algorithms 6

2.1.1 Mathematical Methods 7

2.1.2 Evolutionary Optimization Algorithms 12

2.2 Genetic Algorithm 13

2.2.1 Representation 15

2.2.2 Selection Schemes 16

2.2.3 Crossover Operators 18

2.2.4 Mutation Operators 19

2.2.5 Constraint Handling 20

2.2.6 Convergence Criteria 21

2.3 Improving Genetic Algorithms' Performance 23

2.3.1 Binary Tree Memory 23

2.3.2 Multivariate Approximation 30

3 Thin-Walled Composite Beams 35

3.1 Laminated Composites . . . 35

3.2 Thin-Walled Beam Structures . . . 38

3.3 Problem Formulation . . . 40

3.4 Analysis . . . 42

3.4.1 Strains in the Wall Segments . . . 44

3.4.2 Forces and Moments in the Wall Segments . . . 46

3.4.3 Forces in the Beam . . . 50

3.4.4 Stiffness Matrix . . . 51

3.4.5 Centroid . . . 52

3.4.6 Stiffness and Compliance Matrices in the Centroid Coordinate System . . . 53

3.4.7 Stresses and Strains in Open-Section Beam . . . 55

3.4.8 Classical Laminate Theory . . . 55

4 Optimum Design of Thin-Walled Beam 63

4.1 Optimization Problem Formulation . . . 63

4.1.1 Selection of Design Variables . . . 63

4.1.2 Standard Form of the Optimization Problem . . . 66

4.1.3 Selection of Objective Function . . . 66

4.1.4 Selection of Constraints . . . 66

5 Results 69

5.1 Example of Composite Beam Design . . . 69

5.2 Optimum Design by Improved Genetic Algorithm . . . 71

5.2.1 GA Parameters . . . 73

5.2.2 Effect of GA Improvement . . . 73

5.3 Conclusions . . . 79

Bibliography 81

A Selected Publications 89

LIST OF FIGURES

2.1 A general framework of evolutionary algorithms. 13
2.2 A framework of canonical genetic algorithm 14
2.3 Weighted average ranking 22
2.4 Example binary tree of integers 25
2.5 Evaluation of fitness function using binary tree. 26
2.6 Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where y is not found in the tree 27
2.7 (Continuation of Figure 2.6) Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where y is found in the tree 28

3.1 Unidirectional lamina with local material principal axis (1,2) and the global reference system (ξ,η) 37
3.2 Laminate made of n single layers, coordinate locations 38
3.3 Open section segmented thin-walled beam 40
3.4 Forces in open section thin-walled beam 41
3.5 Coordinate systems employed in the analysis 42
3.6 Displacements of a beam 43
3.7 Coordinates (y_c,z_c) of the centroid C 45
3.8 The cross-section of the segmented open-section beam 46
3.9 The curvatures of the ith wall segment's axis 47

3.10 The force resultants in the ith wall segment 48
3.11 The cross-section of the segmented open-section beam 49

5.1 Cross-sections of segmented beams with different numbers of segments 71
5.2 Response functions 72
5.3 Cross-sections of segmented beams corresponding to the optimum designs 76

LIST OF TABLES

2.1 The discrete search space. 24

4.1 The cases with different numbers of continuous design variables . . . 65

5.1 Ply properties of graphite/epoxy (T300/5208) 70
5.2 Results for the selected designs 70
5.3 Ranges for design variables 72
5.4 GA parameters used in experiments. 74
5.5 Best-known optimal designs using standard GA. 74
5.6 Selection of the best design using weighted average ranking. 75
5.7 Efficiency of multivariate approximations 78
5.8 The average percent savings (S) and the average error of the multivariate approximations (E) as functions of the parameters ϵ and δ with $d_{max} = 0.5$ for the case A 79

Chapter 1

Introduction

1.1 Optimization of Composite Structures by Genetic Algorithms

During the last two decades there has been a growing interest in using genetic algorithm (GA) techniques for the optimum design of structures made of laminated composites. Most effort in this field has been devoted to optimizing the stacking sequence of laminates. Most practical laminate design requires integer or combinatorial optimization because the ply orientations are usually restricted to a certain set of discrete angles, for example, 0°, $\pm 45^\circ$, and 90°. This leads to a nonlinear mixed integer programming problem.

Contributions to the optimization of composite structures by GA methods have been made by a large number of researchers. One of the earliest attempts at optimizing a honeycomb structure using a GA was carried out by Minga (1987). The

application of GAs for optimization of composite structures was reported by Hajela (1989, 1990). Callahan and Weeks (1992) used a GA to maximize strength and stiffness of a laminate under in-plane and flexural loads. Labossiere and Turkkan (1992) used a GA and neural networks for optimization of composite materials. Haftka, Watson, Gürdal and their coworkers (Nagendra et al., 1992; Le Riche and Haftka, 1993; Nagendra et al., 1993a,b; Gürdal et al., 1994; Le Riche, 1994; Soremekun, 1997) have developed specialized GAs for stacking sequence optimization of composite laminates under buckling and strength constraints. Sargent et al. (1995) compared GAs to other random search techniques for strength design of laminated plates.

The applications of GA methods in the field of composite structure optimization include

- the weight minimization of stiffened panels and shells (Harrison et al., 1995; Nagendra et al., 1996; Kallassy and Marcelin, 1997; Jaunky et al., 1998; Kaletta and Wolf, 2000; Gantovnik et al., 2003b; Kang and Kim, 2005);
- the strength optimization of plates with open holes (Todoroki et al., 1995; Sivakumar et al., 1998);
- the improvement of the energy absorption capability of composite structures (Woodson et al., 1995; Averill et al., 1995; Crossley and Laananen, 1996);
- the optimization of sandwich-type composite structures (Malott et al., 1996; Kodiyalam et al., 1996; Wolf, 2001; Gantovnik et al., 2002b; He and Aref, 2003; Lin and Lee, 2004);
- the optimization of dimensional and thermal buckling stability under hygrothermal loads (Le Riche and Gaudin, 1998; Spallino and Thierauf, 2000);
- the strain energy minimization of laminated composite plates and shells (Potgieter and Stander, 1998);

- maximizing the fundamental frequency of the laminated composite structure (Sivakumar et al., 1998);
- the stacking sequence blending of multiple composite laminates (Soremekun et al., 2001, 2002; Adams et al., 2003; Seresta et al., 2004; Adams et al., 2004);
- the optimization of electromagnetic absorption in laminated composite structures (Matous and Dvorak, 2003);
- the optimization of composite structures considering mechanical performance and manufacturing cost (Park et al., 2004);
- the optimization of composite tire reinforcement (Abe et al., 2004);
- the optimization of composites against impact induced failure (Rahul et al., 2005).

A GA is a powerful technique for search and optimization problems with discrete variables, and is therefore particularly useful for optimization of composite laminates. However, to reach an optimal solution with a high degree of confidence typically requires a large number of function evaluations during the optimization search. Performance of GAs is even more of an issue for problems with mixed integer design variables. Several studies have concentrated on improving the reliability and efficiency of GAs. The proposed project is the extension of the study by Kogiso et al. (1994b,a), where, in order to reduce the computational cost, the authors used memory and local improvements so that information from previously analyzed design points is utilized during a search. In the first approach a memory binary tree was employed for a composite panel design problem to store pertinent information about laminate designs that have already been analyzed (Kogiso et al., 1994b). After the creation of a new population of designs, the tree structure is searched for either a design with identical stacking sequence or similar performance, such as a laminate with identical in-plane strains. Depending on the kind of information that can be retrieved from the tree, the analysis for a given laminate may be significantly reduced or may not be required at all. The second method is called local improvement

(Kogiso et al., 1994a). This technique was applied to the problem of maximizing the buckling load of a rectangular laminated composite plate. The information about previously analyzed designs is used to construct an approximation to buckling load in the neighborhood of each member of the population of designs. After that, the approximations are used to search for improved designs in small discrete spaces around nominal designs. These two methods demonstrated substantial improvements in computational efficiency for purely discrete optimization problems. The implementation, however, was not suitable for handling continuous design variables.

New approaches have been proposed to overcome this shortcoming. In particular, a new version of GA has been developed (Gantovnik et al., 2002c), consisting of memory as a function of both discrete and continuous design variables using spline (Gantovnik et al., 2002b) and multivariate (Gantovnik et al., 2002a, 2005) approximations of the fitness function in terms of continuous design variables.

this dissertation work proposes to enhance the efficiency and accuracy of the GA with memory using multivariate approximations of the objective and constraint functions individually instead of direct approximations of the fitness function. The primary motivation for the proposed improvements is the nature of the fitness function in constrained engineering design optimization problems. Since GAs are algorithms for unconstrained optimization, constraints are typically incorporated into the problem formulation by augmenting the objective function of the original problem with penalty terms associated with constraint violations. The resulting fitness function is usually highly nonlinear, which makes the multivariate approximation highly inaccurate unless a large number of exact function evaluations are performed. Since the individual response functions in many engineering problems are smooth mildly nonlinear functions of the continuous variables (although they can be highly nonlinear), high quality approximations to individual functions can be constructed without requiring a large number of function evaluations. The proposed modification is, therefore, expected to improve the efficiency of the memory constructed in terms of the continuous variables.

The dissertation presents the algorithmic implementation of the memory scheme

and demonstrates the efficiency of the combination of the memory and the multivariate approximation procedure for optimization problems with mixed integer design variables. The proposed method is applied to the optimization of a thin-walled segmented composite beam. The results of the standard GA and of the improved GA will be compared to demonstrate the relative efficiency and accuracy of the proposed algorithm.

The dissertation is divided into five chapters. The contents of each chapter are now briefly described.

- Chapter 1 gives an overview of previous work using of the evolutionary methods for optimization of composite structures.

- Chapter 2 gives a brief overview of optimization methods including the conventional methods and evolutionary methods. It also presents the basic concepts of optimization theory and mathematical programming. Some background is given for GAs. The proposed algorithm is described in detail.

- Chapter 3 describes general theory and analysis of segmented thin-walled composite beams.

- Chapter 4 presents the optimization problem formulation, describes the selection of the objective function, the constraint functions, and the design variables.

- Chapter 5 contains the results and conclusions.

Chapter 2

Optimization Methods and Algorithms

2.1 Optimization Algorithms

The choice of an optimization strategy is crucial for the successful solution of the problem. There are many important parameters such as the type of design variables (continuous, discrete or mixed), the type of objective function (smooth or nonsmooth, differentiable, convex, concave, etc.), constrained or unconconstrained problem, shape of feasible design space, the number of design variables, the number of constraints, cost of each simulation, linear or nonlinear functions, availability of first- and second-order derivatives, local and global optima, etc.

Optimization algorithms can be divided into two classes.

1. Deterministic methods: these methods use function and/or gradient information to construct mathematical approximation of the functions, and then they find an optimum point employing hill-climbing methods. These methods work normally with continuous design variables and need a small number of function

evaluations, but they may not find a global optimum point.

2. Nondeterministic methods: the most common methods in this class are random search, genetic algorithms (GAs), evolutionary programming (EP), evolution strategies (ES), simulated annealing (SA), and particle swarm optimization (PSO). These methods work entirely using only function values. These methods can work with discrete variables and (with infinite time) find a global optimum in the presence of several local optima. However, the number of function evaluations can be high even when a global optimum not found.

2.1.1 Mathematical Methods

The objective of optimization is to find the best or optimum point out of the number of possible combinations of parameters defining the problem mathematically. The optimum is either the maximum or minimum of an objective function $f(s)$. The independent variables s are called design variables. In the case of mixed integer programming problems, the objective function depends on integer variables $y \in \mathbb{Z}^\ell$, where ℓ is the number of integer design variables, and continuous variables $x \in \mathbb{R}^m$, where m is the number of continuous design variables, i.e., $s = (y, x)$.

For simplicity, consider the optimization problem with only continuous design variables, i.e., $s = x$. In this case, the design variables lie in the design space $\mathbb{R}^m$. The design variables have to satisfy a set of constraints that normally involve nonlinear functions of the design variables. The constraints divide the design space into feasible $\mathcal{F}$ and infeasible $\mathcal{U}$ regions. The constraints are equality constraints $h_k(x) = 0$, inequality constraints $g_j(x) \leq 0$, and upper and lower bounds for the vector of design variables x. The bounds are inequality constraints of the form $a_i \leq x_i \leq b_i$ for each design variable x_i.

Definition 2.1.1 (Optimization Problem)**.** Let $f : \mathbb{R}^m \to \mathbb{R}$, $g : \mathbb{R}^m \to \mathbb{R}^q$, h :

$\mathbb{R}^m \to \mathbb{R}^p$. The standard form of a typical optimization problem is

$$\begin{aligned}
&\text{Minimize} \quad f(x) \\
&\text{subject to} \\
&\qquad g_j(x) \leq 0, \quad j \in \{1, \ldots, q\}, \\
&\qquad h_k(x) = 0, \quad k \in \{1, \ldots, p\}, \\
&\text{and} \\
&\qquad (x_i)_{min} \leq x_i \leq (x_i)_{max}, \quad i \in \{1, \ldots, m\},
\end{aligned} \tag{2.1.1}$$

where $f(x)$ is the objective function, $x = (x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$ is the vector of design variables, $g_j(x) \leq 0$ are inequality constraints, $h_k(x) = 0$ are equality constraints, $(x_i)_{min}$ and $(x_i)_{max}$ are lower and upper bounds on the ith design variable, q is the number of inequality constraints, p is the number of equality constraints, and m is the number of design variables.

An equality constraint can be replaced by two inequality constraints, and (2.1.1) can be represented as

$$\begin{aligned}
&\text{Minimize} \quad f(x) \\
&\text{subject to} \\
&\qquad g_j(x) \leq 0, \quad j \in \{1, \ldots, q + 2p\}, \\
&\text{and} \\
&\qquad (x_i)_{min} \leq x_i \leq (x_i)_{max}, \quad i \in \{1, \ldots, m\}.
\end{aligned} \tag{2.1.2}$$

Finally, we consider the optimization problem without equality constraints in the

form

$$\begin{aligned} &\text{Minimize} \quad f(x) \\ &\text{subject to} \\ &\qquad g_j(x) \leq 0, \quad j \in \{1, \ldots, q\}, \\ &\text{and} \\ &\qquad (x_i)_{min} \leq x_i \leq (x_i)_{max}, \quad i \in \{1, \ldots, m\}. \end{aligned} \tag{2.1.3}$$

The feasible region $\mathcal{F}$ is described more closely in the following definition.

Definition 2.1.2 (Constraints)**.** $\mathcal{F} := \{x \in \mathbb{R}^m \,|\, g_j(x) \leq 0 \ \forall j \in \{1, \ldots, q\}\}$ is called the feasible region of the problem (2.1.3). The functions $g_j : \mathbb{R}^m \to \mathbb{R}$ define the constraints, and at a point $x \in \mathbb{R}^m$ a constraint g_j is called

$$\begin{aligned} \text{satisfied} &\quad \Leftrightarrow \quad g_j(x) \leq 0, \\ \text{active} &\quad \Leftrightarrow \quad g_j(x) = 0, \\ \text{inactive} &\quad \Leftrightarrow \quad g_j(x) < 0, \text{ and} \\ \text{violated} &\quad \Leftrightarrow \quad g_j(x) > 0. \end{aligned} \tag{2.1.4}$$

The optimization problem is called unconstrained iff $\mathcal{F} = \mathbb{R}^m$; otherwise, constrained.

In the treatment of optimization problems it is important to make a distinction between global and local minima. We invoke the following definitions.

Definition 2.1.3 (Global minimum)**.** Given a function $f : \mathcal{F} \subseteq \mathbb{R}^m \to \mathbb{R}$, $\mathcal{F} \neq \emptyset$, for $x^\star \in \mathcal{F}$ the value $f^\star := f(x^\star) > -\infty$ is called a global minimum iff

$$\forall\, x \in \mathcal{F}, \quad f(x^\star) \leq f(x). \tag{2.1.5}$$

Then $x^\star$ is a global minimum point, f is called the objective function, and the set $\mathcal{F}$ is called the feasible region. The problem of determining a global minimum point is called the global optimization problem.

Definition 2.1.4 (Local minimum)**.** For $\hat{x} \in \mathcal{F}$ the value $\hat{f} := f(\hat{x})$ is called a local minimum and $\hat{x}$ a local minimum point iff

$$\exists\, \varepsilon \in \mathbb{R},\ \varepsilon > 0\ :\ \forall\, x \in \mathcal{F},\ ||x - \hat{x}|| < \varepsilon\ \Rightarrow\ \hat{f} \leq f(x). \tag{2.1.6}$$

These definitions do not restrict generality of the optimization problem, since the identity

$$\max\{f(x) \mid x \in \mathcal{F}\} = -\min\{-f(x) \mid x \in \mathcal{F}\} \tag{2.1.7}$$

holds.

The necessary but not sufficient conditions for an optimum are the Karush-Kuhn-Tucker (KKT) conditions. A local optimum point of (2.1.3) must satisfy the KKT conditions

Definition 2.1.5 (KKT conditions)**.**

$$\begin{aligned}
&\frac{\partial f(x)}{\partial x_i} + \sum_{j=1}^{q} \lambda_j \frac{\partial g_j(x)}{\partial x_i} = 0 \quad \text{at } x = x^\star, \text{ for } i = 1, \ldots, m; \\
&\lambda_j g_j(x) = 0, \quad g_j(x) \leq 0, \quad \text{for } j = 1, \ldots, q; \\
&\lambda_j \geq 0 \quad \text{for } j = 1, \ldots, q,
\end{aligned} \tag{2.1.8}$$

where q is the number of constraints.

The geometric interpretation of these conditions means that the negative gradient vector of the objective function is inside a cone built by the positive gradient vectors of the active constraints.

For most practical problems, it is assumed that all functions, objective function and constraint functions, are nonlinear. For many engineering problems the nonlinearities are small enough that is reasonable to approximate them with linear functions. However, when the nonlinearities are not small, we often must deal directly with nonlinear programming models.

The types of programming problems can be classified as follows:

- linear programming problems (LP): in these problems the variables are continuous and both the objective function and constraints are linear;
- nonlinear programming problems (NLP): in these problems the variables are continuous and the objective function or the constraints can be either linear or nonlinear;
- mixed-integer programming problems (MIP): in these problems the objective function and constraints are functions of integer and continuous variables;
- integer programming problems (IP): in these problems there no continuous variables involved;
- binary programming problems (BP): in these problems the variables have either a value of 0 or 1.

There is no single algorithm that is always used to solve NLP. A number algorithms have been developed, each with its own advantages and disadvantages. There are three basic categories of algorithms: 1) gradient algorithms, where gradient search procedure is modified in a way to keep the search path from penetrating any constraint boundaries; 2) sequential unconstrained algorithms, where the constraints are incorporated into a penalty or barrier function. The role of such penalty function is to impose a penalty for violating constraints or even to consider only feasible solutions; 3) sequential approximation algorithms include linear approximation and quadratic approximation methods. These algorithms replace a nonlinear objective function and nonlinear constraints by a sequence of linear or quadratic approximations.

Algorithms for nonconvex NLP and for NLP with nondifferentiable functions are an area of current research. However, these problems are very difficult. Also most mathematical programming algorithms assume that the design variables are continuous. However, in many practical problems in engineering, the design variables are discrete or mixed. These appear to be promising areas for the application of evolutionary optimization algorithms.

2.1.2 Evolutionary Optimization Algorithms

Evolutionary computation includes several major branches, i.e., evolutionary strategies, evolutionary programming, genetic algorithms (GAs), and genetic programming. At the algorithmic level, they differ mainly in their representations of potential solutions and their operators used to modify the solutions.

Evolution strategies were first proposed by Rechenberg (Rechenberg, 1964) and Schwefel (Schwefel, 1968) as a numerical optimization technique. The original evolution strategy did not use populations. A population was introduced into evolution strategies later (Schwefel, 1981, 1995).

Evolutionary programming was first proposed by Fogel et al. in the mid 1960's as one way to solve artificial intelligence problems (Fogel et al., 1966a,b). Since the late 1980's evolutionary programming has also been applied to various combinatorial and numerical optimization problems.

The current framework of GAs was first proposed by Holland (Holland, 1975) and his student Jong (Jong, 1975) in 1975, and was finally popularized by another of his students, Goldberg (Goldberg, 1989). It is worth noting that some of the ideas of genetic algorithms appeared as early as 1957 in the context of simulating genetic systems (Fraser, 1957). Genetic algorithms were first proposed as adaptive search algorithms, although they have mostly been used as a global optimization algorithm for combinatorial and numerical problems.

A special branch of genetic algorithms is genetic programming. Genetic programming can be regarded as an application of genetic algorithms to evolve tree-

structured chromosomes. The term genetic programming was first used by Koza (Koza, 1989, 1990).

All evolutionary algorithms have two prominent features which distinguish themselves from other search algorithms. First, they are all population based. Second, there is information exchange among individuals in a population. Such information exchange is the result of selection and recombination in evolutionary algorithms. A general framework for evolutionary algorithms can be summarized by Figure 2.1, where the search operators are also called genetic operators for genetic algorithms. Obviously Figure 2.1 specifies a whole class of algorithms, not any particular one. Different representations of individuals and different schemes for implementing fitness evaluation, selection, and search operators define different algorithms.

1. Set $i = 0$;
2. Generate the initial generation $P(i)$ at random;
3. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) Select parents from $P(i)$ based on their fitness;
   (c) Apply search operators to the parents and produce generation $P(i + 1)$;
4. UNTIL the halting criterion is satisfied.

Figure 2.1: A general framework of evolutionary algorithms.

2.2 Genetic Algorithm

Genetic algorithms (Holland, 1975; Goldberg, 1989; Michalewicz, 1996) emphasize genetic encoding of potential solutions into chromosomes and apply genetic operators to these chromosomes. A canonical genetic algorithm (also called simple or standard GA) (Goldberg, 1989) is the one which uses binary representation, one-

1. Generate the initial population $P(0)$ at random and set $i = 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) Select parents from $P(i)$ based on their fitness as follows: Given the fitness of μ individuals as $f_1, f_2, \ldots, f_\mu$. Then select individual i with probability

$$p_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}.$$

   This is called roulette wheel selection or fitness proportional selection.
   (c) Apply crossover to selected parents;
   (d) Apply mutation to crossed-over new individuals;
   (e) Replace parents by the offspring to produce generation $P(i + 1)$;
3. UNTIL the halting criterion is satisfied.

Figure 2.2: A framework of canonical genetic algorithm

point crossover and bit-flipping mutation. A canonical genetic algorithm can be implemented as shown in Figure 2.2.

The GA-based methods can deal with discrete and/or continuous design variables, and are computationally simple. They are not limited by restrictive assumption about the search space. The following GA operators can be adopted: reproduction, crossover, and mutation. GAs work with a coding of the design variable set, they search from a population of designs rather than working with a single design. GAs use function values, and do not need derivatives.

In a GA the chromosomes contain all of the necessary information about the individuals they represent, which, in the present context, is a structural design. The GA randomly creates an initial population of individuals and then breeds new generations using some selection mechanism.

2.2.1 Representation

Encoding is the first operation in a GA. Each variable is represented using a bit-string. Each bit-string is then merged to form a chromosome that represents a design. From a mathematical point of view, we take this problem and encode the design variables as strings or vectors where each component is a symbol from an alphabet $\mathcal{A}$. The elements of the string corresponds to genes, and the values those genes can take to alleles.

Consider binary representation of continuous design variables. In this case, it is necessary to divide the search interval into a number of intervals determined by a tolerance defined by the designer. Assume that chromosomes have fixed length ℓ. The vector $x \in \prod_{i=1}^{m}[(x_i)_{min}, (x_i)_{max}]$ transforms into the chromosome $y = (y_1, \ldots, y_m) \in \mathbb{B}^{\ell}$, where the chromosome is divided into m segments of equal length ℓ_0, such that $\ell = m \times \ell_0$, and each segment $y_i = (y_{i1}, \ldots, y_{il_0}) \in \mathbb{B}^{\ell_0}, i = 1, \ldots, m$ encodes the corresponding design variable x_i. The decoding of a chromosome is two-step process including decoding the genes into the corresponding integer value between 0 and $2^{\ell_0} - 1$ and then mapping that integer to the real interval $[(x_i)_{min}, (x_i)_{max}]$. The decoding function Γ for the ith segment has the following form

$$\Gamma_i(y_{i1}, \ldots, y_{i\ell_0}) = (x_i)_{min} + \frac{(x_i)_{max} - (x_i)_{min}}{2^{\ell_0} - 1} \left(\sum_{k=1}^{\ell_0} y_{i(\ell_0+1-k)} 2^{k-1} \right). \tag{2.2.1}$$

In such representation only grid points are searched instead of the continuous space, and the solution of NLP is expected to be the grid point with the smallest value of objective function instead of the true global minimum point. The resolution Δ_i between two adjacent grid points with respect to dimension i is determined by the number l_0 of genes for encoding design variable x_i and the design variable range $[(x_i)_{min}, (x_i)_{max}]$ by

$$\Delta_i = \frac{(x_i)_{max} - (x_i)_{min}}{2^{\ell_0} - 1}. \tag{2.2.2}$$

Therefore, grid density and accuracy of the results can be increased by increasing

ℓ_0.

2.2.2 Selection Schemes

A selection scheme determines the probability of an individual being selected for producing offspring by crossover and mutation. In order to search for increasingly better individuals, fitter individuals should have higher probabilities of being selected while unfit individuals should be selected only with small probabilities. Different selection schemes have different methods of calculating selection probability. There are three major types of selection schemes, roulette wheel selection (also known as the fitness proportional selection), rank-based selection, and tournament selection.

Roulette Wheel Selection

Let $f_1, f_2, \ldots, f_\mu$ be fitness values of individuals $1, 2, \ldots, \mu$. Then the selection probability for individual i is

$$p_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}.$$

Roulette wheel selection calculates the selection probability directly from individual's fitness values. This method may cause problems in some cases. For example, if an initial population contains one or two very fit but not the best individuals and the rest of the population are not good, then these fit individuals will quickly dominate the whole population (due to their very large selection probabilities) and prevent the population from exploring other potentially better individuals. On the other hand, if individuals in a population have very similar fitness values, it will be very difficult for the population to move towards a better one since selection probabilities for fit and unfit individuals are very similar.

Rank-Based Selection

Rank-based selection does not calculate selection probabilities from fitness values directly. It sorts all individuals according to their fitness values first and then

computes selection probabilities according to their ranks rather than their fitness values. Hence rank-based selection can maintain a constant selection pressure in the evolutionary search and avoid some of the problems encountered by roulette wheel selection. There are many different rank-based selection schemes. Several are introduced below.

- Assume the best individual in a population ranks first. The probability of selecting individual i can be calculated as follows (Baker, 1985):

$$p_i = \frac{1}{\mu}\left(\eta_{max} - (\eta_{max} - \eta_{min})\frac{i-1}{\mu-1}\right), \tag{2.2.3}$$

  where μ is the population size, $\eta_{max} + \eta_{min} = 2$, η_{max} is the probability of the best individual, η_{min} is the probability for the worst individual. Intermediate individuals' ranks are decreased from η_{max} to η_{min} proportionally to their rank. Setting $\eta_{min} = 0$, the maximum selective pressure is obtained.

- A rank-based selection scheme with a stronger selection pressure is the following nonlinear ranking scheme (Yao, 1993) :

$$p_i = \frac{\mu + 1 - i}{\sum_{j=1}^{\mu} j}. \tag{2.2.4}$$

- The exponential function has the following form (Michalewicz, 1996)

$$p_i = \frac{\eta(1-\eta)^{i-1}}{c}, \tag{2.2.5}$$

  where c is the normalization factor chosen so that the sum of the probabilities is unity. In this case a larger value of η implies stronger selection pressure.

Tournament Selection

Both roulette wheel selection and rank-based selection are based on the global information in the whole population. Tournament selection only needs part of the whole population to calculate an individual's selection probability. The idea of this selection consists in finding a better solution in the tournament. The population is divided into subgroups and the best individual from each group is chosen for the next generation. Subgroups may contain 2 or more individuals.

Elitist Selection

Elitist selection is also known as elitism and elitist strategy. It always copies the best individual to the next generation without any modification. More than one individual may be copied, i.e., the best, second best, etc., may be copied to the next generation without any modification. Elitism is usually used in addition to an accepted selection scheme.

2.2.3 Crossover Operators

The crossover operator is emphasized as the most important search operator of a GA. Crossover in a GA with crossover probability p_c selects two parent individuals and recombines them to form two new individuals, i.e., two new designs.

Crossover for integer strings

Common crossover operators for integer strings include k-point crossover, $c'_{\{p_c,k\}}$, $(k \geq 1)$ and uniform crossover, $c'_{\{p_c\}}$.

- k-point crossover. This crossover can be applied to strings of any alphabet. Given two parents of length ℓ, k random numbers, $r_1, r_2, \ldots, r_k$, between 1 and $\ell - 1$ will be generated uniformly without repetition. Then an offspring is produced by taking segments (separated by $r_1, r_2, \ldots, r_k$) of parent strings

alternately, i.e., the first segment from the first parent, the second from the second parent, the third from the first parent, and so on.

- uniform crossover. This crossover is also applicable to strings of any alphabet. An offspring is generated by taking each bit or character from the corresponding bit or character in one of the two parents. The parent that the bit or character is to be taken from is chosen uniformly at random.

Crossover for Real-valued Vectors

The blending method (McMahon et al., 1998) is used to combine parameter values from the two parents into new parameter values in the offspring in the case of real valued design variables. A single offspring parameter value, c, comes from a combination of the two corresponding parent parameter values, p_1 and p_2, as follows

$$\begin{aligned} c_1 &= \mu + \sigma r, \\ c_2 &= \max(c_1, x_{min}), \\ c &= \min(c_2, x_{max}), \end{aligned} \tag{2.2.6}$$

where

$$\mu = \frac{p_1 + p_2}{2}, \quad \sigma = \frac{|p_2 - p_1|}{2}, \tag{2.2.7}$$

r is a normally distributed random number with mean zero and unit standard deviation.

2.2.4 Mutation Operators

Mutation operators, $m'_{\{p_m\}}$, used for vectors of real values are usually based on certain probability distributions, such as uniform, Gaussian (normal), and Cauchy distributions. Mutation for integer strings is usually a bit-flipping operation. Mutation is needed because it allows new genetic patterns to be formed improving the

search method. The mutation probability $p_m \in [0, 1]$ per bit is usually very small. The common setting is $p_m = 0.001$.

Mutation for Integer Strings

- Bit-Flipping. Bit-flipping mutation simply flips a bit from 0 to 1 or from 1 to 0 with a certain probability. This probability is called the mutation probability or mutation rate. Bit-flipping mutation can be generalized to mutate strings of any alphabet. The generalized mutation works as follows: for each character in a string, replace it with another randomly chosen character (not the same as the one to be replaced) in the alphabet with certain mutation probability.

- Random Bit. This mutation does not flip a bit. It replaces a bit by 0 or 1 with equal probability, i.e., 0.5, respectively. The generalized version of this mutation works as follows: for each character in a string, replace it with a randomly chosen character (could be the same as the one to be replaced) in the alphabet with a certain mutation probability.

2.2.5 Constraint Handling

GAs are suited for unconstrained optimization problems. The main problem in applying evolutionary algorithms to solving a constrained problem is how to deal with constraints because evolutionary operators used for manipulating chromosomes may yield infeasible solutions. Extensive overviews of the different constraint handling techniques available are given by Michalewicz (1996) and Coello (2002).

A penalty function strategy is based on the strategies developed for conventional optimization methods in which solutions that are out of the feasible domain are penalized using a penalty coefficient. In other words a constrained optimization problem is transformed to an unconstrained optimization problem.

In this study the method implemented is based on the approach used by Powell and Skolnick (1993) and recently modified by Deb (2000). This method is called the

method of superiority of feasible points. The fitness function $\phi(s)$ has the following form

$$\phi(s) = \begin{cases} f(s), & \text{if } s \text{ is feasible,} \\ f_{max} + \sum_{j=1}^{q} c_j(s), & \text{otherwise,} \end{cases} \tag{2.2.8}$$

where f_{max} is the function value of the worst known feasible solution, the function c_j measures the violation of the j-th constraint as follows

$$c_j(s) = \max\{0, g_j(s)\}, \quad 1 \leq j \leq q. \tag{2.2.9}$$

There is no need for penalty coefficients here because the feasible solutions are always evaluated to be better than infeasible solutions, and infeasible solutions are compared purely on the basis of their constraint violations. The constraints g_j should be normalized.

It is possible to obtain several solutions that will have the same value of the objective function and will satisfy all the constraints. In this situation, the weighted average ranking method is used to obtain the best solution from a set of candidate solutions (Collette and Siarry, 2003). This method is frequently used in multiobjective optimization, and it is based on the algorithm shown in Figure 2.3. The design with smallest rank can be selected as the best optimal solution.

2.2.6 Convergence Criteria

Three convergence criteria are used in this work. If just one of them is reached, then the optimization process terminates. These criteria are

1. when the percentage difference between the average value of all the designs and the best design in a population reaches a very small specified value c_1,

$$\frac{|f_a - f^\star|}{|f_a|} \times 100 \leq c_1, \tag{2.2.10}$$

1. s is a solution to compare;
2. A is a set of solutions, which have best objective function and satisfy all constraints g_j (A does not contain s);
3. $j = 1$;
4. REPEAT
   (a) Compare s with A with respect to the g_j;
   (b) N_j=number of solutions from A better than s with respect to the g_j;
   (c) j=j+1;
5. UNTIL $j > q$
6. The rank assigned to s is $N = \sum_{j=1}^{q} N_j$;

Figure 2.3: Weighted average ranking

where $f^\star$ is the fittest design in a population, f_a is the average objective value in a generation defined by

$$f_a = \frac{1}{\mu} \sum_{j=1}^{\mu} f_j, \tag{2.2.11}$$

and μ is the population size,

2. if the fittest design has not changed for 50 successive generations, or the difference of the fittest design of the current generation, $f_c^\star$, and that of 50 generations before, $f_b^\star$, is less than a small amount c_2, i.e.,

$$\frac{|f_c^\star - f_b^\star|}{|f_c^\star|} \times 100 \leq c_2, \tag{2.2.12}$$

3. if the total number of generations is reached.

2.3 Improving Genetic Algorithms' Performance

The standard GA, being faced with the usual conflict between reliability and computation time, often results in an unsatisfactory compromise, characterized by a slow convergence, when an exact solution is required. The key to improving the performance of the GA is to reduce the time needed to calculate the fitness. By examining the mechanisms of the GA, it is seen that the diversity of the population decreases as the algorithm runs. The fitness values for the same chromosomes are recalculated repeatedly. If previously calculated fitness values can be efficiently saved, computation time will diminish significantly. This suggests an opportunity for performance improvement. By efficiently storing fitness values, GA performance can be dramatically improved.

The size of the discrete search space for the stacking sequence optimization problem with the size of alphabet $N(\mathcal{A})$ and the length of chromosomes ℓ is given by Table 2.1 and is equal to

$$\sum_{i=1}^{\ell} N^i(\mathcal{A}). \tag{2.3.1}$$

For example, if $N(\mathcal{A}) = 6$ and $\ell = 3$, then the discrete search space contains 1092 possible designs. Although this is not an unmanageable size, the problem quickly becomes unwieldy for a slightly larger optimization problem. With current technology, it is not feasible to store large size arrays efficiently. This leads us to consider alternative methods for storing the fitness values.

2.3.1 Binary Tree Memory

Kogiso et al. (1994a) used information about past designs to reduce the number of analyses required by the GA. The information was stored in a binary tree format and was used to construct a set of linear approximations to the buckling load in the neighborhood of each member of the population of designs. The approximations were then used to seek nearby improved designs in a procedure called local improvement. The procedure was applied to the problem of buckling load maximization for

Table 2.1: The discrete search space.

ℓ	$N(\mathcal{A})$		
	2	3	4
1	2	3	4
2	6	12	20
3	14	39	84
4	30	120	340
5	62	363	1364
6	126	1092	5460
7	254	3279	21844
8	510	9840	87380
9	1022	29523	349524
10	2046	88572	1398100
20	2097150	5230176600	1466015503700

a given laminate thickness. Substantial reductions in the number of required analyses were achieved by this use of memory of past designs. The same strategy was applied to the dual problem of the minimum thickness design of laminates subject to buckling and strength constraints (Kogiso et al., 1994b).

Gantovnik et al. (2002b) used an augmented GA with memory that can work with discrete and continuous variable simultaneously. The new algorithm was applied to stacking sequence design of laminated sandwich composite panels that involves both discrete variables and one continuous design variable. Spline approximation was used in the local memory for the continuous part of the design space.

Gantovnik et al. (2003a) used the same approach for a GA with memory that can work with discrete and several continuous design variables. The efficiency of the proposed multivariate approximation based procedure, as well as the use of memory for a GA that can handle continuous variables, were investigated for the weight optimization of a lattice shell with laminated composite skins subjected to axial compressive load. Later this algorithm was supplemented with multivariate approximation of the individual function's responses in terms of continuous design variables (Gantovnik et al., 2003b, 2005). The selected publications about the discussed approaches are presented in Appendix A.

Binary Tree Memory for Discrete Design Variables

A discussion of binary trees can be found in Vowels (1998). A binary tree is a linked list structure in which each node may point to up to two other nodes. In a binary search tree, each left pointer points to nodes containing elements that are smaller than the element in the current node; each right pointer points to nodes containing elements that are greater than the element in the current node. The binary tree is used to store data pertinent to the design such as the design string and its associated fitness and constraint function values. A sample binary tree with integer nodes is shown in Figure 2.4

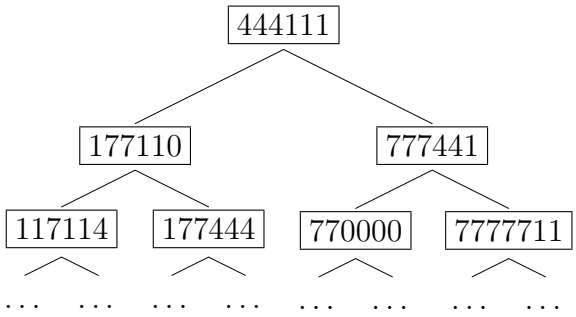


Figure 2.4: Example binary tree of integers

A binary tree has several properties of great practical value, one of which is that the data can be retrieved, modified, and inserted relatively quickly. If the tree is perfectly balanced, the cost of inserting an element in a tree with n nodes is proportional to $\log_2 n$ steps, and rebalancing the tree after an insertion may take as little as several steps, but at most takes $\log_2 n$ steps. Thus, the total time is of the order of $\log_2 n$.

In the standard genetic algorithm, a new population may contain designs that have already been encountered in the previous generations, especially towards the end of the optimization process. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. Algorithm 2.5 shows the pseudo code of the fitness function evaluation with the aid of the binary tree. After a new generation of designs is created by the genetic operations, the binary tree is searched

```
search for the given design in the binary tree;
if found then
   get the fitness function value from the binary tree;
else
   perform exact analysis;
end if
```

Figure 2.5: Evaluation of fitness function using binary tree.

for each new design. If the design is found, the fitness value is retrieved from the binary tree without conducting an analysis. Otherwise, the fitness is obtained based on an exact analysis. This new design and its fitness value are then inserted in the tree as a new node.

Binary Tree Memory for Discrete and Continuous Design Variables

The procedure described above works well for purely discrete optimization problems where designs are completely described by discrete strings (Kogiso et al., 1994a). In the case of mixed integer optimization problems where designs include discrete and continuous variables, the solution becomes more complicated. If the continuous variables are also discretized into a fine discrete set, the possibility of creating a child design that has the same discrete and continuous parts as one of the earlier designs diminishes substantially. In the worst case, if the continuous design variables are represented as real numbers, which is the approach used by most recent research work, it may not be possible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless, and result in many exact analyses even if the real part of the new child is different from one of the earlier designs by a minute amount.

The main idea of the memory approach is to construct a response surface approximation for every constraint function as a function of the continuous variables using historical data values, and estimate from the stored data whenever appropriate. The memory in this case consists of two parts: a binary tree, which consists of the nodes that have different discrete parts of the design, and a storage part at each

if y is not found in the tree **then**
  evaluate $g_0(s), \ldots, g_q(s)$;
  evaluate $\phi(s)$; $n := 1$; $x^{(1)} := x$;
  for $j = 0$ to q **do**
    $c_j := 1$; $\varrho_j := 0.0$; $I_{j1} := 1$; $d_{j1} := 0.0$;
    $T_j := \left(\left\{ \left(I_{ji}, g_j(y, x^{(I_{ji})}), d_{ji} \right) \right\}_{i=1}^{c_j}, c_j, \varrho_j \right)$;
  end for
  $D := \left(\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_q \right)$; add a node corresponding to (y, D);
  return $\phi(s)$;
end if

Figure 2.6: Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where y is not found in the tree

node that keeps the continuous values and the corresponding constraint functions' values. That is, each node contains several real arrays that store the continuous variables' values and their corresponding constraint functions' values.

In order for the memory to be functional, it is necessary to have accumulated a sufficient number of designs with different continuous values for a particular discrete design point so that the approximations can be constructed.

Naturally, some of the discrete nodes will not have more than a few designs with different continuous values. However, it is possible that as the GA search progresses good discrete parts will start appearing repeatedly with different continuous values. In this case, one will be able to construct good quality response surface approximations to the data.

In addition to building the multivariate approximations, it is important to assess accuracy of the multivariate approximation at new continuous points, so that a decision may be made either to accept the approximation or perform exact function evaluation.

Based on the multivariate approximation, the proposed algorithm described by the following pseudo code is then used to decide when to retrieve the constraint function values from the approximations, and when to do an exact analysis and add the new data point to the approximation database.

For the description of the pseudo code, denote a design consisting of continuous

if y is found in the tree **then**
  for $j = 0$ to q **do**
    if $B(j, y, x)$ **then**
      evaluate $g_j(s)$;
    else
      if $c_j < (c_j)_{min}$ **then**
        [add new point x to database]
        evaluate $g_j(s)$; $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$; $c_j := c_j + 1$;
        $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(s), 0.0)\}$; $I_{jc_j} := n$;
        $\varrho_j := \left| \max\limits_{1 \le i \le c_j} \left\{ g_j(y, x^{(I_{ji})}) \right\} - \min\limits_{1 \le i \le c_j} \left\{ g_j(y, x^{(I_{ji})}) \right\} \right|$;
      else
        construct $\tilde{g}_j(x)$ using the data in
        $\left\{ \left(x^{(I_{ji})}, g_j\left(y, x^{(I_{ji})}\right) \right) \right\}_{i=1}^{c_j}$;
        define k and d^* by
        $d^* := d_{jk} - \left\| x - x^{(I_{jk})} \right\| = \max\limits_{1 \le i \le c_j} \left\{ d_{ji} - \left\| x - x^{(I_{ji})} \right\| \right\}$;
        if $d^* \ge 0.0$ and $\left| g_j\left(y, x^{(I_{jk})}\right) - \tilde{g}_j(x) \right| < \delta_j \, \varrho_j$ **then**
          [accept approximation as good enough]
          $g_j(s) := \tilde{g}_j(x)$;
        else
          evaluate $g_j(s)$;
          $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$;
          $c_j := c_j + 1$;
          [update trust region radius for x]
          if $|g_j(s) - \tilde{g}_j(x)| > \epsilon_j$ **then**
            $d_{jc_j} := 0.0$;
          else
            $d_{jc_j} := \min\left\{ (d_j)_{max}, \left\| x - x^{(I_{jk})} \right\| \right\}$; $d_{jk} := d_{jc_j}$;
          end if
          [update node database with information for x]
          $(T_j)_1 := (T_j)_1 \cup \left\{ \left(n, g_j(s), d_{jc_j} \right) \right\}$; $I_{jc_j} := n$;
          $\varrho_j := \left| \max\limits_{1 \le i \le c_j} \left\{ g_j(y, x^{(I_{ji})}) \right\} - \min\limits_{1 \le i \le c_j} \left\{ g_j(y, x^{(I_{ji})}) \right\} \right|$;
        end if
      end if
    end if
  end for
  evaluate $\phi(s)$ using $g_0(s), \ldots, g_q(s)$;
  return $\phi(s)$;
end if

Figure 2.7: (Continuation of Figure 2.6) Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where y is found in the tree

and discrete design variables by $s = (y, x)$, where $y \in \mathbb{Z}^\ell$ is a ℓ-dimensional integer design vector for the discrete space, and $x \in \mathcal{S} = \prod_{i=1}^{m}[(x_i)_{min}, (x_i)_{max}] \in \mathbb{R}^m$ is a real m-dimensional design vector for the continuous variables. Let $g_0(s) = f(s)$ be the corresponding objective function, $g_1(s), \ldots, g_q(s)$ the corresponding constraint functions, where q is the total number of inequality constraints, and $\phi\big(s\big)$ the corresponding fitness value of the individual defined in terms of the constraint functions and the objective function.

Furthermore, define $d \in \mathcal{S}$ to be a real distance corresponding to a trust region radius about a specific point in the database. Let $D = \big(\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_q\big)$ be the set of n observed exact analysis and their corresponding information within a given discrete node, where $T_j = \left(\left\{\big(I_{ji}, g_j(y, x^{(I_{ji})}), d_{ji}\big)\right\}_{i=1}^{c_j}, c_j, \varrho_j\right)$ is the data set associated with the jth constraint function, I_{ji} is the index pointing to the global design data set $\{x^{(i)}\}_{i=1}^{n}$, $g_j(y, x^{(I_{ji})})$ is the value of the jth constraint, d_{ji} is the corresponding trust region radius, c_j is the counter indicating the number of points in the design data set corresponding to the jth constraint, $\varrho_j = \left|\max_{1\le i\le c_j}\left\{g_j(y, x^{(I_{ji})})\right\} - \min_{1\le i\le c_j}\left\{g_j(y, x^{(I_{ji})})\right\}\right|$ is the difference between the current maximum and minimum values of the jth constraint. $\tilde{g}_j(x)$ is the approximation to $g_j(s)$ built from the data in T_j. Finally, each node in the binary tree memory structure records a tuple of the form (y, D). The pseudo code for processing a candidate individual $s = (y, x)$ is defined by Figures 2.6 and 2.7. $B(j, y, x)$ is a Boolean function intended to provide the option of bypassing the algorithm's normal logic, if dictated by a priori knowledge about the function g_j or individual (y, x). The parameters $(c_j)_{min}$ are defined separately for each constraint function, and their values are based on the function complexity and approximation method used for the constraint function. The algorithm uses three real user-specified parameters, d_{max}, δ, and ϵ, all indexed by j. The parameter $d_{max} > 0$ is an upper bound on the trust region radius about each sample point $x^{(i)}$. The parameter δ is chosen to satisfy $0 < \delta < 1$, and in higher dimensions protects against large variations in g_j in unsampled directions. Finally, the parameter $\epsilon > 0$ is the selected acceptable approximation accuracy, and is solely based on engineering considerations.

2.3.2 Multivariate Approximation

The functions used to define the engineering optimization problem often are computationally expensive. Faced with such prohibitive computational costs, the promising approach is to make use of approximating surrogates since they are much less expensive to compute.

A spline-based approximation was used for only one continuous variable (Gantovnik et al., 2002b). In the current work an evolving database of continuous variable points is used to construct multivariate response surface approximations at those discrete nodes that are processed frequently (Gantovnik et al., 2003a).

Local methods are attractive for very large data sets because the interpolation or approximation at any point can be achieved by considering only a local subset of the data. In order for the overall method to be local, it is necessary that the weight functions have local support, that is, be nonzero over a bounded region, or at a limited number of the data points.

The original global inverse distance weighted interpolation method is due to Shepard (1968). All methods of this type may be viewed as generalizations of Shepard's method.

The specific problem for which we provide a constructive solution can be stated as follows: given a set of irregularly distributed points $x^{(i)} \in \mathbb{R}^m$, $i = 1, \ldots, n$, and scalar values $g^{(i)}$ associated with each point satisfying $g^{(i)} = g(x^{(i)})$ for some underlying function $g : \mathbb{R}^m \to \mathbb{R}$, look for for an interpolating function $\tilde{g} \approx g$ such that $\tilde{g}(x^{(i)}) = g^{(i)}$.

Define an approximation to $g(x)$ by

$$\tilde{g}(x) = \frac{\sum_{k=1}^{n} W_k(x) g^{(k)}}{\sum_{i=1}^{n} W_i(x)},$$

where the weight functions $W_k(x)$ are defined in the original paper Shepard (1968)

as

$$W_k(x) = \frac{1}{\left\|x - x^{(k)}\right\|_2^2}.$$

However, this form of the weight functions accords too much influence to data points that are far away from the point of approximation and may be unacceptable in some cases.

Franke and Nielson (1980) developed a modification that eliminates the deficiencies of the original Shepard's method. They modified the weight function $W_k(x)$ to have local support and hence to localize the overall approximation, and replaced $g^{(k)}$ with a suitable local approximation $P_k(x)$. This method is called the local quadratic Shepard method and has the general form

$$\tilde{g}(x) = \frac{\sum_{k=1}^{n} W_k(x) P_k(x)}{\sum_{i=1}^{n} W_i(x)}, \tag{2.3.2}$$

where $P_k(x)$ is a local approximant to the function $g(x)$ centered at $x^{(k)}$, with the property that $P_k\left(x^{(k)}\right) = g^{(k)}$. The choice for the weight functions $W_k(x)$ used by Renka (1988a,b,c) was suggested by Franke and Nielson (1980) and is of the following form

$$W_k(x) = \left[\frac{\left(R_w^{(k)} - d_k(x)\right)_+}{R_w^{(k)} d_k(x)}\right]^2, \tag{2.3.3}$$

where, for real w, w_+ is defined as $w_+ = \max\{0, w\}$, $d_k(x) = \left\|x - x^{(k)}\right\|_2$ is the Euclidean distance between the points x and $x^{(k)}$, and the constant $R_w^{(k)} > 0$ is a radius of influence about the point $x^{(k)}$ chosen just large enough to include N_w points. The data at $x^{(k)}$ only influences $g(x)$ values within this radius.

The polynomial function P_k is written as a Taylor series about the point $x^{(k)}$ with constant term $g^{(k)} = P_k(x^{(k)})$ and coefficients chosen to minimize the weighted

square error

$$\sum_{\substack{i=1 \\ i \neq k}}^{n} \omega_i\big(x^{(k)}\big) \left[P_k(x^{(i)}) - g^{(i)}\right]^2$$

with weights

$$\omega_i(x^{(k)}) = \left[\frac{\big(R_p^{(i)} - d_i(x^{(k)})\big)_+}{R_p^{(i)} d_i(x^{(k)})}\right]^2,$$

and $R_p^{(k)} > 0$ defining a radius about $x^{(k)}$ within which data is used for the least squares fit. R_w and R_p are taken by Franke and Nielson (1980) as

$$R_w = \frac{D}{2}\sqrt{\frac{N_w}{n}}, \qquad R_p = \frac{D}{2}\sqrt{\frac{N_p}{n}},$$

where $D = \max_{i,j} \left\|x^{(i)} - x^{(j)}\right\|_2$ is the maximum distance between any two data points, and N_w and N_p are arbitrary constants. The constant values for R_w and R_p are appropriate assuming uniform data density. If the data density is not uniform, then the radii R_w and R_p should depend on k.

The basis function $P_k(x)$ was the constant $g^{(k)}$ in the original Shepard algorithm (Shepard, 1968), and later variants used a quadratic polynomial (Franke and Nielson, 1980; Renka, 1988a,b,c; Berry and Minser, 1999), a cubic polynomial (Renka, 1999a), and a cosine trigonometric polynomial (Renka, 1999b). The primary disadvantages for large data sets is that a considerable amount of preprocessing is needed to determine closest points and calculate the local approximation. The second order polynomial models have $(m+2)(m+1)/2$ coefficients for m design variables, therefore the number of coefficients for $P_k(x)$ is at least $(m+2)(m+1)/2$, which becomes prohibitive for a typical engineering problem, where $m \gg 5$ and function values are expensive. Also the use of multivariate polynomials for the evaluation of the nodal functions leads to the loss of the main advantage of Shepard's original method, namely its independence in the evaluation phase from the space dimension. In fact, by increasing the space dimension m, the evaluation of the nodal functions $P_k(x)$ can become computationally expensive.

Of course, the use of polynomials of degree < 2 for $P_k(x)$ is inadequate to describe the local behavior of highly nonlinear objective functions, e.g., the penalty function based fitness functions in GAs. However, since the individual response functions (constraints, components of objective function) in many engineering problems are slowly varying smooth functions of continuous variables, high quality approximations to these component functions can be constructed without requiring a large number of function evaluations by using linear local approximations. Examples of such response functions are shown in Figure 5.2.

In the context of a genetic algorithm with a memory binary tree for the discrete and continuous variables, each tree node would have to accumulate $N_p = \Omega\big(m^2\big)$ function values $g^{(k)}$ before an approximation $\tilde{g}(x)$ could be constructed at that node using quadratic $P_k(x)$. These complexity considerations motivate the choice of $P_k(x)$ as linear, which only requires $N_p > m$ function values to construct the local least squares fit $P_k(x)$. The radii R_w and R_p vary with k and are taken to be

$$\begin{aligned} R_w^{(k)} &= 2 \min_{\substack{1 \le i \le n \\ i \neq k}} \left\| x^{(k)} - x^{(i)} \right\|_2, \\ R_p^{(k)} &= \min\{r \mid \overline{B(x^{(k)}, r)} \text{ contains at least } 3m/2 \text{ of the points } x^{(i)}\}, \end{aligned} \tag{2.3.4}$$

where $\overline{B(x,r)}$ is the closed ball of radius r with center x.

The linear Shepard method would choose $P_k(x)$ as

$$P_k(x) = g^{(k)} + \sum_{j=1}^{m} a_j^{(k)} \left(x_j - x_j^{(k)} \right). \tag{2.3.5}$$

Let $S = \big\{i_1, i_2, \ldots, i_{s_k}\big\} = \big\{i \mid i \neq k \text{ and } \omega_i\big(x^{(k)}\big) \neq 0\big\}$, the set of indices corresponding to points and weights $\omega_i\big(x^{(k)}\big) \neq 0$ that determine the local least squares approximation $P_k(x)$. Define the $s_k \times m$ matrix A and s_k-vector b by

$$A_{j\cdot} = \sqrt{\omega_{i_j}\left(x^{(k)}\right)} \left(x^{(i_j)} - x^{(k)}\right)^T, \qquad b_j = \sqrt{\omega_{i_j}\left(x^{(k)}\right)} \left(g^{(i_j)} - g^{(k)}\right).$$

The coefficients $a^{(k)}$ of $P_k(x)$ are then the minimum norm solution of the linear least squares problem

$$\min_{a \in \mathbb{R}^m} \|Aa - b\|_2 ,$$

found by using a complete orthogonal factorization of A via the LAPACK subroutine DGELSX (Anderson et al., 1999).

Chapter 3

Thin-Walled Composite Beams

3.1 Laminated Composites

The mechanics of laminated composite materials is generally studied at two levels: micromechanics and macromechanics. Micromechanics defines the relationship between the properties of the constituents and those of the lamina. For most engineering design applications an analysis addressed to the micromechanical level is unrealistic. At the macromechanical level the properties of the individual layers are assumed to be known a priori. Macromechanics investigates the interaction of the individual layers of a laminate with one another and their effects on the overall response quantities. The use of macromechanical formulations in designing composite laminates for desired material characteristics is well established. Macromechanics is based on continuum mechanics, which models each lamina as homogeneous and orthotropic and ignores the fiber/matrix interface.

Lamination theory is the mathematical modeling technique used to predict the

macromechanical behavior of a laminate based on an arbitrary assembly of homogeneous orthotropic laminae. Two-dimensional modeling is most common; three-dimensional theory is very complex and should be limited to selected problems.

A real structure generally will consist of several laminae. The mechanical characteristics of a unidirectional laminate are very limited in the transverse direction. One can overcome this restriction by making laminates with layers stacked at different fiber angles corresponding to complex loading and stiffness requirements. To minimize the increasing costs and weights for an such approach one has to optimize the laminae angles.

The behavior of a multidirectional laminate is a function of the laminae properties, i.e., their elastic moduli, thickness, angle orientations, and the stacking sequence of the individual layers.

The mechanical modeling requires the following assumptions:

1. There is a monolithic bonding of all laminae, i.e there is no slip between laminae at their interface.

2. Each layer is quasi-homogeneous and orthotropic, but the angle orientations may be different.

3. The strains and displacements are continuous throughout the laminate. The in-plane displacements and strains vary linearly through the laminate thickness.

There are also some rules that guarantee an optimal global laminate behavior:

1. Symmetric laminate staking yields an uncoupled modeling and analysis of in-plane and bending stress-strain relations and avoids distortion in the processing.

2. Laminates should be made up of at least three unidirectional laminae with different fiber angle orientation.

3. Although it is possible to determine an optimum orientation sequence of laminates for any given condition, it is more practical from a fabrication standpoint to limit the number of fiber orientations to a few specific laminae types, e.g. fiber orientations of $0°, \pm 45°$ and $90°$.

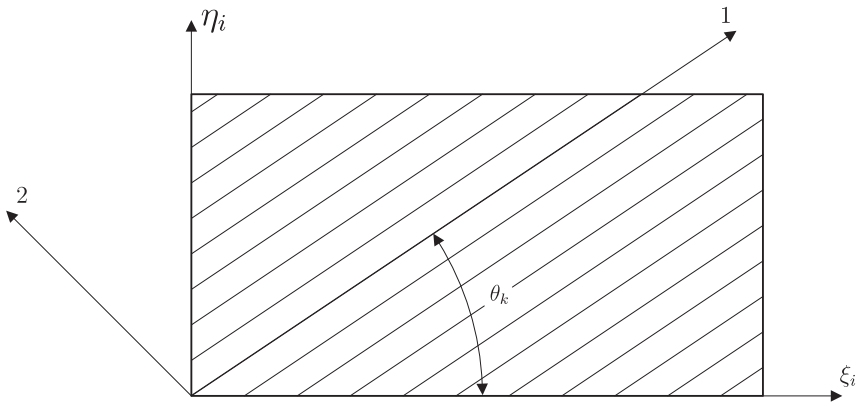


Figure 3.1: Unidirectional lamina with local material principal axis (1,2) and the global reference system (ξ,η)

Each layer of a laminate is defined by its location in the laminate, its material and fiber orientation. Consider ξ–η–ζ orthogonal coordinate system, where the η coordinate is perpendicular to the plane of laminate. The global and local material reference systems are given in Figure 3.1. We consider the ply material axes to be rotated away from the global axes by an angle θ_k, positive in the counterclockwise direction. Examine a laminate made of n plies as shown in Figure 3.2. Each ply has a thickness of

$$h_k = \zeta_k - \zeta_{k-1}, \quad k = 1, \ldots, n, \tag{3.1.1}$$

where ζ_k and ζ_{k-1} are the coordinates of the top and the bottom surface of the laminate. The total laminate thickness is

$$h = \sum_{k=1}^{n} h_k. \tag{3.1.2}$$

The distance from the mid-plane is defined as

$$\zeta_k = -\frac{h}{2} + \sum_{j=1}^{k} h_j. \tag{3.1.3}$$

The coordinates of the top and the bottom surface of the laminate are $\zeta_n = h/2$ and $\zeta_0 = -h/2$, respectively. The orientations of continuous unidirectional plies are specified by the angle θ with respect to the ξ axis. The possible fiber orientations are limited to $0°, \pm 45°$ and $90°$. The laminate code has the following form

$$[\theta_1/\theta_2/\ldots/\theta_n], \tag{3.1.4}$$

where a slash sign separates each ply.

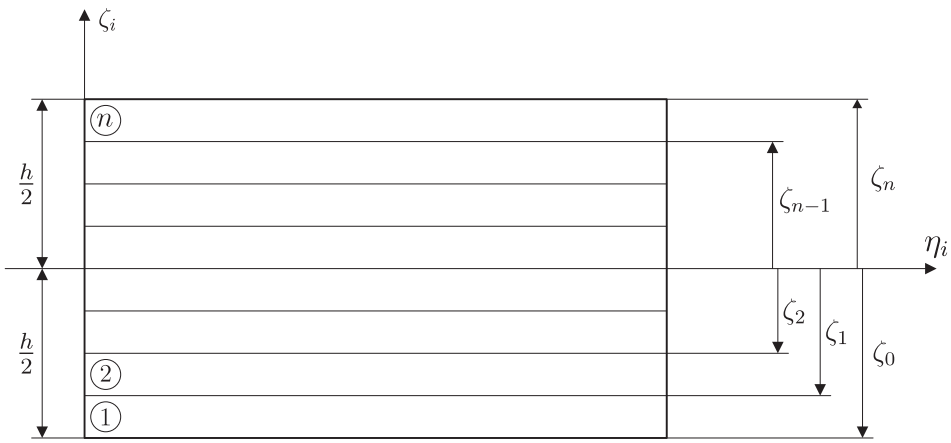


Figure 3.2: Laminate made of n single layers, coordinate locations

3.2 Thin-Walled Beam Structures

Theory of isotropic thin-walled beam open section beams was developed five decades ago by Vlasov (1958, 1961). This theory was extended for orthotropic beams by Bank and Bednarczyk (1988); Barbero et al. (1993). Several authors extended the

classical theories for non-orthotropic beams. Bauld and Tzeng (1984) derived a Vlasov type theory for thin walled open section beams with symmetrical layup only. While the transverse shear deformation was neglected in this derivation, the restrained warping was included. Kobelev and Larichev (1988) extended the Bauld and Tzeng (1984) work by taking the transverse shear deformation into account. Mansfield and Sobey (1979) neglected both the transverse shear deformation and the effect of restrained warping. They obtained the following constitutive equations

$$\begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{Bmatrix} = \begin{bmatrix} \bar{P}_{11} & \bar{P}_{12} & \bar{P}_{13} & \bar{P}_{14} \\ \bar{P}_{21} & \bar{P}_{22} & \bar{P}_{23} & \bar{P}_{24} \\ \bar{P}_{31} & \bar{P}_{32} & \bar{P}_{33} & \bar{P}_{34} \\ \bar{P}_{41} & \bar{P}_{42} & \bar{P}_{43} & \bar{P}_{44} \end{bmatrix} \begin{Bmatrix} \epsilon^{\circ}_{\bar{x}} \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{Bmatrix}, \tag{3.2.1}$$

where $\hat{N}_{\bar{x}}$ is the normal force, $\hat{M}_{\bar{y}}$ and $\hat{M}_{\bar{z}}$ are the bending moments about the $\bar{y}$ and $\bar{z}$ axes, $\hat{T}_{\bar{x}}$ is the Saint Venant torque, $\epsilon^{\circ}_{\bar{x}}$ is the axial strain of the $\bar{x}$ axis, $\kappa_{\bar{y}}$ and $\kappa_{\bar{z}}$ are the curvatures of the $\bar{x}$ axis about the $\bar{y}$ and $\bar{z}$ axes, $\vartheta_{\bar{x}}$ is the rate of the twist, and $\bar{P}$ is the stiffness matrix. Kollar and Pluzsik (2002) derived a general theory for non-orthotropic composite beams. They present explicit expression for the stiffness matrix of thin-walled open and closed section composite beams. There is no restriction on the layup of the wall segments and the local bending stiffnesses are taken into account. They neglected the effects of restrained warping and transverse shear deformations. The effect of this approximation was investigated in Pluzsik and Kollar (2002).

Obviously, it is necessary to give special attention to thin-walled composite structures. Because of their advantages laminated composite beam structures will play an increasing role in the design of future constructions in the aeronautical and aerospace, automotive and naval industries. In addition to the known advantages of high strength and high stiffness to weight ratio, the other elastic and structural characteristics, depending on the laminate stacking sequence, can be successfully investigated in order to enhance the response characteristics of the thin-walled com-

posite beam structures.

3.3 Problem Formulation

Many structural problems lead to the modeling and analysis of complex structures containing thin-walled elements. Such structures have a significant larger dimension in one direction in comparison with the dimensions in transverse directions, and a significant smaller thickness of the walls in comparison with the transverse dimensions.

Consider thin-walled open section prismatic beam shown in Figure 3.3. The walls of the beam may consists of several plies made of composite materials. The beam's wall consists of flat segments denoted by the subscript i, where $i = 1, 2, \ldots, \iota$, and ι is the total number of wall segments. The cross-section may be symmetrical or unsymmetrical, and the layup of the beam is arbitrary.

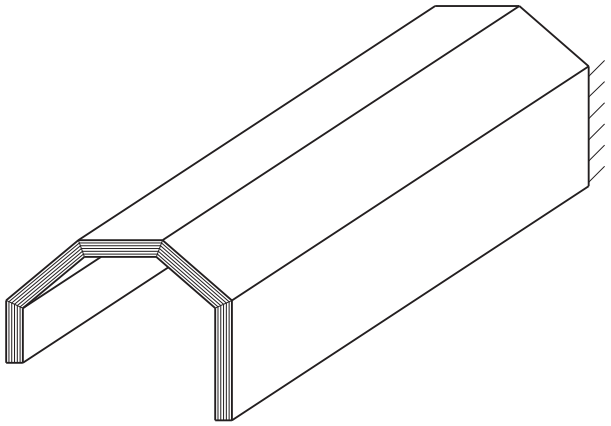


Figure 3.3: Open section segmented thin-walled beam

The beam is subjected to an axial force $\hat{N}_x$, bending moments $\hat{M}_y$ and $\hat{M}_z$, and torque $\hat{T}_x$ acting at the centroid C as shown in Figure 3.4. The centroid C is defined such that the force $\hat{N}_x$ applied at the centroid does not result in the curvatures of the axis of the beam.

We use the following coordinate systems. For the beam we use the x–y–z coordinate system with the origin at the centroid C and the $\bar{x}$–$\bar{y}$–$\bar{z}$ coordinate system

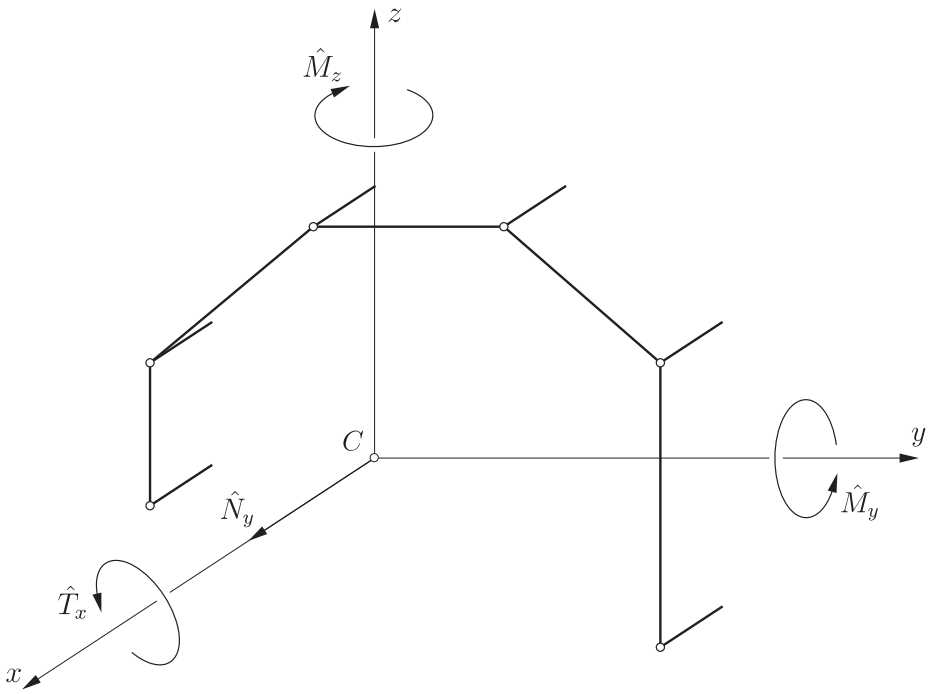


Figure 3.4: Forces in open section thin-walled beam

with the origin at an arbitrary chosen point O as shown in Figure 3.5. In addition, for the ith segment we employ the ξ_i–η_i–ζ_i coordinate system with the origin at the center of the reference plane of the ith segment. The axis ξ_i is parallel to the x coordinate, the axis η_i is along the circumference of the wall, and the axis ζ_i is perpendicular to the circumference of the wall.

The displacements of the longitudinal axis passing through the centroid are u, v, w, and ψ as shown in Figure 3.6, where u is the axial displacement, v and w are the transverse displacements in the y and z directions, respectively, and ψ is the rotation of the cross-section. In the x–y–z coordinate system the relationships between these displacements, the axial strain ϵ_x°, the curvatures κ_y and κ_z of the x axis, and the rate of twist ϑ_x are

$$\epsilon_x^\circ = \frac{\partial u}{\partial x}, \quad \kappa_z = -\frac{\partial^2 v}{\partial x^2}, \quad \kappa_y = -\frac{\partial^2 w}{\partial x^2}, \quad \vartheta_x = \frac{\partial \psi}{\partial x}. \tag{3.3.1}$$

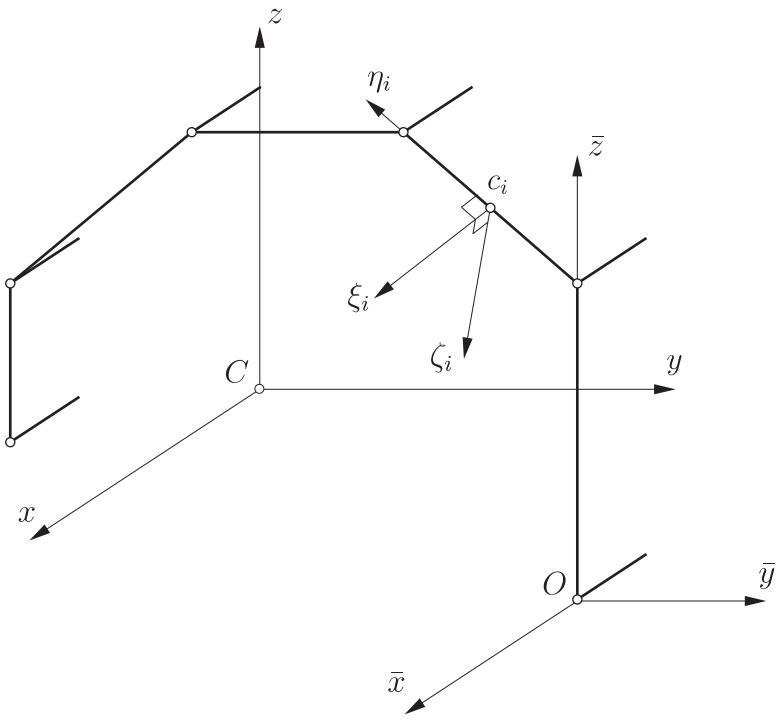


Figure 3.5: Coordinate systems employed in the analysis

In the $\bar{x}$–$\bar{y}$–$\bar{z}$ coordinate system these relationships become

$$\epsilon^{\circ}_{\bar{x}} = \frac{\partial \bar{u}}{\partial \bar{x}}, \quad \kappa_{\bar{z}} = -\frac{\partial^2 \bar{v}}{\partial \bar{x}^2}, \quad \kappa_{\bar{y}} = -\frac{\partial^2 \bar{w}}{\partial \bar{x}^2}, \quad \vartheta_{\bar{x}} = \frac{\partial \bar{\psi}}{\partial \bar{x}}, \tag{3.3.2}$$

where $\epsilon^{\circ}_{\bar{x}}$, $\kappa_{\bar{y}}$, $\kappa_{\bar{z}}$, and $\vartheta_{\bar{x}}$ are the axial strain, the curvatures, and the rate of twist of the longitudinal axis passing through the origin of the $\bar{x}$–$\bar{y}$–$\bar{z}$ coordinate system; $\bar{u}$, $\bar{v}$, $\bar{w}$, and $\bar{\psi}$ are the displacements of the $\bar{x}$ axis.

3.4 Analysis

The analysis of thin-walled open section beams includes four stages:

1. The strains in each wall segment are expressed in terms of the axial strain, curvatures and twist of the beams axis.

2. The forces in each wall segment are determined from the strains in the wall

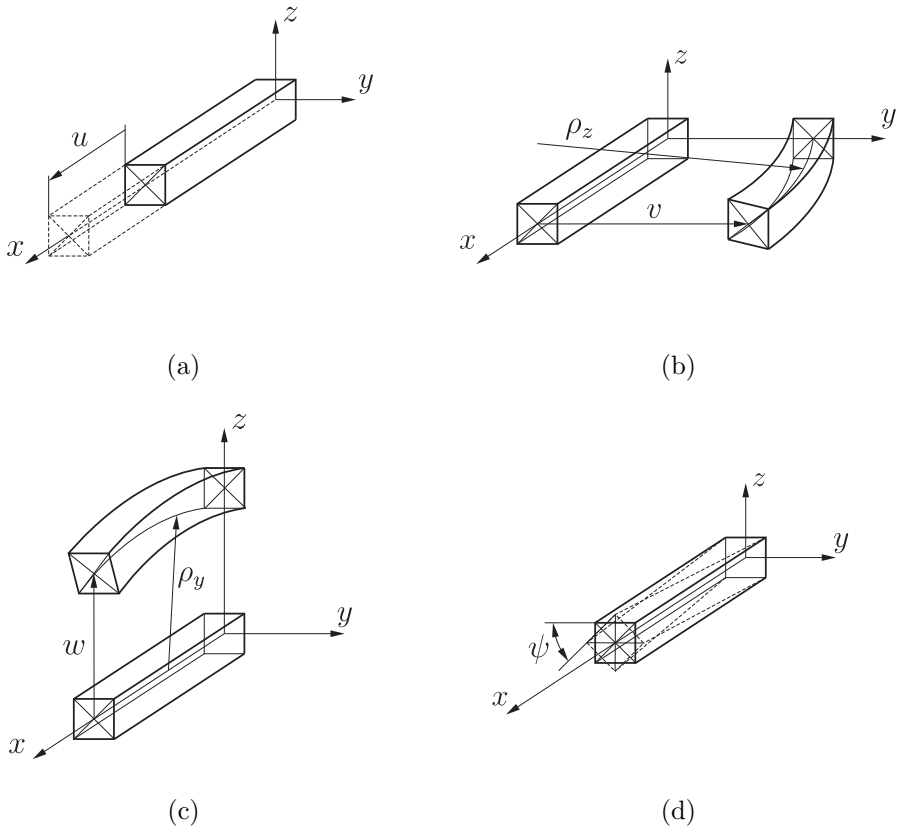


Figure 3.6: Displacements of a beam

segment.

3. The resultant axial force, moments, and torque acting at the axis of the beam are determined from the wall segment forces.

4. The stiffness matrix is established by relating the resultant axial force, moments, and torque to the axial strain, curvatures and twist of the axis of the beam.

3.4.1 Strains in the Wall Segments

According to Bernoulli-Navier hypothesis, at a point on the reference plane of each wall segment the axial strain is calculated by the plane strain condition

$$\epsilon^{\circ}_{\xi i} = \epsilon^{\circ}_{\bar{x}} + \bar{z}\kappa_{\bar{y}} + \bar{y}\kappa_{\bar{z}}, \tag{3.4.1}$$

where $\bar{y}$ and $\bar{z}$ are the coordinates of an arbitrary point on the ith segment's reference surface, and $\epsilon^{\circ}_{\xi i}$ is the axial strain at this point.

The cross-sections of beams with arbitrary layup do not remain plane and Bernoulli-Navier hypothesis (3.4.1) is inapplicable. However, in a long beam, the strains may be considered to be constant in the axial direction, and the plane-strain condition, where stresses and strains vary only in planes perpendicular to the x axis, may be applied in the analysis.

The strains of the axis of the ith wall segment can be expressed as

$$\left\{ \begin{array}{c} \epsilon^{c}_{\xi} \\ \kappa^{c}_{\eta} \\ \kappa^{c}_{\zeta} \\ \vartheta^{c}_{\xi} \end{array} \right\}_i = \underbrace{\left[\begin{array}{cccc} 1 & \bar{z}_i & \bar{y}_i & 0 \\ 0 & \cos(\varphi_i) & -\sin(\varphi_i) & 0 \\ 0 & \sin(\varphi_i) & \cos(\varphi_i) & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]}_{[R_i]} \left\{ \begin{array}{c} \epsilon^{\circ}_{\bar{x}} \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{array} \right\}, \tag{3.4.2}$$

where $\bar{y}_i$ and $\bar{z}_i$ are the coordinates of the ξ_i–η_i–ζ_i coordinate system's origin, which is at the midpoint of the reference plane, as shown in Figure 3.7 and φ_i is the angle between the η_i and $\bar{y}$ coordinate axes as shown in Figure 3.8. The superscript c refers to the segment's longitudinal axis which passes through the midpoint of the reference plane, where $\xi = 0$ and $\zeta = 0$. $\kappa^{c}_{\eta i}$ and $\kappa^{c}_{\zeta i}$ are the curvatures of the ith wall segment's axis in the ξ–ζ and ξ–η planes, respectively (Figures 3.9). The last equation in Eq. (3.4.2), i.e., $\vartheta^{c}_{\xi i} = \vartheta_{\bar{x}}$, is written by observing that the twist of every point in the wall segment is equal to the twist of the beam.

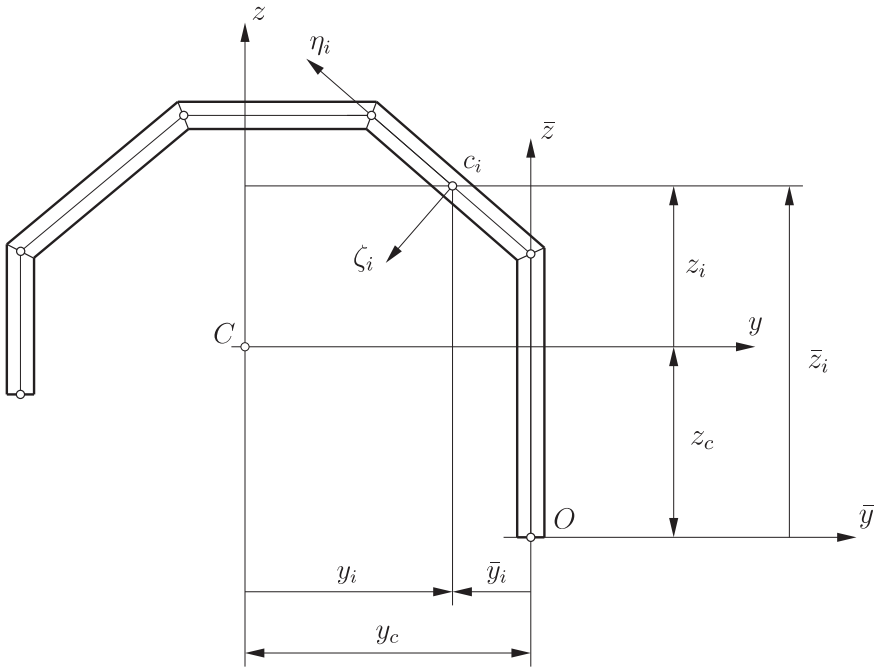


Figure 3.7: Coordinates (y_c,z_c) of the centroid C

The axial strain in the ith segment varies linearly with η, therefore we can write

$$\epsilon^{\circ}_{\xi i} = \epsilon^{c}_{\xi i} + \eta \kappa^{c}_{\zeta i}. \tag{3.4.3}$$

The curvature $\kappa_{\xi i}$ is uniform in each flat segment and is defined by

$$\kappa_{\xi i} = \kappa^{c}_{\eta i} = \kappa_{\bar{y}} \cos(\varphi_i) - \kappa_{\bar{z}} \sin(\varphi_i). \tag{3.4.4}$$

The rate of twist is defined as

$$\vartheta^{c}_{\xi i} = -\frac{1}{2} \kappa_{\xi \eta i}. \tag{3.4.5}$$

Therefore, we have

$$\kappa_{\xi \eta i} = -2 \vartheta^{c}_{\xi i}. \tag{3.4.6}$$

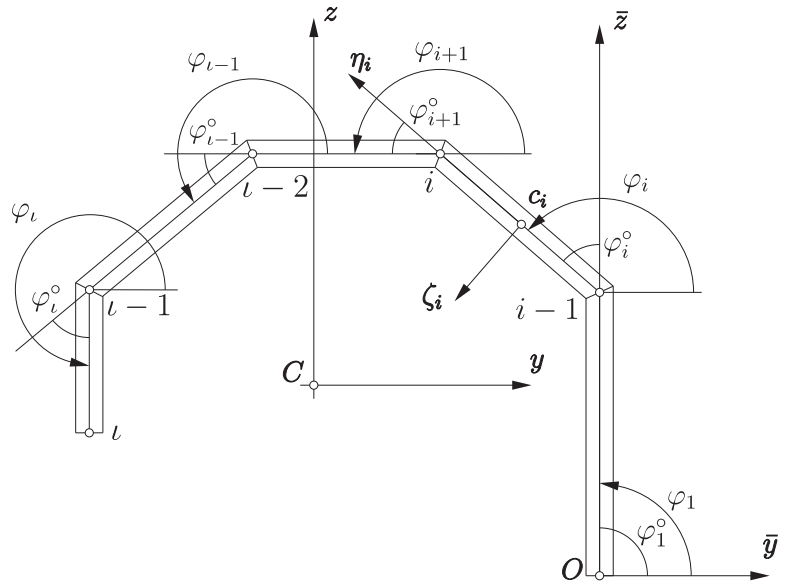


Figure 3.8: The cross-section of the segmented open-section beam

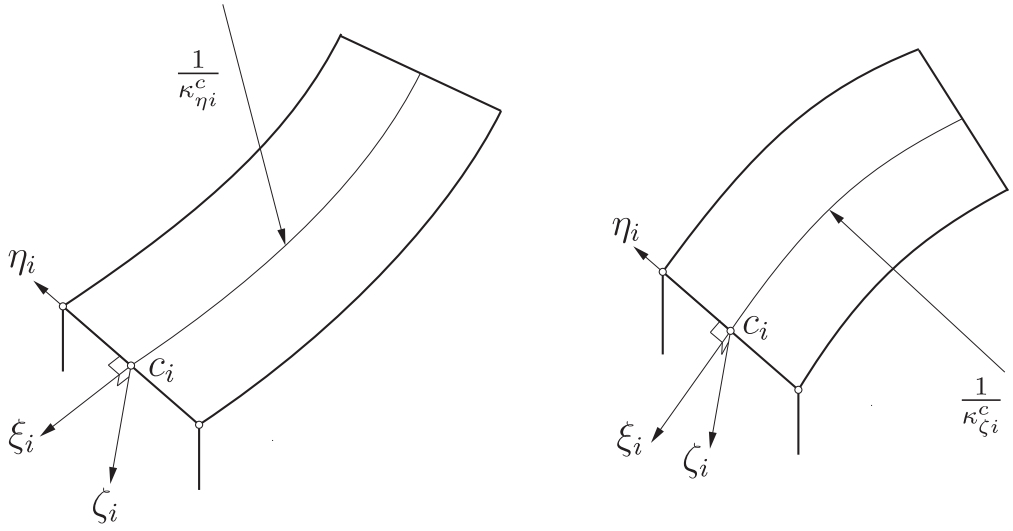
Eqs. (3.4.3),(3.4.4), and (3.4.6) are written in matrix form as

$$\left\{\begin{array}{c}\epsilon^\circ_\xi\\ \kappa_\xi\\ \kappa_{\xi\eta}\end{array}\right\}_i = \underbrace{\left[\begin{array}{cccc}1&0&\eta&0\\0&1&0&0\\0&0&0&-2\end{array}\right]}_{[R_\eta]} \left\{\begin{array}{c}\epsilon^c_\xi\\ \kappa^c_\eta\\ \kappa^c_\zeta\\ \vartheta^c_\xi\end{array}\right\}_i . \tag{3.4.7}$$

3.4.2 Forces and Moments in the Wall Segments

The strain-force relationships in ith segment is given by

$$\left\{\begin{array}{c}\epsilon^\circ_\xi\\ \epsilon^\circ_\eta\\ \gamma^\circ_{\xi\eta}\\ \kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{array}\right\}_i = \left[\begin{array}{cccccc}\alpha_{11}&\alpha_{12}&\alpha_{16}&\beta_{11}&\beta_{12}&\beta_{16}\\ \alpha_{21}&\alpha_{22}&\alpha_{26}&\beta_{21}&\beta_{22}&\beta_{26}\\ \alpha_{61}&\alpha_{62}&\alpha_{66}&\beta_{61}&\beta_{62}&\beta_{66}\\ \beta_{11}&\beta_{21}&\beta_{16}&\delta_{11}&\delta_{12}&\delta_{16}\\ \beta_{12}&\beta_{22}&\beta_{26}&\delta_{21}&\delta_{22}&\delta_{16}\\ \beta_{61}&\beta_{62}&\beta_{66}&\delta_{61}&\delta_{62}&\delta_{66}\end{array}\right]_i \left\{\begin{array}{c}N_\xi\\ N_\eta\\ N_{\xi\eta}\\ M_\xi\\ M_\eta\\ M_{\xi\eta}\end{array}\right\}_i , \tag{3.4.8}$$



(a) (b)

Figure 3.9: The curvatures of the ith wall segment's axis

where $[\alpha]$, $[\beta]$, and $[\delta]$ matrices are defined as

$$\begin{bmatrix} [\alpha] & [\beta] \\ [\beta]^T & [\delta] \end{bmatrix}_i = \begin{bmatrix} [A] & [B] \\ [B] & [D] \end{bmatrix}_i^{-1}. \tag{3.4.9}$$

N_η, $N_{\xi\eta}$, and M_η are zero along the free longitudinal edges and, consequently, are approximately zero everywhere, therefore

$$N_\eta = 0, \quad N_{\xi\eta} = 0, \quad M_\eta = 0. \tag{3.4.10}$$

With these approximations the strain-force relationships for the ith wall segment become

$$\begin{Bmatrix} \epsilon^\circ_\xi \\ \kappa_\xi \\ \kappa_{\xi\eta} \end{Bmatrix}_i = \underbrace{\begin{bmatrix} \alpha_{11} & \beta_{11} & \beta_{16} \\ \beta_{11} & \delta_{11} & \delta_{16} \\ \beta_{16} & \delta_{16} & \delta_{66} \end{bmatrix}_i}_{[\mu_i]} \begin{Bmatrix} N_\xi \\ M_\xi \\ M_{\xi\eta} \end{Bmatrix}_i, \tag{3.4.11}$$

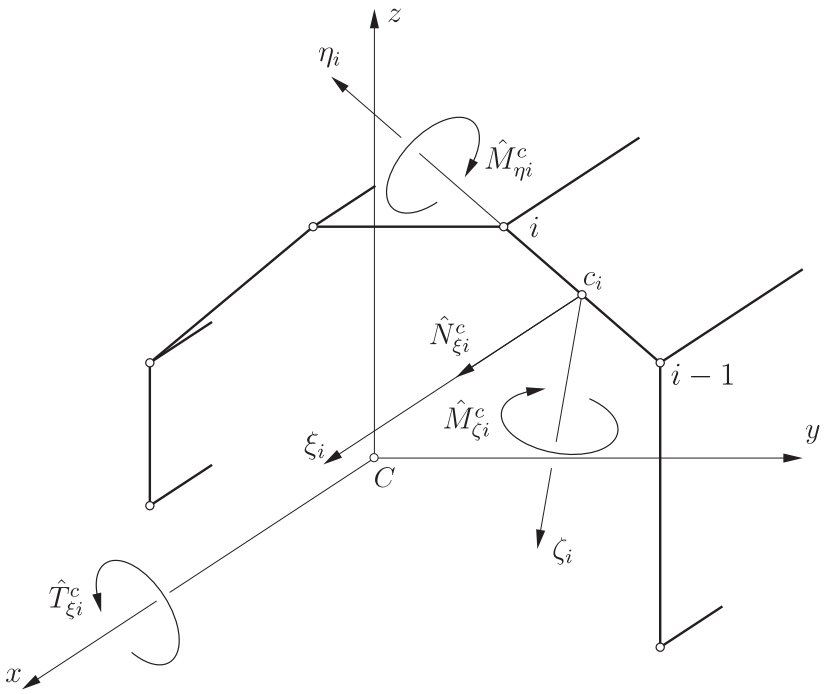


Figure 3.10: The force resultants in the ith wall segment

The stress resultants in the ith segment's coordinate system, shown in Figure 3.10, are

$$\hat{N}^c_{\xi i} = \int_0^{b_i} N_{\xi i}\, d\eta = b_i N_{\xi i} \quad \text{at } \eta = 0, \tag{3.4.12}$$

$$\hat{M}^c_{\eta i} = \int_0^{b_i} M_{\xi i}\, d\eta = b_i M_{\xi i} \quad \text{at } \eta = 0, \tag{3.4.13}$$

$$\hat{M}^c_{\zeta i} = \int_0^{b_i} N_{\xi i}\eta\, d\eta, \tag{3.4.14}$$

where b_i is the width of the wall segment, as shown in Figure 3.11.

The torque is

$$\hat{T}^c_{\xi i} = -2\int_0^{b_i} M_{\xi\eta i}\, d\eta = -2b_i M_{\xi\eta i} \quad \text{at } \eta = 0. \tag{3.4.15}$$

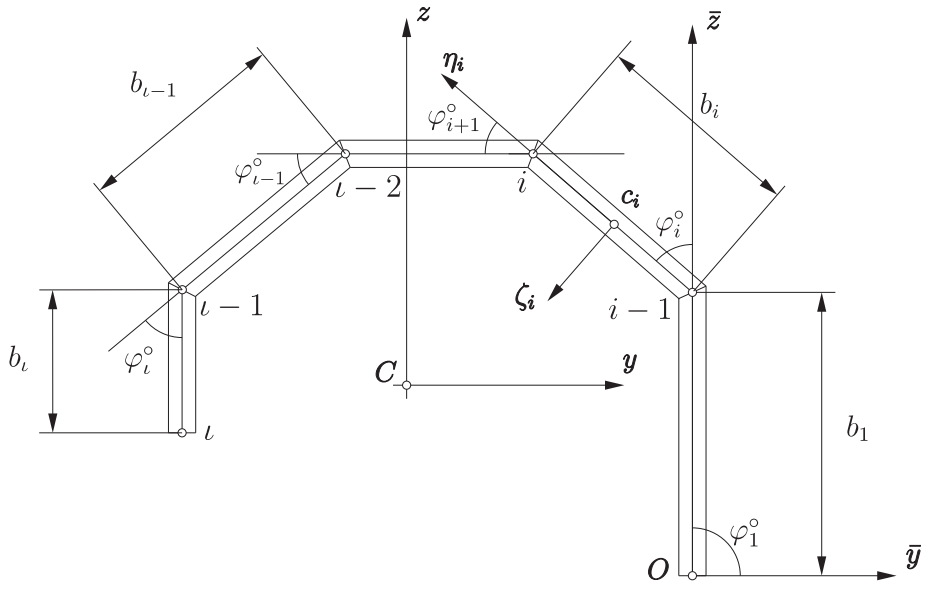


Figure 3.11: The cross-section of the segmented open-section beam

Equations (3.4.12), (3.4.13), (3.4.15), (3.4.11), and (3.4.7) result in

$$\left\{\begin{array}{c}\epsilon^c_\xi\\ \kappa^c_\eta\\ \vartheta^c_\xi\end{array}\right\}_i = \frac{1}{b_i}\left[\begin{array}{ccc}\alpha_{11} & \beta_{11} & -\frac{1}{2}\beta_{16}\\ \beta_{11} & \delta_{11} & -\frac{1}{2}\delta_{16}\\ -\frac{1}{2}\beta_{16} & -\frac{1}{2}\delta_{16} & \frac{1}{4}\delta_{66}\end{array}\right]_i \left\{\begin{array}{c}\hat{N}^c_\xi\\ \hat{M}^c_\eta\\ \hat{T}^c_\xi\end{array}\right\}_i , \tag{3.4.16}$$

To evaluate the integral in Eq. (3.4.14) we assume that the flat segment remains flat, i.e., $\kappa_{\xi i} = 0$ and $\kappa_{\xi\eta i} = 0$, and consider only the curvature $\kappa^c_{\zeta i}$ of the segment. With this approximation the inverse of Eq. (3.4.11) yields

$$\left\{\begin{array}{c}N_\xi\\ M_\xi\\ M_{\xi\eta}\end{array}\right\}_i = \left[\begin{array}{ccc}a_{11} & a_{12} & a_{13}\\ a_{12} & a_{22} & a_{23}\\ a_{13} & a_{23} & a_{33}\end{array}\right]_i \left\{\begin{array}{c}\epsilon^\circ_\xi\\ 0\\ 0\end{array}\right\}_i , \tag{3.4.17}$$

The matrix $[a]_i$ is defined as

$$[a]_i = [\alpha]_i^{-1}. \tag{3.4.18}$$

With this definition $N_{\xi i}$ is

$$N_{\xi i} = a_{11i}\epsilon^{\circ}_{\xi i}. \tag{3.4.19}$$

Substituting this expression and Eq. (3.4.3) into Eq. (3.4.14) and integrating, we obtain

$$\hat{M}^c_{\zeta i} = \frac{a_{11i}b_i^3}{12}\kappa^c_{\zeta i} \tag{3.4.20}$$

Eqs. (3.4.16) and (3.4.20) give the following strain-force relationship in the ξ_i–η_i–ζ_i coordinate system

$$\begin{Bmatrix} \epsilon^c_\xi \\ \kappa^c_\eta \\ \kappa^c_\zeta \\ \vartheta^c_\xi \end{Bmatrix}_i = \underbrace{\frac{1}{b_i}\begin{bmatrix} \alpha_{11} & \beta_{11} & 0 & -\frac{1}{2}\beta_{16} \\ \beta_{11} & \delta_{11} & 0 & -\frac{1}{2}\delta_{16} \\ 0 & 0 & \frac{12}{a_{11i}b_i^2} & 0 \\ -\frac{1}{2}\beta_{16} & -\frac{1}{2}\delta_{16} & 0 & \frac{1}{4}\beta_{66} \end{bmatrix}_i}_{[\Omega_i]} \begin{Bmatrix} \hat{N}^c_\xi \\ \hat{M}^c_\eta \\ \hat{M}^c_\zeta \\ \hat{T}^c_\xi \end{Bmatrix}_i, \tag{3.4.21}$$

3.4.3 Forces in the Beam

In the bar coordinate system the forces in the beam are the sum of the forces in the wall segments

$$\begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{Bmatrix} = \sum_{i=1}^{\iota} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ \bar{z}_i & \cos(\varphi_i) & \sin(\varphi_i) & 0 \\ \bar{y}_i & -\sin(\varphi_i) & \cos(\varphi_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_i}_{[R_i]^T} \begin{Bmatrix} \hat{N}^c_\xi \\ \hat{M}^c_\eta \\ \hat{M}^c_\zeta \\ \hat{T}^c_\xi \end{Bmatrix}_i, \tag{3.4.22}$$

3.4.4 Stiffness Matrix

Eqs. (3.4.22),(3.4.21), and (3.4.2) yield

$$\begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{Bmatrix} = \sum_{i=1}^{\iota} [R_i]^T \begin{Bmatrix} \hat{N}^c_{\xi} \\ \hat{M}^c_{\eta} \\ \hat{M}^c_{\zeta} \\ \hat{T}^c_{\xi} \end{Bmatrix}_i = \sum_{i=1}^{\iota} [R_i]^T [\Omega_i]^{-1} \begin{Bmatrix} \epsilon^c_{\xi} \\ \kappa^c_{\eta} \\ \kappa^c_{\zeta} \\ \vartheta^c_{\xi} \end{Bmatrix}_i , \tag{3.4.23}$$

and then

$$\begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{Bmatrix} = \underbrace{\sum_{i=1}^{\iota} \left([R_i]^T [\Omega_i]^{-1} [R_i] \right)}_{[\bar{P}]} \begin{Bmatrix} \epsilon^{\circ}_{\bar{x}} \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{Bmatrix}, \tag{3.4.24}$$

where $[\bar{P}]$ is the stiffness matrix in the bar coordinate system.

The displacements must be determined by solving the governing equations (3.3.1) and the constitutive equations (3.2.1) for the beam under consideration. These equations are given in the x–y–z coordinate system. Therefore, the stiffness matrices must also be expressed in this coordinate system. From uniform strains the end displacements and the rotation of the cross-section at the end are

$$\begin{aligned} u &= \epsilon^{\circ}_x L, \\ v &= -\kappa_y \frac{L^2}{2}, \\ w &= -\kappa_z \frac{L^2}{2}, \\ \psi &= \vartheta_x L, \end{aligned} \tag{3.4.25}$$

where L is the length of the beam.

3.4.5 Centroid

The centroid C is located such that the beam's axis remains straight when a pure axial force $\hat{N}_x$ is applied at the centroid. While this axis remains straight, the beam may twist about the axis of twist, which does not necessarily coincide with the axis passing through the centroid. The coordinates of the centroid in the bar coordinate system are denoted by y_c and z_c as shown in Figure 3.7.

The force and moment resultants at the origin of the bar coordinate system $\hat{N}_{\bar{x}}$, $\hat{M}_{\bar{y}}$, $\hat{M}_{\bar{z}}$ are related to the force applied at the centroid by the following expressions

$$\hat{N}_{\bar{x}} = \hat{N}_x, \quad \hat{M}_{\bar{y}} = z_c \hat{N}_x, \quad \hat{M}_{\bar{z}} = y_c \hat{N}_x. \tag{3.4.26}$$

The strain-force relationships are

$$\begin{Bmatrix} \epsilon^{\circ}_{\bar{x}} \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{Bmatrix} = \begin{bmatrix} \bar{W}_{11} & \bar{W}_{12} & \bar{W}_{13} & \bar{W}_{14} \\ \bar{W}_{12} & \bar{W}_{22} & \bar{W}_{23} & \bar{W}_{24} \\ \bar{W}_{13} & \bar{W}_{23} & \bar{W}_{33} & \bar{W}_{34} \\ \bar{W}_{14} & \bar{W}_{24} & \bar{W}_{34} & \bar{W}_{44} \end{bmatrix} \begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{Bmatrix}, \tag{3.4.27}$$

where $[\bar{W}]$ is the compliance matrix in the bar coordinate system defined as

$$[\bar{W}] = [\bar{P}]^{-1}. \tag{3.4.28}$$

Eqs. (3.4.27) and (3.4.26) yield the curvatures

$$\begin{Bmatrix} \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \end{Bmatrix} = \begin{bmatrix} \bar{W}_{12} & \bar{W}_{22} & \bar{W}_{23} \\ \bar{W}_{13} & \bar{W}_{23} & \bar{W}_{33} \end{bmatrix} \begin{Bmatrix} 1 \\ z_c \\ y_c \end{Bmatrix}, \tag{3.4.29}$$

Since $\hat{N}$ is applied at the centroid, the curvatures of the beam are zero, i.e.,

$$\kappa_{\bar{y}} = 0, \quad \kappa_{\bar{z}} = 0, \tag{3.4.30}$$

Eqs. (3.4.29) and (3.4.30) yield the location of the centroid with respect to the origin of the bar coordinate system

$$\begin{Bmatrix} z_c \\ y_c \end{Bmatrix} = - \begin{bmatrix} \bar{W}_{22} & \bar{W}_{23} \\ \bar{W}_{23} & \bar{W}_{33} \end{bmatrix}^{-1} \begin{Bmatrix} \bar{W}_{12} \\ \bar{W}_{13} \end{Bmatrix}. \tag{3.4.31}$$

3.4.6 Stiffness and Compliance Matrices in the Centroid Coordinate System

The forcers and moments $\hat{N}_{\bar{x}}$, $\hat{M}_{\bar{y}}$, and $\hat{M}_{\bar{z}}$ in the bar coordinate system are related to the axial force $\hat{N}_x$ in the x–y–z coordinate system by Eq. (3.4.26). The torques in the two coordinate systems are identical, i.e.,

$$\hat{T}_{\bar{x}} = \hat{T}_x. \tag{3.4.32}$$

Eqs. (3.4.26) and (3.4.32) in matrix form are

$$\begin{Bmatrix} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{x}} \\ \hat{T}_{\bar{x}} \end{Bmatrix} = [R_c] \begin{Bmatrix} \hat{N}_x \\ \hat{M}_y \\ \hat{M}_z \\ \hat{T}_x \end{Bmatrix}, \tag{3.4.33}$$

where $[R_c]$ is defined as

$$[R_c] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ z_c & 1 & 0 & 0 \\ y_c & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.4.34}$$

The axial strain of the axis passing through the centroid is related to the strain and curvatures of the axis passing through the origin of the bar coordinate system by the following equation

$$\epsilon_x^\circ = \epsilon_{\bar{x}}^\circ + \kappa_{\bar{y}} z_c + \kappa_{\bar{z}} y_c. \tag{3.4.35}$$

The curvatures and twists per unit length of the x and $\bar{x}$ axes are identical. Therefore, we can write

$$\begin{Bmatrix} \epsilon_x^\circ \\ \kappa_y \\ \kappa_z \\ \vartheta_x \end{Bmatrix} = [R_c]^T \begin{Bmatrix} \epsilon_{\bar{x}}^\circ \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{Bmatrix}. \tag{3.4.36}$$

Substituting Eqs. (3.4.33) and (3.4.36) into Eq. (3.4.27), we obtain the strain-force relationship with respect to the x–y–z coordinate system

$$\begin{Bmatrix} \epsilon_x^\circ \\ \kappa_y \\ \kappa_z \\ \vartheta_x \end{Bmatrix} = \underbrace{[R_c]^T [\bar{W}][R_c]}_{[W]} \begin{Bmatrix} \hat{N}_x \\ \hat{M}_y \\ \hat{M}_z \\ \hat{T}_x \end{Bmatrix}, \tag{3.4.37}$$

where $[W]$ is the compliance matrix in the x–y–z coordinate system.

3.4.7 Stresses and Strains in Open-Section Beam

We consider a thin-walled open section beam subjected to axial force $\hat{N}_{\bar{x}}$, bending moments $\hat{M}_{\bar{y}}$ and $\hat{M}_{\bar{z}}$, and torque $\hat{T}_{\bar{x}}$ acting at the origin of the arbitrarily chosen $\bar{x}$–$\bar{y}$–$\bar{z}$ coordinate system. The axial strain $\epsilon^0_{\bar{x}}$, curvatures $\kappa_{\bar{y}}$ and $\kappa_{\bar{z}}$, and twist of the longitudinal axis $\vartheta_{\bar{x}}$ are defined by Eqs. (3.4.27).

For the ith segment Eqs. (3.4.2) and (3.4.7) give

$$\left\{ \begin{array}{c} \epsilon^\circ_\xi \\ \kappa_\xi \\ \kappa_{\xi\eta} \end{array} \right\}_i = [R_\eta][R_i] \left\{ \begin{array}{c} \epsilon^\circ_{\bar{x}} \\ \kappa_{\bar{y}} \\ \kappa_{\bar{z}} \\ \vartheta_{\bar{x}} \end{array} \right\}, \tag{3.4.38}$$

where ϵ°_ξ is the axial strain, κ_ξ and $\kappa_{\xi\eta}$ are the curvatures of the axis through the midpoint of the wall segment's reference plane.

Eqs. (3.4.27), (3.4.38), and (3.4.11) give

$$\left\{ \begin{array}{c} N_\xi \\ M_\xi \\ M_{\xi\eta} \end{array} \right\}_i = [\mu_i]^{-1}[R_\eta][R_i][\bar{W}] \left\{ \begin{array}{c} \hat{N}_{\bar{x}} \\ \hat{M}_{\bar{y}} \\ \hat{M}_{\bar{z}} \\ \hat{T}_{\bar{x}} \end{array} \right\}, \tag{3.4.39}$$

We recall that in open section beam $N_{\eta i}$, $N_{\xi\eta i}$, and $M_{\eta i}$ are zero. From the three forces $N_{\xi i}$, $N_{\eta i}$, $N_{\xi\eta i}$ and three moments $M_{\xi i}$, $M_{\eta i}$, $M_{\xi\eta i}$ for unit length we can calculate the stresses and strains using the classical laminate plate theory.

3.4.8 Classical Laminate Theory

In this section we determine stiffness matrices for thin flat laminate undergoing small deformation. The analysis is based on the classical laminate theory and is

formulated based on the following approximations:

1. the strains vary linearly across the laminate,
2. the out-of-plane shear deformations are negligible,
3. the out-of-plane normal stress σ_ζ and the shear stresses $\tau_{\xi\zeta}$ and $\tau_{\eta\zeta}$ are small compared with the in-plane stresses σ_ξ, σ_η, and $\tau_{\xi\eta}$.

These approximation imply that the stress-strain relationships under plane-stress conditions may be applied.

Often, the reference plane is taken to be the midplane of the laminate. Unless the laminate is symmetrical with respect to the reference plane, the reference plane is not a neutral plane, and the strains in the reference plane are not zero under pure bending. These strains in the reference plane are

$$\epsilon_\xi^\circ = \frac{\partial \tilde{u}^\circ}{\partial \xi}, \quad \epsilon_\eta^\circ = \frac{\partial \tilde{v}^\circ}{\partial \eta}, \quad \gamma_{\xi\eta}^\circ = \frac{\partial \tilde{u}^\circ}{\partial \eta} + \frac{\partial \tilde{v}^\circ}{\partial \xi}, \tag{3.4.40}$$

where $\tilde{u}$ and $\tilde{v}$ are the ξ and η components of the displacement and the superscript $(^\circ)$ refers to the reference plane.

We adopt the Kirchhoff hypothesis, where the normals to the reference surface remain normal and straight. Accordingly, for small deflections the angles of rotation of the normal of the reference plane $\chi_{\xi\zeta}$ and $\chi_{\eta\zeta}$ are

$$\chi_{\xi\zeta} = \frac{\partial \tilde{w}^\circ}{\partial \xi}, \quad \chi_{\eta\zeta} = \frac{\partial \tilde{w}^\circ}{\partial \eta}, \tag{3.4.41}$$

where $\tilde{w}^\circ$ is the out-of-plane displacement of the reference plane. The total displacements in the ξ and η directions are

$$\tilde{u} = \tilde{u}^\circ - \zeta \tilde{\chi}_{\xi\zeta} = \tilde{u}^\circ - \zeta \frac{\partial \tilde{w}^\circ}{\partial \xi} \tag{3.4.42}$$

and

$$\tilde{v} = \tilde{v}^\circ - \zeta\tilde{\chi}_{\eta\zeta} = \tilde{v}^\circ - \zeta\frac{\partial\tilde{w}^\circ}{\partial\eta}, \tag{3.4.43}$$

respectively. By definition, the strains are

$$\epsilon_\xi = \frac{\partial\tilde{u}}{\partial\xi}, \quad \epsilon_\eta = \frac{\partial\tilde{v}}{\partial\eta}, \quad \gamma_{\xi\eta} = \frac{\partial\tilde{u}}{\partial\eta} + \frac{\partial\tilde{v}}{\partial\xi}. \tag{3.4.44}$$

Substituting Eqs. (3.3.1) and (3.3.1) into Eqs. (3.3.1), we obtain

$$\epsilon_\xi = \frac{\partial\tilde{u}^\circ}{\partial\xi} - \zeta\frac{\partial^2\tilde{w}^\circ}{\partial\xi^2}, \tag{3.4.45}$$

$$\epsilon_\eta = \frac{\partial\tilde{v}^\circ}{\partial\eta} - \zeta\frac{\partial^2\tilde{w}^\circ}{\partial\eta^2}, \tag{3.4.46}$$

$$\gamma_{\xi\eta} = \frac{\partial\tilde{u}^\circ}{\partial\eta} + \frac{\partial\tilde{v}^\circ}{\partial\xi} - \zeta\frac{2\partial^2\tilde{w}^\circ}{\partial\xi\partial\eta}, \tag{3.4.47}$$

or, in matrix form,

$$\left\{\begin{array}{c}\epsilon_\xi\\ \epsilon_\eta\\ \gamma_{\xi\eta}\end{array}\right\}_k = \left\{\begin{array}{c}\epsilon_\xi^\circ\\ \epsilon_\eta^\circ\\ \gamma_{\xi\eta}^\circ\end{array}\right\} + \zeta\left\{\begin{array}{c}\kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{array}\right\}, \tag{3.4.48}$$

where ϵ_ξ°, ϵ_η°, and $\gamma_{\xi\eta}^\circ$ are the strains in the reference plane, κ_ξ, κ_η, and $\kappa_{\xi\eta}$ are the curvatures of the reference plane of the plate defined as

$$\kappa_\xi = -\frac{\partial^2\tilde{w}^\circ}{\partial\xi^2}, \quad \kappa_\eta = -\frac{\partial^2\tilde{w}^\circ}{\partial\eta^2}, \quad \kappa_{\xi\eta} = -\frac{2\partial^2\tilde{w}^\circ}{\partial\xi\partial\eta}. \tag{3.4.49}$$

For plane-stress condition the stress-strain relationships for each ply are

$$\left\{ \begin{array}{c} \sigma_\xi \\ \sigma_\eta \\ \tau_{\xi\eta} \end{array} \right\}_k = \underbrace{\left[\begin{array}{ccc} \bar{Q}_{11} & \bar{Q}_{12} & \bar{Q}_{16} \\ \bar{Q}_{12} & \bar{Q}_{22} & \bar{Q}_{26} \\ \bar{Q}_{16} & \bar{Q}_{26} & \bar{Q}_{66} \end{array} \right]_k}_{[\bar{Q}]_k} \left\{ \begin{array}{c} \epsilon_\xi \\ \epsilon_\eta \\ \gamma_{\xi\eta} \end{array} \right\}, \tag{3.4.50}$$

where $[\bar{Q}]_k$ is the transformed reduced stiffness matrix of the ply in the ξ–η coordinate system. The transformed reduced stiffness matrix $[\bar{Q}]$ is related to the reduced stiffness matrix $[Q]$ by the following relationships

$$\begin{aligned} \bar{Q}_{11} &= c^4 Q_{11} + s^4 Q_{22} + 2c^2 s^2 (Q_{12} + 2Q_{66}), \\ \bar{Q}_{12} &= c^2 s^2 (Q_{11} + Q_{22} - 4Q_{66}) + (c^4 + s^4) Q_{12}, \\ \bar{Q}_{22} &= s^4 Q_{11} + c^4 Q_{22} + 2c^2 s^2 (Q_{12} + 2Q_{66}), \\ \bar{Q}_{16} &= cs(c^2 Q_{11} - s^2 Q_{22} - (c^2 - s^2)(Q_{12} + 2Q_{66})), \\ \bar{Q}_{26} &= cs(s^2 Q_{11} - c^2 Q_{22} + (c^2 - s^2)(Q_{12} + 2Q_{66})), \\ \bar{Q}_{66} &= c^2 s^2 (Q_{11} + Q_{22} - 2Q_{12}) + (c^2 - s^2)^2 Q_{66}, \end{aligned} \tag{3.4.51}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$, and θ is the ply orientation angle.

The in-plane forces and moments acting on a small element are

$$\begin{aligned} \hat{N}_\xi &= \int_{-h/2}^{h/2} \tilde{\sigma}_\xi \, d\zeta, \quad \hat{N}_\eta = \int_{-h/2}^{h/2} \sigma_\eta \, d\zeta, \quad \hat{N}_{\xi\eta} = \int_{-h/2}^{h/2} \tilde{\tau}_{\xi\eta} \, d\zeta, \\ \hat{M}_\xi &= \int_{-h/2}^{h/2} \zeta \sigma_\xi \, d\zeta, \quad \hat{M}_\eta = \int_{-h/2}^{h/2} \zeta \sigma_\eta \, d\zeta, \quad \hat{M}_{\xi\eta} = \int_{-h/2}^{h/2} \zeta \tau_{\xi\eta} \, d\zeta, \end{aligned} \tag{3.4.52}$$

where $\hat{N}$ and $\hat{M}$ are the in-plane forces and moments per unit length, h is the total laminate thickness.

By substituting Eqs. (3.4.50) into Eq. (3.4.52), we obtain

$$\begin{aligned}\begin{Bmatrix}\hat{N}_\xi\\ \hat{N}_\eta\\ \hat{N}_{\xi\eta}\end{Bmatrix} &= \int_{-h/2}^{h/2}\left[[\bar{Q}]\begin{Bmatrix}\epsilon_1^\circ\\ \epsilon_2^\circ\\ \epsilon_{12}^\circ\end{Bmatrix} + [\bar{Q}]\zeta\begin{Bmatrix}\kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{Bmatrix}\right] d\zeta\\ &= \int_{-h/2}^{h/2}[\bar{Q}]\,d\zeta\begin{Bmatrix}\epsilon_\xi^\circ\\ \epsilon_\eta^\circ\\ \epsilon_{\xi\eta}^\circ\end{Bmatrix} + \int_{-h/2}^{h/2}[\bar{Q}]\zeta\,d\zeta\begin{Bmatrix}\kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{Bmatrix},\end{aligned} \tag{3.4.53}$$

$$\begin{aligned}\begin{Bmatrix}\hat{M}_\xi\\ \hat{M}_\eta\\ \hat{M}_{\xi\eta}\end{Bmatrix} &= \int_{-h/2}^{h/2}\eta\left[[\bar{Q}]\begin{Bmatrix}\epsilon_\xi^\circ\\ \epsilon_\eta^\circ\\ \epsilon_{\xi\eta}^\circ\end{Bmatrix} + [\bar{Q}]\eta\begin{Bmatrix}\kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{Bmatrix}\right] d\zeta\\ &= \int_{-h/2}^{h/2}[\bar{Q}]\zeta\,d\zeta\begin{Bmatrix}\epsilon_\xi^\circ\\ \epsilon_\eta^\circ\\ \epsilon_{\xi\eta}^\circ\end{Bmatrix} + \int_{-h/2}^{h/2}[\bar{Q}]\zeta^2\,d\zeta\begin{Bmatrix}\kappa_\xi\\ \kappa_\eta\\ \kappa_{\xi\eta}\end{Bmatrix}.\end{aligned} \tag{3.4.54}$$

The stiffness matrices of the laminate are defined as

$$[A] = \int_{-h/2}^{h/2}[\bar{Q}]\,d\zeta, \quad [B] = \int_{-h/2}^{h/2}[\bar{Q}]\zeta\,d\zeta, \quad [D] = \int_{-h/2}^{h/2}[\bar{Q}]\zeta^2\,d\zeta. \tag{3.4.55}$$

Since $[\bar{Q}]$ is constant across each ply, the integrals may be replaced by summations as follows

$$\begin{aligned} A_{ij} &= \sum_{k=1}^{n} (\bar{Q}_{ij})_k (\zeta_k - \zeta_{k-1}), \\ B_{ij} &= \frac{1}{2} \sum_{k=1}^{n} (\bar{Q}_{ij})_k (\zeta_k^2 - \zeta_{k-1}^2), \\ D_{ij} &= \frac{1}{3} \sum_{k=1}^{n} (\bar{Q}_{ij})_k (\zeta_k^3 - \zeta_{k-1}^3), \quad (i,j = 1,2,6), \end{aligned} \tag{3.4.56}$$

where n is the total number of plies in the laminate, ζ_k and ζ_{k-1} are the distances from the reference plane to the two surfaces of the kth ply, and $(\bar{Q}_{ij})_k$ are the elements of the stiffness matrix of the kth ply. The expressions for the in-plane forces and moments become

$$\begin{Bmatrix} \hat{N}_\xi \\ \hat{N}_\eta \\ \hat{N}_{\xi\eta} \\ \hat{M}_\xi \\ \hat{M}_\eta \\ \hat{M}_{\xi\eta} \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{16} & B_{11} & B_{12} & B_{16} \\ A_{12} & A_{22} & A_{26} & B_{12} & B_{22} & B_{26} \\ A_{16} & A_{26} & A_{66} & B_{16} & B_{26} & B_{66} \\ B_{11} & B_{21} & B_{16} & D_{11} & D_{12} & D_{16} \\ B_{12} & B_{22} & B_{26} & D_{12} & D_{22} & D_{16} \\ B_{16} & B_{26} & B_{66} & D_{16} & D_{26} & D_{66} \end{bmatrix} \begin{Bmatrix} \epsilon_\xi^\circ \\ \epsilon_\eta^\circ \\ \gamma_{\xi\eta}^\circ \\ \kappa_\xi \\ \kappa_\eta \\ \kappa_{\xi\eta} \end{Bmatrix} \tag{3.4.57}$$

$$\begin{Bmatrix} \epsilon_\xi^\circ \\ \epsilon_\eta^\circ \\ \gamma_{\xi\eta}^\circ \\ \kappa_\xi \\ \kappa_\eta \\ \kappa_{\xi\eta} \end{Bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{16} & \beta_{11} & \beta_{12} & \beta_{16} \\ \alpha_{12} & \alpha_{22} & \alpha_{26} & \beta_{12} & \beta_{22} & \beta_{26} \\ \alpha_{16} & \alpha_{26} & \alpha_{66} & \beta_{16} & \beta_{26} & \beta_{66} \\ \beta_{11} & \beta_{21} & \beta_{16} & \delta_{11} & \delta_{12} & \delta_{16} \\ \beta_{12} & \beta_{22} & \beta_{26} & \delta_{12} & \delta_{22} & \delta_{16} \\ \beta_{16} & \beta_{26} & \beta_{66} & \delta_{16} & \delta_{26} & \delta_{66} \end{bmatrix} \begin{Bmatrix} \hat{N}_\xi \\ \hat{N}_\eta \\ \hat{N}_{\xi\eta} \\ \hat{M}_\xi \\ \hat{M}_\eta \\ \hat{M}_{\xi\eta}, \end{Bmatrix} \tag{3.4.58}$$

where $[\alpha]$, $[\beta]$, and $[\delta]$ matrices are defined in Eq. (3.4.9).

The stresses in each layer, referred to the global reference system (ξ,η) are given by Eq. (3.4.50). The stresses in each layer, referred to the principal directions of the layer (1,2) are determined from Eq. (3.4.50) as follows

$$\left\{ \begin{array}{c} \sigma_1 \\ \sigma_2 \\ \tau_{12} \end{array} \right\}_k = [T']_k \left\{ \begin{array}{c} \sigma_\xi \\ \sigma_\eta \\ \tau_{\xi\eta} \end{array} \right\}_k , \tag{3.4.59}$$

with

$$[T']_k = \left[\begin{array}{ccc} c^2 & s^2 & 2sc \\ s^2 & c^2 & -2sc \\ -sc & sc & c^2 - s^2 \end{array} \right], \tag{3.4.60}$$

where $c = \cos(\theta_k)$ and $s = \sin(\theta_k)$.

Under plane-stress condition the quadratic failure criterion has the following form

$$F_1\sigma_1 + F_2\sigma_2 + F_{11}\sigma_1^2 + F_{22}\sigma_2^2 + F_{66}\tau_{12}^2 + 2F_{12}\sigma_1\sigma_2 < 1, \tag{3.4.61}$$

where the strength parameters are defined as

$$\begin{aligned} F_1 &= \frac{1}{t_1^+} - \frac{1}{t_1^-}, \quad F_2 = \frac{1}{t_2^+} - \frac{1}{t_2^-}, \\ F_{11} &= \frac{1}{t_1^+ t_1^-}, \quad F_{22} = \frac{1}{t_2^+ t_2^-}, \\ F_{66} &= \frac{1}{t_{12}^2}, \quad F_{12} = -\frac{1}{2}\sqrt{F_{11}F_{22}}, \end{aligned} \tag{3.4.62}$$

where t is the strength of the material, and the superscripts (+) and (-) refer to tension and compression, respectively. Assume, each stress component increases by the same proportion until failure occurs. This is expressed as

$$\sigma_1^f = R\sigma_1, \quad \sigma_2^f = R\sigma_2, \quad \tau_{12}^f = R\tau_{12}, \tag{3.4.63}$$

where superscript (f) refers to the stress components on the failure surface, and R is called the stress ratio. No failure occurs when $R > 1$, and failure occurs when $R \leq 1$. At failure, where the stress components are designated by the superscript f, Eq. (3.4.61) is

$$F_1\sigma_1^f + F_2\sigma_2^f + F_{11}\left(\sigma_1^f\right)^2 + F_{22}\left(\sigma_2^f\right)^2 + F_{66}\left(\tau_{12}^f\right)^2 + 2F_{12}\sigma_1^f\sigma_2^f = 1, \tag{3.4.64}$$

By substituting Eq. (3.4.63) into Eq. (3.4.64), we observe that the quadratic failure criterion becomes

$$R\left(F_1\sigma_1 + F_2\sigma_2\right) + R^2\left(F_{11}\sigma_1^2 + F_{22}\sigma_2^2 + F_{66}\tau_{12}^2 + 2F_{12}\sigma_1\sigma_2\right) = 1, \tag{3.4.65}$$

Whether or not failure occurs is indicated by the value of the stress ratio R given by the solution of Eq. (3.4.65) as follows

$$R = \frac{-b + \sqrt{b^2 + 4a}}{2a}, \tag{3.4.66}$$

where

$$a = F_{11}\sigma_1^2 + F_{22}\sigma_2^2 + F_{66}\tau_{12}^2 + 2F_{12}\sigma_1\sigma_2, \tag{3.4.67}$$

and

$$b = F_1\sigma_1 + F_2\sigma_2. \tag{3.4.68}$$

Chapter 4

Optimum Design of Thin-Walled Beam

4.1 Optimization Problem Formulation

4.1.1 Selection of Design Variables

The choice of design variables is a key factor in obtaining the optimal structure since it changes the character of the problem by changing the degree of nonlinearity of the objective and constraint functions. In this problem we deal with discrete and continuous design variables simultaneously. The mixed design variable s is mixed vector $s = (y, x)$.

The discrete design variable defines the stacking sequence of the composite laminate. We assume that all wall segments in the beam have exactly same stacking sequence. The maximum number of plies in the laminate, and, respectively, the chromosome length is ℓ. The alphabet of alleles is $\mathcal{A} = \{A_1, A_2, \ldots, A_{N(A)}\}$, which

correlates with the possible ply orientations

$$\{0^\circ, \ldots, \pm\frac{(i-1)\cdot 90^\circ}{N(\mathcal{A})-1}, \ldots, 90^\circ\}, \quad i = 1, \ldots, N(\mathcal{A}), \tag{4.1.1}$$

where $N(\mathcal{A}) = 7$ the total alphabet size. The ℓ-dimensional discrete design vector is $y = (y_1, y_2, \ldots, y_\ell)$, and the corresponding ℓ-dimensional integer chromosome is $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ with $\gamma_i \in \mathcal{A},\ 1 \leq i \leq \ell$. The decoding function Γ directly maps γ into y according to the following rule

$$y_i = \Gamma(\gamma_i) = \frac{90^\circ}{N(\mathcal{A})-1}(\gamma_i - 1). \tag{4.1.2}$$

However, due to manufacturing recommendations we will use only three possible ply orientations $\{0^\circ, \pm 45^\circ, 90^\circ\}$, and all chromosomes in the GA will contain only three alleles $\{1, 4, 7\}$.

The continuous design variables x are represented by the parameters

$$\varphi_i^\circ \in [\varphi_{min}^\circ, \varphi_{max}^\circ], \quad \text{and} \quad b_i \in [b_{min}, b_{max}], \quad i = 1, \ldots, \iota - 1, \tag{4.1.3}$$

where b_i is width of the ith wall segment, and φ_i° is the angle between $i-1$ and i adjacent wall segments as shown in Figure 3.11; $[\varphi_{min}^\circ, \varphi_{max}^\circ]$ and $[b_{min}, b_{max}]$ are the corresponding lower and upper values considered for the design variables. These design variables completely describe the cross-sectional geometry of the beam. In general formulation, the total number of the continuous design variables m is defined by $m = 2\iota - 2$, where ι is the number of wall segments. Since the coordinates of point ι are given, φ_ι° and b_ι are not variables. The length of ιth segment is calculated as follows

$$b_\iota = \sqrt{(\bar{y}_\iota - \bar{y}_{\iota-1})^2 + (\bar{z}_\iota - \bar{z}_{\iota-1})^2}. \tag{4.1.4}$$

Table 4.1: The cases with different numbers of continuous design variables

Case	ι	$x^{(1)}$		$x^{(2)}$		m
		φ_i°	m_1	b_i	m_2	
A	2	φ_1°	1	$b_1 = b$	1	2
B	3	φ_1°, φ_2°	2	$b_1 = b_2 = b$	1	3
C	4	φ_1°, φ_2°, φ_3°	3	$b_1 = b_2 = b_3 = b$	1	4
D	5	φ_1°, φ_2°, φ_3°, φ_4°	4	$b_1 = b_2 = b_3 = b_4 = b$	1	5

The vector of continuous design variables x is defined as follows

$$x = (x^{(1)}, x^{(2)}) = (x_1^{(1)}, \ldots, x_{m_1}^{(1)}, x_1^{(2)}, \ldots, x_{m_2}^{(2)}), \tag{4.1.5}$$

where $x^{(1)}$ and $x^{(2)}$ are two groups of continuous design variables corresponding to the parameters φ° and b, respectively, i.e.,

$$\begin{aligned} x^{(1)} &= (x_1^{(1)}, \ldots, x_{m_1}^{(1)}) = (\varphi_1^\circ, \ldots, \varphi_{m_1}^\circ), \\ x^{(2)} &= (x_1^{(2)}, \ldots, x_{m_2}^{(2)}) = (b_1, \ldots, b_{m_2}), \end{aligned} \tag{4.1.6}$$

and $m = m_1 + m_2$. Combining vectors $x^{(1)}$ and $x^{(2)}$ in one vector, we obtain

$$x = (x_1, \ldots, x_m) \in \prod_{i=1}^{m} [(x_i)_{min}, (x_i)_{max}], \tag{4.1.7}$$

where $x \in \mathbb{R}^m$.

In order to evaluate the proposed algorithm four cases with different numbers of continuous design variables, shown in Table 4.1, were selected.

4.1.2 Standard Form of the Optimization Problem

The considered problem can be defined according to standard definition as follows

$$\begin{aligned}
&\text{Maximize} \quad g_0(s) \\
&\text{subject to} \\
&\qquad g_j(s) \leq 0, \quad j \in \{1, \ldots, p\}, \\
&\qquad s = (y, x), \\
&\qquad y \in \mathbb{Z}^{\ell}, \quad x \in \mathbb{R}^{m}, \\
&\text{and} \\
&\qquad (x_i)_{min} \leq x_i \leq (x_i)_{max}, \quad i \in \{1, \ldots, m\}.
\end{aligned} \tag{4.1.8}$$

4.1.3 Selection of Objective Function

The weight of the segmented beam is to be minimized with the design configuration meeting all the design requirements. The weight of the beam is defined by the following formula

$$W(s) = \sum_{i=1}^{\iota} W_i, \tag{4.1.9}$$

where $W_i = \rho L h b_i$ is the weight of the ith wall segment, ρ is the material density, L is the length of the beam, h is the wall thickness, and b_i is the width of the ith beam segment. For the maximization problem the objective function is

$$g_0(s) = \frac{1}{W(s)}. \tag{4.1.10}$$

4.1.4 Selection of Constraints

We introduce four behavior constraints ($q = 4$), which are imposing limiting values on the cross-sectional area, Tsai-Wu failure criterion, the total displacement and rotation of the cross-section at the end of the beam. These constraints are defined

by the following expressions

$$\begin{aligned} A(s) &\geq A_{min}, \\ R(s) &\geq 1, \\ \hat{u}(s) &\leq \hat{u}_{max}, \\ \psi(s) &\leq \psi_{max}, \end{aligned} \tag{4.1.11}$$

where $A(s)$ is the area of the cross-section of the beam; $R(s)$ is the Tsai-Wu failure criterion defined in Eq. (3.4.66); $\hat{u}(s)$ is the total displacement of the centroid at the end of beam defined as

$$\hat{u}(s) = \sqrt{u^2(s) + v^2(s) + w^2(s)}; \tag{4.1.12}$$

$\psi(s)$ is the rotation of the centroid of the beam cross-section at the end of the beam. The displacements u, v, w, and ψ are defined by Eqs. (3.4.25).

The cross-section of the beam is a polygon defined by points $(\bar{x}_1^{(1)}, \bar{x}_2^{(1)})$ through $(\bar{x}_1^{(\iota)}, \bar{x}_2^{(\iota)})$. The polygon is composed of lines between adjacent vertices, and we assume that $(\bar{x}_1^{(1)}, \bar{x}_2^{(1)})$ is adjacent to $(\bar{x}_1^{(\iota)}, \bar{x}_2^{(\iota)})$. The area of such polygon in terms of the coordinates of the vertices is defined as follows

$$A(s) = \frac{1}{2} \sum_{i=1}^{\iota} \left(\bar{x}_1^{(i)} \bar{x}_2^{(i+1)} - \bar{x}_1^{(i+1)} \bar{x}_2^{(i)} \right), \tag{4.1.13}$$

where $\left(\bar{x}_1^{(\iota+1)}, \bar{x}_2^{(\iota+1)}\right)$ is $\left(\bar{x}_1^{(1)}, \bar{x}_2^{(1)}\right)$. The formula was described by Meister in 1769 and by Gauss in 1795. The formula 4.1.13 result in a positive area if the vertices are enumerated counterclockwise, otherwise the area is negative.

Finally, the inequality constraints (4.1.11) can be transformed to the standard

form according to the definition (4.1.8) as follows

$$
\begin{aligned}
g_1(s) &= \frac{A_{min}}{A(s)} - 1 \leq 0, \\
g_2(s) &= \frac{1}{R(s)} - 1 \leq 0, \\
g_3(s) &= \frac{\hat{u}(s)}{\hat{u}_{max}} - 1 \leq 0, \\
g_4(s) &= \frac{\psi(s)}{\psi_{max}} - 1 \leq 0.
\end{aligned}
\tag{4.1.14}
$$

Chapter 5

Results

5.1 Example of Composite Beam Design

First, we present analysis of a cantilever beam subjected to an axial load $\hat{N}_{\bar{x}} = 500$ N and torque $\hat{T}_x = 3$ N·m applied at the point O. The length of the cantilever beam $L = 1000$ mm. Setting the continuous design variables to their maximum limits, i.e., $\varphi_i^\circ = \varphi_{max}^\circ$, $i = 1, \ldots, \iota-1$, and $b = b_{max}$, we obtain the geometrical configurations of beam cross-sections presented in Figure 5.1. The selected stacking sequence, used in these examples, is $[0_3/90_3]$. The material properties of a ply made of unidirectional fibers are given in Table 5.1. The results for these examples are presented in Table 5.2. All selected designs are not feasible, since they violate constraint g_4. The response functions corresponding to the laminate stacking sequence $[0_3/90_3]$ are shown in Figure 5.2.

Table 5.1: Ply properties of graphite/epoxy (T300/5208)

Property	T300/5208
E_1, GPa	181.0
E_2, GPa	10.3
G_{12}, GPa	7.17
ν_{12},	0.28
t_1^+, MPa	1500.0
t_1^-, MPa	1500.0
t_2^+, MPa	40.0
t_2^-, MPa	246.0
t_{12}, MPa	68.0
ρ, g/cm^3	1.6
h_k, mm	0.5

Table 5.2: Results for the selected designs

Parameter	Case			
	A	B	C	D
ι	2	3	4	5
m	2	3	4	5
x_1	1.0	1.0	1.0	1.0
x_2	1.0	1.0	1.0	1.0
x_3	–	1.0	1.0	1.0
x_4	–	–	1.0	1.0
x_5	–	–	–	1.0
y	111777	111777	111777	111777
g_0	1.0417	0.6944	0.5208	0.4167
g_1	-0.2500	-0.6752	-0.8112	-0.8750
g_2	-0.9269	-0.9508	-0.9628	-0.9701
g_3	-0.9269	-0.9149	-0.9586	-0.9746
g_4	3.4395	1.9597	1.2197	0.7758
ϕ	-4.3995	-3.3997	-3.1397	-3.1758

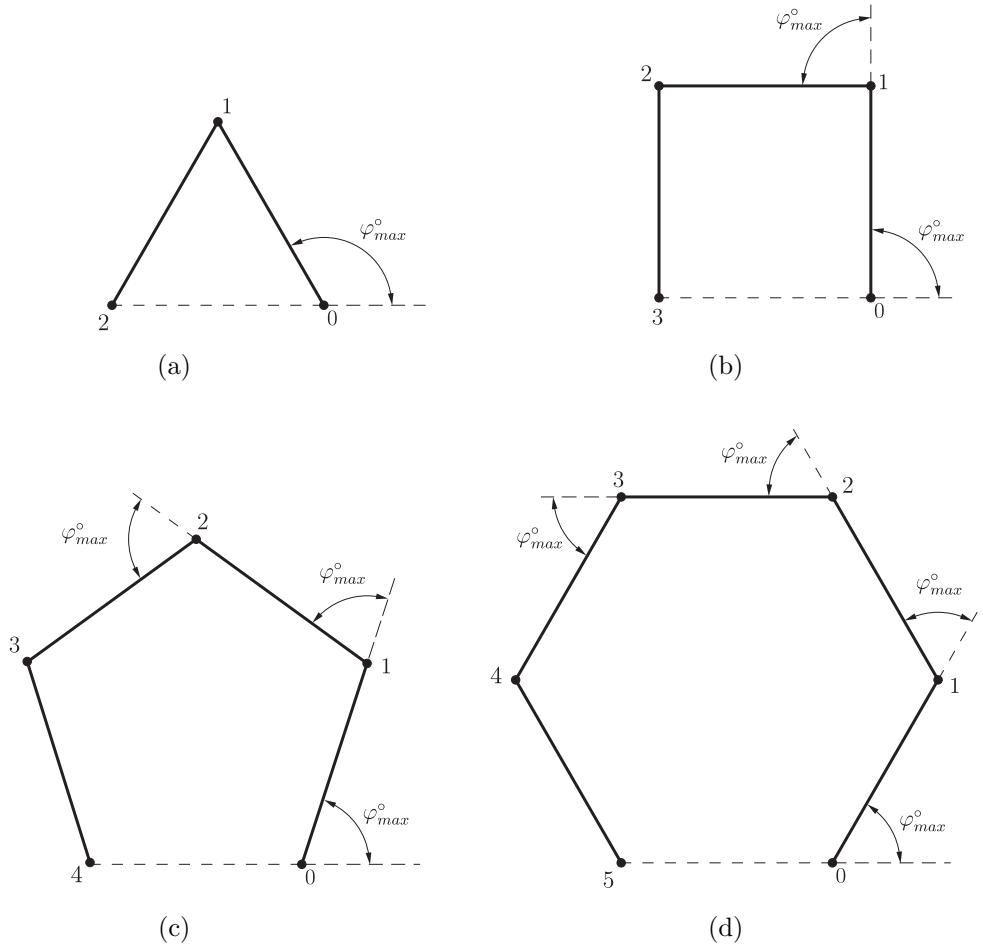


Figure 5.1: Cross-sections of segmented beams with different numbers of segments

5.2 Optimum Design by Improved Genetic Algorithm

Four cross-sections with different number of wall segments are considered according to Table 4.1. In consequence, all four cases have different number of continuous design variables. In the first case, the beam has two segments, and the corresponding optimization problem has two design variables: φ_1° and b; in the second case, three design variables: φ_1°, φ_2°, and b; in the third case, four design variables: φ_1°, φ_2°, φ_3°,

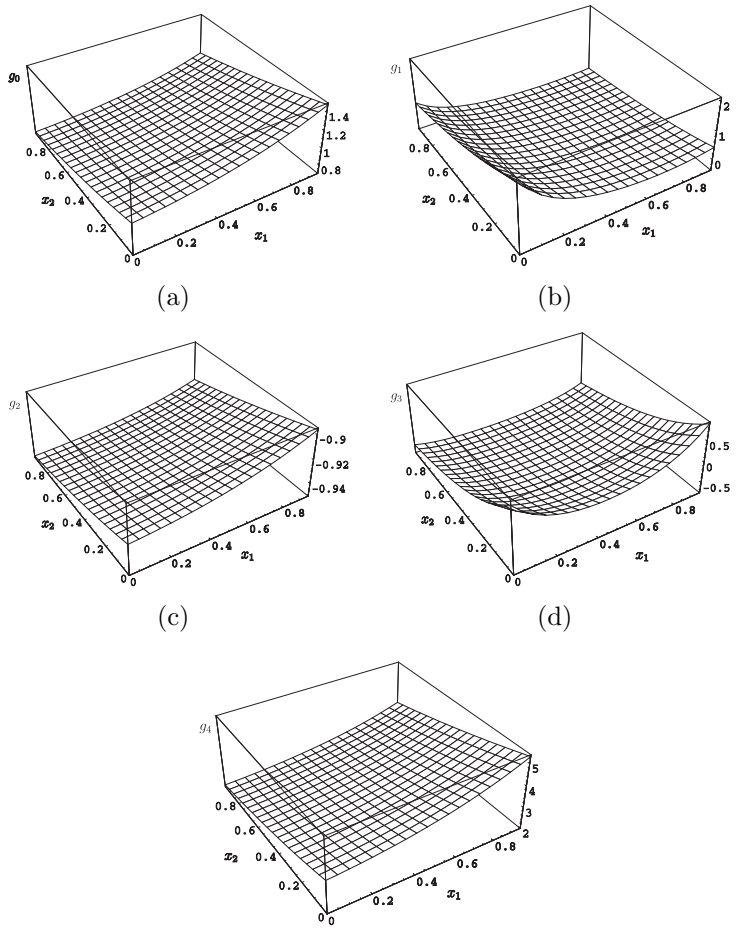


Figure 5.2: Response functions

Table 5.3: Ranges for design variables

Parameter	Range
$b_i \in [b_{min}, b_{max}]$	$[50.0, 100.0] \times 10^{-3}$ m
$\varphi_i^\circ \in [\varphi_{min}^\circ, \varphi_{max}^\circ]$	$[0, 2\pi/(\iota + 1)]$ rad
$n \in [n_{min}, n_{max}]$	[2,6]
θ_k, $k = 1, n$	$\{0^\circ, \pm 45^\circ, 90^\circ\}$

and b; and in the fourth case, five design variables: φ_1°, φ_2°, φ_3°, φ_4° and b. Here we assume that all segments in the wall has same length, i.e., $b_1 = b_2 = \ldots = b_{\iota-1} = b \in [b_{min}, b_{max}]$. The length of last ιth segment b_ι is calculated according to Eq. (4.1.4). The possible ranges for the design variables are given in Table 5.3.

5.2.1 GA Parameters

A Fortran 90 GA framework that was designed in an earlier research effort (McMahon et al., 1998; McMahon and Watson, 2000) was used for the composite laminate structure design. This framework includes a module, encapsulating GA data structures, and a package of GA operators. The module and the package of operators result in what we call a standard GA. The developed algorithm is incorporated within the GA framework to illustrate performance of the binary tree memory and multivariate approximations. An integer alphabet is used to code ply genes. The continuous design variables represented by floating point numbers had already been implemented in the GA framework data structure as geometry chromosomes.

The values of the GA parameters used in the experiments are shown in Table 5.4. The GA stopping condition is a limit on the total number of fitness function evaluations conducted by the standard GA, which is $(n_e)_{max} = 500000$. The best known optimal designs for the different cases obtained by standard GA are presented in Table 5.5, where n_e° is the average number of exact analyses of individuals obtained from 10 runs. Figure 5.3 shows the beam cross-sections corresponding to the the optimum designs.

In many cases, at the end of GA run it is possible to obtain several different feasible designs with the same objective function, and, therefore, with the same fitness function value. The weighted average ranking method is implemented for the purpose of making a decision about the optimal design. Table 5.6 shows the situation where seven feasible designs with the same discrete chromosomes $y = 447440$ and different continuous chromosomes have equal values $g_0 = 1.0316$ of the objective function. The weighted average ranking method selects the design s_3 as the best.

5.2.2 Effect of GA Improvement

The results presented in this section focus on the ability of the proposed algorithm to save computational time during GA optimization with the multivariate approximation used as a memory device. The best designs are identical to the results

Table 5.4: GA parameters used in experiments.

Parameter	Value
Selection type	Elitist
Maximum number of generations	25000
Population size	20
Integer chromosome length, λ	6
Crossover type:	
Integer chromosomes	One-point
Real chromosome	Uniform
Probability of crossover:	
Integer chromosomes	1.0
Real chromosome	1.0
Probability of mutation:	
Integer chromosomes	0.01
Real chromosome	0.01

Table 5.5: Best-known optimal designs using standard GA.

Parameter	Case			
	A	B	C	D
ι	2	3	4	5
m	2	3	4	5
n_e°	211304	157653	244643	155789
x_1	1.0	1.0	0.9834	1.0
x_2	0.5	1.0	0.9602	1.0
x_3	–	0.0	0.9638	1.0
x_4	–	–	0.1646	1.0
x_5	–	–	–	0.0
y	444444	441444	447440	447440
g_0	1.2616	1.2204	1.0316	0.9999
g_1	0.0000	-0.1339	-0.5644	-0.5714
g_2	-0.8513	-0.9622	-0.8341	-0.8302
g_3	-0.0712	-0.4872	-0.6442	-0.7010
g_4	-0.1457	-0.0356	0.0000	-0.0306
ϕ	1.2616	1.2204	1.0316	0.9999

Table 5.6: Selection of the best design using weighted average ranking.

	g	N_j	N
$s_1 = (447440, 0.9972, 0.8613, 0.9033, 0.0994)$			
g_1	-0.5689	0	
g_2	-0.8366	5	11
g_3	-0.6241	6	
$s_2 = (447440, 0.8142, 0.8989, 0.9579, 0.0305)$			
g_1	-0.5432	4	
g_2	-0.8381	2	9
g_3	-0.5941	3	
$s_3 = (447440, 0.9834, 0.9602, 0.9638, 0.1646)$			
g_1	-0.5644	1	
g_2	-0.8341	5	6
g_3	-0.6443	0	
$s_4 = (447440, 0.6448, 1.0000, 1.0000, 0.0037)$			
g_1	-0.5112	6	
g_2	-0.8381	2	13
g_3	-0.5730	5	
$s_5 = (447440, 0.8916, 0.9999, 0.5876, 0.0324)$			
g_1	-0.5639	2	
g_2	-0.8380	2	10
g_3	-0.5552	6	
$s_6 = (447440, 0.7864, 0.9839, 0.929, 0.0549)$			
g_1	-0.5420	5	
g_2	-0.8370	4	11
g_3	-0.5966	2	
$s_7 = (447440, 0.9559, 0.7364, 0.8655, 0.0000)$			
g_1	-0.5602	3	
g_2	-0.8397	0	7
g_3	-0.5879	4	

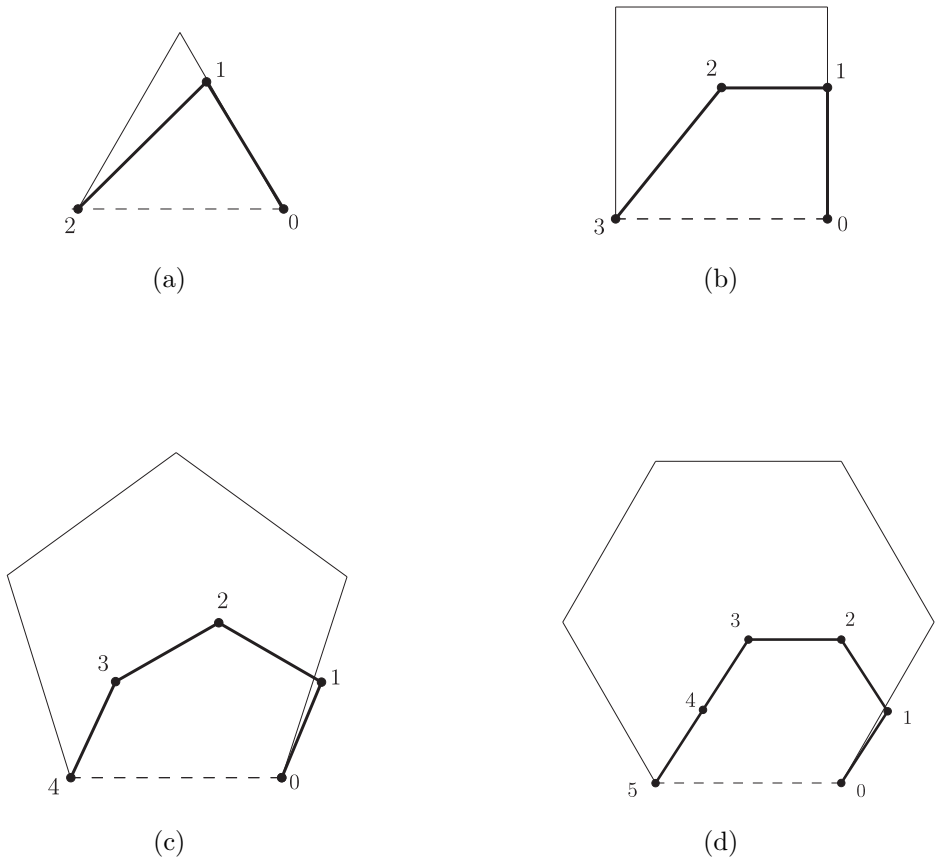


Figure 5.3: Cross-sections of segmented beams corresponding to the optimum designs

presented in Table 5.5 for the baseline algorithm. The performance of the GA with the multivariate approximation is presented in Table 5.7, which shows the average number of attempts to evaluate response functions n_i, the average number of exact analyses n_e, the average percent savings S in terms of response function evaluations, the average percent savings S° in terms of response function evaluations as compared with the standard GA result reported in Table 5.5, and the mean absolute error E due to the approximations. The average percent savings S in terms of number of

response function evaluations is defined by

$$S = \frac{1}{r_{max}} \sum_{r=1}^{r_{max}} \left[1 - \frac{(n_e)_r}{(n_i)_r} \right] \times 100\%, \tag{5.2.1}$$

where r_{max} is the number of runs. The average percent savings S° in terms of number of response function evaluations as compared with the standard GA is defined by

$$S^\circ = \frac{1}{r_{max}} \sum_{r=1}^{r_{max}} \left[1 - \frac{(n_e)_r}{n_e^\circ} \right] \times 100\%. \tag{5.2.2}$$

The average mean absolute error E is defined as

$$E = \frac{1}{r_{max}} \sum_{r=1}^{r_{max}} \left[\frac{1}{n_a} \sum_{i=1}^{n_a} |g(s_i) - \tilde{g}(s_i)| \right], \tag{5.2.3}$$

where $n_a = n_i - n_e$ is the total number of acceptable approximate evaluations for the given response function. This error is computed every time that the algorithm decides to extract an approximation of the constraint value without an exact analysis.

The results of the experiments in Table 5.7 show that the cost of the GA with continuous variables could be reduce up to 90% relative to the standard GA by using the approximation procedure. Moreover, for the problem considered, the computation of the fitness function is not very expensive in terms of CPU time. However, the realistic problems in which evaluation of the response functions may require large finite element analysis models, the computation effort spent on evaluating the fitness function far exceeds that of the memory tree and approximation constructions. Therefore, the approach developed in this project has great potential for problems with expensive fitness functions.

The mean absolute error E due to the approximation and the savings S in terms of the number of constraint evaluations for different values of the parameters ϵ and δ with $d_{max} = 0.5$ for all constraint functions for the case A are shown in Table 5.8.

Table 5.7: Efficiency of multivariate approximations

Case	n_e°	n_i	g	n_e	$S, \%$	$S^\circ, \%$	E
A	211304	153267	g_0	22017	76	90	3.14E–2
			g_1	25119	84	88	1.26E–1
			g_2	45367	70	79	2.84E–3
			g_3	36171	76	83	3.42E–2
			g_4	35404	77	83	6.73E–2
B	157653	121077	g_0	27968	77	82	1.29E–2
			g_1	23769	80	85	7.02E–2
			g_2	19006	84	88	1.24E–3
			g_3	22548	81	86	3.33E–2
			g_4	18917	84	88	5.93E–2
C	244643	179225	g_0	30442	83	88	2.82E–2
			g_1	22758	87	91	1.14E–1
			g_2	29747	83	88	1.73E–3
			g_3	38658	78	84	2.62E–2
			g_4	40185	78	84	5.25E–2
D	155789	114801	g_0	17643	85	89	2.64E–2
			g_1	27160	76	83	1.54E–1
			g_2	24751	78	84	1.83E–3
			g_3	27057	76	83	2.92E–2
			g_4	17527	85	89	6.14E–2

Table 5.8: The average percent savings (S) and the average error of the multivariate approximations (E) as functions of the parameters ϵ and δ with $d_{max} = 0.5$ for the case A

δ	g_0		g_1		g_2		g_3		g_4	
	S, %	E	S, %	E	S, %	E	S, %	E	S, %	E
					$\epsilon = 0.001$					
0.1	42	5.12E–3	44	3.65E–2	32	4.16E–4	41	8.65E–3	41	1.68E–2
0.5	45	5.24E–3	52	3.79E–2	34	4.26E–4	53	8.78E–3	46	1.79E–2
1.0	48	5.32E–3	58	4.14E–2	35	4.68E–4	57	9.95E–3	51	1.94E–2
					$\epsilon = 0.005$					
0.1	64	1.14E–2	63	6.11E–2	68	1.46E–3	62	1.56E–2	65	3.12E–2
0.5	64	1.63E–2	64	7.96E–2	69	1.67E–3	64	1.67E–2	65	3.54E–2
1.0	67	2.27E–2	68	8.61E–2	71	1.87E–3	68	1.81E–2	66	4.12E–2
					$\epsilon = 0.01$					
0.1	76	3.14E–2	84	1.26E–1	70	2.84E–3	76	3.42E–2	77	6.73E–2
0.5	77	4.03E–2	84	2.35E–1	70	3.45E–3	77	5.33E–2	78	8.35E–2
1.0	78	7.19E–2	85	2.96E–1	71	3.68E–3	77	7.15E–2	79	9.17E–2

It is possible to further enhance the performance of the algorithm by more precise tuning of its parameters. Table 5.8 shows the expected trends; both average savings S and average absolute error E increase as either ϵ or δ increases.

5.3 Conclusions

A solution strategy for the multi-constraint design problem, based on memory algorithm and multivariate approximations, has been presented and successfully applied to the weight minimization of segmented open section composite beam. The results obtained for different case studies with different numbers of design variables. The use of memory based on binary tree for integer design variables avoids repeating analyses of previously encountered designs. The multivariate approximation for continuous variables saves unnecessary exact analyses for points close to previous values. Modifications of the standard GA to save previously computed response functions' values provide significant performance improvement.

The results continue developing the memory procedure with multivariate approxi-

mations of response functions, confirming how how the proposed approach may open a new and deeper insight in the field of composite structures design optimization.

Several parts of this work are interesting starting points for further research. First, the binary tree allocates large computer memory for a large optimization problem. One of the possible solution of this problem is to construct binary tree based on several last generations only. Another possible approach is to incorporate a decision making algorithm which will discard nodes with bad designs from building of database for response functions' approximations. Second, there is a need to develop a methodology to quantify the degree of non-linearity of a continuous design space. Third, it is necessary to continue search for new approximation methods which will successfully build surrogate functions to response functions based on scattered data points in high dimensions. Fourth, multiobjective optimization should be incorporates into GA. The developed data structure is well adopted to deal with a multiobjective optimization problems.

BIBLIOGRAPHY

Abe, A., Kamegawa, T., and Nakajima, Y. (2004). Optimization of construction of tire reinforcement by genetic algorithm. *Optimization and Engineering*, 5:77–92.

Adams, D., Watson, L. T., and Gürdal, Z. (2003). Optimization and blending of composite laminates using genetic algorithms with migration. *Mechanics of Advanced Materials and Structures*, 10:183–203.

Adams, D. B., Watson, L. T., Gürdal, Z., and Anderson-Cook, C. M. (2004). Genetic algorithm optimization and bending of composite laminates by locally reducing laminate thickness. *Advances in Engineering Software*, 35:35–43.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition.

Averill, R. C., Punch, W. F., Goodman, E. D., Lin, S. C., Yip, Y. C., and Ding, Y. (1995). Genetic algorithm-based design of energy absorbing laminated composite beams. In *ASME Design Engin. Tech. Conf.*, Boston, MA.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proc. of the 11th Int'l Conf. on Genetic Algorithms and Their Applications*, pages 101–111.

Bank, L. C. and Bednarczyk, P. J. (1988). A beam theory for thin-walled composite beams. *Composite Science and Technology*, 32:265–277.

Barbero, E. J., Lopez-Anido, R., and Davalos, J. F. (1993). On the mechanics of thin-walled laminated composite beams. *Journal of Composite Materials*, 27:806–829.

Bauld, N. R. and Tzeng, L. S. (1984). A vlasov theory of fiber reinforced beams with thin-walled open cross sections. *International Journal of Solids and Structures*, 20:277–297.

Berry, M. W. and Minser, K. S. (1999). Algorithm 798: High-dimensional interpolation using the modified Shepard method. *ACM Transactions on Mathematical Software*, 25(3):353–366.

Callahan, K. J. and Weeks, G. E. (1992). Optimum design of composite laminates using genetic algorithm. *Composites Engineering*, 2(3):149–160.

Coello, C. A. C. (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Comput. Methods Appl. Mech. Engrg.*, 191:1245–1287.

Collette, Y. and Siarry, P. (2003). *Multiobjective Optimization*. Springer, Berlin Heidelberg.

Crossley, W. A. and Laananen, D. H. (1996). Genetic algorithm based optimal design of stiffened composite panels for energy absorption. In *Proc. of 52nd AHS Annual Forum*, pages 1367–1376, Washington, DC.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186:311–338.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966a). Adaptation of evolutionary programming to the prediction of solar flares. report NASA-CR-417, General Dynamics-Convair, San Diego, CA.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966b). *Artificial intellegence through simulated evolution*. John Wiley & Sons, New York.

Franke, R. and Nielson, G. (1980). Smooth interpolations of large sets of scattered data. *International Journal for Numerical Methods in Engineering*, 15:1691–1704.

Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers, i. introduction. *Australian Journal of Biological Sciences*, 10:484–491.

Gantovnik, V. B., Anderson-Cook, C. M., Gürdal, Z., and Watson, L. T. (2002a). A genetic algorithm with memory for mixed discrete-continuous design optimization. In *Proceedings of the 9th AIAA/ ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA. AIAA Paper 2002-5431.

Gantovnik, V. B., Anderson-Cook, C. M., Gürdal, Z., and Watson, L. T. (2003a). A genetic algorithm with memory for mixed discrete-continuous design optimization. *Computers & Structures*, 81:2003–2009.

Gantovnik, V. B., Gürdal, Z., and Watson, L. T. (2002b). A genetic algorithm with memory for optimal design of laminated sandwich composite panels. *Composite Structures*, 58:513–520.

Gantovnik, V. B., Gürdal, Z., and Watson, L. T. (2002c). A genetic algorithm with memory for optimal design of laminated sandwich composite panels. In *Proceedings of the 43rd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, Denver, CO. AIAA Paper 2002-1221.

Gantovnik, V. B., Gürdal, Z., Watson, L. T., and Anderson-Cook, C. M. (2003b). A genetic algorithm for mixed nonlinear programming problems using separate constraint approximations. In *Proceedings of the 44th AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, Norfolk, VA. AIAA Paper 2003-1700.

Gantovnik, V. B., Gürdal, Z., Watson, L. T., and Anderson-Cook, C. M. (2005). Genetic algorithm for mixed integer nonlinear programming problems using separate constraint approximations. *AIAA Journal*, 43(8):1844–1849.

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

Gürdal, Z., Haftka, R. T., and Nagendra, S. (1994). Genetic algorithm for the design of laminated composite panels. *SAMPE Journal*, 30(3):29–35.

Hajela, P. (1989). Genetic search - an approach to the nonconvex optimization problem. In *Proceedings of 30th AIAA/ASME/ASCE/AHS/ASC SDM Conference*, Mobile, Alabama.

Hajela, P. (1990). Genetic search - an approach to the nonconvex optimization problem. *AIAA Journal*, 26(7):1205–1210.

Harrison, P. N., Le Riche, R., and Haftka, R. T. (1995). Design of stiffened panels by genetic algorithm and response surface approximation. In *Proceedings of the 36th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, pages 58–68, New Orleans, LA. AIAA Paper 95-1163-CP.

He, Y. and Aref, A. J. (2003). An optimization design procedure for fiber reinforced polymer web-core sandwich bridge deck systems. *Composite Structures*, 60:183–195.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

Jaunky, N., Knight, N. F., and Ambur, D. R. (1998). Optimal design of general stiffened composite circular cylinders for global buckling with strength constraints. *Composite Structures*, 41:243–252.

Jong, K. A. D. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI. Deptartment Of Computer and Communication Sciences.

Kaletta, P. and Wolf, K. (2000). Optimization of composite aircraft panels using evolutionary computation methods. In *Proc. of the 22nd ICAS Conference*, pages 411.1–411.10, Harrogate, UK.

Kallassy, A. and Marcelin, J. L. (1997). Optimization of stiffened plates by genetic search. *Structural Optimization*, 13:132–141.

Kang, J. H. and Kim, C. G. (2005). Minimum-weight design of compresevelly loaded composite plates and stiffened panels for postbuckling strength by genetic algorithm. *Composite Structures*, 69:239–246.

Kobelev, V. V. and Larichev, A. D. (1988). Model of thin-walled anisotropic rods. *Mehanika Kompozicionnyh Materialov*, 24:102–109.

Kodiyalam, S., Nagendra, S., and DeStefano, J. (1996). Composite sandwich structure optimization with application to satellite components. *AIAA Journal*, 34(3):614–621.

Kogiso, N., Watson, L. T., Gürdal, Z., and Haftka, R. T. (1994a). Genetic algorithm with local improvement for composite laminate design. *Structural Optimization*, 7(4):207–218.

Kogiso, N., Watson, L. T., Gürdal, Z., Haftka, R. T., and Nagendra, S. (1994b). Design of composite laminates by a genetic algorithm with memory. *Mechanics of Composite Materials and Structures*, 1(1):95–117.

Kollar, L. P. and Pluzsik, A. (2002). Analysis of thin-walled composite beams with arbitrary layup. *Journal of Reinforced Plastics and Composites*, 21(16):1423–1465.

Koza, J. R. (1989). Evolving programs using symbolic expressions. In *Proc. of the 11th Int'l Joint Conf. on Artificial Intelligence*, pages 768–774, San Mateo, CA. Morgan Kaufmann.

Koza, J. R. (1990). Genetic algorithm: a paradigm for genetically breeding populations of computer programs to solve problems. Tech. Rep. STAN-CS-90-1314, Department of Computer Science, Stanford University.

Labossiere, P. and Turkkan, N. (1992). Optimization of composite materials with genetic algorithms and neural networks. In Neale, K. W. and Labossiere, P., editors, *Proceedings of Int. Conf. on Advanced Composite Materials in Bridges and Structures*, pages 659–668, Montreal. Canadian Society for Civil Engineering.

Le Riche, R. (1994). *Optimization of composite structures by genetic algorithms*. PhD thesis, Virginia Polytechnic Institute and State University, Dept. of Aerospace Eng.

Le Riche, R. and Haftka, R. T. (1993). Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm. *AIAA Journal*, 31(5):951–956.

Le Riche, R. G. and Gaudin, J. (1998). Design of dimensionally stable composites by evolutionary optimization. *Composite Structures*, 41:97–111.

Lin, C. and Lee, Y. (2004). Stacking sequence optimization of laminated composite structures using genetic algorithm with local improvement. *Composite Structures*, 63:339–345.

Malott, B., Averill, R. C., Goodman, E. D., Ding, Y., and Punch, W. F. (1996). Use of genetic algorithms for optimal design of laminated composite sandwich panels with bending-twisting coupling. In *Proceedings of the 37th AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, pages 1874–1881, Salt Lake City, UT. AIAA Paper 1996-26801.

Mansfield, E. H. and Sobey, A. J. (1979). The fiber composite helicopter blade - part 1: stiffness properties, part 2: prospectors for aeroelastic tailoring. *Aeronautical Quarterly*, 30:413–449.

Matous, K. and Dvorak, G. J. (2003). Optimization of electromagnetic absorption in laminated composite plates. *IEEE transactions on magnetics*, 39(3):1827–1835.

McMahon, M. T. and Watson, L. T. (2000). A distributed genetic algorithm with migration for the design of composite laminate structures. *Parallel Algorithms and Applications*, 14:329–362.

McMahon, M. T., Watson, L. T., Soremekun, G. A., Gürdal, Z., and Haftka, R. T. (1998). A Fortran 90 genetic algorithm module for composite laminate structure design. *Engineering with Computers*, 14(3):260–273.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, Germany.

Minga, A. K. (1987). Honeycomb design using a genetic algorithm. In *Proceedings of the AIAA Southeastern Regional Student Conference*, Atlanta, GA.

Nagendra, S., Haftka, R. T., and Gürdal, Z. (1992). Stacking sequence optimization of simply supported laminates with stability and strain constraints. *AIAA Journal*, 30(8):2132–2137.

Nagendra, S., Haftka, R. T., and Gürdal, Z. (1993a). Design of a blade stiffened composite panel by genetic algorithm. In *Proceedings of the the 34th AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, pages 2418–2436, La Jolla, CA. AIAA Paper No. 93-1584-CP.

Nagendra, S., Haftka, R. T., and Gürdal, Z. (1993b). A genetic algorithm based design procedure for stiffened composite panels under stability and strain constraints. In *Proceedings of the the 10th DOD/ NASA/ FAA Conference on Fibrous Composites in Structural Design*, Hilton Head, SC.

Nagendra, S., Jestin, D., Gürdal, Z., Haftka, R. T., and Watson, L. T. (1996). Improved genetic algorithm for the design of stiffened composite panels. *Computers & Structures*, 58(3):543–555.

Park, C. H., Lee, W. I., Han, W. S., and Vautrin, A. (2004). Simultaneous optimization of composite structures considering mechanical performance and manufacturing cost. *Composite Structures*, 65:117–127.

Pluzsik, A. and Kollar, L. P. (2002). Effects of shear deformation and restrained warping on the displacements of composite beams. *Journal of Reinforced Plastics and Composites*, 21(17):1517–1541.

Potgieter, E. and Stander, N. (1998). The genetic algorithm applied to stiffness maximization of laminated plates: review and comparison. *Structural Optimization*, 15:221–229.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–430, San Mateo, CA. Morgan Kaufmann.

Rahul, Chakraborty, D., and Dutta, A. (2005). Optimization of FRP composites against impact induced failure using island model parallel genetic algorithm. *Composites Science and Technology*, 65:2003–2013.

Rechenberg, I. (1964). Cybernatic solution path of an experimental problem. *Royal Aircraft Establishment, Library Translation 1122, Farnborough, Hants, Aug. 1965.* English translation of the unpublished wriiten summary of the lecture, derived at the joint annual meeting of the WGLR and DGRR, Berlin.

Renka, R. J. (1988a). Algorithm 660: QSHEP2D: quadratic Shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 14(2):149–150.

Renka, R. J. (1988b). Algorithm 661: QSHEP3D: quadratic Shepard method for trivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 14(2):151–152.

Renka, R. J. (1988c). Multivariate interpolation of large sets of scattered data. *ACM Transactions on Mathematical Software*, 14(2):139–148.

Renka, R. J. (1999a). Algorithm 790: CSHEP2D: cubic Shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 25(1):70–73.

Renka, R. J. (1999b). Algorithm 791: TSHEP2D: Cosine series Shepard method for bivariate interpolation of scattered data. *ACM Transactions on Mathematical Software*, 25(1):74–77.

Sargent, P. M., Ige, D. O., and Ball, N. R. (1995). Design of laminate composite layups using genetic algorithms. *Engineering with Computers*, 11:59–69.

Schwefel, H. P. (1968). Experimentelle Optimierung einer Zweiphasenduse Teil I. Report 35 for the project MHD-Strastrahlrohr, AEG Research Institute, Berlin.

Schwefel, H. P. (1981). *Numerical optimization of computer models.* John Wiley & Sons, Chichester.

Schwefel, H. P. (1995). *Evolution and Optimum Seeking.* John Wiley & Sons, New York.

Seresta, O., Gürdal, Z., Adams, D. B., and Watson, L. T. (2004). Optimal design of composite wing structures with blended laminates. In *Proceedings of the 10th AIAA/ ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Albany, NY. AIAA Paper 2004-4349.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly spaced data. *Proceedings of the 23rd National Conference, ACM*, pages 517–523.

Sivakumar, K., Iyengar, N. G. R., and Kalyanmoy, D. (1998). Optimimum design of laminated composite plates with cutouts using a genetic algorithm. *Comp. Struct.*, 42:265–279.

Soremekun, G., Gurdal, Z., Kassapoglou, C., and Toni, D. (2001). Stacking sequence blending of multiple composite laminates using genetic algorithms. In *Proceedings of the 42nd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, Seattle, WA. AIAA Paper No. 2001-1203.

Soremekun, G., Gürdal, Z., Kassapoglou, C., and Toni, D. (2002). Stacking sequence blending of multiple composite laminates using genetic algorithms. *Composite Structures*, 56:53–62.

Soremekun, G. A. (1997). Genetic algorithms for composite laminate design and optimization. Master's thesis, Department of Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Spallino, R. and Thierauf, G. (2000). Thermal buckling optimization of composite laminates by evolution strategies. *Computers & Structures*, 78:691–697.

Todoroki, A., Watanabe, K., and Kobayashi, H. (1995). Application of genetic algorithms to stiffness optimization of laminated composite plates with stress-concentrated open holes. *JSME Internat. Journal*, 38(4):458–464.

Vlasov, V. Z. (1958). *Tonkostennye uprugie sterzhni.* Gosudarstvennoe izdatelstvo fiziko-matematicheskoi literatury, Moscow, Russia, 2nd edition. in Russian.

Vlasov, V. Z. (1961). *Thin-Walled Elstic Beams*. Israel Program for Scientific Translations, Jerusalem, Israel. Translated from the Russian *Tonkostennye uprugie sterzhni* by the Israel Program for Scientific Translations for the National Science Foundation, available from the Office of Technical Services, U.S. Department of Commerce, Washington.

Vowels, R. A. (1998). *Algorithms and Data Structures in F and Fortran*. Unicomp, Inc, Tucson, Arizona.

Wolf, K. (2001). Optimization of composite sandwich panels using evolutionary computational methods. In *Proceedings of the 42nd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference*, Seattle, WA. AIAA Paper No. 2001-1277.

Woodson, M. B., Johnson, E. R., and Haftka, R. T. (1995). Optimal design of composite fuselage frames for progressive failure and energy absorption by genetic algorithms. Technical Report AIAA Paper 95-1218, AIAA, Reston, VA.

Yao, X. (1993). An empirical study of genetic operators in genetic algorithms. *Microprocessing and Microprogramming*, 38:707–714.

Appendix A

Selected Publications

ELSEVIER

Composite Structures 58 (2002) 513–520

COMPOSITE STRUCTURES

www.elsevier.com/locate/compstruct

A genetic algorithm with memory for optimal design of laminated sandwich composite panels

Vladimir B. Gantovnik [a], Zafer Gürdal [a], Layne T. Watson [b,*]

[a] Department of Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0219, USA
[b] Departments of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, 630 McBryde Hall, Blacksburg, VA 24061-0106, USA

Abstract

This paper is concerned with augmenting genetic algorithms (GAs) to include memory for continuous variables, and applying this to stacking sequence design of laminated sandwich composite panels that involves both discrete variables and a continuous design variable. The term "memory" implies preserving data from previously analyzed designs. A balanced binary tree with nodes corresponding to discrete designs renders efficient access to the memory. For those discrete designs that occur frequently, an evolving database of continuous variable values is used to construct a spline approximation to the fitness as a function of the single continuous variable. The approximation is then used to decide when to retrieve the fitness function value from the spline and when to do an exact analysis to add a new data point for the spline. With the spline approximation in place, it is also possible to use the best solution of the approximation as a local improvement during the optimization process. The demonstration problem chosen is the stacking sequence optimization of a sandwich plate with composite face sheets for weight minimization subject to strength and buckling constraints. Comparisons are made between the cases with and without the binary tree and spline interpolation added to a standard GA. Reduced computational cost and increased performance index of a GA with these changes are demonstrated.

Keywords: Genetic algorithm; Composite panel structure; Spline approximation

1. Introduction

Traditionally, the problem of composite laminate stacking sequence optimization has been defined as a continuous design problem and solved using gradient-based techniques [1]. However, because of manufacturing considerations, the orientations of the fibers in plies are typically confined within a discrete set, for example, (0°, +45°, −45°, 90°). The stacking sequence optimization in this case could be formulated as an integer-programming problem [2]. Recent studies have shown that genetic algorithms (GAs) are highly suitable for the solution of the composite laminate design problems with discrete design variables [3,4]. The solution of such problems by GA is possible, because the method does not require gradient or Hessian information. A GA is a powerful technique for search and optimization problems with discrete variables, and is therefore particularly useful for optimization of composite laminates. However, to reach an optimal solution with a high degree of confidence, it typically requires a large number of analyses during the optimization search. Performance of GAs is even more of an issue for problems that include continuous variables.

Several studies have concentrated on improving the reliability and efficiency of GAs. Hybrid algorithms formed by the combination of a GA with local search methods provide increased performance when compared to a GA with a discrete encoding of real numbers or local search alone [5]. In order to reduce the computational cost, two of the authors earlier used local improvements and memory so that information from previously analyzed design points is utilized during a search [6,7]. In the first method a memory binary tree was employed to store pertinent information about laminate designs that have already been analyzed [6]. After the creation of a new population of designs, the tree structure is searched for either a design with identical stacking sequence or similar performance, such as a laminate with identical in-plane strains. Depending on

* Corresponding author. Tel.: +1-540-231-7540; fax: +1-540-231-6075.
E-mail address: ltw@cs.vt.edu (L.T. Watson).

0263-8223/02/$ - see front matter © 2002 Published by Elsevier Science Ltd.
PII: S0263-8223(02)00128-9

the kind of information that can be retrieved from the tree, the analysis for a given laminate may be significantly reduced or may not be required at all. The second method is called local improvement [7]. This technique was applied to the problem of maximizing the buckling load of a rectangular laminated composite plate. The information about previously analyzed designs is used to construct an approximation to buckling load in the neighborhood of each member of the population of designs. After that the approximations are used to search for improved designs in small discrete spaces around nominal designs. These two methods demonstrated substantial improvements in computational efficiency for purely discrete optimization problems. The implementation, however, was not suitable for handling continuous design variables.

The objective of the present work is to find a suitable algorithm for a GA with memory that can work with discrete and continuous variables simultaneously. A local memory for the continuous part of the design space based on spline approximation is proposed for problems with a single continuous variable. The efficiency of the proposed spline based procedure, as well as the use of memory for a GA that can handle continuous variables, are investigated for the weight optimization of a sandwich plate with composite face sheets subjected to strength and buckling constraints.

2. Genetic algorithm package

A state-of-the-art Fortran 90 GA framework that was designed in an earlier research effort was used for the composite laminate structure design [8]. This framework includes a module, encapsulating GA data structures, and a package of GA operators. The module and the package of operators result in what we call a standard GA. The proposed algorithm is incorporated within the GA framework as a sample test program that illustrates performance of the binary tree memory and spline interpolation. An integer alphabet is used to code ply genes. It should be pointed out that the continuous variables had already been implemented in the GA data structure as geometry chromosomes, represented directly as real numbers and not discrete binary approximations. Fig. 1 from [8] shows the data structure of a population for the most general case of composite structure design. This data structure supports advanced features such as migration between independently evolving subpopulations, use of multiple materials for designing hybrid laminates, and multiple laminated substructures represented by multiple laminate chromosomes. In general, the laminate chromosomes are for discrete variables, and the geometry chromosomes are for continuous variables. Migration is not used here, however, so there is just one subpopulation identical to the population, and there is just one laminate chromosome with one material type. Here only the ply orientation genes are used for the stacking sequence definition and a single geometry gene is used for the single continuous variable.

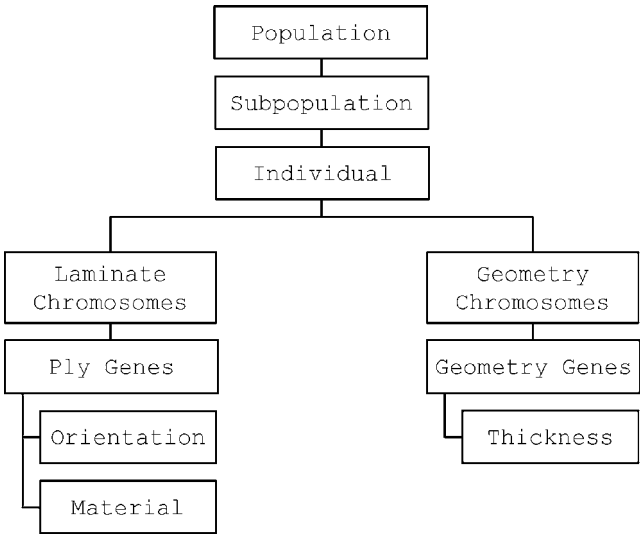


Fig. 1. The structure of a population.

3. Binary tree memory

A binary tree is a linked list structure in which each node may point to up to two other nodes. In a binary search tree, each left pointer points to nodes containing elements that are smaller than the element in the current node; each right pointer points to nodes containing elements that are greater than the element in the current node, as shown in Fig. 2. A binary tree has several properties of great practical value, one of which is that the data can be retrieved, modified, and inserted relatively quickly. If the tree is perfectly balanced, the cost of inserting of an element in a tree with n nodes is proportional to $\log_2 n$ steps, and rebalancing the tree after an insertion may take as little as several steps, but at most takes $\log_2 n$ steps. Thus, the total time is of the order of $\log_2 n$ [9].

In the standard GA, a new population may contain designs that have already been encountered in the

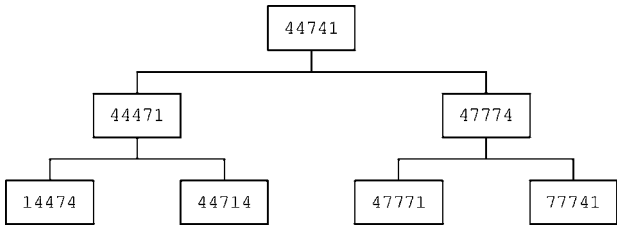


Fig. 2. An example of a binary tree.

```
Procedure Evaluation of the fitness function using a binary tree
begin
    search for the given design in the binary tree;
    if found then
      get the fitness function value from the binary tree;
    else
      perform exact analysis;
    end if
end
```

Fig. 3. Evaluation of fitness function using binary tree.

previous generations, especially towards the end of the optimization process. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. The binary tree, keyed on the discrete design string encodings, is used to store data pertinent to the design such as the design string and the fitness and constraint functions. In Fig. 2, for example, the integers in the boxes represent design string encodings, used as keys for comparison to traverse the tree. The boxes (binary tree nodes) would also contain fitness and constraint function values (not shown) for that discrete design. Fig. 3 shows the pseudocode for the fitness function evaluation with the aid of the binary tree.

After a new generation of designs is created by the genetic operations, the binary tree is searched for each new design. If the design is found, the fitness value is retrieved from the binary tree without conducting an analysis. Otherwise, the fitness is obtained based on an exact analysis. This new design and its data are then inserted in the tree as a new node.

4. Spline interpolation

The procedure described above works well for designs that are completely described by discrete strings. In case of designs that include a continuous variable, the solution is more complicated. If the continuous variable is also discretized into a fine discrete set, the possibility of creating a child design that has the same discrete and continuous parts as one of the earlier designs diminishes substantially. In the worst case, if the continuous design variable is represented as a real number, which is the approach used by most recent research work, it may be impossible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless.

The main idea of the approach proposed in this paper is to construct approximations for the fitness function as a function of the continuous variable using a spline function fitted to the historical data, and interpolate from the stored data whenever possible. The memory in this case consists of two parts. A binary tree, which consists of the nodes that have different discrete parts of the design, and a storage part at each node that keeps the continuous values and the associated fitness function. That is, each node contains a real array that stores the continuous variable and its corresponding value of the fitness function. In order for the memory to be functional, it is necessary to have accumulated enough designs with different continuous values for a particular discrete design part. Naturally, not all the nodes will have more than a few designs with different continuous values. However, it is possible that as the evolution progresses good discrete parts will start appearing again and again with different continuous values. In this case, one will be able to construct a good quality spline interpolant to the data.

In addition to building a spline approximation it is important to develop a scheme in which accuracy of the spline approximation at a new continuous point may be assessed, so that a decision may be made either to accept the approximation or perform exact function evaluation. In the proposed approach for all the design points with identical discrete variables and different continuous variable values, a measure of distance within which a good quality approximation may be obtained is used to make the decision. Based on the stored information, the procedure described by pseudocode in Fig. 4 decides whether an exact analysis should be performed and stored in the tree, or to extract an approximation of the fitness value without an exact analysis.

For the description of the pseudocode, let v be a discrete vector variable, x a scalar real variable, $\Phi(v,x)$ the fitness of the individual defined by (v,x), and d a real number representing the radius of an interval within which a good quality approximation is possible. Each

```
Procedure Evaluation of the fitness function using a binary tree and spline
begin
get a candidate individual (v, x) for insertion in the tree;
search for v of the given design in the binary tree;
if v is not found in the tree, then
  evaluate Φ(v, x);
  add a node corresponding to (v, x, Φ(v, x), 0);
  return with value Φ(v, x) to the GA;
else
  let (v, x_i, Φ_i, d_i), 1 ≤ i ≤ n, be all the node data matching v;
  compute a spline S interpolating the data (x_i, Φ_i), 1 ≤ i ≤ n;
  evaluate the spline S(x);
  if max_{1≤i≤n}{d_i − |x − x_i|} ≥ 0, then
    return with the value S(x) to the GA;
  else
    evaluate Φ(v, x);
    if |Φ(v, x) − S(x)| > ε, then
      add a node corresponding to (v, x, Φ(v, x), 0);
      return with the value Φ(v, x) to the GA;
    else
      define k and d by |x − x_k| = min_{1≤i≤n} |x − x_i|, d = min{d_0, |x − x_k|};
      change the data at node (v, x_k, Φ_k, d_k) to (v, x_k, Φ_k, d);
      add a node corresponding to (v, x, Φ(v, x), d);
      return with the value Φ(v, x) to the GA;
    end if
  end if
end if
end
```

Fig. 4. Evaluation of fitness function using binary tree and spline.

node in the binary tree memory structure records the data $(v, \{x_i, \Phi(v,x_i), d_i\}_{i=1}^{n})$, where n is the number of times the design is evaluated with different values of the real design variable corresponding to the same discrete part. For the first three values of the continuous variable at a given discrete branch, the points are automatically evaluated and entered into the spline database with radius values d of zero. Since the initial spline points will have zero radius, many of the additional points x will not satisfy the proximity condition $\max_{1\leqslant i\leqslant n}\{d_i - |x - x_i|\} \geqslant 0$ initially, and therefore their fitness $\Phi(v,x)$ will be computed exactly. It is possible to add the new points either with a zero radius or a positive computed value for the radius. If the actual value of the fitness $\Phi(v,x)$ and the spline approximation $S(x)$ are within acceptable error limit ϵ, then the radius value d for the new point and the closest point x_k to the new point are both changed to the distance between the two points, $d = \min\{d_0, |x - x_k|\}$, and the new point is added to the spline database with this computed value. Otherwise, the new point is added to the database with a zero radius value (see Fig. 4, for the pseudocode). The parameter d_0 is an upper bound for the distance d that protects the algorithm against large errors between exact and spline function values.

The approximation of the fitness function is based on spline interpolation of order less than or equal to four with the "not-a-knot" boundary condition. The "not-a-knot" condition is used when no information about the end point derivatives of an approximated function is known. The subroutine CUBSPL from the Fortran 90 version of Carl de Boor's cubic spline package was used to construct the spline approximations [10].

5. Local improvement

Local improvement is essentially an addition to improve the performance of the GA with spline interpolation. The value of the continuous variable at a given discrete node is either randomly assigned or obtained through a crossover operation. If an explicit spline approximation is available at a given node, it is possible to use a spline interpolant to generate a good candidate for the continuous variable for the next point at that node rather than depend on random action from the crossover operator. That is, after construction of an initial approximation $S(x)$ of $\Phi(v,x)$ based on the objective function values obtained at the design sites, one can easily find the point x^* that minimizes the spline function $S(x)$ in the range $[x_1, x_n]$. This optimal x^* value is stored at the discrete node in addition to the rest of the spline database. If, in future generations, a discrete node that has a stored x^* value is reached through the crossover operation on the discrete part v of the design, then, rather than performing crossover on the real part (v, x^*) is used as the child design for the next generation. This child design will then be treated like the other new designs in the child population and will be checked if an approximation to it can be used without exactly evaluating it.

6. Analysis

The optimum design of a honeycomb sandwich construction for strength and buckling constraints is treated in this paper. The core depth, which is a real variable, the ply orientation angles and the number of layers, which are discrete variables, are taken as the design variables. Consider a honeycomb sandwich plate with length a, width b, face ply thickness t_p, core depth t_h, core cell wall thickness t_c, and cell size diameter t_d. When such a structure is subjected to biaxial compression, there are several modes of sandwich plate failure. One obvious way is overstressing. The strength of faces is determined by the maximum strain criterion. The plate is assumed to fail if any of the ply strains, $\{\epsilon_1^i, \epsilon_2^i, \gamma_{12}^i\}_{i=1}^{n}$, exceeds its allowable value. The critical strength failure load factor is defined as

$$\lambda_s = \min_{1\leqslant i\leqslant n}\left[\min\left(\frac{\epsilon_1^*}{\epsilon_1^i}, \frac{\epsilon_2^*}{\epsilon_1^i}, \frac{\gamma_{12}^*}{\gamma_{12}^i}\right)\right], \tag{1}$$

where $\{\epsilon_1^*, \epsilon_2^*, \gamma_{12}^*\}$ are the ultimate allowable strains. The strength constraint is represented by

$$G_s = \lambda_s - 1.0 \geqslant 0. \tag{2}$$

Before overstressing, it is also possible that the structure can buckle. The reliability for the optimum design of the honeycomb sandwich construction is enhanced by adding several buckling constraints. Three modes of instability are considered as buckling constraints: overall buckling, face wrinkling, and face dimpling.

The overall buckling constraint for a honeycomb sandwich panel is represented by

$$G_b = \min\left[\frac{P_{cr}^*}{N_x}, \frac{P_{cr}^*}{N_y}\right] - 1.0 \geqslant 0, \tag{3}$$

where P_{cr}^* is the lowest value of the critical buckling load.

The computation of the buckling load is based on the mathematical model proposed by Hwu and Hu [11]. A closed-form solution of the buckling load was used for the composite sandwich plate consisting of crossply symmetric laminate faces with all edges simply supported:

$$P_{cr} = \frac{(1+\eta)P_0}{[1 + k(a/b)^2(n/m)^2](1 + \eta_x + \eta_y + \eta_0)}, \tag{4}$$

where

$$\eta = \frac{\pi^2 D^*}{t_h}\left[\frac{1}{G_{yz}}\left(\frac{m}{a}\right)^2 + \frac{1}{G_{xz}}\left(\frac{n}{b}\right)^2\right], \qquad \eta_0 = \frac{D^* P_0 m^2 n^2}{a^2 t_h^2 G_{xz} G_{yz}},$$

$$\eta_x = \frac{\pi^2}{t_h G_{xz}}\left[D_{11}\left(\frac{m}{a}\right)^2 + D_{66}\left(\frac{n}{b}\right)^2\right],$$

$$\eta_y = \frac{\pi^2}{t_h G_{yz}}\left[D_{66}\left(\frac{m}{a}\right)^2 + D_{22}\left(\frac{n}{b}\right)^2\right],$$

$$P_0 = \frac{\pi^2 a^2}{m^2}\left[D_{11}\left(\frac{m}{a}\right)^4 + 2(D_{12}+2D_{66})\left(\frac{mn}{ab}\right)^2 + D_{22}\left(\frac{n}{b}\right)^4\right],$$

$$D^* P_0 = \frac{\pi^2 a^2}{m^2}\left[D_{11}D_{66}\left(\frac{m}{a}\right)^4 + (D_{11}D_{22} - D_{12}(D_{12} + 2D_{66}))\left(\frac{mn}{ab}\right)^2 + D_{22}D_{66}\left(\frac{n}{b}\right)^4\right],$$

$k = N_y/N_x$ is the loading ratio; m and n are the number of buckling half-waves within the deformed surface; D_{ij} are the bending stiffnesses of the laminate with respect to the midsurface of the core; G_{xz} and G_{yz} are the transverse shear moduli of the core material in the $(x-z)$ and $(y-z)$ planes.

Face wrinkling may occur across many cells of the honeycomb core. The face wrinkling constraint can be expressed as

$$G_{fw} = \min\left[\frac{\bar{N}_x^{fw}}{N_x}, \frac{\bar{N}_y^{fw}}{N_y}\right] - 1.0 \geqslant 0, \tag{5}$$

where

$$\bar{N}_x^{fw} = \left[\left(\frac{1}{N_x(1+G_s)}\right)^3 + \left(\frac{1}{N_x^{fw}}\right)^3\right]^{-1/3},$$

$$\bar{N}_y^{fw} = \left[\left(\frac{1}{N_y(1+G_s)}\right)^3 + \left(\frac{1}{N_y^{fw}}\right)^3\right]^{-1/3},$$

$$N_x^{fw} = \left[\left(D_{11} - \frac{D_{12}^2}{D_{22}}\right)E_c G_{xz}\right]^{1/3},$$

$$N_y^{fw} = \left[\left(D_{22} - \frac{D_{12}^2}{D_{11}}\right)E_c G_{yz}\right]^{1/3}.$$

In honeycomb core sandwiches, a third type of instability, which is generally referred to as the face dimpling, may occur because the face sheet over one cell of the honeycomb can buckle as a small plate supported by the cell walls. The face dimpling constraint can be written as

$$G_{fd} = \min\left[\frac{\bar{N}_x^{fd}}{N_x}, \frac{\bar{N}_y^{fd}}{N_y}\right] - 1.0 \geqslant 0, \tag{6}$$

where

$$\bar{N}_x^{fd} = \left[\left(\frac{1}{N_x(1+G_s)}\right)^3 + \left(\frac{1}{N_x^{fd}}\right)^3\right]^{-1/3},$$

$$\bar{N}_y^{fd} = \left[\left(\frac{1}{N_y(1+G_s)}\right)^3 + \left(\frac{1}{N_y^{fd}}\right)^3\right]^{-1/3},$$

$$N_x^{fd} = \frac{9}{t_f^2}\left[D_{11} - \frac{D_{12}^2}{D_{22}}\right]\left(\frac{t_f}{t_d}\right)^{3/2},$$

$$N_y^{fd} = \frac{9}{t_f^2}\left[D_{22} - \frac{D_{12}^2}{D_{11}}\right]\left(\frac{t_f}{t_d}\right)^{3/2}.$$

7. Optimization problem

The optimization problem can be formulated as finding the stacking sequences of the face sheet and the core thickness in order to minimize the weight W of the panel. The set of design variables is expressed as a vector $\tilde{t} = (\theta_1, \ldots, \theta_n, t_h)$, where n is an implicit design variable dictated by the number of layers in the face sheet stacking sequence. The optimization problem with displacement and buckling constraints can be expressed as

$$\min_{\tilde{t}} W(\tilde{t}), \tag{7}$$

such that

$G_s(\tilde{t}) \geqslant 0$ (strength constraint),
$G_b(\tilde{t}) \geqslant 0$ (overall buckling),
$G_{fw}(\tilde{t}) \geqslant 0$ (face wrinkling),
$G_{fd}(\tilde{t}) \geqslant 0$ (face dimpling),
$t_h \in [t_h^l, t_h^u]$ (core thickness),
$\theta_i \in \{0°, \pm 45°, 90°\}, \quad i = 1, \ldots, n$ (ply angles),

where $W(\tilde{t}) = [(2t_f)\rho_f + t_h(8/3)(t_c/t_d)\rho_c](ab)$ is the structural weight of the plate; ρ_c and ρ_f are material densities of the core and face ply, respectively; $t_f = \sum_{i=1}^{n/2} t_{ply}^i$ is the total face thickness; t_h^l and t_h^u are lower and upper bounds of the core thickness, respectively; θ_i is the orientation of the ith ply; n is the total number of plies.

The critical constraint is defined as

$$G_{cr}(\tilde{t}) = \min\{G_s, G_b, G_{fw}, G_{fd}\}, \tag{8}$$

and the constrained optimization problem is transformed into an unconstrained maximization problem for the GA. This is done by using penalty parameters. The fitness function Φ to be maximized is defined as

$$\Phi(\tilde{t}) = \begin{cases} -W(\tilde{t}) + G_{cr}\delta, & G_{cr} \geqslant 0, \\ -W(\tilde{t})(1 - G_{cr})^p, & G_{cr} < 0, \end{cases} \tag{9}$$

where δ and p are bonus and penalty parameters, respectively.

8. Results

A T300/N5208 graphite-epoxy unidirectional ply is used for the laminate face construction. The core material is HFT fiberglass reinforced phenolic honeycomb. The material properties of the face and the core are listed in Table 1.

The panel has the longitudinal and lateral dimensions of $a = 20$ in. and $b = 10$ in., respectively. The allowed range of the core thickness is from 0.1 to 1.5 in. The GA was applied for one load case with $N_x = 8000.0$ lb/in. and $N_y = 0.0$ lb/in. The GA stopping condition is either a limit on the total number of function evaluations conducted by the standard GA, or when there is no change in the fitness function between generations, whichever occurs first. For all the results here (other than Table 2, for which a limit of 125,000 was used as the stopping condition), the limit was set large enough that it was never reached.

Table 1
Material properties

Face place	Core
$E_1 = 20.5 \times 10^6$ psi	$E_c = 23,000$ psi
$E_2 = 1.65 \times 10^6$ psi	$G_{xz} = 19,000$ psi
$G_{12} = 0.75 \times 10^6$ psi	$G_{yz} = 7500$ psi
$v_{12} = 0.32$	$t_d = 0.125$ in.
$t_{ply} = 0.006$ in.	$t_c = 1.0$ in.
$\rho_f = 0.0055$ lbs/in.^{3}	$\rho_c = 0.00174$ lbs/in.^{3}

Table 2
Optimal design

$\bar{n}_i$	t	v	Φ
125,000	1.0	1 1 1 1 1 1 1	−1.3616

Table 3
The efficiency of the binary tree

$\bar{n}_i$	$\bar{n}_e$	$\bar{n}_r$	$\bar{\xi}$ (%)	v	Φ
84,572	43,937	40,635	47.0	1 1 1 1 1 1 1	−1.3616

Table 4
Optimal design

$\bar{n}_i$	t	v	Φ
1,181,040	0.4220	1 1 1 1 1 1 1	−1.1882

8.1. Effect of binary tree

In order to demonstrate the efficiency that can be achieved through the use of a discrete memory, first consider the problem with the core thickness variable fixed to a constant value. The best known global optimal design for the specified load case obtained by the standard GA is presented in Table 2. The table gives the average number from ten runs of individuals ($\bar{n}_i$), the thickness (t), ply orientations (v), and fitness function value (Φ). This design was obtained with the constant core thickness, $t_h = 1.0$ in, in about 125,000 function evaluations.

The efficiency of the binary tree for the fitness function evaluation is shown in Table 3. This table shows the averages from ten runs for the number of individuals (total number of attempts to evaluate fitness function) (n_i), the number of nodes in the binary tree (n_e), the number of repeated designs (n_r), savings in terms of amount of fitness evaluations (ξ), ply orientations (v), and fitness function value (Φ). The chromosome length is 20. It is evident from this table that 40–50% of the analyses can be avoided by using the GA with memory. This reduction of the total number of analyses agrees well with the results from the previous studies of the GA with memory [6,7].

8.2. Effect of spline interpolation

The results of this experiment were obtained with the core thickness varying in the range [0.1, 1.5] in. The best known optimal design is presented in Table 4. The design was obtained using a standard GA with a large number of iterations requiring over a million function evaluations.

The performance of the GA with the spline interpolation is presented in Table 5, which shows averages from ten runs. Table 5 shows the best design after n_i attempts to evaluate the fitness function, n_s accepted spline values, and with $\epsilon = 0.005$ and $d_0 = 0.01$. The chromosome length is 7. The best known discrete vector v is $\{1111111\}$. The maximum absolute error δ_{max} and the mean absolute error δ_{av} are defined as

$$\delta_{max} = \max_{1 \leqslant i \leqslant n_s} |\Phi(x_i) - S(x_i)|,$$

$$\delta_{av} = \frac{1}{n_s} \sum_{i=1}^{n_s} |\Phi(x_i) - S(x_i)|, \tag{10}$$

where n_s is the total number of acceptable spline evaluations. These errors are computed every time that the

Table 5
The efficiency of the spline interpolation

$\bar{n}_i$	$\bar{n}_s$	$\bar{\xi}$ (%)	t	Φ	$\bar{\delta}_{max}$	$\bar{\delta}_{av}$
969,011	896,487	71.3	0.4220	−1.1882	0.11978	0.00006

algorithm decides to extract an approximation of the fitness value without an exact analysis. The data show the cost of the GA with a continuous variable could be reduced up to 70% relative to the standard GA by using the spline local approximation procedure. For the problem considered, the computation of the fitness function is not very expensive in terms of CPU time. However, this procedure has great potential in problems with expensive objective functions.

Table 6 shows he maximum absolute error δ_{max} and the mean absolute error δ_{av} for different values of the parameters ϵ and d_0. Table 7 shows the savings (ξ) in terms of number of fitness evaluations for different values of the parameters ϵ and d_0.

Table 6
The errors of the spline interpolation $(\delta_{max}, \delta_{av})$ as a function of the parameters ϵ and d_0

ϵ	d_0		
	0.005	0.01	0.02
0.005	0.03434, 0.00003	0.11978, 0.00006	0.20111, 0.00022
0.01	0.06142, 0.00004	0.09826, 0.00009	0.23898, 0.00024
0.02	0.04859, 0.00005	0.09676, 0.00012	0.32709, 0.00030
0.04	0.05555, 0.00012	0.10540, 0.00024	0.34188, 0.00042

Table 7
The percent savings (ξ) as a function of the parameters ϵ and d_0

ϵ	d_0		
	0.005	0.01	0.02
0.005	50.5	70.8	75.7
0.01	59.6	72.1	78.5
0.02	63.0	76.2	80.8
0.04	64.9	77.8	86.4

Table 6 shows the expected trends; δ_{av} decreases as either ϵ or d_0 decreases, but δ_{max} is essentially random, being determined early in the iterations. Note that the average approximation error δ_{av} is quite acceptable, several orders of magnitude less than ϵ and d_0. Table 7 shows that the saving are significant, but of course these percentages will be smaller for larger chromosomes.

8.3. Effect of local improvement

The fitness as a function of the natural logarithm of the number of exact fitness evaluations is plotted in Fig. 5. Since the analysis is a finite deterministic computation, involving no iterative algorithms, computation time corresponds exactly to number of function evaluations, which is shown in Fig. 5. If iterative processes were involved, then time would be more meaningful for the abscissa in the figure. It can be observed that the algorithm with local improvement converges faster in terms of number of fitness function evaluations than the one with spline approximation. The algorithm with only spline interpolation demonstrates good convergence in comparison with the standard GA, and noticeably decreases the number of exact analyses. After about 20,000 exact function evaluations the fitness values of the standard GA, the GA with spline interpolation, and the GA with local improvement are $\Phi = -1.2095$, $\Phi = -1.2031$, and $\Phi = -1.1882$, respectively.

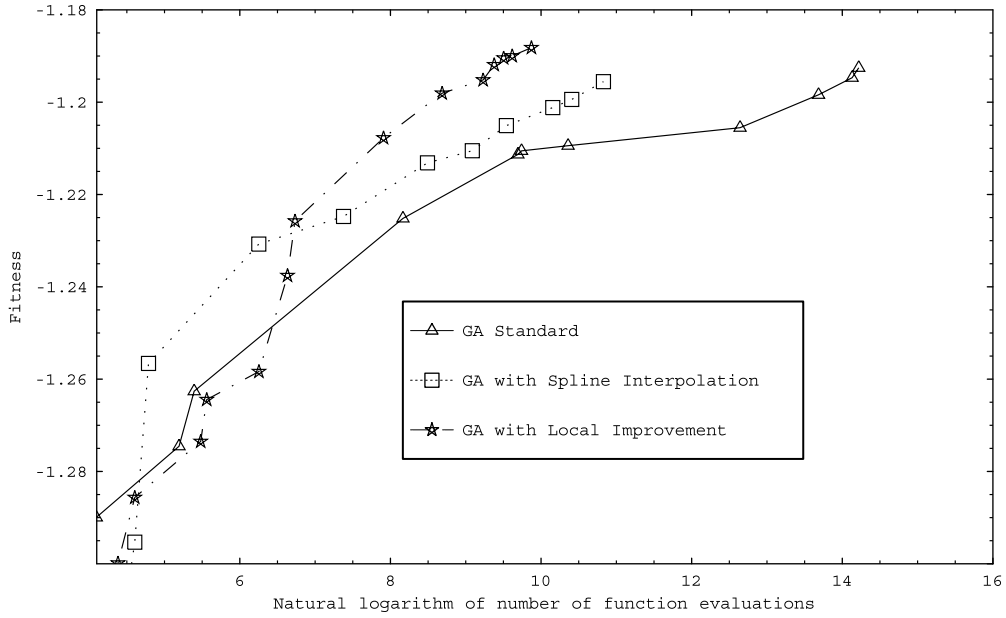


Fig. 5. Comparison of the performance (fitness vs. number of exact analyses).

9. Conclusions

A GA with memory along with spline interpolation was applied to the problem of weight minimization of a laminated sandwich composite panel with mixed discrete face sheet design variables and a continuous thickness design variable. The use of memory based on binary tree for discrete part of the design variables, and spline approximation for a continuous variable avoids repeating analyses of previously encountered designs. Moreover, it is also demonstrated that the spline approximation can be used to provide local improvement during the search and further reduce the number of exact function evaluations required to reach an improved solution. For FEM based analyses/simulations in which the computational cost is not trivial, the percent savings demonstrated in Tables 3, 5 and 7 will result in substantial computational savings. Even if less sophisticated analysis than FEM is used, the use of the binary tree will save time for all but extremely trivial function evaluations. Future work on this procedure might include development of a robust algorithm that is capable of handling several continuous variables using multivariate approximation techniques.

Acknowledgements

The authors are indebted to Christine Anderson-Cook for suggesting improvements in the approximation algorithm. This research was supported in part by Air Force Office of Scientific Research grant F49620-99-1-0128 and National Science Foundation grant DMS-9625968.

References

[1] Schmit LA, Farshi B. Optimum design of laminated fiber composite plates. International Journal of Numerical Methods in Engineering 1977;11:623–40.
[2] Haftka RT, Walsh JL. Stacking sequence optimization for buckling of laminated plates by integer programming. AIAA Journal 1992;3(3):814–9.
[3] Nagendra S, Haftka RT, Gürdal Z. Stacking sequence optimization of simply supported laminates with stability and strain constraints. AIAA Journal 1992;30(8):2132–7.
[4] LeRiche R, Haftka RT. Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm. AIAA Journal 1993;31(5):951–6.
[5] Seront G, Bersini H. A new GA-local search hybrid for optimization based on multilevel single linkage clustering. In: Proceedings Genetic and Evolutionary Computation Conference (GECCO-2000) Las Vegas, Nevada, 10–12 July 2000.
[6] Kogiso N, Watson LT, Gürdal Z, Haftka RT. Genetic algorithms with local improvement for composite laminate design. Structural Optimization 1994;7(4):207–18.
[7] Kogiso N, Watson LT, Gürdal Z, Haftka RT, Nagendra S. Design of composite laminates by a genetic algorithm with memory. Mechanics of Composite Materials and Structures 1994;1(1):95–117.
[8] McMahon MT, Watson LT, Soremekun GA, Gürdal Z, Haftka RT. A Fortran 90 genetic algorithm module for composite laminate structure design. Engineering with Computers 1998;14:260–73.
[9] Vowels RA. Algorithms and data structures in F and Fortran. Tucson, Arizona: Unicomp, Inc.; 1998.
[10] de Boor C. A practical guide to splines. Berlin: Springer-Verlag; 1978.
[11] Moh J, Hwu C. Optimization for buckling of composite sandwich plates. AIAA Journal 1997;35(5):863–8.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
PERGAMON
Computers and Structures 81 (2003) 2003–2009
Computers & Structures
www.elsevier.com/locate/compstruc

A genetic algorithm with memory for mixed discrete–continuous design optimization

Vladimir B. Gantovnik [a,*], Christine M. Anderson-Cook [b], Zafer Gürdal [c], Layne T. Watson [d]

[a] *Department of Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*
[b] *Department of Statistics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*
[c] *Departments of Aerospace and Ocean Engineering, and Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*
[d] *Departments of Computer Science, and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*

Received 3 October 2002; accepted 8 May 2003

Abstract

This paper describes a new approach for reducing the number of the fitness function evaluations required by a genetic algorithm (GA) for optimization problems with mixed continuous and discrete design variables. The proposed additions to the GA make the search more effective and rapidly improve the fitness value from generation to generation. The additions involve *memory* as a function of both discrete and continuous design variables, multivariate approximation of the fitness function in terms of several continuous design variables, and localized search based on the multivariate approximation. The approximation is demonstrated for the minimum weight design of a composite cylindrical shell with grid stiffeners.

Keywords: Optimization; Genetic algorithm; Response surface approximation; Composite structure

1. Introduction

Mixed discrete–continuous design optimization is an active research topic. There are many diverse applications that are mathematically modelled in terms of mixed discrete–continuous variables. The optimization of such models is typically difficult because of potential existence of multiple local minima in the search space. The most general methods for solving such problems are branch and bound method, simulated annealing (SA) method, and genetic algorithms (GA) [1]. These methods do not require gradient or Hessian information. However, to reach an optimal solution with a high degree of confidence, they typically require a large number of analyses during the optimization search. Performance of these methods is even more of an issue for problems that include continuous variables. The number of analyses required is an important characteristic of any method in multidisciplinary optimization. Several studies have concentrated on improving the reliability and efficiency of GAs. Hybrid algorithms formed by the combination of a GA with local search methods provide increased performance when compared to a GA with a discrete encoding of real numbers or local search alone [2]. In order to reduce the computational cost, two of the authors earlier used local improvements and memory for discrete problems so that information from previously analyzed design points is utilized in later searches [3,4]. In the first method a memory binary tree was employed for a composite panel design problem to store pertinent

* Corresponding author.
E-mail address: vlgantov@vt.edu (V.B. Gantovnik).

0045-7949/03/$ - see front matter
doi:10.1016/S0045-7949(03)00253-0

information about laminate designs that have already been analyzed [3]. After the creation of a new population of designs, the tree structure is searched for either a design with identical stacking sequence or similar performance, such as a laminate with identical in-plane strains. Depending on the kind of information that can be retrieved from the tree, the analysis for a given laminate may be significantly reduced or may not be required at all. The second method is called local improvement [4]. This technique was applied to the problem of maximizing the buckling load of a rectangular laminated composite plate. The information about previously analyzed designs is used to construct an approximation to buckling load in the neighborhood of each member of the population of designs. After that, the approximations are used to search for improved designs in small discrete spaces around nominal designs. These two methods demonstrated substantial improvements in computational efficiency for purely discrete optimization problems. The implementation, however, was not suitable for handling continuous design variables.

The objective of the present work is to find a suitable algorithm for a GA with memory that can work with discrete and several continuous variables simultaneously. A local memory for the continuous part of the design space at each discrete node of a binary tree based on multivariate approximation is proposed for problems with several continuous variables. The efficiency of the proposed multivariate approximation based procedure, as well as the use of memory for a GA that can handle continuous variables, are investigated for the weight optimization of a lattice shell with laminated composite skins subjected to axial compressive load. The composite shell design problem is used as a demonstration problem, rather than a synthetic constrained optimization problem. Results are generated to demonstrate the advantages of the proposed improvements to a standard GA.

2. Genetic algorithm package

A Fortran 90 GA framework that was designed in an earlier research effort was used for the composite laminate structure design [5]. This framework includes a module, encapsulating GA data structures, and a package of GA operators. The module and the package of operators result in what we call a standard GA. The proposed algorithm is incorporated within the GA framework as a sample test program that illustrates performance of the binary tree memory and multivariate approximation. An integer alphabet is used to code ply genes. The continuous variables represented by floating-point numbers had already been implemented in the GA framework data structure as geometry chromosomes.

3. Binary tree memory

A binary tree is a linked list structure in which each node may point to up to two other nodes. In a binary search tree, each left pointer points to nodes containing elements that are smaller than the element in the current node; each right pointer points to nodes containing elements that are greater than the element in the current node. The binary tree is used to store data pertinent to the design such as the design string and its associated fitness and constraint function values. A binary tree has several properties of great practical value, one of which is that the data can be retrieved, modified, and inserted relatively quickly. If the tree is perfectly balanced, the cost of inserting of an element in a tree with n nodes is proportional to $\log_2 n$ steps, and rebalancing the tree after an insertion may take as little as several steps, but at most takes $\log_2 n$ steps. Thus, the total time is of the order of $\log_2 n$ [6].

In the standard GA, a new population may contain designs that have already been encountered in the previous generations, especially towards the end of the optimization process. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. Algorithm 1 shows the pseudo-code of the fitness function evaluation with the aid of the binary tree.

After a new generation of designs is created by the genetic operations, the binary tree is searched for each new design. If the design is found, the fitness value is retrieved from the binary tree without conducting an analysis. Otherwise, the fitness is obtained based on an exact analysis. This new design and its fitness value are then inserted in the tree as a new node.

Algorithm 1. Evaluation of fitness function using binary tree.

```
search for the given design in the binary tree;
if found then
   get the fitness function value from the binary tree;
else
   perform exact analysis;
endif
```

4. Response surface approximation

The procedure described above works well for purely discrete optimization problems where designs are completely described by discrete strings. In case of mixed optimization problems where designs include discrete and continuous variables, the solution becomes more complicated. If the continuous variables are also discretized into a fine discrete set, the possibility of creating

a child design that has the same discrete and continuous parts as one of the earlier designs diminishes substantially. In the worst case, if the continuous design variables are represented as a real numbers, which is the approach used by most recent research work, it may not be possible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless, and result in exact analysis even if the real part of the new child is different from one of the earlier designs by a minute amount.

The main idea of the *memory* approach proposed in this work is to construct a response surface approximation for the fitness function as a function of the continuous variables using historical data values, and estimate from the stored data whenever appropriate. The *memory* in this case consists of two parts: a binary tree, which consists of the nodes that have different discrete parts of the design, and a storage part at each node that keeps the continuous values and their associated fitness values. That is, each node contains a real array that stores the continuous variable points' value and their corresponding fitness function values. In order for the *memory* to be functional, it is necessary to have accumulated a sufficient number of designs with different continuous values for a particular discrete design point so that the approximation can be constructed. Naturally, some of the discrete nodes will not have more than a few designs with different continuous values. However, it is possible that as the evolution progresses good discrete parts will start appearing repeatedly with different continuous values. In this case, one will be able to construct a good quality response surface approximation to the data.

The response surface approximation approach is an extension of the previous work by the authors where a spline-based approach was used for only one continuous variable [7]. An evolving database of continuous variable points is used in the current work to construct a multivariate response surface approximation at those discrete nodes that are processed frequently. The modified quadratic Shepard's method is a local smoothing method used for the approximation of scattered data for the case of two independent continuous design variables [8]. This method may be the best known among all scattered data interpolants for a general number of variables. Shepard's method for fitting a surface to data values has the advantage of small storage requirements and easy generalization to more than two independent variables.

In addition to building a multivariate approximation, it is important to assess accuracy of the multivariate approximation at new continuous points, so that a decision may be made either to accept the approximation or perform exact function evaluation. Based on the bivariate approximation, the proposed algorithm described by the following pseudo-code is then used to decide when to retrieve the fitness function value from the approximation, and when to do an exact analysis and add the new data point to the approximation database. For the description of the pseudo-code, let $v \in Z^k$ be a k-dimensional integer design vector for the discrete space, $x^{(i)} \in E^m$ a real m-dimensional design vector for the continuous variables, and $f(v, x^{(i)})$ the corresponding fitness value of the individual defined by $(v, x^{(i)})$. Furthermore, define $d_i \in E$ to be a real distance corresponding to point $x^{(i)}$ to measure its proximity any given point, $c \in Z$ an integer counter, initially zero, and $r \in E$ a real range value. Let T be the set of observed exact analyses and their corresponding information within a given discrete node, precisely

$$T = \left\{(x^{(i)}, f(v, x^{(i)}), d_i)\right\}_{i=1}^{n}.$$

Each node in the binary tree memory structure records a tuple of the form (v, T, c, r). The pseudo-code for processing a candidate individual (v, x) is defined by Algorithm 2.

The algorithm uses three real user-specified parameters, d_0, δ, and ϵ. The parameter $d_0 > 0$ is an upper bound on the trust region radius about each sample point $x^{(i)}$. The parameter δ is chosen to satisfy $0 < \delta < 1$, and in higher dimensions protects against large variations in f in unsampled directions. Finally, the parameter $\epsilon > 0$ is the selected acceptable approximation accuracy, and is solely based on engineering considerations.

5. Local improvement

Once a multivariate approximation is in place at a given discrete node, a local improvement procedure may be implemented to improve the performance of the GA. The values of the continuous variables at a given discrete node are either randomly assigned (if mutation operator is used) or obtained through a crossover operation. If an explicit multivariate approximation and its first partial derivatives are available at a given node, it is possible to generate good candidates for the continuous design variables for the next child at that node rather than depend on random action from the crossover operator. That is, after construction of an initial approximation $S(x)$ of $f(x)$ based on the objective function values obtained at the design sites, one can easily find the point x^* that optimizes the approximate function $S(x)$ in some compact subset $\Omega \subset E^m$. This optimal x^* value is stored at the discrete node in addition to the rest of the T database. If, in future generations, a discrete node that has a stored x^* value is reached through the crossover operation on the discrete part v of the design, then, rather than performing crossover on the real part, (v, x^*) is used as the child design for the next generation. This

child design will then be treated like the other new designs in the child population and will be checked if an approximation to it can be used without exact analysis.

Algorithm 2. Evaluation of fitness function using binary tree and m-dimensional approximation.

if v is not found in the tree **then**
  evaluate $f(v,x)$;
  $T := \{(x, f(v,x), 0)\}$;
  add a node corresponding to $(v, T, 1, 0)$;
  return $f(v,x)$;
else
  if $c < (m+2)(m+1)$ **then**
    evaluate $f(v,x)$;
    $T := T \cup \{(x, f(v,x), 0)\}$;
    $c := c + 1$;
    $r := \max_{t\in T} t_2 - \min_{t\in T} t_2$;
    return $f(v,x)$;
  else
    construct an approximation $S(x)$ using
    the data in $T = \{(x^{(i)}, f(v,x^{(i)}), d_i)\}_{i=1}^{n}$;
    define k and d^* by $d^* = d_k - \|x - x^{(k)}\| = \max_{1\leqslant i\leqslant n} d_i - \|x - x^{(i)}\|$;
    if $d^* \geqslant 0$ and $|f(v,x^{(k)}) - S(x)| < \delta r$ **then**
    return $S(x)$;
    else
    evaluate $f(v,x)$;
    if $|f(v,x) - S(x)| > \epsilon$ **then**
    $T := T \cup \{(x, f(v,x), 0)\}$;
    else
    $d := \min\{d_0, \|x - x^{(k)}\|\}$; $d_k := d$;
    $T := T \cup \{(x, f(v,x), d)\}$;
    end if
    $r := \max_{t\in T} t_2 - \min_{t\in T} t_2$;
    return $f(v,x)$;
    end if
  end if
end if

6. Design optimization problem

The design of a lattice shell with specified radius, length, and axial load level is used to demonstrated the improvements introduced into the standard GA. Such shells supported by a lattice have been considered as a replacement to solid shells, stiffened shells and honeycomb structures [9–11]. Consider a lattice cylindrical shell loaded with compressive axial force P. Proper design would involve, in general, determination of rib parameters (dimension of cross-section, material, spacing, and orientation angle, φ), skin parameters (the number of layers, their materials, thicknesses, and orientation angles, θ_k) that satisfy strength and stability constraints while minimizing the weight of the shell. Constraints considered include rib strength constraint (C_1), skin strength constraint (C_2), rib local buckling constraint (C_3), and general buckling constraint (C_4). All constraint equations are based on a simplified lattice cylindrical shell model developed by Bunakov [12–14].

The mixed optimization problem considered here operates on three design variables v, x_1, and x_2. The discrete variable is the stacking sequence of the skins, $v = \{\theta_1, \ldots, \theta_n\}$, where n is an implicit design variable dictated by the number of layers in the skin stacking sequence. We shell restrict our consideration to two continuous design variables, namely, the angle of helical ribs, $x_1 = \varphi$, and the rib height, $x_2 = H$. The optimization problem can be formulated as finding the stacking sequences of the skins, the angle of helical ribs, and the rib height in order to minimize the mass of the shell, M. The set of design variables is expressed as a vector $\tau = (v, x_1, x_2)$. The design problem is typically formulated to provide a minimum mass structure:

$$M = 4\pi\rho L\left[h\left(R + h + \frac{H}{2}\right) + H\frac{\delta}{a}(R + h)\right], \tag{1}$$

where ρ is the material density, L is the length of the shell, R is the shell radius, $h = \sum_{k=1}^{n} h_0^{(k)}$ is the skin thickness, h_0 is the single ply thickness, δ is the rib width, a is the rib spacing. The optimization problem can be written as

$$\min_{\tau} M(\tau) \tag{2}$$

such that

$$\begin{aligned}
&C_1(\tau) \geqslant 0 \quad \text{(rib strength)},\\
&C_2(\tau) \geqslant 0 \quad \text{(skin strength)},\\
&C_3(\tau) \geqslant 0 \quad \text{(rib local buckling)},\\
&C_4(\tau) \geqslant 0 \quad \text{(general buckling)},\\
&H \in [H_{\min}, H_{\max}],\\
&\varphi \in [\varphi_{\min}, \varphi_{\max}],\\
&\theta_k \in \{0^\circ, \pm 45^\circ, 90^\circ\} \quad (k = 1, n),\\
&n \in [n_{\min}, n_{\max}],
\end{aligned}$$

where $H_{\min}$ and $H_{\max}$ are lower and upper bounds of the rib height; $\varphi_{\min}$ and $\varphi_{\max}$ are are lower and upper bounds of the angle of helical ribs, θ_k is the ply orientation angle in the kth skin ply, n is the total number of skin plies, $n_{\min}$ and $n_{\max}$ are minimum and maximum possible values of n. The above problem may not be a realistic composite design formulation. A standard laminate optimization typically includes additional constraints such as ply contiguity, interlaminar stress, core strength, etc.

The constrained optimization problem is transformed into an unconstrained maximization problem using a

Table 1
The material properties of the skin (T300/5208)

Stiffness parameters, GPa				Strength parameters, MPa				
E_1	E_2	G_{12}	v_{12}	X_t	Y_t	X_c	Y_c	S
181.0	10.3	7.17	0.28	1500.0	57.0	1340.0	212.0	68.0

penalty function approach. The critical constraint is defined as

$$C_{cr}(\tau) = \min_{i=1,4}\{C_i(\tau)\}. \tag{3}$$

The fitness function Φ to be maximized is defined as

$$\Phi(\tau) = \begin{cases} -M(\tau) + C_{cr}(\tau)q, & C_{cr}(\tau) \geqslant 0, \\ -M(\tau)(1 - C_{cr}(\tau))^p, & C_{cr}(\tau) < 0, \end{cases} \tag{4}$$

where q and p are bonus and penalty parameters, respectively.

7. Results

A cylindrical lattice shell is made of fiberglass-epoxy composite material with density $\rho = 2100$ kg/m^3. The specified axial compressive load is $P = 10^6$ N/m. The shell radius and length are $R = 1.0$ m and $L = 1.5$ m, respectively. The lattice shell has $\pm\varphi$ unidirectional helical ribs with elastic modulus $E = 45.0$ GPa, and shear modulus $G = 1.0$ GPa. The compressive strength of ribs is $\bar{\sigma}^r = 240.0$ MPa. The shell has external and internal skins made of T300/5208 graphite-epoxy unidirectional plies. The material properties of the skin plies are given in Table 1. The possible ranges for the design variables are given in Table 2. The range of the shell mass throughout the entire design space is approximately $31.12 \leqslant M \leqslant 303.74$ kg.

7.1. Genetic algorithm parameters

The values of the GA parameters used in the experiments are shown in Table 3. The GA stopping condition is a limit on the total number of function evaluations conducted by the standard GA, $(n_i)_{max} = 500{,}000$. The best known global optimal design obtained by the standard GA is presented in Table 4. The table gives the average number of exact analyses from 10 runs of individuals ($\bar{n}_e$), the continuous design variables (x_1, x_2), the discrete design variable (v), the critical constraint value (C_{cr}), the mass (M), and fitness function value (Φ). This design was obtained in an average of about 274,545 function evaluations by the standard GA. Table 5 shows reliability (fraction of runs in which the optimum was found, out of 50 runs) for various geometry chromosome mutation probabilities p_m and population sizes. The conclusion from the table indicates that

Table 2
Ranges for the design variables

Design variable	Range
$H \in [H_{min}, H_{max}]$	$[0.001, 0.1]$ m
$\varphi \in [\varphi_{min}, \varphi_{max}]$	$[5°, 85°]$
$\theta_k,\ k = 1, n$	$\{0°, \pm 45°, 90°\}$
$n \in [n_{min}, n_{max}]$	$[2, 20]$

Table 3
GA parameters used in the experiments

Parameter	Value
Maximal number of generations	25,000
Population size	20
Laminate chromosome length	7
Probability of crossover (p_c):	
for laminate chromosomes	1.0
for geometry chromosomes	1.0
Probability of mutation (p_m):	
for laminate chromosomes	0.05
for geometry chromosomes	0.01
Crossover type:	
for laminate chromosomes	Two-point
for geometry chromosomes	One-point

that the GA works better with the large population size along with very small mutation probability. A small population size causes the GA to quickly converge on a local minimum, because it insufficiently samples the parameter space. Note that the results given in Table 4 and in the rest of study provided below were obtained with the population size of 20 and with geometry chromosome mutation probability of 0.01.

7.2. Effect of continuous memory

The results presented in this section focus on the ability of the proposed algorithm to save computational time during GA optimization with the multivariate approximation used as a memory device. The performance of the GA with the multivariate approximation is presented in Table 6, which shows averages from 10 runs. The table shows the best design (x_1, x_2, v, Φ) after n_i attempts to evaluate the fitness function, of which n_e are the number of exact fitness function evaluations with $\epsilon = 0.01$, $\delta = 0.1$, $d_0 = 0.5$. In addition, Table 6 contains the average percent savings ($\bar{\xi}$) in terms of fitness

Table 4
The best known optimal design using standard GA

$\bar{n}_e$	x_1	x_2	v	C_{cr}	M	Φ
274,545	1.0000	0.0052	1,111,100	0.0	43.8228	−0.1443

Table 5
The percent reliability for various geometry chromosome mutation probabilities p_m and population sizes

p_m	Population size		
	20	50	100
>0.5	0.00	0.00	0.00
0.5	0.04	0.02	0.00
0.4	0.10	0.00	0.02
0.3	0.14	0.20	0.00
0.2	0.46	0.38	0.14
0.1	0.60	0.78	0.52
0.01	0.52	0.72	0.96
0.001	0.36	0.78	0.94
0.0	0.28	0.44	0.96

function evaluations, mass (M) and critical constraint (C_{cr}) corresponding to the best design which are identical to the results presented earlier for the baseline algorithm. The percent savings (ξ) in terms of number of fitness function evaluations is defined by

$$\xi = \left(1 - \frac{n_e}{n_i}\right) \times 100\%. \tag{5}$$

The influence of the number of generations on the function evaluation savings are shown in Table 7. This table contains average values for fitness function ($\bar{\Phi}$), savings ($\bar{\xi}$), and reliability (fraction of runs the optimum was found, out of 20 runs). The number of successful runs and the percent savings ($\bar{\xi}$) increase when the algorithm is run for a longer time, as expected.

These results show that the cost of the GA with continuous variables could be reduced up to 60% relative to the standard GA by using the approximation procedure. For the problem considered, the computation of the fitness function is not very expensive in terms of CPU time. However, this procedure has great potential in problems with expensive objective functions.

The mean absolute error $\bar{\varepsilon}$ due to the approximation and the savings ($\bar{\xi}$) in terms of the number of fitness evaluations for different values of the parameters ϵ, and δ with $d_0 = 0.5$ are shown in Table 8. The mean absolute error $\bar{\varepsilon}$ is defined as

$$\bar{\varepsilon} = \frac{1}{n_s} \sum_{i=1}^{n_s} |\Phi(x_i) - S(x_i)|, \tag{6}$$

where $n_s = n_i - n_e$ is the total number of acceptable approximate evaluations. This error is computed every time that the algorithm decides to extract an approximation of the fitness value without an exact analysis. It is possible to further enhance the performance of the algorithm by a more precise tuning of its parameters. Table 8 shows the expected trends; both average savings ($\bar{\xi}$) and average absolute error $\bar{\varepsilon}$ increase as either ϵ or δ increases. Table 8 also shows that the saving are significant, but of course these percentages are likely to be smaller for longer chromosomes that results in a larger design space.

Table 7
The performance of the GA with multivariate approximation for different number of generations

Number of generations	Reliability	$\bar{\Phi}$	$\bar{\xi}$ (%)
5000	0.1	−0.1508	42.8
10000	0.4	−0.1464	49.9
20000	0.9	−0.1444	53.3

Table 8
The error of the multivariate approximation ($\bar{\varepsilon}$) and the percent savings ($\bar{\xi}$) as a function of the parameters ϵ and δ with $d_0 = 0.5$

ϵ	δ	$\bar{\varepsilon}$	$\bar{\xi}$ (%)
0.001	0.1	0.00097	47.9
	0.5	0.00128	50.5
	1.0	0.00445	52.2
0.005	0.1	0.00185	55.6
	0.5	0.01440	56.5
	1.0	0.03040	58.3
0.01	0.1	0.02490	59.4
	0.5	0.03310	62.0
	1.0	0.04170	66.8

Table 6
The efficiency of the multivariate approximation

$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$ (%)	x_1	x_2	v	C_{cr}	M	Φ
183,995	74,626	59.4	1.0000	0.0052	1,111,100	0.0	43.8228	−0.1443

Table 9
The performance comparison of the GA modifications

Modification	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$ (%)	$\bar{\zeta}$ (%)
Standard	274,545	274,545	0.0	0.0
With approximation	183,995	74,626	59.4	72.8
With approximation and local improvement	53,690	38,280	28.7	86.1

For the above results, Algorithm 2 used the test $c < 20$ rather than $c < (m+2)(m+1) = 12$, because of constraints in the Shepard algorithm code from [8].

7.3. Effect of local improvement

Finally, the performance comparison of the considered GA modifications, namely, (1) standard GA, (2) GA with approximation, and (3) GA with approximation and local improvement are presented in Table 9. The average number of attempts to evaluate the fitness function $\bar{n}_i$, the number of exact fitness function evaluations $\bar{n}_e$, the percent savings $\bar{\xi}$ in terms of number of exact fitness evaluations, and the percent savings $\bar{\zeta}$ defined as

$$\bar{\zeta} = \left(1 - \frac{\bar{n}_e}{\bar{n}_i(\text{Standard GA})}\right) \times 100\% \tag{7}$$

in terms of number of exact fitness evaluations as compared with the standard GA are shown in the table.

As one would expect, the GA with local improvement converges faster in terms of number of fitness function evaluations than the GA with approximation. The two algorithms with approximation demonstrate good convergence in comparison with the standard GA, and very substantial decrease in the number of exact analyses required to find the optimal solution.

8. Conclusions

A GA with memory along with multivariate approximation was applied to the problem of weight minimization of a lattice shell with mixed discrete design variable and two continuous design variables. The use of memory based on binary tree for discrete part of the design variables avoids repeating analyses of previously encountered designs. The multivariate approximation for continuous variables saves unnecessary exact analyses for points close to previous values. Moreover, it is also demonstrated that the multivariate approximation can be used to provide local improvement during the search and further reduce the number of exact function evaluations required to reach an improved solution.

Acknowledgements

The authors are indebted to Dr. Samy Missoum for suggesting improvements in the multivariate approximation algorithm. This research was supported in part by Air Force Office of Scientific Research grant F49620-99-1-0128 and National Science Foundation grant DMS-9625968.

References

[1] Stelmack MA, Nakashima N, Batill SM. Genetic algorithms for mixed discrete–continuous optimization in multidisciplinary design. 38th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, AIAA Paper No. 1998-2033. Long Beach, CA; 1998.
[2] Seront G, Bersini H. A new GA-local search hybrid for optimization based on multi level single linkage clustering. Genetic and Evolutionary Computation Conference (GECCO-2000). Las Vegas, NV; 2000.
[3] Kogiso N, Watson LT, Gürdal Z, Haftka RT. Genetic algorithms with local improvement for composite laminate design. Struct Optimizat 1994;7(4):207–18.
[4] Kogiso N, Watson LT, Gürdal Z, Haftka RT, Nagendra S. Design of composite laminates by a genetic algorithm with memory. Mech Compos Mater Struct 1994;1(1):95–117.
[5] McMahon MT, Watson LT, Soremekun GA, Gürdal Z, Haftka RT. A Fortran 90 genetic algorithm module for composite laminate structure design. Eng Comput 1998;14:260–73.
[6] Vowels RA. Algorithms and data structures in F and Fortran. Tucson, Arizona: Unicomp, Inc; 1998.
[7] Gantovnik VB, Gürdal Z, Watson LT. A genetic algorithm with memory for optimal design of laminated sandwich composite panels. 43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA Paper No. 2002-1221. Denver, CO; 2002.
[8] Renka RJ. Multivariate interpolation of large sets of scattered data. ACM Trans Math Software 1988;14(2):139–48.
[9] Vasiliev VV, Lopatin AV. Theory of lattice and stiffened composite shells. In: Tarnopolskii YM, editor. Mechanics of composite materials. Riga: Zinatne; 1992. p. 82–8 [in Russian].
[10] Vasiliev VV, Barynin VA, Rasin AF. Anisogrid lattice structures—survey of development and application. Compos Struct 2001;54:361–70.
[11] Slinchenko D, Verijenko VE. Structural analysis of composite lattice shells of revolution on the basis of smearing stiffness. Compos Struct 2001;54:341–8.
[12] Bunakov VA, Protasov VD. Cylindrical lattice composite shells. Mech Compos Mater 1989;6:1046–53 [in Russian].
[13] Belousov PS, Bunakov VA. Bending of cylindrical lattice composite shells. Mech Compos Mater 1992;2:225–31 [in Russian].
[14] Bunakov VA. Design of axially compressed composite cylindrical shells with lattice stiffeners. In: Vasiliev VV, Gürdal Z, editors. Optimal design. Lancaster, PA: Technomic Publishing Co.; 1999. p. 207–46.

AIAA JOURNAL
Vol. 43, No. 8, August 2005

Genetic Algorithm for Mixed Integer Nonlinear Programming Problems Using Separate Constraint Approximations

Vladimir B. Gantovnik,* Zafer Gürdal,† Layne T. Watson,‡ and Christine M. Anderson-Cook§
Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

A new approach is described for reducing the number of the fitness and constraint function evaluations required by a genetic algorithm (GA) for optimization problems with mixed continuous and discrete design variables. The proposed additions to the GA make the search more effective and rapidly improve the fitness value from generation to generation. The additions involve memory as a function of both discrete and continuous design variables and multivariate approximation of the individual functions' responses in terms of several continuous design variables. The approximation is demonstrated for the minimum weight design of a composite cylindrical shell with grid stiffeners.

Nomenclature

c	=	counter
D	=	database at a given tree node
d	=	trust region radius, subscripted
E	=	real numbers
$f(v, x)$	=	fitness function
$g_j(v, x)$	=	constraint functions, $1 \leq j \leq p$
$g_0(v, x)$	=	objective function
n_e	=	number of exact analyses
n_i	=	number of requests for constraint evaluations
r	=	variation measure
$\boldsymbol{v}$	=	integer design vector of dimension k
$\boldsymbol{x}$	=	real design vector of dimension m
Z	=	integers
δ, ϵ	=	tolerance parameters
ζ	=	percent savings over standard genetic algorithm
ξ	=	percent savings due to approximations only

I. Introduction

MANY diverse applications are mathematically modeled in terms of mixed discrete–continuous variables. The optimization of such models is typically difficult due to their combinatorial nature and potential existence of multiple local minima in the search space. The engineering problems that contain integer, discrete, zero–one, and continuous design variables are often referred to as mixed integer nonlinear programming (MINLP) problems.

Genetic algorithms (GAs) are powerful tools for solving MINLP problems. These methods do not require gradient or Hessian information. However, to reach an optimal solution with a high degree of confidence, they typically require a large number of analyses during the optimization search. Performance of these methods is even more of an issue for problems that include continuous variables. Several studies have concentrated on improving the reliability and efficiency of GAs. Hybrid algorithms formed by the combination of a GA with local search methods provide increased performance when compared to a GA with a discrete encoding of real numbers or local search alone.[1]

Although GAs are robust global optimizers, they typically require a very large number of fitness function evaluations. Moreover, it is commonly observed that fitness values are frequently recalculated for some designs that appear repeatedly during the evolution of the population. This suggests an opportunity for performance improvement. To reduce the computational cost, the authors earlier used local improvements and memory for discrete problems so that information from previously analyzed design points is stored and utilized in later searches.[2,3] In the first approach, a memory binary tree was employed for a composite panel design problem to store pertinent information about laminate designs that have already been analyzed.[2] After the creation of a new population of designs, the tree structure is searched for either a design with identical stacking sequence or similar performance, such as a laminate with identical in-plane strains. Depending on the kind of information that can be retrieved from the tree, the analysis for a given laminate may be significantly reduced or may not be required at all. The second method is called local improvement.[3] This technique was applied to the problem of maximizing the buckling load of a rectangular laminated composite plate. The information about previously analyzed designs is used to construct an approximation to buckling load in the neighborhood of each member of the population of designs. After that, the approximations are used to search for improved designs in small discrete spaces around nominal designs. These two methods demonstrated substantial improvements in computational efficiency for purely discrete optimization problems. The implementation, however, was not suitable for handling continuous design variables.

New approaches have been proposed to overcome this shortcoming. In particular, a new version of GA has been recently developed,[4] consisting of memory as a function of both discrete and continuous design variables using spline[5] and multivariate[6] approximations of the constraint functions in terms of continuous design variables.

The work here proposes to enhance the efficiency and accuracy of the GA with memory using multivariate approximations of the objective and constraint functions individually instead of direct approximations of the overall fitness function. The primary motivation for the proposed improvements is the nature of the fitness function in constrained engineering design optimization problems. Because GAs are algorithms for unconstrained optimization, constraints are typically incorporated into the problem formulation by augmenting the objective function of the original problem with penalty terms associated with individual constraint violations. The resulting fitness function is usually highly nonlinear and discontinuous, which makes the multivariate approximation highly inaccurate unless a large number of exact function evaluations are performed. Because the individual response functions in many engineering problems are mostly smooth functions of the continuous variables (although they can be highly nonlinear), high-quality approximations to individual

Received 29 July 2003; revision received 21 March 2005; accepted for publication 4 April 2005.

*Graduate Research Assistant, Department of Engineering Science and Mechanics. Member AIAA.
†Professor, Departments of Aerospace and Ocean Engineering and Engineering Science and Mechanics; currently Aerospace Structures Chair, Delft University of Technology, 2629 HS Delft, The Netherlands. Associate Fellow AIAA.
‡Professor, Departments of Computer Science and Mathematics.
§Associate Professor, Department of Statistics.

functions can be constructed without requiring a large number of function evaluations. The proposed modification is, therefore, expected to improve the efficiency of the memory constructed in terms of the continuous variables. The paper presents the algorithmic implementation of the proposed memory scheme and demonstrates the efficiency of the proposed multivariate approximation procedure for the weight optimization of a lattice shell with laminated composite skins subjected to axial compressive load. The composite shell design problem is used as a demonstration problem, instead of a synthetic constrained optimization problem. Results are generated to demonstrate the advantages of the proposed improvements to a standard GA.

II. GA Package

A FORTRAN 90 GA framework that was designed in an earlier research effort was used for the composite laminate structure design.[7] This framework includes a module, encapsulating GA data structures, and a package of GA operators. The module and the package of operators result in what we call a standard GA. The proposed algorithm is incorporated within the GA framework to illustrate performance of the binary tree memory and multivariate approximation. An integer alphabet is used to code ply genes. The continuous variables represented by floating-point numbers had already been implemented in the GA framework data structure as geometry chromosomes.

III. Binary Tree Memory

A binary tree is a linked list structure in which each node may point to up to two other nodes. In a binary search tree, each left pointer points to nodes containing elements that are smaller than the element in the current node; each right pointer points to nodes containing elements that are greater than the element in the current node. The binary tree is used to store data pertinent to the design such as the design string and its associated fitness and constraint function values. A binary tree has several properties of great practical value, one of which is that the data can be retrieved, modified, and inserted relatively quickly. If the tree is perfectly balanced, the cost of inserting an element in a tree with n nodes is proportional to $\log_2 n$ steps, and rebalancing the tree after an insertion may take as little as several steps, but at most takes $\log_2 n$ steps. Thus, the total time is of the order of $\log_2 n$ (Ref. 8).

When the mechanisms of the GA operators are examined, it is observed that the diversity of a population trends to decrease as the algorithm runs longer. The fitness values for the same chromosomes are recalculated repeatedly, especially toward the end of the optimization process. If previously calculated fitness values can be efficiently saved and retrieved, computation time will decrease significantly. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. Algorithm 1 shows the pseudocode of the fitness function evaluation with the aid of the binary tree. After a new generation of designs is created by the genetic operations, the binary tree is searched for each new design. If the design is found, the fitness value is retrieved from the binary tree without conducting an analysis. Otherwise, the fitness is obtained based on an exact analysis. This new design and its fitness value are then inserted in the tree as a new node. The major improvement proposed here is to store not just the fitness value but the values of every function that can contribute to the computation of the fitness function.

Algorithm 1; Evaluation of fitness function using binary tree:
search for the given design in the binary tree;
if found **then**
  get the fitness function value from the binary tree;
else
  perform exact analysis;
end if

IV. Response Surface Approximations

The procedure just described works well for purely discrete optimization problems where designs are completely described by discrete strings. In the case of mixed optimization problems where designs include discrete and continuous variables, the solution becomes more complicated. If the continuous variables are also discretized into a fine discrete set, the possibility of creating a child design that has the same discrete and continuous parts as one of the earlier designs diminishes substantially. In the worst case, if the continuous design variables are represented as real numbers, which is the approach used in most recent research work, it may not be possible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless, and result in many exact analyses even if the real part of the new child is different from one of the earlier designs by a minute amount.

The main idea of the memory approach for problems with mixed discrete–continuous variables is to construct a response surface approximation for every constraint function as a function of the continuous variables using historical data values and to estimate from the stored data whenever appropriate. The memory in this case consists of two parts: a binary tree, which consists of the nodes that have different discrete parts of the design, and a storage part at each node that keeps the continuous values and the corresponding constraint functions' values. That is, each node contains several real arrays that store the continuous variables' values and their corresponding constraint functions' values. For the memory to be functional, it is necessary to have accumulated a sufficient number of designs with different continuous values for a particular discrete design point so that the approximations can be constructed. Naturally, some of the discrete nodes will not have more than a few designs with different continuous values. However, it is possible that as the GA search progresses, promising discrete parts will start appearing repeatedly with different continuous values. In this case, one will be able to construct good quality response surface approximations to the data.

The response surface approximation approach is an extension of the previous work by the authors in which a spline-based approach was used for only one continuous variable.[5] An evolving database of continuous variable points is used in the current work to construct multivariate response surface approximations at those discrete nodes that are processed frequently. The modified quadratic Shepard method is a local smoothing method used for the approximation of scattered data for the cases of two and three independent continuous design variables (see Refs. 9–11). It has been suggested that the modified quadratic Shepard method overcomes the drawbacks of a well-known interpolation scheme given by Shepard.[12] This method may be the best known among all scattered data interpolants for a general number of variables and has the advantage of numerical efficiency, stability, small storage requirements, and easy generalization to more than two independent variables. It, therefore, seems to be the most suitable candidate for handling a very large amount of data and for use in the case of a high number of independent variables.

In addition to building the multivariate approximations, it is important to assess accuracy of the multivariate approximation at new continuous points that have not been encountered before, so that a decision may be made either to accept the approximation or perform exact function evaluation. Based on the multivariate approximation, the proposed algorithm described by the following pseudocode is then used to decide when to retrieve the constraint function values from the approximations and when to do an exact analysis and add the new data point to the approximation database.

For the description of the pseudocode, let $\boldsymbol{v} \in Z^k$ be a k-dimensional integer design vector for the discrete space, $\boldsymbol{x} \in E^m$ a real m-dimensional design vector for the continuous variables, $g_0(\boldsymbol{v}, \boldsymbol{x})$ the corresponding objective function, $g_1(\boldsymbol{v}, \boldsymbol{x}), \ldots, g_p(\boldsymbol{v}, \boldsymbol{x})$ the corresponding constraint functions, and $f(\boldsymbol{v}, \boldsymbol{x})$ the corresponding fitness value of the individual defined in terms of the constraint functions and the objective function.

Define the symbol $d \in E$ with subscripts to be a real distance corresponding to a trust region radius about a specific point in the database. Let $D = (\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_p)$ contain the set of n observed exact analysis points and their corresponding information within a given discrete node, where

$$T_j = \left(\left\{ \left(I_{ji}, g_j\left(v, x^{(I_{ji})}\right), d_{ji} \right) \right\}_{i=1}^{c_j}, c_j, r_j \right)$$

is the data set associated with the jth constraint function, I_{ji} is the index pointing to the global design data set $\{x^{(i)}\}_{i=1}^{n}$, $g_j(\boldsymbol{v}, \boldsymbol{x}^{(I_{ji})})$ is

the value of the jth constraint, d_{ji} is the corresponding trust region radius, c_j is the counter indicating the number of points in the design data set corresponding to the jth constraint, and

$$r_j = \left| \max_{1 \le i \le c_j} \left\{ g_j\left(v, x^{(I_{ji})}\right) \right\} - \min_{1 \le i \le c_j} \left\{ g_j\left(v, x^{(I_{ji})}\right) \right\} \right|$$

is the difference between the current maximum and minimum values of the jth constraint. Components of the tuples D and T_j are denoted by subscripts, for example, $(T_j)_1$ is the first component of T_j. Finally, each node in the binary tree memory structure records a tuple of the form $(\boldsymbol{v}, D)$. The pseudocode, with comments in brackets, for processing a candidate individual $(\boldsymbol{v}, \boldsymbol{x})$ is defined by Algorithms 2 and 3:

$B(j, \boldsymbol{v}, \boldsymbol{x})$ is a Boolean function intended to provide the option of bypassing the algorithm's normal logic, if dictated by a priori knowledge about the function g_j or individual $(\boldsymbol{v}, \boldsymbol{x})$. The parameters $c_j^{\min}$ [minimum number of data points required to construct the approximation $s_j(\boldsymbol{x})$ to $g_j(\boldsymbol{v}, \boldsymbol{x})$, determined in the present work by the requirements of Shepard's algorithm] are defined separately for each constraint function, and their values are based on the function complexity and approximation method used for the constraint function. The algorithm uses three real user-specified parameters, d^0, δ, and ϵ, all indexed by j. The parameter $d^0 > 0$ is an upper bound on the trust region radius about each sample point $x^{(i)}$. The parameter δ is chosen to satisfy $0 < \delta < 1$ and, in higher dimensions, protects against large variations in f in unsampled directions. Finally, the

---

Algorithm 2; Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where $\boldsymbol{v}$ is not found in the tree:

if v is not found in the tree **then**
  evaluate $g_0(v, x), \ldots, g_p(v, x)$;
  evaluate $f(v, x)$; $n := 1$; $x^{(1)} := x$;
  for $j = 0$ to p **do**
    $c_j := 1$; $r_j := 0.0$; $I_{j1} := 1$; $d_{j1} := 0.0$;
    $T_j := (\{(I_{ji}, g_j(v, x^{(I_{ji})}), d_{ji})\}_{i=1}^{c_j}, c_j, r_j)$;
  end for
  $D := (\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_p)$; add a node corresponding to (v, D);
  return $f(v, x)$;
end if

Algorithm 3 (continuation of Algorithm 2); Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where $\boldsymbol{v}$ is found in the tree:

if v is found in the tree **then**
  for $j = 0$ to p **do**
    if $B(j, v, x)$ **then**
      evaluate $g_j(v, x)$;
    else
      if $c_j < c_j^{min}$ **then**
        [add new point x to database]
        evaluate $g_j(v, x)$; $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$; $c_j := c_j + 1$;
        $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(v, x), 0.0)\}$; $I_{jc_j} := n$;
        $r_j := |\max_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\} - \min_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\}|$;
      else
        construct $s_j(x)$ using the data in
        $\{(x^{(I_{ji})}, g_j(v, x^{(I_{ji})}))\}_{i=1}^{c_j}$;
        define k and d^* by
        $d^* := d_{jk} - \|x - x^{(I_{jk})}\| = \max_{1 \le i \le c_j}\{d_{ji} - \|x - x^{(I_{ji})}\|\}$;
        if $d^* \ge 0.0$ and $|g_j(v, x^{(I_{jk})}) - s_j(x)| < \delta_j r_j$ **then**
          [accept approximation as good enough]
          $g_j(v, x) := s_j(x)$;
        else
          evaluate $g_j(v, x)$;
          $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$;
          $c_j := c_j + 1$;
          [update trust region radius for x]
          if $|g_j(v, x) - s_j(x)| > \epsilon_j$ **then**
            $d_{jc_j} := 0.0$;
          else
            $d_{jc_j} := \min\{d_j^0, \|x - x^{(I_{jk})}\|\}$; $d_{jk} := d_{jc_j}$;
          end if
          [update node database with information for x]
          $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(v, x), d_{jc_j})\}$; $I_{jc_j} := n$;
          $r_j := |\max_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\} - \min_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\}|$;
        end if
      end if
    end if
  end for
  evaluate $f(v, x)$ using $g_0(v, x), \ldots, g_p(v, x)$;
  return $f(v, x)$;
end if

parameter $\epsilon > 0$ is the selected acceptable approximation accuracy and is based solely on engineering considerations.

V. Local Improvement

Local improvement is essentially an addition to improve the performance of the GA with memory binary tree and separate multivariate approximations. The effectiveness of local improvement was shown in previous work.[5,6] The values of the continuous variables at a given discrete node are either randomly assigned or obtained through the GA operations. If explicit multivariate approximations for all constraint functions are available at a given node, it is possible to generate good candidates for the continuous design variables for the next child at that node thorough local optimization rather than depending on random actions from the GA operators. After construction of all initial approximations $\tilde{g}_i$ of g_i, one can easily find the approximate fitness function $\tilde{f}(\boldsymbol{x})$ whose evaluations do not require any exact function evaluations. Next it is possible to find the design vector $\boldsymbol{x}^*$ that optimizes the approximate function $\tilde{f}(\boldsymbol{x})$ in some compact subset $\Omega \subset E^m$ containing the real data points $\boldsymbol{x}^{(i)}$ associated with that node. This optimal $\boldsymbol{x}^*$ vector is stored at the discrete node in addition to the rest of the database D. If, in future generations, a discrete node that has a stored $\boldsymbol{x}^*$ vector is reached through the GA operations on the discrete part $\boldsymbol{v}$ of the design, then, rather than performing crossover or mutation on the real part, $(\boldsymbol{v}, \boldsymbol{x}^*)$ is used as the child design for the next generation. This child design is treated like the other new designs in the child population to which Algorithm 2 is applied to avoid exact analysis. In an effort to reduce premature convergence, the local improvement procedure is applied with some probability.

VI. Design Optimization Problem

The design of a fiber-reinforced composite lattice shell with specified radius, length, and axial load level is considered as a demonstration problem for the procedure described. Such shells supported by a lattice have been considered as a replacement to solid shells, stiffened shells, and honeycomb structures.[13–15] Consider a lattice cylindrical shell loaded with compressive axial load P. In general, proper design would involve determination of rib parameters (dimension of cross section, material, spacing, and orientation angle φ), skin parameters (the number of layers, their materials, thicknesses, and orientation angles θ_k) that satisfy strength and stability constraints while minimizing the weight of the shell. Constraints considered include rib strength constraint g_1, skin strength constraint g_2, rib local buckling constraint g_3, and general buckling constraint g_4. All constraint equations are based on the lattice cylindrical shell model developed by Bunakov and Protasov,[16] Belousov and Bunakov,[17] and Bunakov.[18]

The mixed optimization problem considered here operates on three design variables $\boldsymbol{v}$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$. The discrete variable is the stacking sequence encoding of the skins, $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$, where k is an implicit design variable dictated by the number of layers in the skin stacking sequence and $\boldsymbol{v}_i$ is the integer encoding for the ith fiber angle θ_i. We shall restrict our consideration to two continuous design variables, namely, the helical rib height, $\boldsymbol{x}_1 = h$, and the orientation angle of helical ribs with respect to the axial direction, $\boldsymbol{x}_2 = \varphi$. The optimization problem can be formulated as finding the stacking sequences of the skins, the angle of helical ribs, and the helical rib height to minimize the mass of the shell, g_0, and satisfy all constraints. The set of design variables is expressed as a vector $\boldsymbol{\tau} = (\boldsymbol{v}, \boldsymbol{x}_1, \boldsymbol{x}_2)$. The optimization problem can be written as

$$\min_{\tau} g_0(\boldsymbol{\tau}) \tag{1}$$

such that rib strength, skin strength, rib local buckling, and general shell buckling are, respectively,

$$g_1(\tau) \geq 0, \qquad g_2(\tau) \geq 0, \qquad g_3(\tau) \geq 0, \qquad g_4(\tau) \geq 0$$

and where

$$h \in [h_{\min}, h_{\max}], \qquad \varphi \in [\varphi_{\min}, \varphi_{\max}]$$

$$\theta_i \in \{0, \pm 45, 90\} \qquad (i = 1, \ldots, k)$$

$$k \in [k_{\min}, k_{\max}]$$

where $h_{\min}$ and $h_{\max}$ are the lower and upper bounds of the rib height, $\varphi_{\min}$ and $\varphi_{\max}$ are the lower and upper bounds of the angle of helical ribs, θ_i is the ply orientation angle (in degrees) in the ith skin ply, k is the total number of skin plies, and $k_{\min}$ and $k_{\max}$ are minimum and maximum possible values of k. The preceding problem may not be a realistic composite design formulation, but it is used instead of a completely artificial constrained optimization problem. A standard laminate optimization typically includes additional constraints such as ply contiguity, interlaminar stress, core strength, etc.

The constrained optimization problem is transformed into an unconstrained maximization problem using a penalty function approach. The critical constraint is defined as

$$g_{\text{cr}}(\tau) = \min_{1 \leq i \leq 4} \{g_i(\tau)\} \tag{2}$$

Note that min/max functions such as g_{cr} are typically nonsmooth and are much harder to approximate directly than are their constituent functions g_i. The fitness function f to be maximized is defined as

$$f(\tau) = \begin{cases} -g_0(\tau) + \alpha g_{\text{cr}}(\tau), & g_{\text{cr}}(\tau) \geq 0 \\ -g_0(\tau) + \beta g_{\text{cr}}(\tau), & g_{\text{cr}}(\tau) < 0 \end{cases} \tag{3}$$

where α is a bonus parameter and β is a user-defined penalty parameter.

VII. Results

A cylindrical lattice shell considered in this study is made of fiberglass–epoxy composite material with density $\rho = 1600$ kg/m^3. The shell radius and length are $R = 0.7$ m and $L = 1.8$ m, respectively. The specified axial compressive load is $P = 10^6$ N/m. The shell has external and internal skins made of T300/5208 graphite–epoxy unidirectional plies with basic ply thickness $h_0 = 0.125$ mm. The material properties of the skin plies are given in Table 1. The lattice shell has $\pm\varphi$ unidirectional helical ribs with elastic modulus $E = 45$ GPa and shear modulus $G = 1$ GPa. The ribs have initial rectangular cross sections of width $b = 4$ mm and height h. The helical rib spacing is $a = 40$ mm. The compressive strength of ribs is $\sigma^r_{\max} = 240$ MPa. The possible ranges for the design variables are given in Table 2.

A. GA Parameters

The values of the GA parameters used in the experiments are shown in Table 3. The GA stopping condition is a limit on the total

Table 1 Material properties of skin (T300/5208)

Parameter	Value
Stiffness	
E_1	181.0
E_2	10.3
G_{12}	7.17
ν_{12}	0.28
Strength	
X_t	1500.0
Y_t	40.0
X_c	1500.0
Y_c	246.0
S	68.0

Table 2 Ranges for design variables

Design variable	Range
$h \in [h_{\min}, h_{\max}]$	[0.001, 0.1], m
$\varphi \in [\varphi_{\min}, \varphi_{\max}]$	[5, 85], deg
$\theta_k, k = 1, n$	{0, ±45, 90}, deg
$n \in [n_{\min}, n_{\max}]$	[2, 14]

Table 3 GA parameters used in experiments

Parameter	Value
Selection type	Elitist
Maximum number of generations	25,000
Population size	20
Laminate chromosome length, λ	7
Crossover type	
Laminate chromosomes	One-point
Geometry chromosomes	Uniform
Probability of crossover, p_c	
Laminate chromosomes	1.0
Geometry chromosomes	1.0
Probability of mutation, p_m	
Laminate chromosomes	0.05
Geometry chromosomes	0.01
Bonus parameter α	0.0
Penalty parameter β	10.0

Table 4 Best-known optimal design using standard GA

Parameter	Value
$\bar{n}_e^0$	113,615
x_1	0.0100
x_2	62.8192
v^{a}	1100000
g_0	22.0723
j_{cr}	2
g_{cr}	0.0
f	−22.0723

[a]Integer encodings 1, 0 in v mean 0 deg and no ply, respectively.

Table 5 Efficiency of multivariate approximations

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, %	$\bar{\zeta}$, %	E
g_1	23,168	3383	84.88	97.02	$9.2915E{-}03$
g_2	23,168	3058	86.28	97.31	$1.0860E{-}05$
g_3	23,168	3931	82.39	96.54	$3.2553E{-}01$
g_4	23,168	3410	84.78	97.00	$3.8846E{-}04$

number of fitness function evaluations conducted by the standard GA, $(n_e^0)_{\max} = 500{,}000$. The best known global optimal design obtained by the standard GA is presented in Table 4. Table 4 gives the average number $\bar{n}_e^0$ of exact analyses of individuals from 10 runs, the continuous design variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the discrete design variable $\boldsymbol{v}$, the objective function value g_0, the critical constraint number j_{cr}, the critical constraint value g_{cr}, and fitness function value f. This design was obtained in an average of about 113,615 function evaluations by the standard GA.

B. Effect of GA Improvements

The results presented in this section focus on the ability of the proposed algorithm to save computational time during GA optimization with the multivariate approximation used as a memory device. The best design is identical to the results presented in Table 4 for the baseline algorithm. The performance of the GA with the multivariate approximation is presented in Table 5, which shows averages from 10 runs with the prescribed parameters $\epsilon = 0.01$, $\delta = 0.1$, and $d^0 = 0.5$. Table 5 shows the average number of attempts to evaluate constraint functions $\bar{n}_i$, the average number of exact analyses $\bar{n}_e$, the average percent savings $\bar{\xi}$ in terms of constraint function evaluations, the average percent savings $\bar{\zeta}$ in terms of constraint function evaluations as compared with the standard GA result reported in Table 4, and the mean absolute error E due to the approximations. The average percent savings $\bar{\xi}$ in terms of number of constraint function evaluations is defined by

$$\bar{\xi} = \frac{1}{r}\sum_{k=1}^{r}\left[1 - \frac{(n_e)_k}{(n_i)_k}\right] \times 100\% \tag{4}$$

where r is the number of the runs. The average percent savings $\bar{\zeta}$ in terms of constraint function evaluations as compared with the standard GA is defined by

$$\bar{\zeta} = \frac{1}{r}\sum_{k=1}^{r}\left(1 - \frac{(n_e)_k}{\bar{n}_e^0}\right) \times 100\% \tag{5}$$

The average mean absolute error E is defined as

$$E = \frac{1}{r}\sum_{k=1}^{r}\left(\frac{1}{n_s}\sum_{i=1}^{n_s}|g(x_i) - s(x_i)|\right)_k \tag{6}$$

where $n_s = n_i - n_e$ is the total number of acceptable approximate evaluations for the given constraint. This error is computed every time that the algorithm decides to extract an approximation of the constraint value without an exact analysis.

The results of the experiments in Table 5 show that the cost of the GA with continuous variables could be reduced up to 97% relative to the standard GA by using the approximation procedure. The results shown in Table 5 are for the same problem presented in Ref. 6 with very minor changes in the overall dimensions of the cylindrical shell. Compared to the improvements achieved in Ref. 6, in which the percent savings in terms of number of exact fitness evaluations compared to the standard GA was at most 86% (Table 9 of Ref. 6), the superiority of the approach proposed here is evident. Moreover, for the problem considered, the computation of the fitness functions is not very expensive in terms of CPU time. However, for realistic problems in which evaluation of the objective and/or constraint functions may require large finite element analysis models, the computation effort spent on evaluating the fitness function far exceeds that of the memory tree and approximation constructions. Therefore, the approach developed here has great potential for problems with expensive fitness functions.

Table 6 contains information about the problem design space. Table 6 includes the laminate chromosome length λ, the number of the possible alphabet elements q, the maximum number of nodes in the binary tree, that is, the number of all possible combinations, $N_{\max}$; the actual average number of nodes in the binary tree, $\bar{N}_t$; the average number of nodes containing at least one working approximation, $\bar{N}_a$; and the average number of design points used to construct approximations in all nodes of the binary tree, $\bar{N}_p$: The GA with memory converges fast and uses about 8% of all available binary tree nodes to obtain the optimal solution.

The mean absolute error E due to the approximation and the savings $\bar{\xi}$ in terms of the number of constraint evaluations for different values of the parameters ϵ and δ with $d^0 = 0.5$ for all constraint functions are shown in Table 7. It is possible to further enhance the performance of the algorithm by a more precise tuning of its parameters. Table 7 shows the expected trends; both average savings $\bar{\xi}$ and average absolute error E increase as either ϵ or δ increases.

Table 8 shows the performance of the GA with separate constraint approximations and local improvement. The local improvement procedure is a quasi-Newton optimization method. In an effort to reduce premature convergence of the GA, the local improvement procedure is applied with probability 0.5. As expected, the GA with approximations and local improvement converges faster in terms of number of fitness function evaluations than the GA with just the separate constraint approximations. Both algorithms with separate

Table 6 Design space for GA with multivariate approximation

Parameter	Value
λ	7
q	3
$N_{\max}$	3280
$\bar{N}_t$	272
$\bar{N}_a$	20
$\bar{N}_p$	8101

Table 7 Average percent savings $\bar{\xi}$ and average error of multivariate approximations E as functions of parameters ϵ and δ with $d^0 = 0.5$

	g_1		g_2		g_3		g_4	
δ	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E
				$\epsilon = 0.001$				
0.1	42.23	1.50E−03	44.38	3.14E−06	32.15	3.82E−03	41.23	1.21E−05
0.5	45.54	1.57E−03	55.16	3.26E−06	34.15	3.64E−02	51.26	1.68E−05
1.0	60.06	2.05E−03	62.35	4.28E−06	60.12	4.22E−02	58.14	1.95E−04
				$\epsilon = 0.005$				
0.1	76.65	2.36E−03	64.29	6.25E−06	65.27	4.26E−02	69.26	2.68E−04
0.5	77.26	5.26E−03	68.25	8.26E−06	71.26	4.59E−02	76.65	2.57E−04
1.0	78.96	6.27E−03	70.15	1.03E−05	74.92	6.26E−02	82.64	3.26E−04
				$\epsilon = 0.01$				
0.1	84.88	9.29E−03	86.28	1.09E−05	82.39	3.26E−01	84.78	3.88E−04
0.5	85.16	1.02E−02	86.31	3.49E−05	82.69	6.26E−01	85.06	6.57E−04
1.0	85.21	8.27E−02	86.65	5.32E−05	82.98	8.26E−01	85.65	8.27E−04

Table 8 Efficiency of multivariate approximations and local improvement with $\epsilon = 0.01$, $\delta = 0.1$, and $d^0 = 0.5$

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, %	$\bar{\zeta}$, %	E
g_1	10,761	2094	80.54	98.16	3.4354E−02
g_2	10,761	1932	82.05	98.30	1.0475E−02
g_3	10,761	2346	78.20	97.94	1.1426E−01
g_4	10,761	2105	80.44	98.15	2.4638E−02

constraint approximations demonstrate good convergence in comparison with the standard GA and noticeably decrease the number of exact analyses. However, note that the mean absolute error is increased for the results with the local improvement procedure. A tradeoff analysis between the acceptable approximation accuracy and the overall modified GA performance is indicated.

VIII. Conclusions

Modifications of the standard GA to save previously computed fitness values provide significant performance improvement. A GA with memory along with multivariate approximations of the objective and constraint functions individually was applied to the problem of weight minimization of a lattice shell with mixed discrete and continuous design variables. The use of memory based on a binary tree for the discrete part of the design variables avoids repeating analyses of previously encountered designs. The multivariate approximation for continuous variables saves unnecessary exact analyses for points close to previous values.

Acknowledgments

This research was supported in part by Air Force Office of Scientific Research Grant F49620-99-1-0128 and National Science Foundation Grant DMS-9625968.

References

[1]Seront, G., and Bersini, H., "A New GA-Local Search Hybrid for Continuous Optimization Based on Multi-Level Single Linkage Clustering," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00)*, edited by L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, Morgan Kaufmann, San Francisco, 2000, pp. 90–95.

[2]Kogiso, N., Watson, L. T., Gürdal, Z., and Haftka, R. T., "Genetic Algorithms with Local Improvement for Composite Laminate Design," *Structural Optimization*, Vol. 7, No. 4, 1994, pp. 207–218.

[3]Kogiso, N., Watson, L. T., Gürdal, Z., Haftka, R. T., and Nagendra, S., "Design of Composite Laminates by a Genetic Algorithm with Memory," *Mechanics of Composite Materials and Structures*, Vol. 1, No. 1, 1994, pp. 95–117.

[4]Gantovnik, V. B., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Panels," AIAA Paper 2002-1221, April 2002.

[5]Gantovnik, V. B., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Panels," *Composite Structures*, Vol. 58, No. 4, 2002, pp. 513–520.

[6]Gantovnik, V. B., Anderson-Cook, C. M., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Mixed Discrete-Continuous Design Optimization," *Computers and Structures*, Vol. 81, No. 20, 2003, pp. 2003–2009.

[7]McMahon, M. T., Watson, L. T., Soremekun, G. A., Gürdal, Z., and Haftka, R. T., "A Fortran 90 Genetic Algorithm Module for Composite Laminate Structure Design," *Engineering with Computers*, Vol. 14, No. 3, 1998, pp. 260–273.

[8]Vowels, R. A., *Algorithms and Data Structures in F and Fortran*, Unicomp, Inc., Tucson, AZ, 1998, pp. 89–98.

[9]Renka, R. J., "Multivariate Interpolation of Large Sets of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 139–148.

[10]Renka, R. J., "Algorithm 660: QSHEP2D: Quadratic Shepard Method for Bivariate Interpolation of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 149, 150.

[11]Renka, R. J., "Algorithm 661: QSHEP3D: Quadratic Shepard Method for Trivariate Interpolation of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 151, 152.

[12]Shepard, D., "A Two-dimensional Interpolation Function for Irregularly Spaced Data," *Proceedings of the 23rd National Conference*, Association for Computing Machinery, Brandon/Systems Press, Princeton, NJ, 1968, pp. 517–523.

[13]Vasiliev, V. V., and Lopatin, A. V., "Theory of Lattice and Stiffened Composite Shells," *Mechanics of Composite Materials*, edited by Y. M. Tarnopolskii, Zinatne, Riga, Latvia, 1992, pp. 82–88 (in Russian).

[14]Vasiliev, V. V., Barynin, V. A., and Rasin, A. F., "Anisogrid Lattice Structures—Survey of Development and Application," *Composite Structures*, Vol. 54, No. 2–3, 2001, pp. 361–370.

[15]Slinchenko, D., and Verijenko, V. E., "Structural Analysis of Composite Lattice Shells of Revolution on the Basis of Smearing Stiffness," *Composite Structures*, Vol. 54, No. 2–3, 2001, pp. 341–348.

[16]Bunakov, V. A., and Protasov, V. D., "Cylindrical Lattice Composite Shells," *Mechanics of Composite Materials*, Vol. 25, No. 6, 1989, pp. 1046–1053 (in Russian).

[17]Belousov, P. S., and Bunakov, V. A., "Bending of Cylindrical Lattice Composite Shells," *Mechanics of Composite Materials*, Vol. 28, No. 2, 1992, pp. 225–231 (in Russian).

[18]Bunakov, V. A., "Design of Axially Compressed Composite Cylindrical Shells with Lattice Stiffeners," *Optimal Design*, edited by V. V. Vasiliev and Z. Gürdal, Technomic, Lancaster, PA, 1999, pp. 207–246.

E. Livne
Associate Editor

Vita

Vladimir Gantovnik was born in April 12, 1975 in Khovu-Aksy (Tuva, Russia). He graduated from high school in Berikul (Kemerovo Region, Russia) in 1992 with *silver medal*. In 1998 he received a M.S. with *red diploma* (*summa cum laude*) in Aerospace Engineering from Siberian Aerospace Academy (Krasnoyarsk, Russia). In 1999 he received a M.S. in Materials Science & Engineering from Iowa State University (Ames, IA, USA).

Vladimir Gantovnik

Blacksburg, August 2005