Hierarchical Topology Optimization Problems In Three-Dimensions

By

Giuseppe C. A. DeRose Jr.

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

1996

Abstract

Hierarchical Topology Optimization Problems In Three-Dimensions

By

Giuseppe C. A. DeRose Jr.

Hierarchical topology optimization is introduced as a mechanism to reduce computational resources needed to solve large scale, 3-dimensional problems. In hierarchical topology optimization, octree data structures are used to describe a given design during the optimization process. Using a sequence of hierarchically described models, increasingly detailed descriptions of the optimal shape are produced. Results obtained using this method are comparable to standard methods and provide substantial computational savings. Spatial moment constrained topology optimization is introduced as a method to provide additional control of an optimum shape. In spatial moment constrained topology optimization, constraints are placed on the shape's spatial moments. Spatial moments are used to describe the topological attributes of shapes as scalars. Results show that enforcing constraints on spatial moments may significantly enhance the optimal design, adding shape control to standard topology optimization formulations.

To my parents for their continued guidance and support.

It is from their strong will and values that I attribute this success.

Acknowledgments

My sincere appreciation is extended to my major professor, Dr. Alejandro Díaz, for his continued encouragement and support. As my teacher, advisor, and mentor, his efforts have been invaluable.

I would also like to thank the members of my thesis committee. Dr. Ronald Averill's experienced teaching provided me with knowledge and confidence necessary to complete this task. I am particularly grateful for the encouragement shared by Dr. Ronald Rosenberg. I am not only indebted to his tutelage, but also to his infinite guidance.

Lastly, I would like to thank Dr. George Stockman for always taking time to discuss various issues.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The evolution of structural optimization has created three classes of problem formulations. The first class of structural optimization is sizing problems. Sizing problems consist of fixed topology and configuration, concentrating on optimizing section properties. An example of a sizing problem is a loaded truss with a fixed number of members and fixed connectivity. In such problems, an optimization of the structure stiffness may be performed where the cross-sectional areas of the structure members are allowed to vary. The second class of structural optimization is fixed-topology problems. Fixed-topology problems consist of problems where section properties and structure configuration may vary; however, the structure connectivity is set. An example of a fixed-topology problem consists of a loaded 2-dimensional plate with a hole. In this type of problem, a minimization of the stress may be performed where the size of the hole may vary. The last class of structural optimization is gener-

alized topology optimization problems introduced by Bendsøe and Kikuchi [2]. In generalized topology optimization problems the section property, structure configuration, and structure topology are allowed to vary. Initially, only the design domain and loading conditions are specified for a complete problem definition. The result of generalized topology optimization is the *optimal distribution* of material. Usually, *optimal* corresponds to minimum compliance with a constraint on the amount of material used and *distribution* corresponds to the placement of material in the domain. The result of topology optimization is a *shape* and a *connectivity.* The main benefit of generalized topology optimization is that the general shape and connectivity of the optimum design are not specified a priori but are determined from the optimization. This is the type of structural optimization that will be addressed here. This thesis focuses on the development of new methods in generalized topology optimization. In particular, it tackles the optimization problem in 3-dimensional domains.

Figure 1.1 displays a sample 3-dimensional generalized topology optimization problem definition. The entire domain, Ω, is divided into the design domain, Ω_s, and the non-design domain, Ω_v. These regions are specified along with the essential boundary conditions and loading conditions. The first step in the topology optimization methods discussed here begins with a discretization of the entire domain. In the problems addressed, Ω is restricted to a regular rectangular parallelepiped. This implies that an effective finite element discretization is achieved simply by using many small cube elements. These cubes, named *voxels*, are the 3-dimensional equivalent of 2-dimensional *pixels* used in image progressing. With a discretized domain and associated boundary conditions, the generalized topology optimization problem may

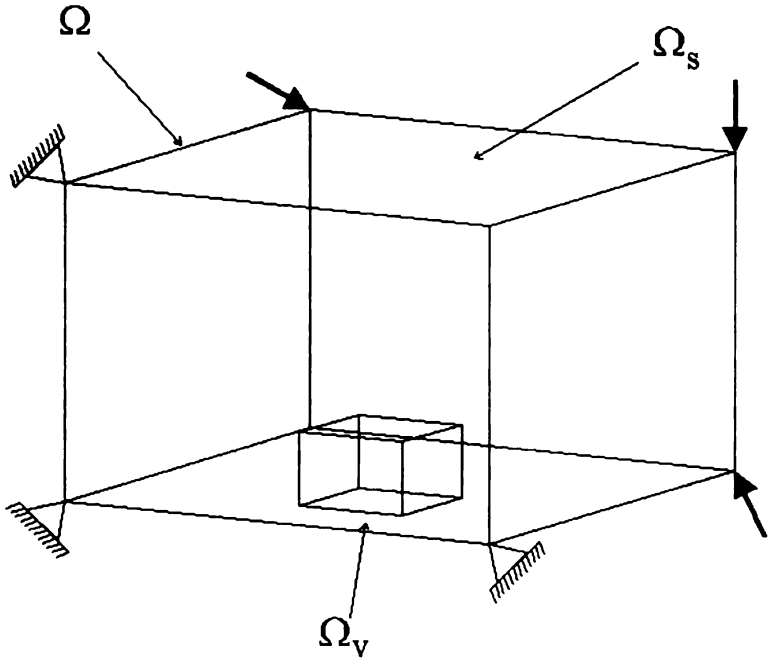

Figure 1.1: Topology optimization problem definition in 3-dimensions

be addressed.

1.1.1 Macroscopic Topology Optimization

Macroscopic topology optimization is noted as the first attempt to solve the generalized topology optimization problem. The objective is to find the stiffest structure formed by distributing a fixed amount of material in the design domain subject to the specified loading conditions. The material is distributed in binary fashion, designating each element in the design domain as either full of material or empty. Formally, this may be expressed as

$$
\begin{array}{lll}
\text{Minimize:} & \underset{\mathbf{x}}{} & \mathbf{f}^T\mathbf{u} \\
\text{Subject To:} & & \sum_{e=1}^{n} \rho_e \, v_e \; - \; \text{Vol} \; \leq \; 0 \\
 & & \mathbf{K}(\mathbf{x})\mathbf{u} \; - \; \mathbf{f} \; = \; \mathbf{0} \\
\text{Where,} & n & = \text{number of elements} \\
 & \mathbf{x} & = (\rho_1, \ldots, \rho_n) \\
 & \rho_i & = \text{element density for element } i, \text{ 1 or 0} \\
 & v_i & = \text{element volume/area/length} \\
 & \mathbf{K} & = \text{stiffness matrix} \\
 & \text{Vol} & = \text{the prescribed volume of material} \\
 & \mathbf{u} & = \text{nodal displacement vector} \\
 & \mathbf{f} & = \text{force vector}
\end{array}
\tag{1.1}
$$

This type of formulation may be interpreted as an approach that attempts to determine the optimum mixture of two materials, solid or void, at the macroscopic level. This type of problem is not well-posed as shown by Kohn and Strang [7, 8, 9] and Murat and Tartar [11]. The sequence of designs that minimize compliance does not converge in the space of macroscopically mixed materials, which leads to solutions that contain "chattering." However, determining the optimum mixture of two materials at the microscopic level is well-posed and discussed in the next section.

1.1.2 Topology Optimization With Microstructure

To overcome the problems associated with the macroscopic topology optimization, the topology optimization problem is solved using a homogenization method. Rather than simply distributing material at the macroscopic scale, material constructed with microscopic voids is distributed over the design domain. Thus, each element in the design domain contains a mixture of material described by a microstructure. The microstructure in homogenization methods is characterized by a microscale cell. In

the problems addressed, the microscale cell used is shown in Figure 1.2. The variables a, b, and c describe the hole in the microcell and determine the amount of material in the cell while the variables ψ, ϕ, and θ represent the cell orientation.

Application of these ideas was first introduced to topology optimization in 2-dimensions by Bendsøe and Kikuchi [2]. This idea was then expanded and applied to 3-dimensional problems by Suzuki and Kikuchi [17, 18]. In topology optimization based on a homogenization strategy, the material in each element is constructed with a matrix of the microcells. Therefore, the effective material properties of each element are described by the cell hole size (a_e, b_e, c_e) and the cell orientation (ψ_e, ϕ_e, θ_e). Formally, a topology optimization based on a homogenization strategy may be expressed as

$$
\begin{array}{lll}
\underset{\mathbf{x}}{\text{Minimize:}} & \frac{1}{L}\sum_{i=1}^{L}\int_{\Omega} < \mathbf{f}^{(i)}, \mathbf{u}^{(i)} > \, d\mathbf{u} & \\
\text{Subject To:} & \int_{\Omega} \rho(\mathbf{x})\, d\Omega \; - \; \text{Vol} \; \leq \; 0 & \\
 & 0 \; \leq \; \rho_e \; \leq \; 1 \quad \text{for } e = 1, \text{n} & \\
 & \mathbf{K(x)U} \; - \; \mathbf{F} \; = \; \mathbf{0} & \\
\text{Where,} & \mathbf{x} \; = \; (a_1, b_1, c_1, \psi_1, \phi_1, \theta_1, \ldots, & \\
 & \quad a_n, b_n, c_n, \psi_n, \phi_n, \theta_n) & \\
 & \rho_e \; = \; 1 \; - \; (1 - a_e)(1 - b_e)(1 - c_e) & \\
 & L \; = \; \text{number of load cases} & \\
 & \mathbf{U} \; = \; \left[\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(L)}\right] & \\
 & \mathbf{F} \; = \; \left[\mathbf{f}^{(1)}, \ldots, \mathbf{f}^{(L)}\right] &
\end{array}
\tag{1.2}
$$

Here, $u^{(i)}$ corresponds to the displacement field associated with the load case $f^{(i)}$. Unlike the macroscopic topology optimization problem, homogenization-based topology optimization is well-posed. In the remaining sections of this thesis, the term topology optimization or generalized topology optimization will be used to indicate

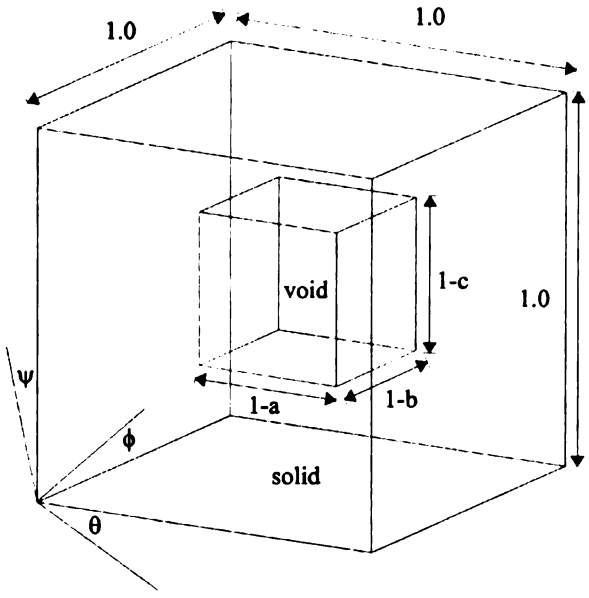

Figure 1.2: 3-dimensional microscale unit cell

homogenization-based topology optimization.

1.2 Recent Advances In Topology Optimization

Much of the recent work in topology optimization has concentrated in two major areas: reducing the resources necessary to solve large-scale problems and modifying the standard formulation stated in (1.2) to provide more control over the shape of the optimal design. The needs to reduce computational resources in topology optimization are mostly seen in areas of 3-dimensional analysis. Often, a large number of design elements are needed to estimate the design domain within acceptable engineering accuracy. Growth of the number of design elements is directly related to a growth in the size of the topology optimization problem. Additional control over the optimal shape is desired to produce more acceptable engineering solutions. Under some loading conditions in various classes of problems, the optimal shape may

not be a practical solution to a given engineering problem. Often, these solutions involve structures with a large number of members or contain extensive areas where the structure is defined by an intermediate stage of material mixture (i.e., a composite). These types of solutions may not be desirable for manufacturing reasons, since the associated structures are either too costly to produce or impossible to manufacture.

Attempts to reduce the resources necessary to solve large-scale problems have concentrated on the solution of finite element models. The solution of a finite element model of the discretized domain is required during each iteration of the optimization problem. As problems grow larger, this operation is the determining factor in the amount of resources necessary to obtain a solution. Hence, reducing resources involved in solving the finite element model significantly reduces the resources necessary to solve topology optimization problems. In-core skyline solvers are considered standard practice in finite element analysis. However, with the introduction of larger problems, in-core skyline solvers are only applicable on super computers. As in standard finite element analysis, element-by-element solvers and sparse solvers were introduced to topology optimization to alleviate this problem. These solvers require less in-core memory and help reduce the resources necessary for topology optimization.

Attempts to provide control of the optimum shape are focused on the re-definition of an optimal design. This is achieved by modifying the objective function in (1.2), by modifications to the optimization solution algorithm, by addition of constraints, or by post-processing of standard topology optimization results. One such effort, suggested by Sigmund [16], provides control of the optimal shape by introducing filter techniques into the optimization solution algorithm. There, a series of topology optimization

problems with varying material properties is solved to provide control on minimum member size in optimal 2-dimensional structures. Another example of optimal shape control is seen in work by Haber, Jog, and Bendsøe [4]. Minimization of compliance is performed with the additional constraint on structure perimeter in 2-dimensions. A final example is seen in the work of Allaire *el. al* [1]. There, post-processing is used after convergence of a standard topology optimization sequence. Penalization techniques are used to remove composite regions from the optimal design. In general, these methods provided tools to control the *shape* of the optimal *structure*.

1.3 Objective

There are two main objectives in this thesis. First, reformulate the generalized topology optimization problem and develop a new solution algorithm to significantly reduce the resources necessary to solve large-scale, 3-dimensional problems. Second, develop a general approach to provide additional shape control over the optimum structure.

Clearly, most of the recent work in solving large-scale topology optimization problems have utilized the advances in general finite element analysis. Unfortunately, the basis of these advances in finite element analysis concentrate on the analysis of structures where the structure topology remains fixed. In topology optimization, the shape of structure changes with each iteration. Here, we desire algorithms that take advantage of the varying structure. The first objective will be achieved through use of hierarchical data structures to reduce the number of finite elements needed to analyze a given structure. In current practice, the initial discretization of the domain

remains static during the topology optimization. Here, a method is introduced to allow a dynamic discretization of the domain. Thus, rather than solving a possible large-scale topology optimization problem directly, a sequence of smaller problems is solved, where the level of discretization of the domain is dependent on the current design or shape.

With few exceptions (e.g., the work of Allaire *el. al.*), recent developments in shape control have been applied to 2-dimensional problems. Here, the goal is to apply some of the methods used in 2-dimensional shape control to 3-dimensional problems. The second objective will be achieved through use of *scalar shape descriptors* to force the optimal shape to process various shape attributes. Unlike the methods mentioned previously, this approach to shape control in topology optimization provides a large set of shape descriptors. Similar to the perimeter control method, constraints on various scalar shape descriptors are incorporated into the standard topology optimization formulation. These shape descriptors, spatial moments, provide an extensive and powerful set of shape descriptors to achieve optimum shape control in topology optimization.

1.4 Outline Of Thesis

The rest of this thesis consists of Chapters 2 through 6. Chapter 2 describes *octrees*, a hierarchical data structure, which is introduced to reduce the resources necessary to solve large-scale 3-dimensional problems. Chapter 3 introduces the idea of scalar shape descriptors and how they may be used to provide structure control in topology

optimization for 3-dimensional problems. Chapter 4 provides illustrative examples of the solutions to topology optimization problems using hierarchical techniques; while Chapter 5 provides examples of topology optimization problems solved with scalar shape descriptors. Concluding remarks and suggestions for future areas of investigation are discussed in Chapter 6.

Chapter 2

Hierarchical Data Structures And Topology Optimization

2.1 Introduction

The focus of this chapter is on the re-definition of the topology optimization problem to significantly reduce the resources necessary to solve large-scale, 3-dimensional problems. This is achieved with a modified region octree methodology applied to a sequence of smaller topology optimization problems. The resulting process is called hierarchical topology optimization. Below is a brief discussion of the conditions in which this process may be used.

In order to use hierarchical topology optimization, one must define a domain and boundary conditions as in standard topology optimization. Unlike standard topology optimization, there exist limitations on the type of domain discretization. The initial discretization must consist exclusively of equally sized cube finite elements that are

aligned in a rectangular parallelepiped. This initial discretization is the finest discretization that may be achieved during the hierarchical topology optimization. The cube elements are termed *voxels*, so this level of discretization is named the *voxel discretization.* As in standard topology optimization, hierarchical topology optimization is an iterative procedure. Each iteration in this optimization technique involves the re-definition of the finite element mesh describing the domain based on the current material distribution. The octree hierarchical data structure will be used to generate this mesh. Since octrees use voxel grayscale values in the construction of the hierarchical representation, the effective densities of the voxels are used in the octree application. The mesh from the octree analysis is called the *octree discretization.* In general, the number of elements in the octree discretization is far fewer than the number of voxels in the voxel discretization. This is the main source of computational savings in this new method.

The remaining sections of this chapter will discuss the development and implementation of this algorithm. First, the reasoning for selecting region octrees as the data structure used is described, followed by examples of applied region octrees. Then, a detailed discussion of the interaction between octrees and topology optimization is provided. There, a formal definition of the new topology optimization sequence is stated. This is followed by a discussion of strategies to solve the hierarchical topology optimization problem.

2.2 Benefits Of Region Octrees

The efficient storage of 3-dimensional volume data has been most aggressively addressed in the area of computer aided design for geometric modeling. Various storage techniques for solid modeling applications have been investigated with the recent growth of solid modeling in engineering. Samet [14, 15] and Mantyla [10] have provided excellent descriptions of various data structures used in solid representation. Below is a description of a few of these structures:

1. *Boundary Representation* - A solid is represented by its boundary. The solid's boundary is described by a set of faces, edges, and vertices.

2. *Sweep Methods* - A solid is represented by a 2-dimensional shape swept along a curve in 3-dimensional space.

3. *Primitive Instancing* - All possible object shapes are defined parametrically. A solid is represented by varying the scale and dimension of a given parameterized object.

4. *Constructive Solid Geometry* - A set of primitive instances are available. A solid is represented by a boolean combination of the primitives.

5. *Spatial Enumeration* - A solid is represented by set of voxels in 3-dimensions in the same fashion as 2-dimensional shapes are represented by pixels.

6. *Cell Decomposition* - A solid is represented by a set of cells in space, similar to finite elements.

The most common methods of solid representation are Constructive Solid Geometry, Boundary Representation, and Sweep Methods. However, these solid representations would not be practical for the problem at hand. A data structure is needed to hierarchically describe a shape that varies during an iteration sequence. Most importantly, the shape is not known a priori; thus, realistic shape primitives may not be constructed a priori. This eliminates the use of Primitive Instancing to represent the solid. In addition, the goal of the data structure is to develop a discretization of the current shape that minimizes the number of finite elements necessary to describe the shape. This eliminates the use of Spatial Enumeration. The final solid representation is Cell Decomposition. Region octrees are contained in this class of shape descriptors. Hence, region octrees are used for hierarchical topology optimization.

The use of region octrees in topology optimization has many advantages. In hierarchical topology optimization, the domain is initially defined at the voxel discretization level. This is a spatial enumeration of the domain. Region octrees allow the analysis of this type of domain to produce a hierarchical description. This is achieved by aggregation techniques based on the density distribution of the design. The result of the octree encoding is a new description of the domain composed of elements built from the aggregation of neighboring voxels of similar density. In most cases, the final result will contain far fewer elements than the number of voxels and the accuracy remains at the voxel level in areas of rapidly varying densities. This is the underlying principle of the use of region octrees for hierarchical topology optimization.

2.3 Examples Of Region Octrees

Region octrees are the 3-dimensional version of 2-dimensional quadtree representation. The idea of a pixel in 2-dimensions is analogous to the voxel in 3-dimensions. Like quadtrees, region octrees were first applied to represent binary data hierarchically. Instead of binary pixel data, 3-dimensional region octrees were applied to binary volume data. Volume data implies a spatial enumeration of a given domain where every point in space is approximated by a voxel. An octree representation of a domain is created using recursive subdivisions of cube regions into eight *octants*. The subdivision of a given octant is complete when all the voxels in the octant possess the same binary value. At this point, the octant is called an *octree cube*. Figures 2.1 through 2.5 display the form of an octree representation of a simple 3-dimensional domain consisting of 64 voxels. Figure 2.1 displays the original binary voxel data. Note that voxels of binary data value 0 are not displayed. Figure 2.2 displays an enumeration of the 64 voxels in Figure 2.1 where each voxel is assigned an unique identification number. Note that the lower back corner of Figure 2.1 corresponds to voxel 0 in Figure 2.2. Comparing Figure 2.1 and Figure 2.2, O_b, the set of all black (binary data value 1), may be defined as

$$O_b = \{0, 1, \ldots, 9, 11, 12, 13, 15, \ldots, 25, 27, 28, 29, 32, 33, 34, 35, 48\} \qquad (2.1)$$

O_w, the set of all white (binary data value 0), may be defined as all the voxels in Figure 2.1 that are not listed in O_b. Applying octree algorithms on the sample data,

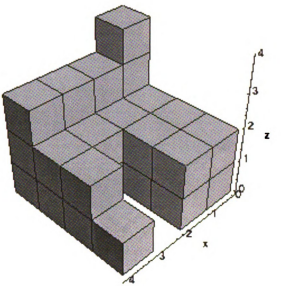

Figure 2.1: Sample 3-dimensional domain displayed at the voxel level

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Level 0

16	20	24	28
17	21	25	29
18	22	26	30
19	23	27	31

Level 1

32	36	40	44
33	37	41	45
34	38	42	46
35	39	43	47

Level 2

48	52	56	60
49	53	57	61
50	54	58	62
51	55	59	63

Level 3

Figure 2.2: Complete enumeration of the voxels

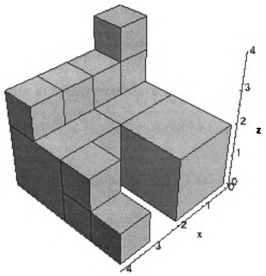

Figure 2.3: Region octree represent of the sample 3-dimensional domain

an octree representation of the domain may be determined. This representation is displayed graphically in Figure 2.3.

Next, a mechanism for storing octree data is necessary. Here, a hierarchical data structure is used to store the data efficiently. A hierarchy of octree octants is built to represent the voxel data. An octree octant is a group of voxels that are spatially aligned so they may be treated as a cube. Therefore, the number of voxels in a given octree octant must be equal to an integer power of 2 (i.e., 0,2,4,8,16,32,...). Because the octree octants are cubes and the domain is enumerated as shown in Figure 2.2, only four attributes are necessary to define octree octant: the type of element, the location of the element, the size of the element, and the list of the element's children. Next, a more detailed description of these attributes is presented.

A binary octree may possess three types of octants: black, white, or mixed. The meaning of the octree octants are as follows:

1. Black octree octants are regions of data that consist of purely black voxels (e.g., in topology optimization, a region made of isotropic material).

2. White octree octants are regions of data that consist of purely white voxels (e.g., in topology optimization, essentially void space).

3. Mixed octree octants are regions of data that consist of both black and white voxels.

The location of an octree octant may be represented many ways. In the investigations presented here, the octree octant location is determined by the lowest voxel number in the octant. The size of the octant is represented by the number of voxels along its edge. For example, an octree octant that contains the entire domain in Figure 2.1 would be a mixed octree octant, with location 0, and size 4. This octree octant can now be subdivided to create more octree octants to describe the domain more accurately. This subdivision is stored as the octant's children. The final goal is to obtain a complete list of the black and white octants (i.e., octree cubes). This list of octree cubes will accurately represent the domain requiring less cubes than voxels in most cases. The data structure that is used to represent this example is shown in Figure 2.4 and Figure 2.5. Figure 2.4 describes the data structure while Figure 2.5 is the actual octree representation of the current example.

These ideas may be expanded to grayscale 3-dimensional data where each voxel is described by a grayscale level. In grayscale analysis, the subdivision of an octant is

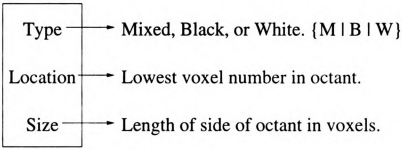

Figure 2.4: Octree data structure definition

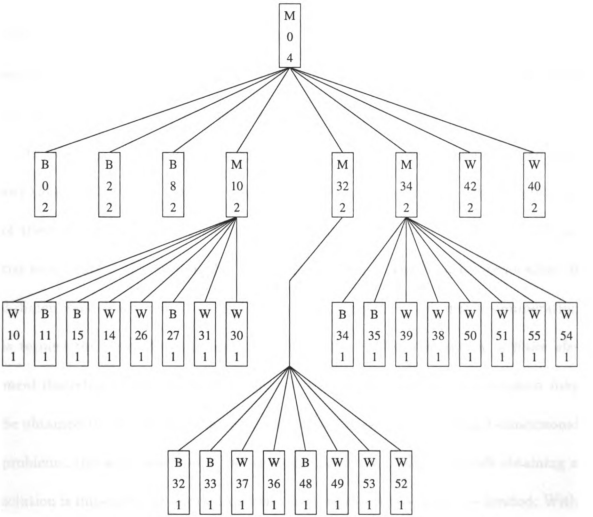

Figure 2.5: Octree data structure for the sample 3-dimensional domain

complete when all voxels in the octant possess *similar* grayscale values. The definition of similar grayscale values is dependent on the application at hand. In the case of topology optimization, similar voxels are voxels that satisfy a merge criterion based on the average voxel grayscale value (density) and an associated tolerance. This issue will be addressed in more detail in the following sections.

2.4 Octrees And 3D Hierarchical Discretization

The basic features of region octree representation were introduced in the previous sections. Here, the use of region octree representations to develop finite element models for grayscale volume data is discussed.

In topology optimization, grayscale values may be considered equivalent to density values. A *shape* in a topology optimization problem is defined by the distribution of these densities in a specified domain. Initially, the problem is defined by a spatial enumeration of a rectangular prism. Each voxel is described by its location in 3-dimensional space and its grayscale value (i.e., density). This spatial enumeration is termed the *voxel discretization.* Based on the voxel discretization, a *finite element* discretization of the shape is generated. A finite element discretization may be obtained by identifying each voxel with a finite element. In typical 3-dimensional problems, this approach would lead to extremely large models in which obtaining a solution is impossible or impractical when computational resources are limited. With the introduction of octree hierarchical data structures, the voxels may be *merged* together to form larger cube construction blocks. Thus, the same shape defined by the

voxel discretization can be approximated using the octree building blocks, the *octree cubes.* Each octree cube may be considered as a single finite element. Thus, with the introduction of octrees, the voxel discretization is used to characterize the *shape* of the structure, while the region octree discretization is used to characterize the finite element *mesh.*

2.4.1 Octrees And Finite Element Analysis

The simplest examples of hierarchical representation of volume data are problems where the volume data is binary, where the voxel discretization may be efficiently described by a string of 0s and 1s. Figure 2.6 displays a voxel description of a 3-dimensional truss (only the non-zero voxels are displayed). When applying region octrees to voxel data for finite element analysis, the truss is approximated by the hierarchical data structure's octree cubes. The octree representation of the same 3-dimensional truss is displayed in Figure 2.7. A significant reduction in the number of elements necessary to describe the truss is accomplished. The number of finite elements necessary to describe the 3-dimensional truss is about 26% of the number of non-zero voxels necessary to describe the same truss.

Because the voxel description in the truss is binary, the octree approximation of the original structure is equivalent to the voxel discretization description. Now the issue of grayscale voxel data is addressed. Figure 2.8 displays another 3-dimensional truss of a slightly different structure and with a grayscale density distribution. Figure 2.9 displays the octree representation of the grayscale voxel mesh. As in the binary

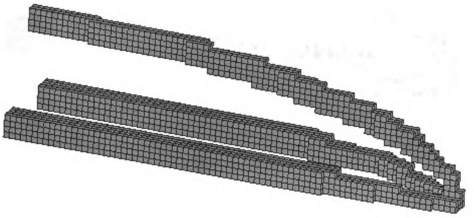

Figure 2.6: Voxel representation of a binary 3-dimensional truss: 2837 voxels

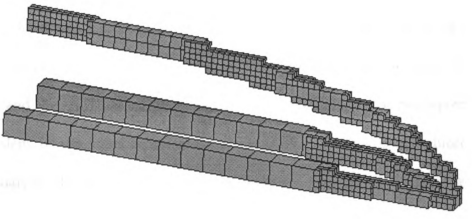

Figure 2.7: Octree representation of a binary 3-dimensional truss: 751 elements

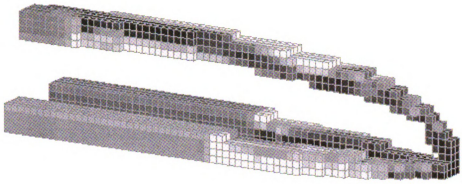

Figure 2.8: Voxel representation of a grayscale 3-dimensional truss: 2519 voxels

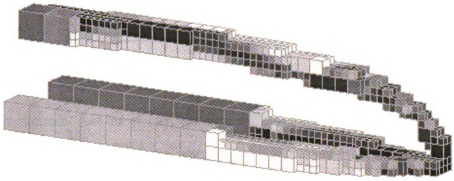

Figure 2.9: Octree representation of a grayscale 3-dimensional truss: 797 elements

example, the octree mesh contains far fewer elements than the number of voxels in the voxel level discretization. However, one difference is noted between the binary example and the grayscale example. In the binary example, the octree representation is equivalent to the voxel representation; no additional error is introduced by the octree analysis. In the grayscale example, there are portions of the truss where the boundary may be modified as results of the octree application. In addition, some of the density distribution is modified when voxels of "similar" density are merged. This is one drawback of octree representation of grayscale images. However, the benefits associated with the reduction in the model size heavily outweigh this drawback.

2.4.2 Octree Encoding Algorithm

The next issue to be addressed is choosing an algorithm to generate grayscale hierarchical representation using region octrees. Consider a *shape* discretization of domain Ω characterized by voxel densities, $r_1, r_2, \ldots, r_N$, where N represents the number of voxels in the domain and the voxels densities, $r_i \in [0,1]$, are analogous to voxel grayscale levels. A region octree encoding algorithm applied to this sequence can be summarized as follows:

Octree Encoding Algorithm

Step 1. Let $O = \{\Omega\}$, the initial octant partition, i.e., one cube containing the entire domain. Let $M = \{\emptyset\}$, the initial finite element mesh.

Step 2. For each element $o \in O$,

Break o into octants $o^{(i)}$, $i = 1, 2, \ldots, 8$. For each $o^{(i)}$, add $o^{(i)}$ to O if any of the voxels, j, in $o^{(i)}$ violate

$$\rho_{avg}^{(i)} - \rho_{tol}^{(i)} \leq \rho_j \leq \rho_{avg}^{(i)} + \rho_{tol}^{(i)}$$

where,

$$\rho_{tol}^{(i)} = \max\left[\rho_{tol_{min}}, \rho_{tol_{max}}\left(1 - \rho_{avg}^{(i)\,f}\right)\right] \tag{2.2}$$

$\rho_{avg}^{(i)}$ – Average density in the i^{th} octant
$\rho_{tol_{min}}$ – Minimum density tolerance
$\rho_{tol_{max}}$ – Maximum density tolerance
f – Density exponential weighting factor

else, add $o^{(i)}$ to M.

Step 3. Stop if O and M are unchanged. Otherwise, return to Step 2.

Step 1 initialized the octant partition, O, and the set of octree cubes, M. The set O initially contains one octant, which encloses the entire domain. The set M is initially empty. During the octree encoding, the set O will contain all the octants that contain voxel data that is inconsistent, i.e., those that do not meet the merge criterion in Step 2. The set M will contain all the octants that are consistent, labeling them as octree cubes.

The algorithm recursively examines the octants in the problem domain. Upon reaching an octant that does not meet the merge criterion, the octant is subdivided into eight sub-octants. To guarantee that the recursive subdivision of the initial octant aligns spatially with the voxel discretization, the voxel discretization must be enclosed in a cube. In addition, the dimension of one side of the cube measured in voxels, N, must be defined as $N = 2^n$ for some integer n. Therefore, Step 1 is complete when a single octant of size $2^n \times 2^n \times 2^n$ containing the domain is created and the set M is initialized.

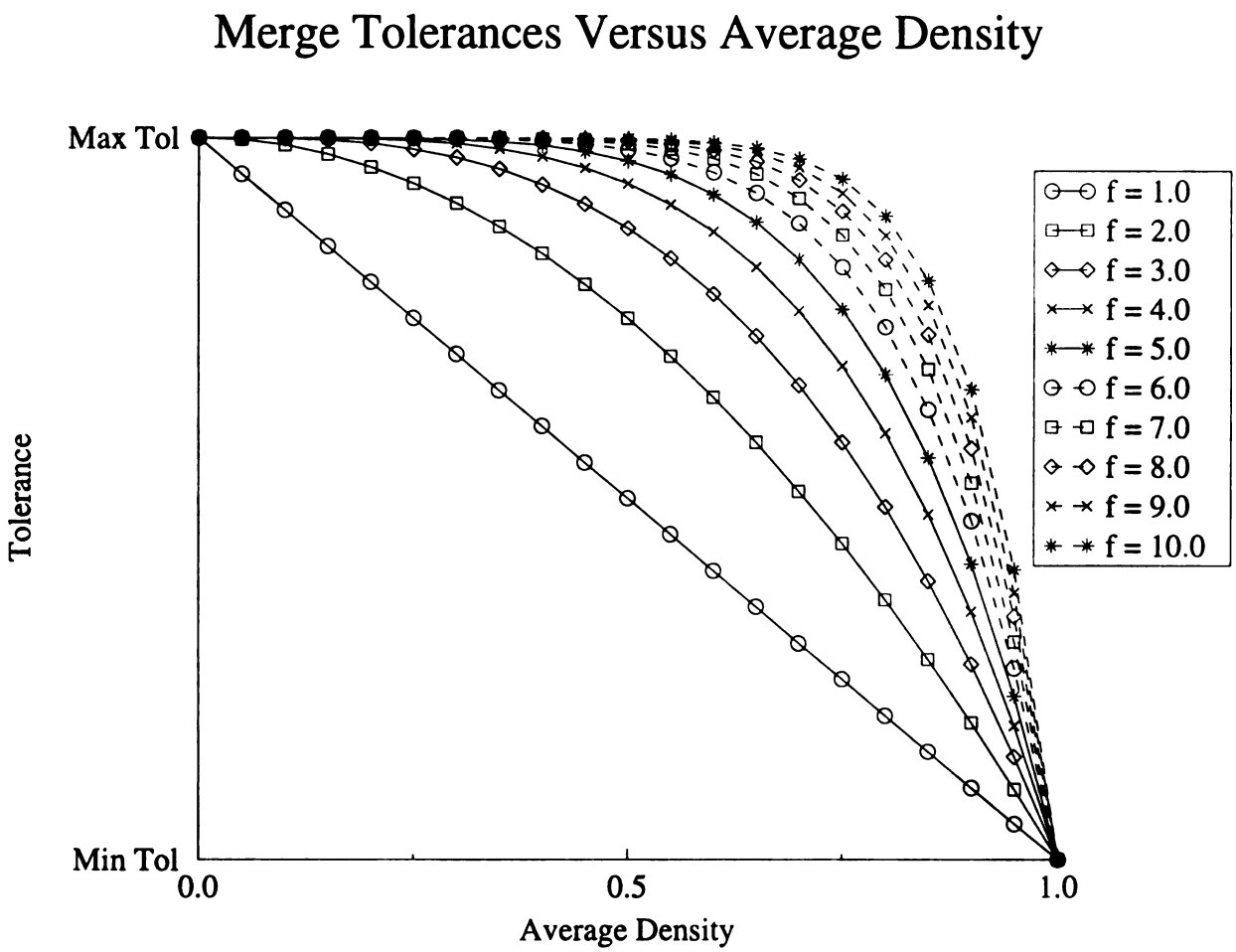

Figure 2.10: Voxel merge tolerances for various exponential weighting factors.

Step 2 contains the most notable difference between the octree encoding algorithm presented here and the algorithms used in binary octrees. In binary volume data analysis, voxels of proper spatial relationship (i.e., voxels that form a cube) are merged into an octree cube if all the voxel densities are equal. In this approach, voxels of proper spatial relationship are merged if all of the voxel densities are within a given tolerance. Furthermore, the tolerance is a function of the cube's average voxel density. Figure 2.10 displays sample density tolerances over a range of average voxel densities for various density exponential weighting factors. The selection of the minimum density tolerance, $\rho_{tol_{min}}$, the maximum density tolerance, $\rho_{tol_{max}}$, and density exponential weighting factor, f, are used to control the level of discretization achieved from the region octree representation. This type of merge criterion was developed to insure two basic effects:

1. Octree cubes consisting of low density voxels contain high merge tolerance values and merge easily. In topology optimization, large regions of low densities represent low material regions that are not essential to supporting the specified load. A coarse discretization is promoted in these regions.

2. Octree cubes consisting of high density voxels contain low merge tolerance values and are more difficult to merge. In topology optimization, these regions constitute the current structure, which is highly active in supporting the specified load. A fine discretization in these regions is beneficial.

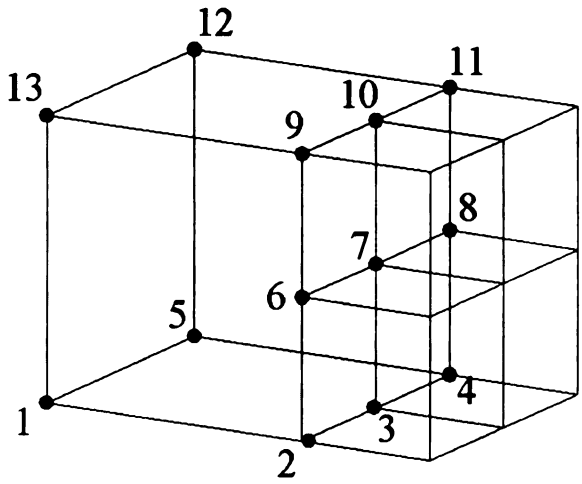

Figure 2.11: Octree mesh constrained nodes

2.4.3 Compatibility Across Element Boundaries

Octree-based finite element meshes typically violate compatibility conditions across boundaries between octants of different sizes. Constraint equations must be added to problem statement (1.2) to guarantee the appropriate smoothness of interpolation functions across element boundaries. These constraints apply to nodes on *edges* or *faces* shared by elements of different sizes. This is illustrated in Figure 2.11. Assuming that elements are the standard, isoparametric 8-noded bricks, equations

$$\begin{aligned} \mathbf{u}_3 &= \tfrac{1}{2}\mathbf{u}_2 + \tfrac{1}{2}\mathbf{u}_4 \\ \mathbf{u}_6 &= \tfrac{1}{2}\mathbf{u}_2 + \tfrac{1}{2}\mathbf{u}_9 \\ \mathbf{u}_8 &= \tfrac{1}{2}\mathbf{u}_4 + \tfrac{1}{2}\mathbf{u}_{11} \\ \mathbf{u}_{10} &= \tfrac{1}{2}\mathbf{u}_9 + \tfrac{1}{2}\mathbf{u}_{11} \\ \mathbf{u}_7 &= \tfrac{1}{4}\mathbf{u}_2 + \tfrac{1}{4}\mathbf{u}_4 + \tfrac{1}{4}\mathbf{u}_9 + \tfrac{1}{4}\mathbf{u}_{11} \end{aligned} \tag{2.3}$$

are needed to guarantee compatibility across edges (2-4), (2-9), (4-11), and (9-11), and face (2-4-9-11). In general, the equilibrium equations $\mathbf{Ku} - \mathbf{f} = \mathbf{0}$ in (1.2) are

replaced by

$$\begin{aligned} \text{Minimize:} \quad & F(\mathbf{u}) = \tfrac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{u}^T\mathbf{f} \\ \text{Subject To:} \quad & h(\mathbf{u}) = \mathbf{P}\mathbf{u} = \mathbf{0} \end{aligned} \tag{2.4}$$

where P is a $m \times k$ matrix of constant coefficients and m is the number compatibility constraints. The exact solution to this problem can be obtained using an augmented Lagrangian formulation. The augmented Lagrangian may be expressed as

$$L_c(\mathbf{u},\mathbf{v}) = L(\mathbf{u},\mathbf{v}) + \frac{1}{2}c\mathbf{h}^T\mathbf{h} \tag{2.5}$$

where L is the Lagrangian associated with 2.4 and may be expressed as

$$L(\mathbf{u},\mathbf{v}) = \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{u}^T\mathbf{f} + \{\mathbf{P}\mathbf{u}\}^T\mathbf{v} \tag{2.6}$$

and $\mathbf{v}$ is a vector of Lagrangian multipliers. At a stationary point, L_c must satisfy

$$\nabla L_c = \mathbf{K}\mathbf{u} - \mathbf{f} + \mathbf{P}^T\mathbf{v} + c\mathbf{P}^T\mathbf{P}\mathbf{u} = \mathbf{0} \tag{2.7}$$

and, if $\mathbf{u}$ is feasible

$$\mathbf{P}\mathbf{u} = \mathbf{0} \tag{2.8}$$

Combing (2.7) and (2.8), the system of equations becomes

$$\begin{bmatrix} \left[\mathbf{K} + c\mathbf{P}^T\mathbf{P}\right] & \mathbf{P}^T \\ \mathbf{P} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{2.9}$$

or in compact form

$$\tilde{\mathbf{K}}\tilde{\mathbf{u}} = \tilde{\mathbf{F}} \tag{2.10}$$

In this development, c is an arbitrary (finite but sufficiently large) scalar to make $\mathbf{K} + c\mathbf{P}^T\mathbf{P}$ non-singular.

2.5 Solution Algorithm For Hierarchical Topology Optimization Problems

An algorithm to solve hierarchical topology optimization problems will be introduced in this section. This discussion is based on the work of DeRose and Díaz[6]. The goal is to generate a sequence of solutions to topology optimization problems with increasing shape detail. First, the solution strategy used to solve the hierarchical topology optimization problems is presented. A brief discussion of the solution strategies implemented for the solution of finite element models with compatibility equations follows.

2.5.1 Solution Of Hierarchical Topology Optimization Problems

Octree discretizations can now be used to generate a sequence of solutions to the topology optimization problem of increasing shape detail. The process consists of six basic steps: *initialization*, *merge parameter determination*, *octree encoding*, *design database update*, *topology optimization*, and *octree convergence*. Figure 2.12 displays a flow chart of this procedure. The individual steps are outlined below.

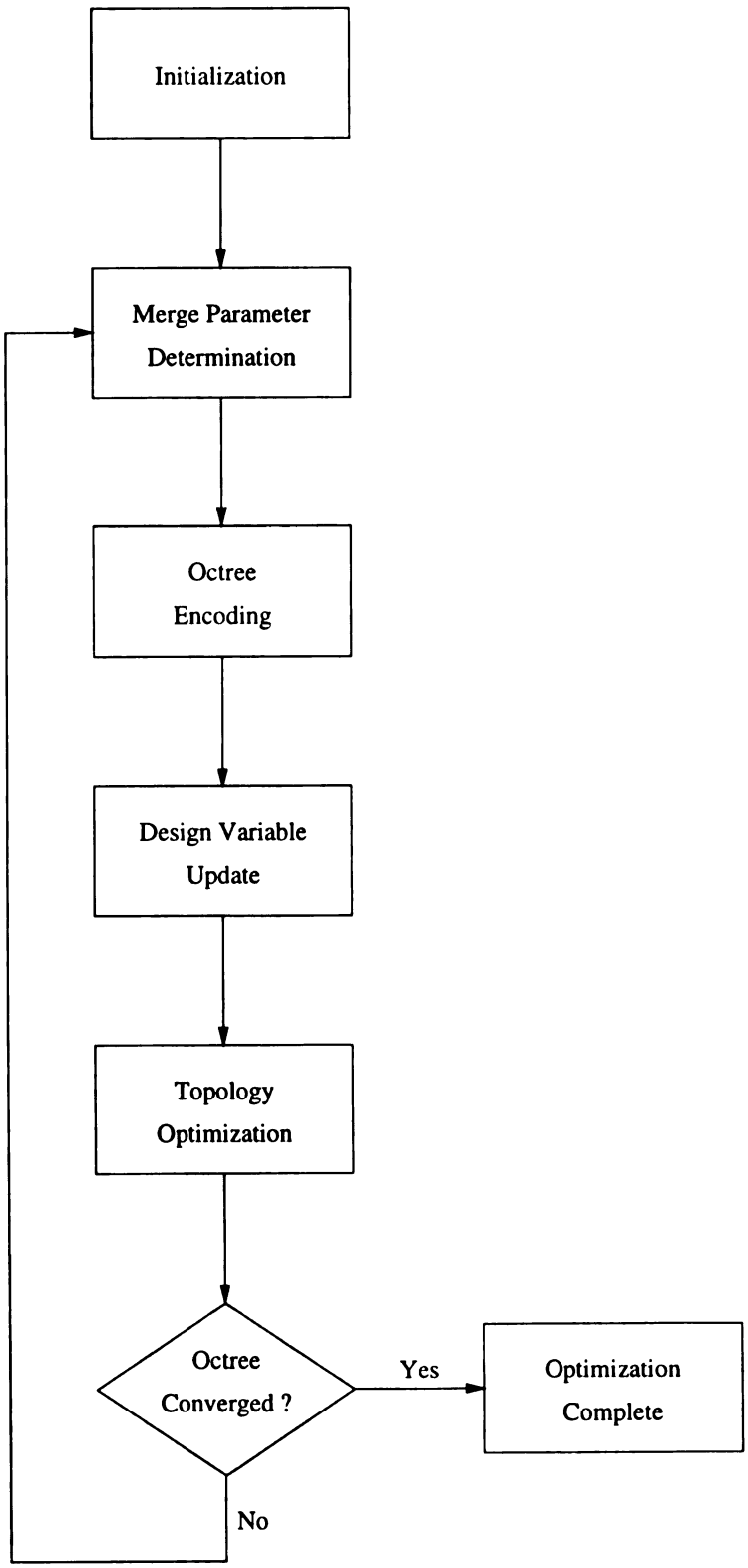

Figure 2.12: Octree iteration flow chart

1. **Initialization:** Set iteration counter $\epsilon = 0$. This step results in the generation of data that describe the design domain at the voxel level.

   (a) Start with the finest discretization of Ω. Fix the number of voxels, N, and set all voxel densities to $r_i = Vol/meas(\Omega)$.

   (b) Set the number of elements $n = N$ and $\rho_e = r_e$ for all elements e.

   (c) Initialize design variables (a_e, b_e, c_e) so that

$$\begin{aligned} a_e &= b_e = c_e \\ \rho_e &= 1 - (1 - a_e)(1 - b_e)(1 - c_e) \\ \psi_e &= \phi_e = \theta_e \ = \ 0 \end{aligned} \tag{2.11}$$

2. **Merge Parameter Determination:** Here, the merge parameters $\rho^{(\epsilon)}_{tol_{min}}$, $\rho^{(\epsilon)}_{tol_{max}}$, and $f^{(\epsilon)}$ are set. As in the work described in DeRose and Díaz [6], the values of $\rho^{(\epsilon)}_{tol_{min}}$ and $\rho^{(\epsilon)}_{tol_{max}}$ remain constant during the hierarchical analysis. However, the density exponential weighting factor is allowed to vary. For the remaining discussion, the merge parameters $\rho^{(\epsilon)}_{tol_{min}}$ and $\rho^{(\epsilon)}_{tol_{max}}$ will be referred to as $\rho_{tol_{min}}$ and $\rho_{tol_{max}}$. To determine the exponential weighting factor at each iteration, the following algorithm is used.

   (a) Apply the octree encoding as described in Step 3 for varying values of f at a fixed $\rho_{tol_{min}}$ and $\rho_{tol_{max}}$.

   (b) Determine a value for $f^{(\epsilon)}$ that yields a mesh with the desired number of elements.

   This requires another parameter to determine the number of elements that is desired in the next octree iteration. This growth factor parameter, g_f, controls

the percent increase in mesh size.

3. **Octree Encoding:** Here a new octree discretization of the domain Ω is generated. The goal is to (1) merge similar voxels into larger octants and (2) refine regions were the shape is changing. A convolution filter is applied to the voxel densities to accomplish these tasks [3, 16]. This step results in a new finite element mesh of size $n^{(\epsilon)}$ and a new set of element effective densities is achieved as follows.

   (a) Assign densities r_i for each voxel i so that $r_i = \rho_e$ whenever voxel i is contained in element e.

   (b) Apply the convolution filter

$$\hat{r}_i = \frac{1}{\sum_{j=1}^{N} H_{i,j}} \sum_{j=1}^{N} H_{i,j} r_j \tag{2.12}$$

   and,

$$H_{i,j} = \max[0, (R - \text{dist}(i,j))] \tag{2.13}$$

   where R corresponds to the filter radius, and the function, $\text{dist}(i,j)$, corresponds to the distance between the centroids of the i^{th} and j^{th} voxels. This particular filter has two qualities needed for successful octree analysis: (1) in large regions of similar density, a majority of the voxels are relatively unaffected and (2) regions of rapidly varying densities, such as structure boundaries, are heavily modified. This step results in a new sequence of voxel densities $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_N$

(c) Apply the octree encoding algorithm of Section 2.4.2 to generate a new octree partition $O^{(\epsilon)}$ and new finite element mesh $M^{(\epsilon)}$ from the sequence of voxel densities $\hat{r}_1, \hat{r}_2, \ldots, \hat{r}_N$. Densities $\rho_1, \rho_2, \ldots, \rho_{n^{(\epsilon)}}$ are assigned to each element in the partition. The amount of voxel merging achieved here depends on the parameters $\rho_{tol_{min}}$, $\rho_{tol_{max}}$, and $f^{(\epsilon)}$.

4. **Design Database Update:** In this step an updated set of design variables (element material microstructure parameters) is generated from the updated element densities.

   (a) For each element $e = 1, 2, \ldots, n^{(\epsilon)}$, generate an updated set of design variables $(\hat{a}_e, \hat{b}_e, \hat{c}_e)$ and $(\psi_i, \phi_i, \theta_i)$ from the densities $\rho_1, \rho_2, \ldots, \rho_{n^{(\epsilon)}}$ where

$$\begin{aligned} \hat{a}_i &= \alpha a_i \\ \hat{b}_i &= \alpha b_i \\ \hat{c}_i &= \alpha c_i \end{aligned} \tag{2.14}$$

   and α is obtained from

$$\rho_i = 1 - (1 - \hat{a}_i)(1 - \hat{b}_i)(1 - \hat{c}_i) \tag{2.15}$$

   The angles for each element, ψ_i, ϕ_i, and θ_i, are taken from the voxel with the largest strain energy contained in the element.

5. **Topology Optimization:** Here a full cycle of a standard topology optimization algorithm is applied to convergence. This step results in an updated set of design variables and corresponding effective densities $\rho_1, \rho_2, \ldots, \rho_{n^{(\epsilon)}}$

   (a) Solve the topology optimization problem of size $n^{(\epsilon)}$,

$$
\begin{array}{lll}
\text{Minimize:} & \frac{1}{L}\sum_{i=1}^{L}\int_{\Omega} < \mathbf{f^i}, \mathbf{u^i} > d\mathbf{u} & \\
\mathbf{x} & & \\
\text{Subject To:} & \int_{\Omega}\rho(\mathbf{x})\, d\Omega \;-\; \text{Vol} \;\leq\; 0 & \\
 & 0 \;\leq\; \rho_e \;\leq\; 1 \quad \text{for } e = 1, n^{(\epsilon)} & \\
 & \tilde{\mathbf{K}}\tilde{\mathbf{U}} - \tilde{\mathbf{F}} \;=\; \mathbf{0} & \\
\text{Where,} & \mathbf{x} \;=\; (a_1, b_1, c_1, \psi_1, \phi_1, \theta_1, \ldots, & \\
 & \quad a_{n^{(\epsilon)}}, b_{n^{(\epsilon)}}, c_{n^{(\epsilon)}}, \psi_{n^{(\epsilon)}}, \phi_{n^{(\epsilon)}}, \theta_{n^{(\epsilon)}}) & \\
 & \rho_e \;=\; 1 - (1-a_e)(1-b_e)(1-c_e) & \\
 & L \;=\; \text{number of load cases} &
\end{array}
\tag{2.16}
$$

6. **Octree convergence:** Declare convergence when the number of elements in the new mesh $n^{(\epsilon)}$ and the new density exponential factor, $f^{(\epsilon)}$, satisfy the convergence criteria.

2.5.2 Discussion Of The Hierarchical Topology Optimization Solution Algorithm

Section 2.5.1 presented an algorithm for solving the hierarchical topology optimization problem. Here, a more detailed discussion of the operations is provided. Each of the six procedures of *initialization*, *merge parameter determination*, *octree encoding*, *design database update*, *topology optimization*, and *octree convergence* will be addressed.

Initialization

This stage of the solution algorithm defines the voxel discretization. This discretization is the finest mesh that may be achieved during the analysis. (The goal, of course,

is never to solve a finite element problem at this level of discretization.) After initialization is complete, a complete voxel description of the domain is available, where the density of each voxel in the design domain is set to the same value adjusted to meet the volume constraint.

Merge Parameter Determination

This stage of the solution algorithm uses the octree discretizations for varying values of the density exponent weighting factor, f, to obtain a value for $f^{(\epsilon)}$ at each iteration. This is done by determining the number of elements introduced by performing the octree encoding algorithm for various values of f. A value of $f^{(\epsilon)}$ is determined by interpolating a value of f that produces a mesh with the number of elements as desired. In the problems solved here, the value of $f^{(\epsilon)}$ at each iteration is set to provide a prescribed increase in the number of elements describing the domain. This is controlled by a growth factor, g_f, usually set at a value of 10% to 25% to control the increase in the number of elements. Figure 2.13 shows a graphical representation of a relation between the density exponent weighting factor and the number of elements obtained in the octree discretization.

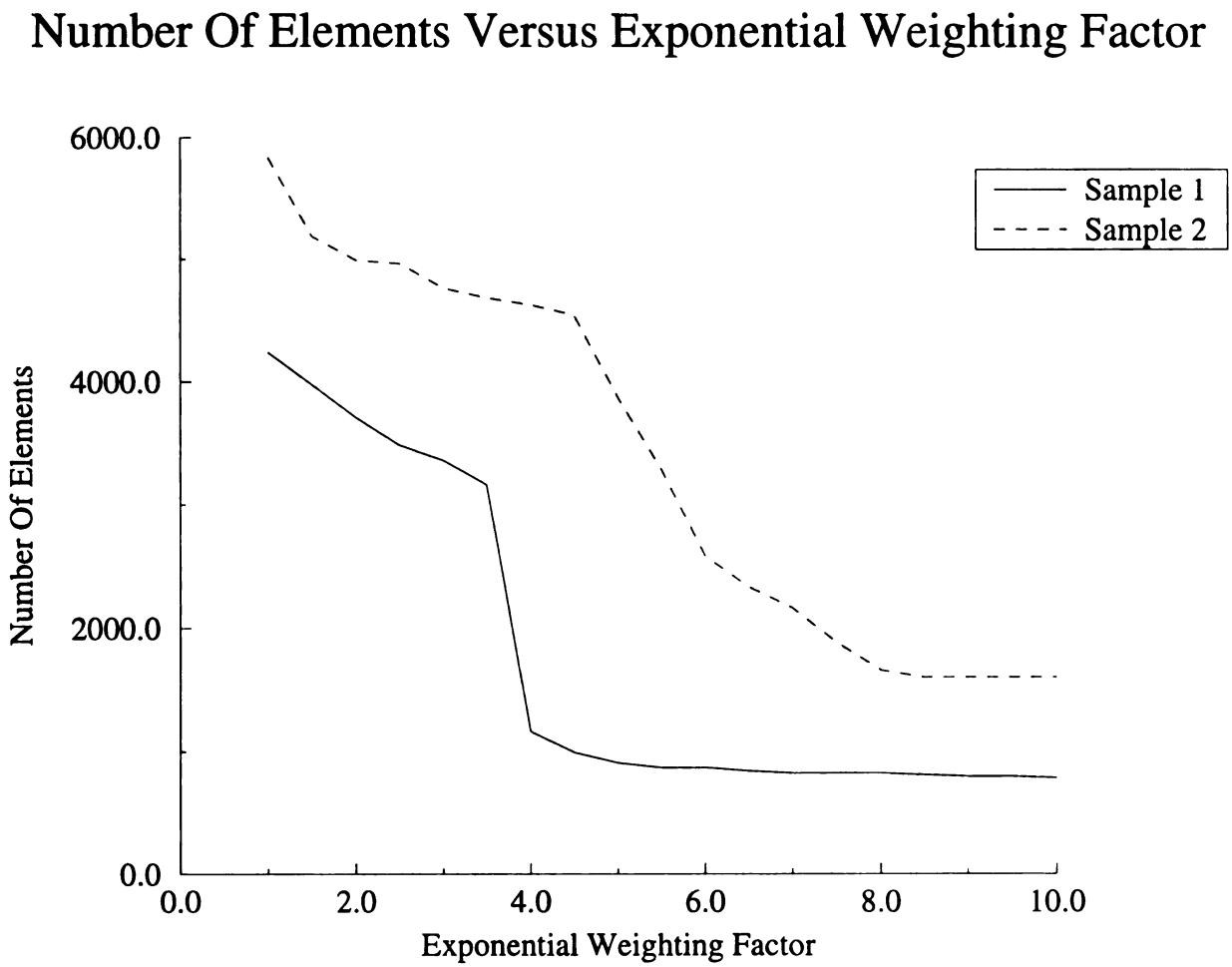

Figure 2.13: Sample relationship between the number of elements and the exponential weighting factor

Octree Encoding

The octree encoding stage is best described by distinguishing its application in two different instances:

1. The first octree encoding application, immediately following the initialization step

2. All other octree encoding applications

In the first instance of the octree encoding, all design voxels have been assigned the same density. The assignment of the voxel densities, $r_i = \rho_e$, is equivalent to the initialization stage. As the voxel densities are constant, the only octants that fail to contain consistent voxels are the octants that cross the boundary of the design domain. Therefore, the first application of octree encoding produces the coarsest finite element mesh that may be achieved based on the *geometry* of the design domain alone. In some situations, this procedure may lead to a very coarse mesh which may actually hinder the hierarchical analysis. In these cases, a maximum element size is specified to prevent this problem.

In all other applications of the octree encoding, a non-constant density distribution exists through the design domain and each step in the octree encoding strategy is relevant. First, the densities of the voxels contained in each finite element are assigned to the density of the element. This operation creates a spatial enumeration of the finite element density distribution at the voxel discretization level. Obviously, applying the octree encoding algorithm to this voxel density distribution would produce an octree representation containing, at most, the same number of elements that were

used originally to assign the voxels. This is due to the spatial dependence of the octree encoding algorithm and the element-wise constant density assignment to the voxels. However, with the application of a convolution filter at the voxel level, the voxel densities near high density gradient areas will be modified. This leads to non-constant voxel density distribution in areas of rapidly varying densities. The voxels that were once along the edges of large elements are modified by neighboring voxels when the filter is applied. The type of interaction enhances the border of the shape by forcing the large elements created in previous octree encodings to possess non-constant density distributions. Finally, the octree encoding algorithm is applied to the modified densities. Now, the large elements with modified density distributions may subdivide. Unlike the first instance of the octree encoding, this application will create finite element models based on the *geometry* of the design domain and the current *shape*.

Design Database Update

This stage simply solves equation (2.15), a cubic equation in α, for each new element to obtain the new design variables for the corresponding new density distribution. One issue arises with the assignment of the homogenization microcell orientation. How should cell angles be assigned when merging elements of similar densities but different angles? The orientation angles from the element with the highest strain energy are selected since this element contains the dominate material direction for the area of interest.

Topology Optimization

This stage of the hierarchical topology optimization algorithm uses standard topology optimization to solve the current octree generated finite element mesh. A complete description of the solution technique is provided in Appendix A. The only modifications necessary are related to the issue retaining compatibility across element boundaries, as discussed in Section 2.4.3. Section 2.5.3 will discuss the issue to a greater detail. Note, even though the hierarchical analysis may greatly reduce the number of finite elements in a given design domain, this step is still the most time consuming.

Octree Convergence

This stage of the hierarchical topology optimization algorithm determines if the iteration of hierarchical meshes have converged. There are five cases in which hierarchical topology optimization convergence is declared:

1. The number of elements in the current octree partition, $n^{(\epsilon)}$, exceeds a desired maximum number of elements.

2. The density exponential weighting factor, $f^{(\epsilon)}$, is lower than the minimum acceptable weighting factor.

3. The selection of the density exponential weighting factor, $f^{(\epsilon)}$, converges. (i.e., $f^{(\epsilon)}$ stabilizes for a few hierarchical iterations.)

4. The maximum number of octree iterations is reached.

5. No value for the density exponential weighting factor will produce enough elements to satisfy the growth factor, g_f.

The mechanisms for declaring convergence of the hierarchical topology optimization may be controlled by various parameters. These parameters are the maximum number of elements allowed, the minimum value of $f^{(e)}$, the maximum number of hierarchical iterations, and the growth factor, g_f. Depending on the nature of the problem addressed, these convergence parameters must be set insightfully to obtain meaningful results.

2.5.3 Solution Of Octree Generated Finite Element Models

From the description of the solution algorithm for the hierarchical topology optimization problem presented in Section 2.5.2, it is clear that much of the computation resources are still used to solve the equilibrium equations. With the use of hierarchical data structures, a significant reduction in the number of elements is achieved at the expense of introducing the associated constraint equations to guarantee compatibility as discussed in Section 2.4.3. This section discusses various solution techniques that were developed to solve the constrained problem and the benefits and shortcomings of each.

The first attempt is to solve the complete system of equations as stated in equation (2.17). Four different types of solvers are examined: constrained skyline, element-by-element, sparse, and penalized skyline. Again, the goal of this section is to obtain solutions for the displacement vector, $\mathbf{u}$, in the system of equations

$$\begin{bmatrix} [\mathbf{K} + c\mathbf{P}^T\mathbf{P}] & \mathbf{P}^T \\ \mathbf{P} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{2.17}$$

Below is a brief description of each one of these strategies

- **Constrained Skyline** - In the constrained skyline solver, the matrix, $\mathbf{K}$, is assembled and stored in skyline form. The constraint matrix, $\mathbf{P}$, is stored in sparse form. Therefore, the storage requirements for the data alone are similar to those of the standard skyline approach. The full set of equations, $\tilde{\mathbf{K}}$, is then constructed. The main disadvantage of the constrained skyline solver lies in its use of memory and slow speed during the LU decomposition. Here, the additional storage necessary to store the LU decomposition of $\tilde{\mathbf{K}}$ and time required to deal with the constraints makes the constrained skyline solver very inefficient for most applications with more than a few constraints. Unfortunately, in the sequence of octree generated finite element mesh, the number of constraint equations may grow rapidly, decreasing the effectiveness of the constrained skyline solver.

- **Element-By-Element** - This solver is implemented using a preconditioned conjugate gradient technique as shown by Winget and Hughes [20]. The main benefit of the element-by-element solver is the reduction of memory necessary to solve large systems of equations. The main drawback is that the solution is achieved via an iterative algorithm, where all the element stiffness matrices must be recomputed at each iteration. This solver is implemented to solve the constrained set of equations (2.17) and requires substantially less memory than

the constrained skyline solver. However, it requires the most CPU time of the solvers examined.

- **Sparse** - This solver is implemented using sparse matrix techniques. The main advantage of this solver is a reduction in the amount of memory necessary to solve the system of equations compared to the constrained skyline solver. This solver also provides results in less time. The main disadvantage of the sparse solution is the interaction between elemental stiffness matrices and the sparse storage technique used to store the global stiffness matrix. Unlike skyline storage techniques, the sparse matrix storage does not easily allow the construction of the system equations from a traversal of the finite elements. See [13] for a more complete discussion of the sparse solvers.

- **Penalized Skyline** - In a penalty approach, the system

$$\left[\mathbf{K} + c\mathbf{P}^T\mathbf{P}\right] \mathbf{u} = \mathbf{f} \tag{2.18}$$

is solved for $\mathbf{u}$. This set of equations results in reductions in both computation memory and time requirements compared to the constrained skyline solver. Also, the constraint penalty terms $c\mathbf{P}^T\mathbf{P}$ may simply be added to a standard skyline representation of $\mathbf{K}$, which implies that memory requirements are essentially independent of the number of constraints. The solution of this problem is implemented with a standard skyline solution technique. Setting the penalty parameter, c, to be sufficiently large (e.g., equal to one thousand times the largest entry in the stiffness matrix) provides displacement results essentially

identical to those obtained with the other constrained solvers. This solver requires less memory than the constrained skyline solver and produces results in less time.

The solution of the reduced system of equations is not implemented in the sparse solver or element-by-element solver, but reasonable gains are expected in each.

2.6 Summary

This chapter introduced the basic concepts of hierarchical data structures and how they may be applied to topology optimization. In this discussion, the theory of hierarchical analysis of binary data was presented with examples. These concepts were then expanded with grayscale data with a corresponding example. Next, an algorithm for hierarchical topology optimization was presented. Here, the methods of applying hierarchical data structures to topology optimization were discussed. In the following chapters, examples of these ideas will be presented.

Chapter 3

Shape Analysis And Topology Optimization

3.1 Introduction

Chapter 2 discussed the methods of applying hierarchical data structures to topology optimization to achieve reduction in computational resources. This chapter demonstrates the benefits that may be achieved in the area of *shape control*. The main objective is to develop shape analysis tools that provide additional control of the optimal shape obtained from standard topology optimization procedures.

3.2 Shape Analysis

The main focus in shape analysis has centered on the area of image processing for pattern recognition. Tomita and Tsuji [19] and Pratt [12] provide an excellent overview

of shape analysis in the field of pattern recognition. Below is a brief discussion of the main tools used in shape analysis.

- *Scalar Shape Descriptors* - Scalar descriptors are among the simplest shape descriptors to compute. Examples of scalar shape descriptors are area, perimeter, circularity, size, and spatial moments. The main advantage of scalar shape description is the efficiency of its computation.

- *Fourier Shape Descriptors* - Fourier descriptors are most often used to analyze the perimeter of a given shape. Using periodic curvature function over the perimeter length, the curvature is expanded in a Fourier series. The coefficients of this expansion are used in comparing various shapes. The main advantage of Fourier descriptors is that a shape may be efficiently approximated by a truncated set of Fourier coefficients.

- *Medial-Axis Transformation* - The Medial-Axis Transformation provides a description of the distance between the medial-axis and the edge of the shape. The main advantage of the medial-axis description is that operations such as region thinning and region decomposition may be easily performed with the resulting data.

- *Boundary Segmentation* - Boundary Segmentation analysis approximates the shape's boundary by a set of line or curve segments. This is one of the most common algorithms used for shape analysis. The main benefit of Boundary Segmentation is the simplicity of the results obtained from the analysis, namely, a set of line or curve segments.

From the list of shape analysis methods listed above, scalar shape descriptors were chosen for the work presented here. The main reasons for their selection are as follows:

- *Computational Resources* - Scalar shape descriptors require relatively small amounts of computational resources.

- *Ease of Computation* - Most shape analysis tools are developed for the analysis of 2-dimensional shapes. Due to the ease of computing scalar shape descriptors, only small modifications are necessary to transform the 2-dimensional operators to 3-dimensional operators.

- *Interaction With Optimization Problems* - Since scalar shape descriptors are *scalars*, they may be more easily included in standard optimization problems as additional constraints or as additional terms in the objective function.

The next section provides a more detailed discussion of scalar shape descriptors.

3.3 Scalar Shape Descriptors

In 2-dimensional image analysis, scalar shape descriptors are used to roughly describe the image. The most common shape descriptors are area, perimeter, circularity, size, and spatial moments. In the analysis of 2-dimensional images, we may define an intensity function, $f(x, y)$, which defines the intensity level of each location of the image. In most image processing applications, the intensity is defined over a discrete set points (i.e., pixels). Figure 3.1 displays a 2-dimensional setting where scalar shape

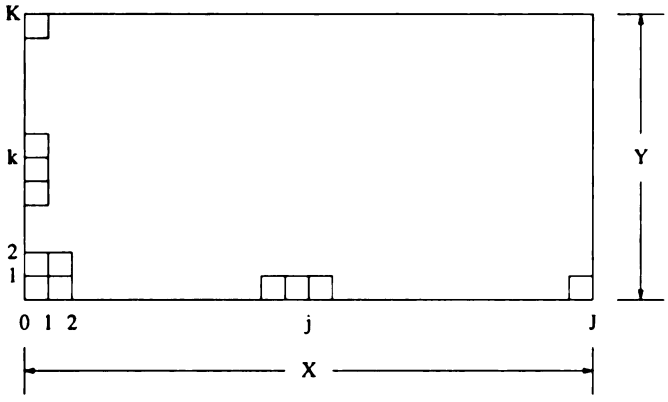

Figure 3.1: 2-dimensional domain for spatial moment application

descriptors may be applied. The center of a pixel (x_j, y_k) may be determined by

$$x_j = j - \frac{1}{2} \tag{3.1}$$

$$y_k = k - \frac{1}{2} \tag{3.2}$$

With a given image description and corresponding intensity function, spatial moments in 2-dimensions may be defined as follows:

- *Unscaled Spatial Moments* -

$$M_U(r,s) = \sum_{j=1}^{J} \sum_{k=1}^{K} x_j^r y_k^s f(x_j, y_k) \tag{3.3}$$

- *Scaled Spatial Moments* -

$$M(r,s) = \frac{1}{X^r Y^s} \sum_{j=1}^{J} \sum_{k=1}^{K} x_j^r y_k^s f(x_j, y_k) \tag{3.4}$$

- *Unscaled Spatial Central Moments* -

$$U_U(r,s) = \sum_{j=1}^{J}\sum_{k=1}^{K}[x_j - \bar{x}]^r[y_k - \bar{y}]^s f(x_j, y_k) \tag{3.5}$$

where,

$$\bar{x} = \frac{M(1,0)}{M(0,0)}; \quad \bar{y} = \frac{M(0,1)}{M(0,0)} \tag{3.6}$$

- *Scaled Spatial Central Moments* -

$$U(r,s) = \frac{1}{X^r Y^s}\sum_{j=1}^{J}\sum_{k=1}^{K}[x_j - \hat{x}]^r[y_k - \hat{y}]^s f(x_j, y_k) \tag{3.7}$$

where,

$$\hat{x} = \frac{U_U(1,0)}{U_U(0,0)}; \quad \hat{y} = \frac{U_U(0,1)}{U_U(0,0)} \tag{3.8}$$

The benefits of the spatial moments are clear from their definition. First, the spatial moments are relatively simple to compute. Second, the spatial moments may vary in spatial order. Namely, varying the parameters r and s will enforce various spatial weightings for a given type of spatial moment.

3.4 Shape Descriptors And Topology Optimization

As noted previously, recent work of Haber, Jog, and Bendsøe [4] has focused on solving 2-dimensional topology optimization problems with constraints on the structure's perimeter. Here, spatial moments as discussed previously are slightly modified to interact with topology optimization problems. In image processing, spatial moments are defined as operators on the intensity level. In this topology optimization problems, spatial moments are defined as operators on the density levels. In addition, the common 2-dimensional definitions of spatial moments are expanded to 3-dimensions. The first type spatial moment considered is that centered about a point. These spatial moments are defined in continuous form as follows:

- *Unscaled Spatial Point Moments* -

$$M_{pU}(r,s,t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[x-\tilde{x}]^r[y-\tilde{y}]^s[z-\tilde{z}]^t\,\rho(x,y,z)\,dx\,dy\,dz \quad (3.9)$$

- *Scaled Spatial Point Moments* -

$$M_p(r,s,t) = \frac{1}{X^r Y^s Z^t}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}[x-\tilde{x}]^r[y-\tilde{y}]^s[z-\tilde{z}]^t\,\rho(x,y,z)\,dx\,dy\,dz \quad (3.10)$$

In both unscaled and scaled spatial point moments, $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ specify the location at which the spatial moment is centered. In three dimensions, it is often useful to consider a moment centered about a line in space rather than a single point. These spatial moments are defined in continuous form as follows:

- *Unscaled Spatial Line Moments* -

$$M_{l_U}(r) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} d(x,y,z)^r \, \rho(x,y,z)\, dx\, dy\, dz \tag{3.11}$$

- *Scaled Spatial Line Moments* -

$$M_l(r) = \frac{1}{X^r Y^s Z^t}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} d(x,y,z)^r \, \rho(x,y,z)\, dx\, dy\, dz \tag{3.12}$$

In both the unscaled and scale spatial line moments, the function $d(x,y,z)$ represents the distance between the point (x,y,z) and the line that the moment in centered about.

When solving topology optimization problems, a discretized model of the continuous domain is developed. The entire optimization process depends upon this discretization. Hence, the definitions for the discrete versions of the 3-dimensional spatial moments are provided. The following definitions assume that the domain of size $(X \times Y \times Z)$ is divided into N elements of equal size, similar to voxels. In addition, the location of an element's centroid of a given element, e, may be expressed as (x_e, y_e, z_e); and the element's density values may be expressed as ρ_e. Assuming that all the elements in the domains are the same size, the 3-dimensional version of the

discrete spatial moments may be defined as follows:

- *Discrete Unscaled Spatial Point Moments* -

$$M_{pU}^{D}(r,s,t) \;=\; \sum_{e=1}^{N} [x_e - \tilde{x}]^r [y_e - \tilde{y}]^s [z_e - \tilde{z}]^t \, \rho_e \tag{3.13}$$

- *Discrete Scaled Spatial Point Moments* -

$$M_{p}^{D}(r,s,t) \;=\; \frac{1}{X^r Y^s Z^t} \sum_{e=1}^{N} [x_e - \tilde{x}]^r [y_e - \tilde{y}]^s [z_e - \tilde{z}]^t \, \rho_e \tag{3.14}$$

- *Discrete Unscaled Spatial Line Moments* -

$$M_{lU}^{D}(r) \;=\; \sum_{e=1}^{N} d(x_e, y_e, z_e)^r \, \rho_e \tag{3.15}$$

- *Discrete Scaled Spatial Line Moments* -

$$M_{l}^{D}(r) \;=\; \frac{1}{X^r Y^s Z^t} \sum_{e=1}^{N} d(x_e, y_e, z_e)^r \, \rho_e \tag{3.16}$$

Once again, the overall goal is to provide additional control of the optimal shape found using standard topology optimization techniques. To achieve this, the standard problem statement (1.2) is modified to include spatial moment constraints. Assuming that some information is known about the desired shape, a number of spatial moments

may be computed for the desired shape. The optimization is then constrained to produce optimal shapes that possess similar spatial moments. In practice, this may be useful in at least two situations:

1. A standard topology optimization solution exists for the problem at hand and designers are not satisfied the with standard results. Often, designers desire modifications to the optimal shape without significant decay in the structural stiffness.

2. Additional constraints, such as manufacturing issues, exist prior to solving the problem that places addition shape restrictions. In these situations, the design domain may be unaffected by the manufacturing issues, such as connectivity and slenderness. However, there may exist a desire to address these issues during the optimization, producing results that satisfy the manufacturing requirements.

Assuming that some information is available that leads to the use of spatial moment constraints, the issue of formulating a topology optimization problem with spatial moment constraints is addressed.

Problem statement (3.17), which is based on problem statement (1.2), provides the desired optimization problem that integrates standard topology optimization and spatial moment control. Here, one may constrain an arbitrary number of spatial moments, N_M, in a standard topology optimization setting. The moments of interest, M_i, may be any of the spatial moments describe previously and may be of any spatial order, controlled by the data vectors: $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{t}$.

$$
\begin{aligned}
\text{Minimize:} \quad & \frac{1}{L}\sum_{i=1}^{L}\int_{\Omega} < \mathbf{f}^{(i)}, \mathbf{u}^{(i)} > d\mathbf{u} \\
\mathbf{x} & \\
\text{Subject To:} \quad & \int_{\Omega} \rho(\mathbf{x})\, d\Omega \;-\; \text{Vol} \;\leq\; 0 \\
& 0 \;\leq\; \rho_e \;\leq\; 1 \quad \text{for } e = 1, \text{n} \\
& M_i(r_i, s_i, t_i) \;=\; \overline{M}_i \quad \text{for } i = 1, N_M \\
& \mathbf{KU} \;-\; \mathbf{F} \;=\; \mathbf{0}
\end{aligned}
\tag{3.17}
$$

Where,

N_M	=	number of constrained spatial moments.
M_i	=	i^{th} spatial moment of interest.
r_i	=	i^{th} spatial moment x-component exponent.
s_i	=	i^{th} spatial moment y-component exponent.
t_i	=	i^{th} spatial moment z-component exponent.
$\overline{M}_i$	=	i^{th} constrained spatial moment value.

Next, the issue of solving the topology optimization problem with spatial moments is addressed.

3.5 Solving Topology Optimization Problems With Shape Descriptors

The topology optimization problem expressed in problem statement (3.17) may not be solved using the same optimality conditions algorithm used to solve the standard topology optimization problem. Two solution techniques were considered in order to solve this problem:

1. Use sequential non-linear programming techniques to solve the problem as stated.

2. Reformulate the problem to be compatible with a modified version of the optimality conditions algorithm.

Although the problem could be solved directly as stated using sequential non-linear programming, it requires development of a completely new solution technique. Hence, the second option was selected. In order to use the optimality conditions algorithm, the problem is first modified using exterior penalty function methods, see Haftka and Gurdal [5]. With penalization methods, the moment constraints were modeled as penalty terms added to the objective function. This new formulation may be seen in problem statement (3.18)

$$
\begin{array}{lll}
\underset{\mathbf{x}}{\text{Minimize:}} & \frac{1}{L}\sum_{i=1}^{L}\int_{\Omega} < \mathbf{f^i}, \mathbf{u^i} > d\mathbf{u} \;+\; \alpha_p \sum_{j=1}^{N_M}\left(\frac{M(r_i,s_i,t_i)}{\overline{M}_i} - 1\right)^2 & \\
\text{Subject To:} & \int_{\Omega}\rho(\mathbf{x})\, d\Omega \;-\; \text{Vol} \;\leq\; 0 & \\
 & 0 \;\leq\; \rho_e \;\leq\; 1 \quad \text{for } e = 1, \text{n} & \\
 & \mathbf{KU} \;-\; \mathbf{F} \;=\; \mathbf{0} &
\end{array}
\tag{3.18}
$$

where the scalar, α_p, is the penalty associated with the violation of the moment constraints. In restating the problem as shown in (3.18), two issues arise:

1. Solving an optimization problem that models a set of equality constraints as a penalty term added to the objective function

2. Solving the topology optimization problem with a penalty term in the objective function

The first issue is relatively straight-forward. Much work has been performed in modeling constrained optimization problems as unconstrained optimization problems with penalization. These ideas may be expanded to handle constrained optimization with penalization as is the case here. Of the available methods, exterior penalty methods are selected to solve topology optimization with penalized spatial moment constraints.

The second issue has also been addressed in previous work, but in a slightly different manner. The same type of situation arises in topology optimization when a restriction is placed on the optimal structure's eigenvalues. The following sections will address these issues more closely.

3.5.1 Exterior Penalty Methods

In order to explain exterior penalty methods, a simplified problem statement will be addressed as shown in (3.19).

$$\begin{aligned} &\underset{\mathbf{x}}{\text{Minimize:}} && f(\mathbf{x}) \\ &\text{Subject To:} && g_i(\mathbf{x}) \leq 0 \qquad \text{for } i = 1, m \\ &&& h_j(\mathbf{x}) = 0 \qquad \text{for } j = 1, n \end{aligned} \tag{3.19}$$

Here, standard optimization nomenclature has been used, $\mathbf{x}$ represents the design vector, f represents the objective function, g represents the m inequality constraints, and h represents the n equality constraints. Problem statement (3.19) is denoted as the standard optimization problem. To use penalization methods, one must modify the problem statement as shown in (3.20).

$$\begin{aligned} &\underset{\mathbf{x}}{\text{Minimize:}} && f(\mathbf{x}) + \alpha_p \textstyle\sum_{j=1}^{n} h_j(\mathbf{x})^2 \\ &\text{Subject To:} && g_i(\mathbf{x}) \leq 0 \qquad \text{for } i = 1, m \end{aligned} \tag{3.20}$$

The optimization problem has been reduced from a problem with both equality and inequality constraints to a problem with purely inequality constraints. This is not always the case but is the type of problem addressed here. Problem statement (3.20)

is denoted as the penalized optimization problem. Now, the algorithm for solving the penalized optimization problem is presented. The basis of the algorithm is to solve the penalized optimization problem for increasing values of α_p, where the increase in α_p is controlled by a penalty growth factor, r_p. This is an iterative process that terminates when the optimal solution of penalized optimization problem produces results that satisfy each of the equality constraints from the standard optimization problem, $h_j(\mathbf{x})$, within a given tolerance, e_j. Below is more detailed description of the penalty method.

Penalty Method Algorithm

Step 1. Initialize the penalty parameter $\alpha_p^{(0)}$ and the counter $\epsilon = 0$.

Step 2. Solve the optimization using standard methods to convergence using $\alpha_p^{(\epsilon)}$ in the penalty term. Assign the optimum results to $\mathbf{x}^{(\epsilon)}$.

Step 3. Check the violation of the constraints modeled as penalty terms. If the error tolerance in the constraints is met, $h_j(\mathbf{x}^{(\epsilon)}) < e_j$ for $j = 1, n$, declare $\mathbf{x}^{(\epsilon)}$ as the optimum solution of (3.19) and terminate, else continue.

Step 4. Update the penalty parameter with $\alpha^{(\epsilon+1)} = r_p \alpha^{(\epsilon)}$. Increment the counter, $\epsilon = \epsilon + 1$. Goto Step 2.

The convergence of the penalty method may be controlled by setting the initial value of the penalty parameter, $\alpha_p^{(0)}$; the penalty growth factor, r_p; and the tolerances, e_j. These parameters must be selected carefully to promote rapid convergence of the penalized optimization problem at each iteration. Now that a general procedure has

been developed to solve penalized optimization problems, its application to spatial moment constrained topology optimization problems is presented.

3.5.2 Solution of Moment Constrained Topology Optimization Problems

Problem statement (3.17) provides a general description of the constrained topology optimization problem of interest. This problem statement is expanded and converted to a penalized topology optimization problem as seen in (3.18). Now, we must determine a method for solving this problem. As stated previously, the solution technique to be used is based upon a slightly modified version of the optimality conditions algorithm used in standard topology optimization.

First, the penalized spatial moment constrained topology optimization statement must be defined in complete form. For simplicity, only a single load case is considered.

$$
\begin{array}{lll}
\text{Minimize:} & & \\
\mathbf{x} & \int_\Omega < \mathbf{f}, \mathbf{u} > d\mathbf{u} \;+\; \alpha_p \sum_{j=1}^{N_M} \left(\frac{M(r_i, s_i, t_i)}{\bar{M}_i} - 1 \right)^2 & \\
 & & \\
\text{Subject To:} & \int_\Omega \rho(\mathbf{x})\, d\Omega \;-\; \text{Vol} \;\leq\; 0 & \\
 & -a_e \leq 0 & \text{for } e = 1, \text{N} \\
 & a_e - 1 \leq 0 & \text{for } e = 1, \text{N} \\
 & -b_e \leq 0 & \text{for } e = 1, \text{N} \\
 & b_e - 1 \leq 0 & \text{for } e = 1, \text{N} \\
 & -c_e \leq 0 & \text{for } e = 1, \text{N} \\
 & c_e - 1 \leq 0 & \text{for } e = 1, \text{N} \\
 & \mathbf{K(x)u} \;-\; \mathbf{f} \;=\; \mathbf{0} &
\end{array}
\tag{3.21}
$$

$$
\begin{array}{lll}
\text{Where,} & \mathbf{x} \;=\; (a_1, b_1, c_1, \psi_1, \phi_1, \theta_1, \ldots, & \\
 & \quad a_N, b_N, c_N, \psi_N, \phi_N, \theta_N) & \\
 & \rho_e \;=\; 1 \;-\; (1 - a_e)(1 - b_e)(1 - c_e) & \\
 & N \;=\; \text{The number of finite elements.} &
\end{array}
$$

When using an optimality conditions algorithm, one must determine under which

properties an optimal solution for a given problem must satisfy. Using standard optimization techniques, the Lagrangian function may be developed for the penalized moment constrained optimization statement. In this situation, the Lagrangian may be expressed as

$$
\begin{aligned}
\mathcal{L} \;=\; \int_{\Omega} <\mathbf{f},\mathbf{u}>\, d\mathbf{u} \;&+\; \alpha_p \textstyle\sum_{j=1}^{N_M} \left(\frac{M(r_i,s_i,t_i)}{\overline{M}_i} - 1\right)^2 \\
&+\; \Lambda\left[\int_{\Omega} \rho\, d\Omega \;-\; Vol\right] \\
&+\; \sum_{e=1}^{N} \alpha_e^{+}(a_e \;-\; 1) \;+\; \sum_{e=1}^{N} \alpha_e^{-}(-a_e) \\
&+\; \sum_{e=1}^{N} \beta_e^{+}(b_e \;-\; 1) \;+\; \sum_{e=1}^{N} \beta_e^{-}(-b_e) \\
&+\; \sum_{e=1}^{N} \gamma_e^{+}(c_e \;-\; 1) \;+\; \sum_{e=1}^{N} \gamma_e^{-}(-c_e) \\
&-\; \mu^T[\mathbf{f} \;-\; \mathbf{K}\mathbf{u}]
\end{aligned}
\tag{3.22}
$$

where, Λ represents the Lagrangian multiplier for the volume constraint and μ contains the Lagrangian multipliers for the equilibrium equation constraint. In addition, $\alpha_e^+, \beta_e^+, \gamma_e^+, \alpha_e^-, \beta_e^-$, and γ_e^- represent the Lagrangian multipliers for the upper and lower constraints on the design variables. In discrete form, the Lagrangian may be represented as

$$
\begin{aligned}
\mathcal{L}_D \;=\; \mathbf{f}^T\mathbf{u} \;&+\; \alpha_p \textstyle\sum_{j=1}^{N_M} \left(\frac{M(r_i,s_i,t_i)}{\overline{M}_i} - 1\right)^2 \\
&+\; \Lambda\left[\sum_{e=1}^{N} \rho_e v_e \;-\; Vol\right] \\
&+\; \sum_{e=1}^{N} \alpha_e^+(a_e \;-\; 1) \;+\; \sum_{e=1}^{N} \alpha_e^-(-a_e) \\
&+\; \sum_{e=1}^{N} \beta_e^+(b_e \;-\; 1) \;+\; \sum_{e=1}^{N} \beta_e^-(-b_e) \\
&+\; \sum_{e=1}^{N} \gamma_e^+(c_e \;-\; 1) \;+\; \sum_{e=1}^{N} \gamma_e^-(-c_e) \\
&-\; \mu^T[\mathbf{f} \;-\; \mathbf{K}\mathbf{u}]
\end{aligned}
\tag{3.23}
$$

The main difference between standard topology optimization and penalized spatial moment constrained topology optimization arise in the stationary conditions related to the design variables,

$$
\frac{\partial \mathcal{L}_\mathcal{D}}{\partial a_e} \;=\; \frac{\partial \mathcal{L}_\mathcal{D}}{\partial b_e} \;=\; \frac{\partial \mathcal{L}_\mathcal{D}}{\partial c_e} \;=\; 0 \quad \text{for } e \;=\; 1,\; N \tag{3.24}
$$

All three of these stationary conditions possess for the same structure for each element. Below the expression for $\frac{\partial \mathcal{L}_\mathcal{D}}{\partial a_e}$ is displayed

$$
\begin{aligned}
\frac{\partial \mathcal{L}_\mathcal{D}}{\partial a_e} \;=\; 2\alpha_p &\left[\sum_{j=1}^{N_M} \left(\frac{M(r_i,s_i,t_i)}{\overline{M}_i} - 1\right)\left(\frac{1}{\overline{M}_i}\frac{M(r_i,s_i,t_i)}{\partial \rho_e}\left(\frac{\partial \rho_e}{\partial a_e}\right)\right)\right] \\
&+ \Lambda \frac{\partial \rho_e}{\partial a_e} v_e \;+\; \alpha_e^+ \;-\; \alpha_e^- \;-\; \mathbf{u}^T \frac{\partial \mathbf{K}}{\partial a_e}\mathbf{u} \;=\; 0
\end{aligned}
\tag{3.25}
$$

In standard topology optimization using an optimality conditions algorithm, a procedure termed the design variable update is used to determine new designs. One of the must important aspects of the update algorithm is the relationship of the current

design at iteration k and the updated design for iteration $k+1$. In standard topology optimization, the following relationship is used:

$$x_i^{(k+1)} = \left[\frac{\mathbf{u}^T \frac{\partial \mathbf{K}}{\partial x_i}\mathbf{u}}{\Lambda \frac{\partial \rho_e}{\partial x_i}}\right]^{\eta} x_i^{(k)} \tag{3.26}$$

From (3.25) a strategy for updating a current design in a penalized moment constrained topology optimization sequence may also be developed. This leads to the following relationship:

$$x_i^{(k+1)} = \left[\frac{\frac{\mathbf{u}^T \frac{\partial \mathbf{K}}{\partial x_i}\mathbf{u}}{\frac{\partial \rho_e}{\partial x_i}}}{\Lambda + 2\alpha_p \left[\sum_{j=1}^{N_M} \left(\frac{M(r_i,s_i,t_i)}{\overline{M}_i} - 1\right)\left(\frac{1}{\overline{M}_i}\frac{M(r_i,s_i,t_i)}{\partial \rho_e}\right)\right]}\right]^{\eta} x_i^{(k)} \tag{3.27}$$

Note, in both (3.26) and (3.27), the parameter η controls the optimization convergence rate and stability. Using the design variable update in (3.27), a penalized spatial moment constrained topology optimization may be solved using an optimality conditions algorithm.

3.6 Summary

This chapter introduced the basic concepts of shape analysis and scalar shape descriptors. These ideas were expanded and applied to topology optimization to provide a degree of shape control in an topology optimization setting. Next, penalty methods were discussed as a solution algorithm for solving this class of problems. Lastly, the necessary modifications to standard optimality conditions algorithm were discussed. In following chapters, examples of these ideas will be presented.

Chapter 4

Hierarchical Topology Optimization Examples

4.1 Introduction

This chapter provides examples of the hierarchical topology optimization that is discussed in Chapter 2. A total of five examples are discussed, ranging in complexity to provide an accurate evaluation of hierarchical analysis. Once again, the goal of hierarchical topology optimization is to achieve results similar to standard topology optimization performed on fine mesh models with a series of coarse hierarchical models. Below is a brief discussion of the examples provided:

1. Symmetric Cube Example - A cube design domain with a centrally located loading condition

2. Anti-Symmetric Cube Example - A cube design domain with an edge loading condition

3. Suspension Arm Example - A non-symmetric design domain with three loading conditions

4. Symmetric Tower Example - A symmetric design domain with symmetric loading conditions

5. Anti-Symmetric Tower Example - A symmetric design domain with non-symmetric loading conditions

The examples will be presented in the following manner. First, a general description of the problem will be provided with reasoning of why the particular example is selected. Next, a pictorial representation of the design domain and loading condition(s) is provided. This is followed by a description of the voxel discretization used to obtain the *standard solution* for the problem. Due to the size of the problems addressed, all standard solutions are obtained using element-by-element solvers to find the standard solution. The results obtained from this analysis will then be used to evaluate the effectiveness of the hierarchical solution strategy. Here, computational resources (in-core memory and CPU time) will be compared and discussed. Lastly, a comparison between a standard solution and the hierarchical solution at the voxel level discretization will be provided. Note, all CPU times and memory requirements are obtained using a INTEL-Pentium 90 workstation.

Before presenting examples of problems solved with hierarchical analysis, a brief discussion of how the results will be analyzed and modifications to the hierarchi-

cal topology optimization algorithm are presented. These two topics focus on the following ideas.

1. Discussing how to evaluate the effectiveness of hierarchical topology optimization techniques

2. Discussing what modifications to the hierarchical topology optimization algorithm are necessary to achieve the best results

4.2 Evaluation Of Hierarchical Topology Optimization

In order to evaluate hierarchical topology optimization, we must develop some measure of the solution's effectiveness. First, we need to ensure that the results obtained from the hierarchical analysis are *similar* to those obtained from standard methods. The solution of topology optimization problems may be represented with two descriptors: *shape* and *compliance*. Therefore, in hierarchical topology optimization we desire results that produce shape and compliance values similar to the standard methods.

The evaluation of similar shape is achieved visually. By examining the hierarchical results and standard results, one may deduce the effectiveness of the hierarchical methods. Obviously, this method is quite subjective; however, after examining the results, the apparent accuracy of the hierarchical methods should be clear.

The evaluation of the compliance values is achieved through post-processing the

hierarchical results. In most cases, the solution of a given problem using hierarchical methods will possess lower compliances than the standard methods. This lower compliance is due to "artificial stiffness" added to the structure due to the compatibility equations. To determine a reasonable value for the final compliance of the hierarchical analysis, the hierarchical solution is analyzed at the voxel discretization level. In this procedure, the voxels in each finite element in the hierarchical analysis are assigned the material properties of the finite element at the optimal solution of the hierarchical analysis. The evaluation of the voxel level representation of the hierarchical solution produces compliance values that do not possess artificial stiffness attributes.

4.3 Modifications Of Hierarchical Topology Optimization

In using the hierarchical topology optimization presented in Chapter 2, a few issues arise. In some examples, complete sections of the final shape are missing from the hierarchical results compared to standard analysis. This issue is corrected by varying the percentage of the design domain that may be "filled" with material during the hierarchical analysis. In the examples presented, the volume percentage begins at a value higher than the desired amount and is reduced during the first few iterations of hierarchical analysis. This is performed because a very coarse mesh needs more material to approximate a structure when compared to a finer mesh description. Although this approach corrects the issue of missing portions of the optimal solution,

another problem arises due to its use. Namely, solutions from hierarchical methods contain regions of material in large elements that do not subdivide as they should when compared to standard results. Due to the increased amount of material used in the early stages of hierarchical analysis, sections of the solution are unaffected by the octree encoding and do not subdivide during the analysis, creating very coarse results. To correct this, the design variable update step is removed from the algorithm and each octree iteration starts with a gray design domain. Hence, the octree encoding algorithm produces the mesh that is analyzed in the next iteration, but not the initial design. Overall, this new method takes more CPU time due to the extra topology optimization iterations necessary at each iteration of the hierarchical analysis, but produces the most robust results.

4.4 Symmetric Cube Example

This example is based on one of the simplest design domains for hierarchical analysis and is provided to demonstrate the benefits of hierarchical topology optimization in a simple to visualize cube design domain. The graphical representation of the problem definition is shown in Figure 4.1. From Figure 4.1, the symmetry of the proposed problem is clear. One of the main reasons for solving this problem is to evaluate if the hierarchical solution technique effectively preserves the symmetry. For this problem, 35% design material is used.

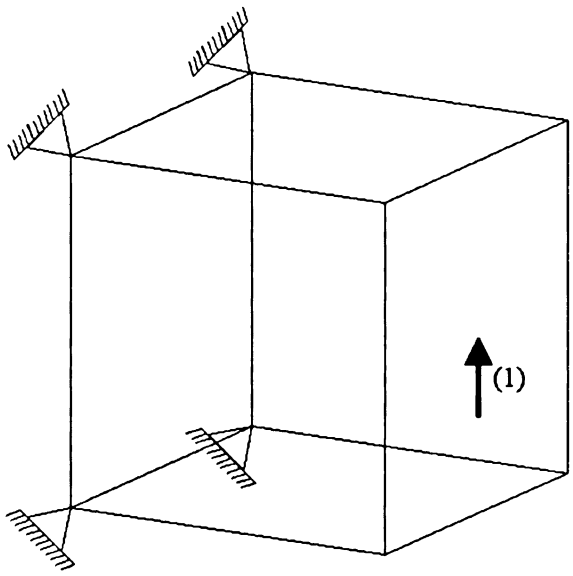

Figure 4.1: Symmetric cube example problem definition

4.4.1 Standard Solution Results

The finite element mesh, based on the voxel level discretization, used to solve this problem using standard topology optimization is show in Figure 4.2. The standard mesh has voxel dimensions of $24 \times 24 \times 24$ leading to 13,824 voxels. Treating each voxel as a finite element leads to a standard topology optimization problem that contains 46,863 degrees of freedom. Using element-by-element solution techniques, the topology optimization problem requires 15.11 megabytes of in-core memory and 18:48 (hours:minutes) of CPU time producing an optimal shape with compliance of 4.20. The standard solution may be seen in Figure 4.3. For visualization purposes, only elements with densities greater than 0.9 are displayed.

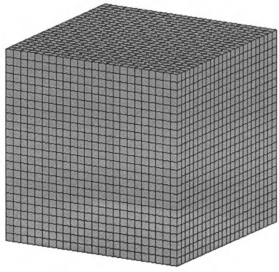

Figure 4.2: Symmetric cube example voxel level discretization

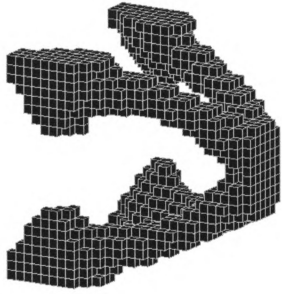

Figure 4.3: Symmetric cube example standard solution

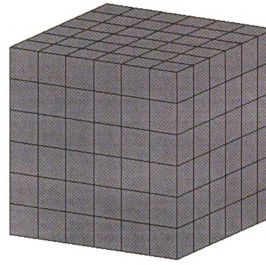

Figure 4.4: Symmetric cube example initial hierarchical mesh

4.4.2 Hierarchical Solution Results

Next, the problem is solved using hierarchical topology optimization techniques. The problem is solved with the minimum density tolerance, $\rho_{tol_{min}} = 0.05$, the maximum density tolerance, $\rho_{tol_{max}} = 0.25$, the growth factor, $g_f = 0.10$, and the density filter radius equal to 1.5 times the size of a voxel's edge. The initial hierarchical mesh is shown in Figure 4.4.

Table 4.1 displays the tabular results of the hierarchical analysis. Figures 4.5 through 4.8 display various solutions obtained during the hierarchical analysis. In the graphical representation of the hierarchical restarts, elements with densities greater than 0.8 are displayed. A brief discussion of Table 4.1 is necessary since this tabular format is used in all the examples provided in this chapter.

- Octree Iteration - This entry of the table corresponds to the iteration counter for hierarchical topology optimization algorithm.

- Design Elements - This entry lists the number of design elements that exist during a given iteration of the hierarchical analysis.

- Volume Percent - This entry corresponds to the amount of the design domain that may be filled with material at a given iteration.

- Density Exponent - This entry corresponds to the value of the density exponential weight factor obtained during the Merge Parameter Determination phase of the hierarchical analysis.

- Memory - This entry corresponds to the amount of in-core memory that is required to solve the topology optimization problem at a given iteration in the hierarchical analysis.

- CPU Time - This entry corresponds to the amount of CPU time (hours:minutes) that is required to solve the topology optimization problem at a given iteration in the hierarchical analysis.

- Average Compliance - This entry lists the average compliance of the optimal shape achieved from the topology optimization during a given iteration of the hierarchical analysis.

The final solution obtained from the hierarchical analysis is displayed at the voxel level in Figure 4.9. As discussed earlier, the density of a given octree cube is assigned to the voxels contained within it. The compliance of the hierarchical solution at the voxel level discretization is 4.38.

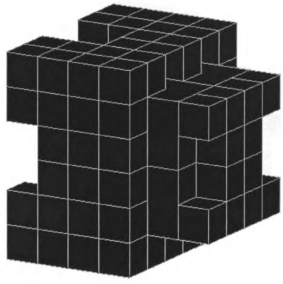

Figure 4.5: Symmetric cube example - octree iteration 0 solution

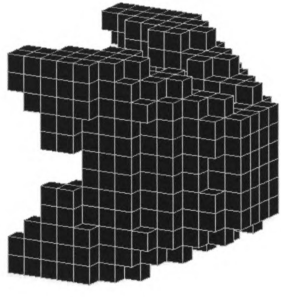

Figure 4.6: Symmetric cube example - octree iteration 3 solution

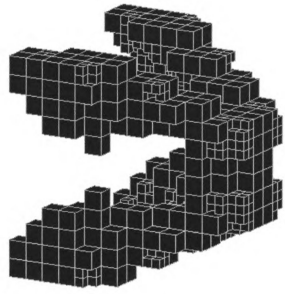

Figure 4.7: Symmetric cube example - octree iteration 5 solution

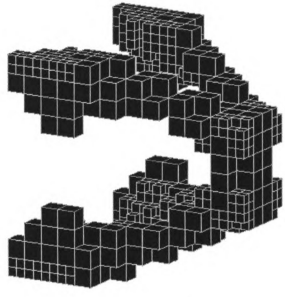

Figure 4.8: Symmetric cube example - octree iteration 6 solution

Table 4.1: Symmetric cube example hierarchical results

Octree Iteration	Design Elements	Constraint Equations	Volume Percent	Density Exponent	Memory (Meg)	CPU Time	Average Compliance
0	160	216	0.55	10.00	1.54	0:04	1.00
1	888	1812	0.50	10.00	14.92	1:08	2.01
2	1196	1296	0.45	8.17	19.38	1:37	2.04
3	1336	1284	0.40	6.02	22.78	0:38	2.11
4	1504	2868	0.35	6.04	31.68	1:33	3.72
5	1672	4140	0.35	4.94	40.03	1:26	2.21
6	1868	4500	0.35	2.63	46.19	1:06	3.86

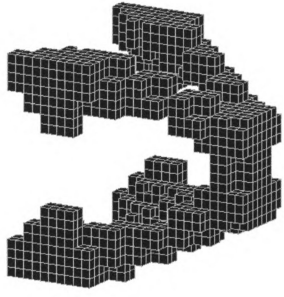

Figure 4.9: Symmetric cube example - octree iteration 6 solution at voxel level

4.4.3 Discussion

This example provides a simple introduction to the use of hierarchical topology optimization. From the figures displaying the standard results and hierarchical results, it appears that the hierarchical methods are sufficient in finding a reasonable approximation of the optimal shape for this example. The compliance value achieved with the hierarchical model (3.86) is slightly lower than standard method compliance value (4.26) as discussed earlier, but the hierarchical solution solved at the voxel discretiza-

tion level produces a compliance value (4.38) slightly higher than the standard value. Overall, the ability of the hierarchical methods to produce similar results is clear in this example.

The next issue is that of performance. Hierarchical methods converge to a final solution in a total CPU time of 7:32 (hours:minutes) and maximum in-core memory requirements of 46.19 megabytes. Here, the hierarchical methods converge in less than half the time of the standard methods but require about three times as much in-core memory. Recall, the solution algorithm used for the standard solution algorithm, an element-by-element solver, is very memory efficient. Hence, the comparison in memory use penalizes the hierarchical methods substantially. This is addressed more completely in the following sections.

4.5 Anti-Symmetric Cube Example

This example is an extension of the previous example and is provided to demonstrate the benefits of hierarchical topology optimization in a problem that is not symmetric. The graphical representation of the problem definition is shown in Figure 4.10. For this problem, 45% design material is used.

4.5.1 Standard Solution Results

The voxel level discretization for this problem is identical to that of the previous problem. Again, the standard mesh of voxel dimensions $24 \times 24 \times 24$ leads to 13,824 finite elements with 46,863 degrees of freedom. Using element-by-element solution

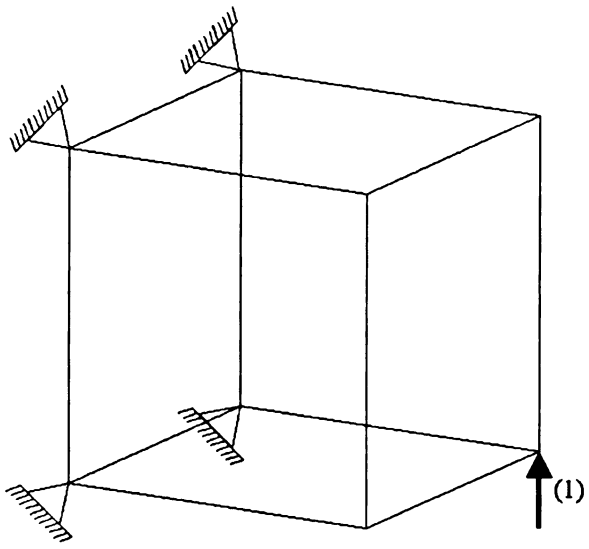

Figure 4.10: Anti-symmetric cube example problem definition

techniques, the topology optimization problem requires 15.11 megabytes of in-core memory and 19:39 (hours:minutes) of CPU time producing an optimal shape with compliance of 8.19. The standard solution may be seen in Figure 4.11 where elements with densities greater than 0.7 are displayed.

4.5.2 Hierarchical Solution Results

Next, the problem is solved using hierarchical topology optimization techniques. The problem is solved with the minimum density tolerance, $\rho_{tol_{min}} = 0.05$, the maximum density tolerance, $\rho_{tol_{max}} = 0.25$, the growth factor, $g_f = 0.25$, and the density filter radius equal to 1.5 times the size of a voxel's edge. The initial hierarchical mesh is identical to the one used in the previous example.

Table 4.2 displays the tabular results of the hierarchical analysis. Figures 4.12 through 4.16 display various solutions obtained during the hierarchical analysis. Note, for octree iteration 0, 3, 5, and 7, elements with densities greater than 0.5 are displayed

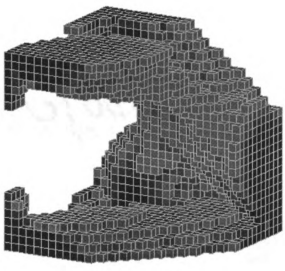

Figure 4.11: Anti-symmetric cube example standard solution

to demonstrate the formation of the final structure. For octree iteration 9, densities greater than 0.7 are displayed for comparison to the standard results.

Table 4.2: Anti-symmetric cube example hierarchical results

Octree Iteration	Design Elements	Constraint Equations	Volume Percent	Density Exponent	Memory (Meg)	CPU Time	Average Compliance
0	160	216	0.60	10.000	1.55	0:01	1.99
1	503	1191	0.55	5.177	7.59	0:04	4.10
2	720	1815	0.50	5.405	13.49	0:10	8.13
3	937	2541	0.45	4.941	20.25	0:24	8.16
4	1147	3462	0.45	4.813	27.32	0:21	8.17
5	1259	3834	0.45	4.757	30.74	0:23	8.17
6	1308	4029	0.45	4.732	32.69	0:25	8.17
7	1350	4188	0.45	4.702	34.03	0:26	8.17
8	1406	4407	0.45	4.668	36.43	0:29	8.17
9	1469	4536	0.45	4.627	38.17	1:37	8.17

The final solution obtained from the hierarchical analysis is displayed at the voxel level discretization in Figure 4.17. The compliance of the hierarchical solution at the voxel level discretization is 8.29.

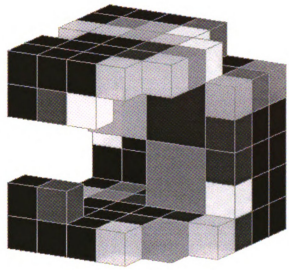

Figure 4.12: Anti-symmetric cube example - octree iteration 0 solution

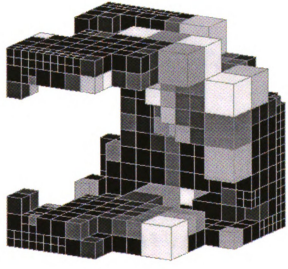

Figure 4.13: Anti-symmetric cube example - octree iteration 3 solution

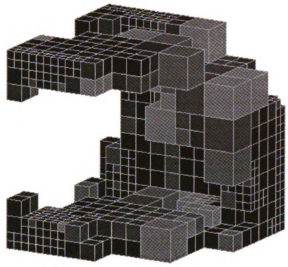

Figure 4.14: Anti-symmetric cube example - octree iteration 5 solution

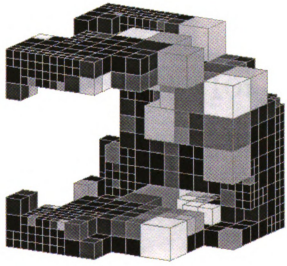

Figure 4.15: Anti-symmetric cube example - octree iteration 7 solution

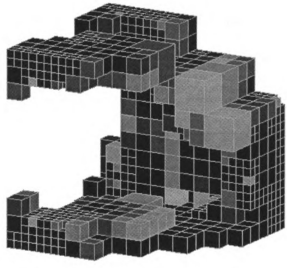

Figure 4.16: Anti-symmetric cube example - octree iteration 9 solution

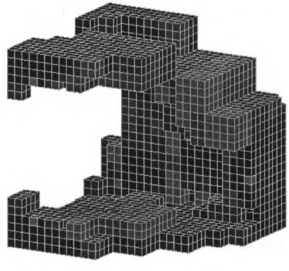

Figure 4.17: Anti-symmetric cube example - octree iteration 9 solution at voxel level

4.5.3 Discussion

As seen in the previous example, the hierarchical analysis provides a reasonable approximation to both the shape and compliance compared to the standard. The compliance of the hierarchical analysis is 8.17 compared to the compliance of 8.19 determined from the standard method. The hierarchical analysis finds an optimal requiring 4:20 of CPU time compared to the 19:39 of CPU time needed in standard methods. It is also reassuring that similar performance is seen in this anti-symmetric problem. Once again, the hierarchical methods require more in-core memory, but considerable savings are made in CPU time.

From the first two examples, the clear gain is seen in the amount of CPU time that is necessary to solve large-scale problems. However, the use of in-core memory is still in question. On the workstation used to solve these examples, approximately 60 megabytes of in-core is available for solving the problems at hand. Both of these examples could be solved using element-by-element finite element analysis using under 16 megabytes. However, the element-by-element solution technique is the only available method for solving the standard problems in the available in-core memory of the workstation. Attempts of using standard skyline solvers require approximately 720 megabytes of in-core memory. The use of sparse solvers require approximately 550 megabytes of in-core memory. Using frontal solvers is another alternative, but their use requires large disk space resources and considerable CPU time. Hence, the hierarchical analysis provides a reasonable approximation to standard solution in far less time than the only other possible solution technique. This benefit highly outweighs

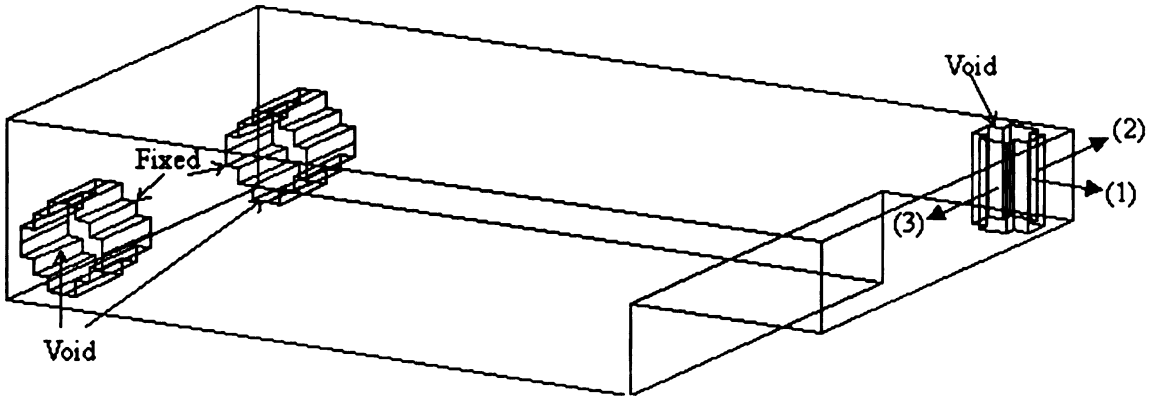

Figure 4.18: Suspension arm example problem definition

the increase of in-core memory necessary to solve these examples.

4.6 Suspension Arm Example

The previous examples demonstrate the effectiveness of hierarchical analysis in both symmetric and anti-symmetric problems with a single load case. This example introduces an anti-symmetric design domain with three loading conditions. The graphical representation of the problem definition is shown in Figure 4.18. For this problem, 35% design material is used.

4.6.1 Standard Solution Results

The voxel level discretization used to solve this problem with standard topology optimization is show in Figure 4.19. The standard mesh contains 19,992 design voxels. Treating each voxel as a finite element leads to a standard topology optimization problem that contains 69,234 degrees of freedom. Using element-by-element solution techniques, the topology optimization problem requires 39.91 megabytes of in-core

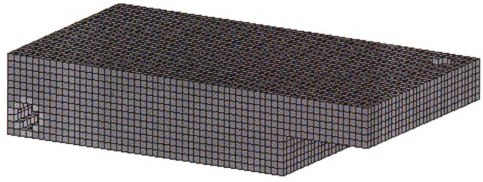

Figure 4.19: Suspension arm example voxel level discretization

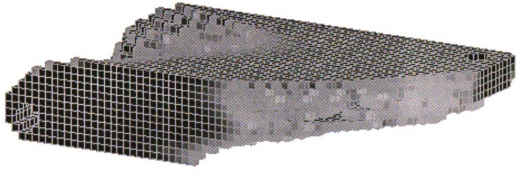

Figure 4.20: Suspension arm example standard solution

memory and 79:52 (hours:minutes) of CPU time producing an optimal shape with compliance of 40.07. The standard solution may be seen in Figure 4.20 where elements with densities greater than 0.3 are displayed.

4.6.2 Hierarchical Solution Results

Next, the problem is solved using hierarchical topology optimization techniques. The problem is solved with the minimum density tolerance, $\rho_{tol_{min}} = 0.05$, the maximum density tolerance, $\rho_{tol_{max}} = 0.30$, the growth factor, $g_f = 0.10$, and the density filter radius equal to 1.5 times the size of a voxel's edge. The initial hierarchical mesh is shown in Figure 4.21. Here, a maximum element size is set to 4 voxels along an edge

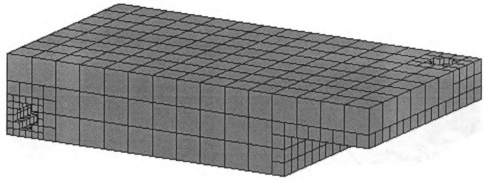

Figure 4.21: Suspension arm example initial hierarchical mesh

to produce a realistic initial mesh.

Table 4.3 displays the tabular results of the hierarchical analysis. Figures 4.22 through 4.25 display various solutions obtained during the hierarchical analysis. Note, for all graphic hierarchical solutions, elements with densities greater than 0.3 are displayed. The final solution obtained from the hierarchical analysis is displayed

Table 4.3: Suspension arm example hierarchical results

Octree Iteration	Design Elements	Constraint Equations	Volume Percent	Density Exponent	Memory (Meg)	CPU Time	Average Compliance
0	714	1371	0.50	10.00	7.49	0:47	16.47
1	1316	3009	0.45	4.47	16.57	2:15	25.07
2	1498	3468	0.40	4.45	19.83	3:41	31.87
3	1687	3705	0.35	0.00	24.97	5:21	39.93

at the voxel level in Figure 4.26. The compliance of the hierarchical solution at the voxel level discretization is 42.97.

4.6.3 Discussion

Once again, the hierarchical analysis provides results that approximate the standard solution optimal shape very well. The average compliance value of the hierarchical

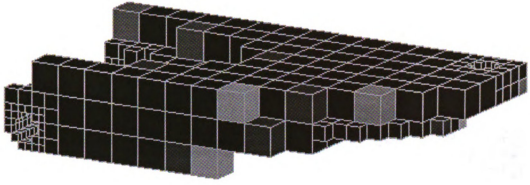

Figure 4.22: Suspension arm example - octree iteration 0 solution

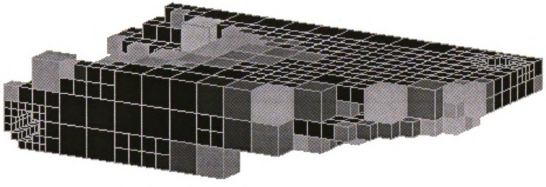

Figure 4.23: Suspension arm example - octree iteration 1 solution

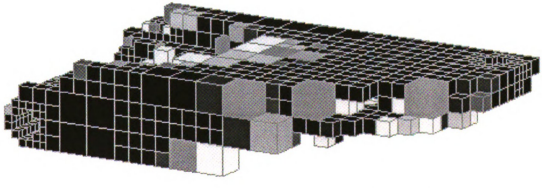

Figure 4.24: Suspension arm example - octree iteration 2 solution

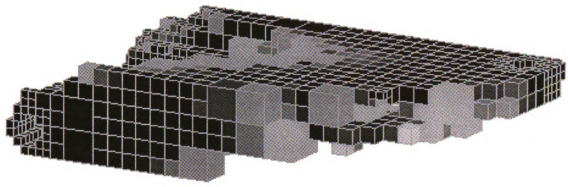

Figure 4.25: Suspension arm example - octree iteration 3 solution

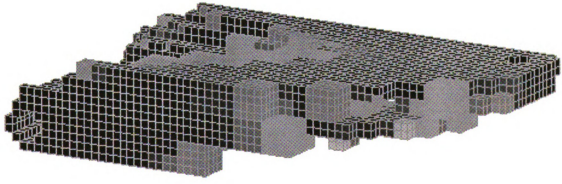

Figure 4.26: Suspension arm example - octree iteration 3 solution at voxel level

solution is slightly lower than that of the standard solution. Upon solving the hierarchical solution at the voxel discretization level, the average compliance value becomes slightly higher than the standard value. Most importantly, this example demonstrates the ability of the hierarchical methods to achieve similar results to standard methods in far less CPU time and less in-core memory. This example requires about 25 megabytes of in-core memory compared to the standard method that requires nearly 40 megabytes.

4.7 Tower Example

The previous examples consisted of optimal solutions that were composed of primarily *thick* components. This example consists of a tower design domain and loading conditions that lead to optimal solutions that possess *thin* regions. The graphical representation of the problem definition is shown in Figure 4.27, where 35% design material is used. These structures are more difficult to solve hierarchically due to the level of detail that is necessary to describe the thin regions of the structure. Hence, this example will show the ability of the hierarchical solution strategies to solve problems of this type.

4.7.1 Standard Solution Results

The finite element mesh, or voxel mesh, to solve this problem using standard topology optimization is show in Figure 4.28. The standard mesh contains 20,480 design voxels. Treating each voxel as a finite element leads to a standard topology optimization

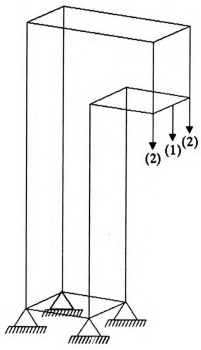

Figure 4.27: Tower example problem definition

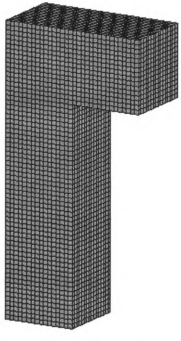

Figure 4.28: Tower example voxel level discretization

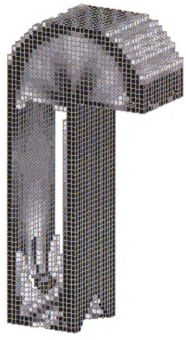

Figure 4.29: Tower example standard solution

problem that contains 70,215 degrees of freedom. Using element-by-element solution techniques, the topology optimization problem requires 29.41 megabytes of in-core memory and 56:20 (hours:minutes) of CPU time producing an optimal shape with compliance of 6.84. The standard solution may be seen in Figure 4.29 where elements with densities greater than 0.2 are displayed.

4.7.2 Hierarchical Solution Results

Next, the problem is solved using hierarchical topology optimization techniques. The problem is solved with the minimum density tolerance, $\rho_{tol_{min}} = 0.05$, the maximum density tolerance, $\rho_{tol_{max}} = 0.30$, the growth factor, $g_f = 0.10$, and the density filter radius equal to 1.5 times the size of a voxel's edge. The initial hierarchical mesh is shown in Figure 4.30. Here, the maximum element size is set to 4 voxels along an

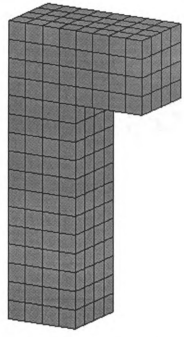

Figure 4.30: Tower example initial hierarchical mesh

edge to produce a realistic initial mesh.

Table 4.4 displays the tabular results of the hierarchical analysis. Figures 4.31 through 4.34 display various solutions obtained during the hierarchical analysis. Again, finite elements with densities greater than 0.2 are displayed. The final solution obtained from the hierarchical analysis is displayed at the voxel level in Figure 4.35. The compliance of the hierarchical solution at the voxel level discretization is 5.37.

4.7.3 Discussion

Although this problem was intended to demonstrate some of the weaknesses of hierarchical analysis, relatively good results were obtained. From the graphical representation of the hierarchical solution and standard solution, the hierarchical method

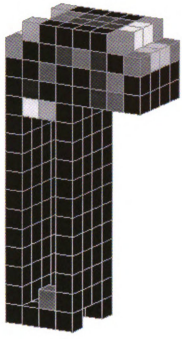

Figure 4.31: Tower example - octree iteration 0 solution

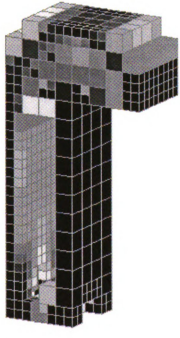

Figure 4.32: Tower example - octree iteration 2 solution

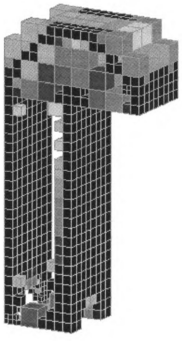

Figure 4.33: Tower example - octree iteration 7 solution

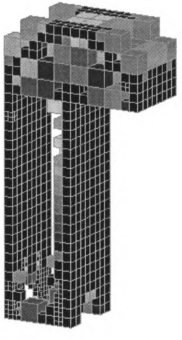

Figure 4.34: Tower example - octree iteration 9 solution

Table 4.4: Tower example hierarchical results

Octree Iteration	Design Elements	Constraint Equations	Volume Percent	Density Exponent	Memory (Meg)	CPU Time	Average Compliance
0	320	0	0.60	10.00	1.86	0:03	2.41
1	866	1341	0.55	6.17	8.52	0:20	3.66
2	936	1539	0.50	4.45	10.17	0:11	3.86
3	1006	1845	0.45	4.11	11.42	0:06	4.67
4	1062	1776	0.40	9.53	11.54	0:46	4.61
5	1160	1947	0.35	4.57	12.95	0:35	4.61
6	1258	2304	0.35	4.42	14.74	0:53	4.51
7	1356	2688	0.35	4.22	16.83	0:50	5.02
8	1398	2832	0.35	3.98	17.70	0:33	5.16
9	1496	3192	0.35	4.07	19.92	1:26	5.01

clearly produces an optimal shape that is similar to the standard method. Unlike the previous examples, the hierarchical solution and the hierarchical solution solved at the voxel level possess optimal solutions with lower average compliance values than the standard solution. From a designer's viewpoint, this type of behavior is desirable. Here, a better design is found with the hierarchical methods. Theoretically, what does this say about the optimal solution achieved during the standard analysis? Two possible explanations exist to describe these results.

1. The standard solution converged to a local optimum rather than a global optimum.

2. The higher number of topology optimizations that are performed during the hierarchical analysis produce a set of material orientation angles that provide a less compliant solution.

In practice, a combination of the two explanations is expected. Namely, the effects of both may be present in the optimal solutions found. Lastly, the hierarchical methods

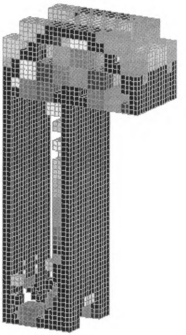

Figure 4.35: Tower example - octree iteration 9 solution at voxel level

produce these results using less in-core memory and a fraction of the CPU time when compared to standard methods.

4.8 Anti-Symmetric Tower Example

This example examines a slight variation of the previous tower example. One of the load conditions is modified to create a torsion load on the tower. The domain definition loading conditions may be seen in Figure 4.27. This loading condition will increase the amount of thin regions in the optimal design making the hierarchical solution more difficult to obtain efficiently. As before, 35% material percentage is used.

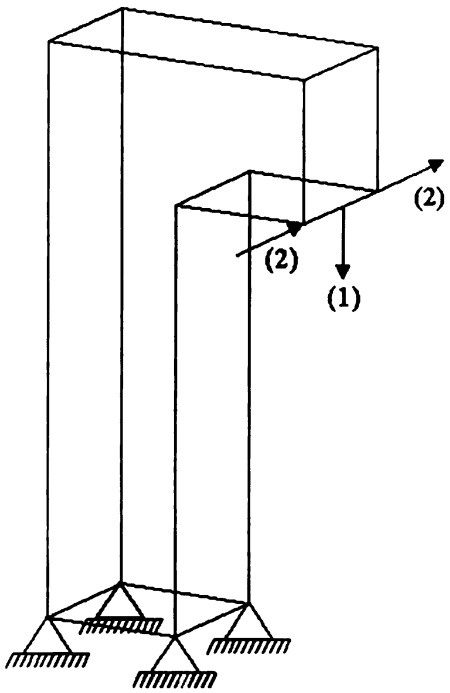

Figure 4.36: Anti-symmetric tower example problem definition

4.8.1 Standard Solution Results

The voxel discretization used in the standard topology optimization is the same as the mesh used in the previous example displayed in Figure 4.28. Again, the standard mesh contains 20,480 design voxels with 70,215 degrees of freedom. Using element-by-element solution techniques, the topology optimization problem requires 29.41 megabytes of in-core memory and 80:49 (hours:minutes) of CPU time producing an optimal shape with compliance of 1.35. The standard solution may be seen in Figure 4.37. An additional view of the standard in Figure 4.38 where a portion of the structure as been removed to examine the interior portions of the structure. In both Figure 4.37 and Figure 4.38, elements with densities greater than 0.3 are displayed.

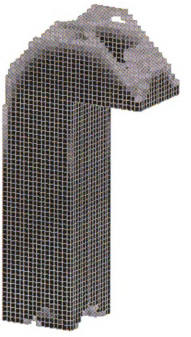

Figure 4.37: Anti-symmetric tower example standard solution

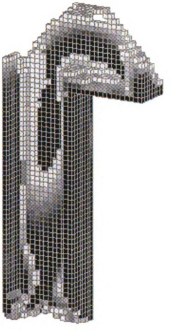

Figure 4.38: Anti-symmetric tower example standard solution. (Front half of structure removed)

4.8.2 Hierarchical Solution Results

Next, the problem is solved using hierarchical topology optimization techniques. The problem is solved with the minimum density tolerance, $\rho_{tol_{min}} = 0.05$, the maximum density tolerance, $\rho_{tol_{max}} = 0.30$, the growth factor, $g_f = 0.10$, and the density filter radius equal to 1.5 times the size of a voxel's edge. The initial hierarchical mesh is the same as shown in previous example in Figure 4.30.

Table 4.5 displays the tabular results of the hierarchical analysis. Figures 4.39 through 4.43 display various solutions obtained during the hierarchical analysis. In all hierarchical figures, elements with densities greater than 0.3 are displayed.

Table 4.5: Anti-symmetric tower example hierarchical results

Octree Iteration	Design Elements	Constraint Equations	Volume Percent	Density Exponent	Memory (Meg)	CPU Time	Average Compliance
0	320	0	0.60	10.00	1.86	0:07	0.446
1	1076	1746	0.55	10.00	11.77	0:50	0.570
2	1244	1791	0.50	4.97	14.10	0:45	0.607
3	1370	1965	0.45	4.05	16.47	0:14	0.856
4	1510	2622	0.40	3.98	19.80	0:32	0.892
5	1636	3141	0.35	4.37	23.14	0:50	0.943
6	1720	3312	0.35	4.49	24.75	1:22	0.916
7	1888	3837	0.35	4.80	28.86	1:50	0.908
8	2056	4377	0.35	4.24	32.88	1:29	0.930
9	2112	4737	0.35	4.82	34.36	3:00	0.903

The final solution obtained from the hierarchical analysis is displayed at the voxel level in Figure 4.44. Here, the density of a given octree cube is assigned to the voxels contained within it. The compliance of the hierarchical solution at the voxel level discretization is 1.45.

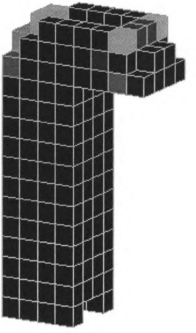

Figure 4.39: Anti-symmetric tower example - octree iteration 0 solution

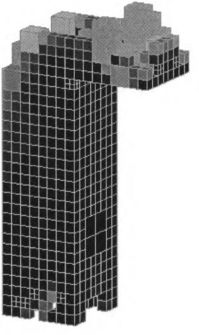

Figure 4.40: Anti-symmetric tower example - octree iteration 4 solution

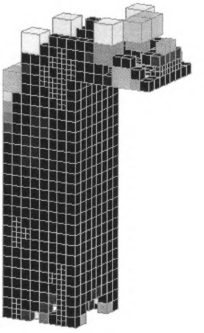

Figure 4.41: Anti-symmetric tower example - octree iteration 7 solution

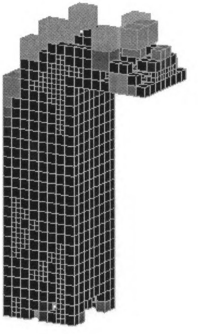

Figure 4.42: Anti-symmetric tower example - octree iteration 9 solution

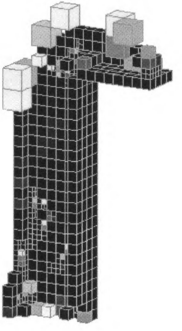

Figure 4.43: Anti-symmetric tower example - octree iteration 9 solution. (Front half of structure removed)

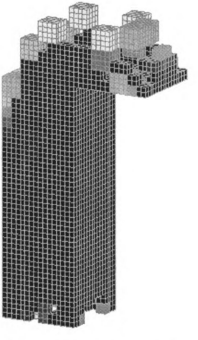

Figure 4.44: Anti-symmetric tower example - octree iteration 9 solution at voxel level

4.8.3 Discussion

The average compliance value obtained from the hierarchical analysis is somewhat lower than the value achieved using standard methods. In addition, the average compliance of the hierarchical solution solved at the voxel level is higher than the value achieved using standard methods. Problems are also observed in the optimal shapes obtained from the two methods. Clearly, this example demonstrates the difficultly encountered where attempting to solve problems that contain a high degree of thin regions in the optimal solution with the propose hierarchical method. The reason for the difficulty in solving such problems in the large number of elements that are necessary to accurately model the domain. Examining the figures of the later iterations of the hierarchical analysis reveals the attempts of the hierarchical methods to introduce a more refined mesh in the areas of the thinnest members. The inability of the hierarchical methods to introduce more elements efficiently leads to two major differences in the hierarchical solution compared to the standard solution. First, the top of the optimal shape that exists in the standard solution is not present in the hierarchical solution. Second, the section of the solution opposite from the load is also not completely filled as seen in the Figure 4.43. Although these differences exist, the most important attributes of the solutions are the same. In this example, the hierarchical solution is obtained in far less CPU time while requiring slightly more in-core memory.

4.9 Summary

Table 4.6 and 4.7 summarize the results obtained in the previous examples. These tables present the numerical results of the hierarchical analysis and compare it to the result obtained from standard topology optimization. The first table contains the following information:

- Hierarchical Methods:
  - Memory - The maximum amount of in-core memory in megabytes to solve the given example using hierarchical methods.
  - CPU - The total CPU time needed in hours to solve the given example using hierarchical methods.
  - Compliance - The compliance of the optimal design using hierarchical methods.
- Hierarchical Results At The Voxel Discretization (Hier. @ Voxel):
  - Compliance - The compliance of the optimal design for the hierarchical analysis processed at the voxel descretization.
- Standard Methods:
  - Memory - The amount of in-core memory in megabytes to solve the given example using standard methods.
  - CPU - The total CPU time needed in hours to solve the given example using standard methods.

- Compliance - The compliance of the optimal design using standard methods.

In addition, a total of four performance parameters are provided. These quantities help describe the effectiveness of the hierarchical methods and are defined as follows:

- Memory Comparison (E_{MEM}) - This parameter compares the amount of in-core memory necessary to solve the hierarchical problem and the standard problem.

$$E_{MEM} = \frac{\text{Hierarchical Maximum Memory Needed}}{\text{Standard Memory Needed}} \tag{4.1}$$

- CPU Comparison (E_{CPU}) - This parameter compares the amount of total CPU time necessary to solve the hierarchical problem and the standard problem.

$$E_{CPU} = \frac{\text{Hierarchical Total CPU Time}}{\text{Standard CPU Time}} \tag{4.2}$$

- Compliance Comparison 1 (E_{C_1}) - This parameter compares the average compliance values determined by the hierarchical problem and the standard problem.

$$E_{C_1} = \frac{\text{Hierarchical Compliance}}{\text{Standard Compliance}} \tag{4.3}$$

- Compliance Comparison 2 (E_{C_2}) - This parameter compares the average compliance values determined by the hierarchical problem solved at the voxel discretization and the standard problem.

$$E_{C_2} = \frac{\text{Hierarchical Compliance At The Voxel Level}}{\text{Standard Compliance}} \tag{4.4}$$

Table 4.6: Summary of hierarchical and standard results

Example	Hierarchical Methods			Hier. @ Voxel	Standard Methods		
	Memory	CPU	Compliance	Compliance	Memory	CPU	Compliance
Sym Cube	46.19	7.53	3.86	4.38	15.11	18.8	4.2
A-Sym Cube	38.17	4.33	8.17	8.29	15.11	19.65	8.19
Susp Arm	24.97	12.07	39.93	42.97	39.91	79.87	40.07
Sym Tower	19.92	5.72	5.01	5.37	29.41	56.33	6.84
A-Sym Tower	34.36	10.98	0.903	1.45	29.41	80.81	1.35

Table 4.7: Summary of hierarchical and standard performance parameters

Example	E_{MEM}	E_{CPU}	E_{C_1}	E_{C_2}
Sym Cube	3.06	0.40	0.919	1.043
A-Sym Cube	2.53	0.22	0.997	1.012
Susp Arm	0.63	0.15	0.997	1.072
Sym Tower	0.68	0.10	0.733	0.785
A-Sym Tower	1.17	0.14	0.669	1.074

From the tabular data, the basic trends of the hierarchical analysis maybe determined. In most cases, hierarchical topology optimization produces optimal solutions with compliance values the same as or lower than the standard topology optimization method. In all cases, solutions are obtained in far less time than the standard method using element-by-element methods. Lastly, these results are achieved by using a range of memory requirements depending on the problem at hand.

Only two issues remain in discussing the hierarchical algorithm presented in this thesis.

1. Under which circumstances does hierarchical topology optimization fail to produce results similar in shape to standard methods?

2. How can the memory requirement of hierarchical topology optimization be lowered?

As for the first issue, experience has indicated that hierarchical topology optimization behaves best for optimal structures that contain thick regions or thick components. So, attempting to solve topology optimization problems for which thin structures are optimal will most likely lead to difficulties when using hierarchical methods. Again, thin structures require high levels of mesh refinement, which cause the hierarchical methods to lose their effectiveness.

The second issue is addressed in the following manner. First, all examples used the penalized-skyline solver or the sparse solver with the full constraint equations during the hierarchical topology optimization. Using a sparse solver with the penalized equations as in the penalized-skyline solver would definitely reduce the necessary in-core memory with only a slight execute time increase. Recall the great differences in the in-core memory requirements needed for the standard solution in the first two examples using standard skyline and sparse solvers. This type of memory distribution is common among most problems for these solvers. Once again, although hierarchical topology optimization may require more memory than the element-by-elements methods in some cases, it still produces reasonable results in far less memory than the other solvers. Hence, in many situations, hierarchical topology optimization may be the only choice other than element-by-element methods for a given problem and

available computational resources.

Chapter 5

Constrained Spatial Moment Topology Optimization Examples

5.1 Introduction

Chapter 3 presented the formulation and solution algorithm for the spatial moment constrained topology optimization. The examples presented in this chapter will demonstrate the effectiveness of this theory. Although the theory accounted for multiple load cases and multiple loading conditions, the examples in this thesis concentrate on single load case and single moment constrained problems. This promotes simple results to truly understand the advantages and disadvantages of applying moment constraints to a given problem. Four examples are provided to show the extent in which shape control may be applied to topology optimization using spatial moment constraints. Below is a brief discussion of the examples provided:

1. 2-Dimensional Truss - A 2-dimensional design domain with a single load case

2. 3-Dimensional Truss - A simply supported cube domain with a single load case

3. Beam - A cantilever beam with a tip load

4. Tower - A tower design domain with a single load case

The examples are presented in the following manner. First, a general description of the problem is provided with a pictorial representation of the design domain and the moment constraint. Then, solutions to a series of spatial moment constrained problems is discussed.

5.2 2-Dimensional Truss Example

Although all other work presented in this thesis focuses on 3-dimensional problems, this example is a simple 2-dimensional problem to help demonstrate the concept of moment constrained topology optimization. Figure 5.1 displays the problem definition where 35% design material is used.

5.2.1 Standard Solution Results

The result obtained with standard topology optimization procedures is displayed in Figure 5.2. The optimal design possess a compliance of 1.80. In addition, computing the moment about the axis as displayed in Figure 5.1 with spatial order $r = 2.0$, reveals $M_l(2) = 85.0$.

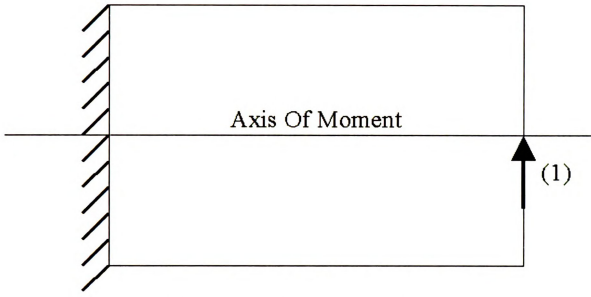

Figure 5.1: 2-dimensional truss example problem definition

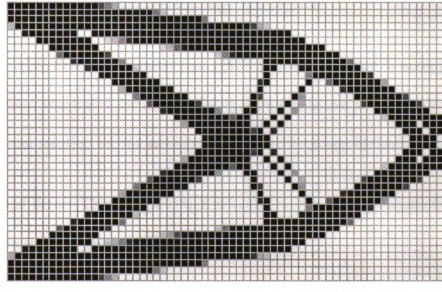

Figure 5.2: 2-dimensional truss example standard solution

5.2.2 Moment Constrained Solution Results

An attempt is now made to alter the the optimal shape with the addition of a moment constraint. Four moment constrained topology optimization solutions are presented; enforcing constraints of 80, 70, 60, and 55 on the moment $M_l(2)$ as shown in Figure 5.1.

Table 5.1 displays the numerical results of the moment constrained analysis, while Figures 5.3 through 5.6 display the optimal shapes. Tables with identical format to Table 5.1 will be provided for the remaining moment constrained examples. Below is a brief description the data these tables contain.

- Constraint - This entry lists the value of the constraint placed on the moment of interest

- Actual Moment - This entry displays the moment of the optimal structure found for the moment constrained topology optimization

- Allowable Error - This entry lists the maximum error that is acceptable in the moment constraint for a potential optimum solution to be considered as the optimal solution for the moment constrained topology optimization

- Actual Error - This entry displays the amount of error in the moment constraint at constrained optimal solution

- Compliance - This entry displays the compliance of the moment constrained optimal solution

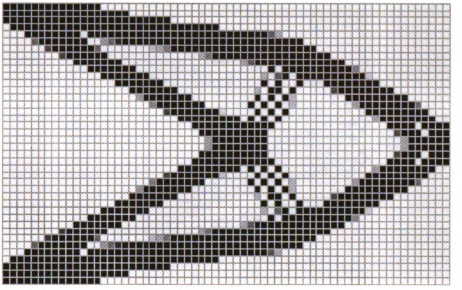

Figure 5.3: 2-dimension truss example - moment constraint $M_l(2) = 80.0$ solution

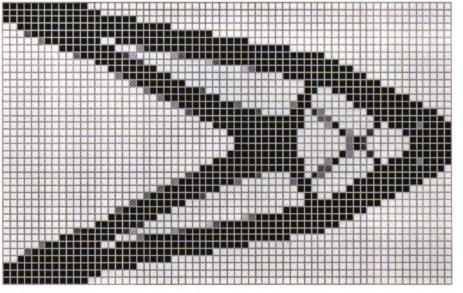

Figure 5.4: 2-dimension truss example - moment constraint $M_l(2) = 70.0$ solution

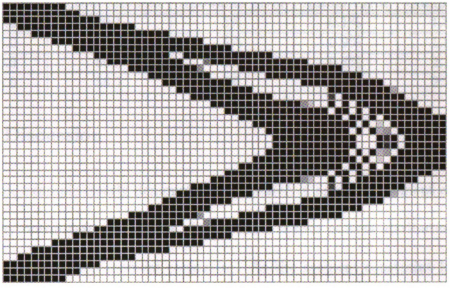

Figure 5.5: 2-dimension truss example - moment constraint $M_l(2) = 60.0$ solution

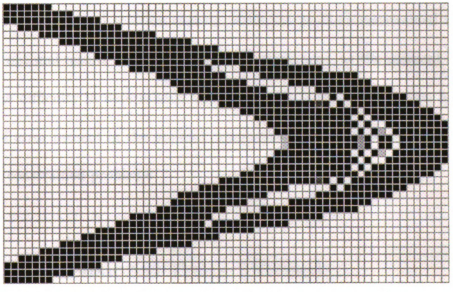

Figure 5.6: 2-dimension truss example - moment constraint $M_l(2) = 55.0$ solution

Table 5.1: 2-dimensional truss moment constrained example results

Constraint	Actual Moment	Allowable Error	Actual Error	Compliance
80.0	80.1	5%	0.1%	1.85
70.0	70.6	5%	0.9%	1.96
60.0	60.5	5%	0.8%	2.19
55.0	56.7	5%	3.1%	2.24

5.2.3 Discussion

This simple example clearly demonstrates the effect of applying moment constraints to topology optimization problems. The optimal shapes possess attributes that correspond with the given moment constraint. As the moment constraint lowers, the optimal shapes contain more material near the axis of the applied moment. In addition, a steady increase in the compliance is observed as the value of the moment constraint lowers, which is expected. There is an issue of stability for the results presented here. The moment constraint problem with constraints of 80, 70, and 60 converged to an optimal solution rapidly, while the problem constrained to 55 converged to optimal results much less rapidly. In most cases, as the moment constraint lowers, the difficulty in determining the solution increases. This is seen here in the amount of error present in the problem constrained at a moment of 55 compared to the other constrained problems.

5.3 3-Dimensional Truss Example

This example is based on a variation of the previous example expanded to three dimensions. Figure 5.7 displays the problem definition where 35% design material is

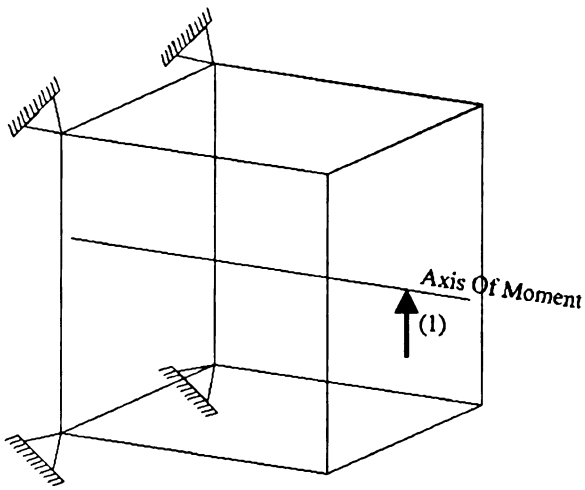

Figure 5.7: 2-dimensional truss example problem definition

used.

5.3.1 Standard Solution Results

The result obtained with standard topology optimization procedures is displayed in Figure 5.8 where elements with densities greater than 0.9 are displayed. The optimal design possesses a compliance of 0.451. In addition, computing the moment about the axis as displayed in Figure 5.7 with spatial order $r = 2.0$, reveals $M_l(2) = 1.13\text{E+}04$.

5.3.2 Moment Constrained Solution Results

As in the 2-dimensional example, an attempt is made to alter the the optimal shape with the addition of the a moment constraint. Four moment constrained topology optimization solutions are presented; enforcing constraints of 1.0E+04, 9.0E+03, 8.5E+03, and 8.0E+03 on the moment $M_l(2)$ as shown in Figure 5.7. Table 5.2 displayed the numerical results of the constrained problems. Figures 5.9 through 5.12

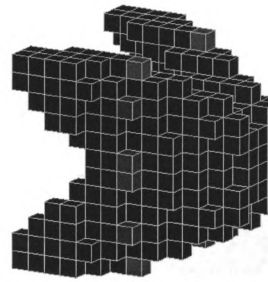

Figure 5.8: 3-dimensional truss example standard solution

display the optimal shape determined for each problem where elements with densities greater than 0.9 are displayed.

Table 5.2: 3-dimensional truss moment constrained example results

Constraint	Actual Moment	Allowable Error	Actual Error	Compliance
1.0E+04	1.05E+03	5%	4.96%	0.471
9.0E+03	9.39E+03	5%	4.32%	0.482
8.5E+03	8.88E+03	5%	4.46%	0.491
8.0E+03	8.35E+03	5%	4.44%	0.480

5.3.3 Discussion

The results of this 3-dimensional problem are promising. Although it may be more difficult to observe in 3-dimensions, a thinning of the structure is observed as the moment constraint value decreases. The effect is most clearly seen in comparing Figure 5.12 to the standard results. As in the 2-dimensional example, all the moment constrained optimal solutions are more compliant than the standard solution. Yet,

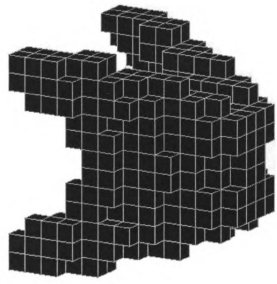

Figure 5.9: 3-dimension truss example - moment constraint $M_l(2) = 1.0\text{E}{+}04$ solution

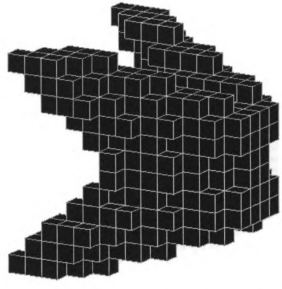

Figure 5.10: 3-dimension truss example - moment constraint $M_l(2) = 9.0\text{E}{+}03$ solution

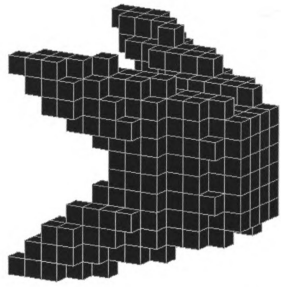

Figure 5.11: 3-dimension truss example - moment constraint $M_l(2)$ = 8.5E+03 solution

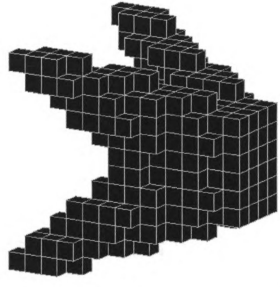

Figure 5.12: 3-dimension truss example - moment constraint $M_l(2)$ = 8.50E+03 solution

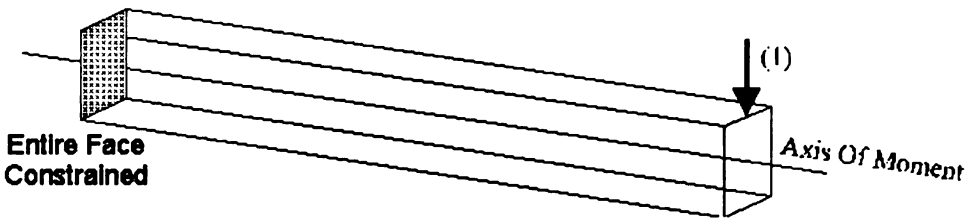

Figure 5.13: Beam example problem definition

the compliance values do not increase with lower moment constraints. This may be seen with the compliance of the $M_l(2)$ = 8.0E+03 constrained solution. To explain this, we must realize the moment constraint of 8.0E+03 is difficult to achieve for this problem, requiring many topology optimization iterations. Hence, the solutions to the lower value moment constraints will usually contain less gray material and better angle orientations. Lastly, the degree of the actual moment constraint error is much higher in this example compared to the 2-dimensional example. This trend is common with 3-dimensional results since obtaining moment constrained results is more difficult than simple 2-dimensional examples.

5.4 Beam Example

This example is a typical support beam design problem. Figure 5.13 displays the problem definition where 35% design material is used.

5.4.1 Standard Solution Results

The result obtained with standard topology optimization procedures is displayed in Figure 5.14 and Figure 5.15 where elements with densities greater than 0.3 are displayed. Figure 5.15 is provided to help visualize the standard solution. The optimal

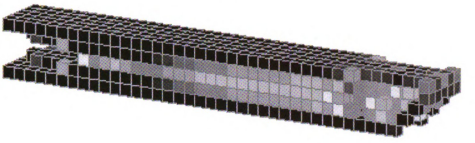

Figure 5.14: Beam example standard solution

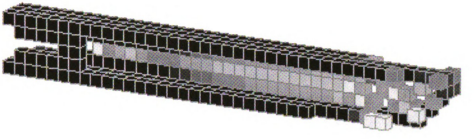

Figure 5.15: Beam Example standard solution. (Front rows removed)

design possess a compliance of 6.42. In addition, computing the moment about the axis as displayed in Figure 5.13 with spatial order $r = 2.0$, reveals $M_l(2) = 2.68\text{E+}03$.

5.4.2 Moment Constrained Solution Results

Three moment constrained topology optimization problems are displayed for this example. The constrained moments are of values 2.50E+03, 2.25E+03, and 2.00E+03. The tabular results for this example are displayed in Table 5.3. Graphical representation of optimal moment constrained solutions are shown in Figures 5.16 through 5.19.

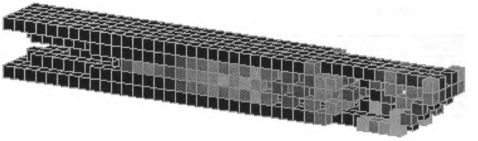

Figure 5.16: Beam example - moment constraint $M_l(2) = 2.5\text{E}{+}03$ solution

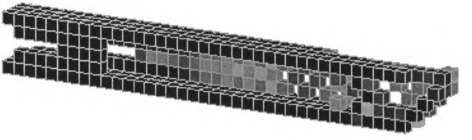

Figure 5.17: Beam example - moment constraint $M_l(2) = 2.5\text{E}{+}03$ solution. (Front Rows removed)

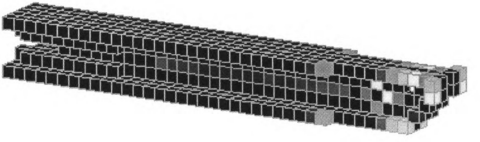

Figure 5.18: Beam example - moment constraint $M_l(2) = 2.25\text{E}{+}03$ solution

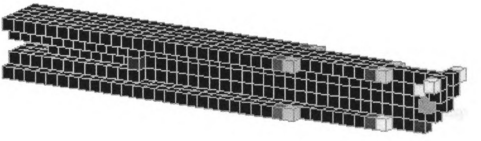

Figure 5.19: Beam Example - moment constraint $M_l(2) = 2.0\text{E}{+}03$ solution

Table 5.3: Beam moment constrained example results

Constraint	Actual Moment	Allowable Error	Actual Error	Compliance
2.50E+03	2.50E+03	5%	0.0%	6.41
2.25E+03	2.18E+03	5%	3.1%	6.24
2.00E+03	2.05E+03	5%	2.5%	6.25

5.4.3 Discussion

This simple example demonstrates some of the problems associated with standard 3-dimensional topology optimization problems. In many situations where bending modes dominate the loading conditions, hollow structures are determined as the optimal solution. From the standard solution displayed in Figure 5.14 and Figure 5.15, this effect is clearly observed. In many situations, this type of solution is not desired for manufacturing considerations. Here, the use of moment constrained topology optimization removes these manufacturing concerns by producing a version of an I beam as seen most clearly in Figure 5.19.

5.5 Tower Example

This example is similar to the tower examples examined with hierarchical analysis. Figure 5.20 displays the problem definition where 35% design material is used.

5.5.1 Standard Solution Results

The result obtained with standard topology optimization procedures is displayed in Figure 5.21 where elements with densities greater than 0.3 are displayed. The optimal design possess a compliance of 1.97. In addition, computing the moment about the

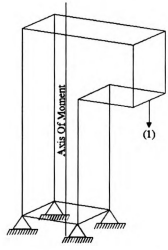

Figure 5.20: Tower example problem definition

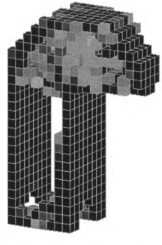

Figure 5.21: Tower example standard solution

axis as displayed in Figure 5.20 with spatial order $r = 2.0$, reveals $M_l(2) = 1.89\text{E+}04$.

5.5.2 Moment Constrained Solution Results

In this example, six moment constrained topology optimization problems are displayed. The tabular results for this example are displayed in Table 5.4. Graphical representation of optimal moment constrained solutions are shown in Figures 5.22

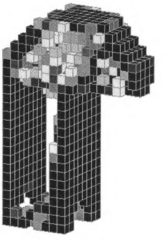

Figure 5.22: Tower example - moment constraint $M_l(2)$ = 1.8E+04 solution

through 5.27.

Table 5.4: Tower moment constrained example results

Constraint	Actual Moment	Allowable Error	Actual Error	Compliance
1.80E+04	1.80E+04	5%	0.0%	1.96E+00
1.60E+04	1.67E+04	5%	4.4%	1.88E+00
1.40E+04	1.46E+04	5%	4.3%	1.92E+00
1.20E+04	1.26E+04	5%	5.0%	1.99E+00
1.00E+04	1.04E+04	5%	4.0%	2.14E+00
8.00E+03	8.24E+03	5%	3.0%	2.51E+00

5.5.3 Discussion

The goal of applying moment constraints to this problem is to demonstrate the range of solutions that may be obtained though this new method. Examining the tabular data shows that only the few lowest constrained moment problems produce optimal solutions with compliances much different than the standard solution. Not surprisingly, the the graphical representation of the optimal solutions demonstrate the same behavior. As the moment constraint lowers, there exists a definite thinning of the

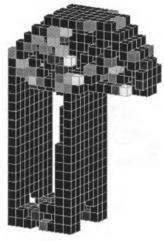

Figure 5.23: Tower example - moment constraint $M_l(2) = 1.6\text{E}{+}04$ solution

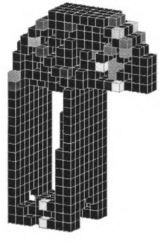

Figure 5.24: Tower example - moment constraint $M_l(2) = 1.4\text{E}{+}04$ solution

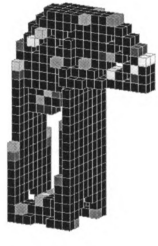

Figure 5.25: Tower example - moment constraint $M_l(2) = 1.2\text{E}{+}04$ solutions

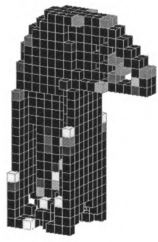

Figure 5.26: Tower example - moment constraint $M_i(2) = 1.0\text{E}{+}04$ solution

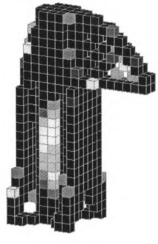

Figure 5.27: Tower example - moment constraint $M_i(2) = 8.0\text{E}{+}03$ solution

upper region of the optimal design (Figures 5.22, 5.23, 5.24, and 5.25). As the moment constraint is reduced more, the I beam characteristics from the previous example are present (Figure 5.26 and Figure 5.27). Clearly, the spatial constrained topology optimization algorithm provides the desired shape control in this situation.

5.6 Summary

This chapter presented various examples of spatial moment constrained topology optimization solutions. From these examples, it is clear that the initial goal of providing additional control of the optimal shape has been obtained. The last remaining issue focuses on the selection of the actual moment constraint values. Namely, how does one chose a moment constraint and corresponding value for a given problem and how does one guarantee that the solution procedure will be stable?

In order to determine which moment constraint(s) to use and what constraint values to chose, one should first obtain a standard topology optimization solution for the problem and perform a full moment analysis on the optimal shape. Here, solve the given problem using standard methods and compute all the moments of interest for the given optimal shape. Next, determine which attributes of the shape are not desired and find moments that penalized those attributes. In the examples presented, a removal of exterior regions is desired (i.e., thinning of the structure). This led to a selection of the $M_l(2)$ moment constraint with values lower than the standard results. In general, the process may require a few iterations.

In order to guarantee that the solution procedure will be stable, one must carefully

choose the initial penalty parameter, $\alpha_p^{(0)}$, the penalty growth factor, r_p, and the moment constraint values. In most cases, these values should be selected together. For example, in problems where large reductions in a given moment are desired, low values of $\alpha_p^{(0)}$ and r_p should be used to provided smooth convergence. Unfortunately, selecting parameters in this fashion results in more iterations of the optimization sequence. Conversely, in problems where only small reductions in a given moment desired, the value of $\alpha_p^{(0)}$ and r_p may be set more aggressively to promote rapid convergence.

As stated previously, spatial moment constrained topology optimization offers a method of shape control to standard topology optimization practices. However, with this control is an associated cost of additional computational resources to solve the iterative problem and the necessary experience needed to select the optimization parameters wisely.

Chapter 6

Conclusions

6.1 Summary

This research has provided two new methods for solving topology optimization problems. The objectives of this research focused on reducing resources necessary in solving large scale, 3-dimensional topology optimization problems and in formulating a method to introduce the idea of shape control to topology optimization problems. From these goals, the new methods of *Hierarchical Topology Optimization* and *Spatial Moment Constrained Topology Optimization* were developed, implemented, and demonstrated.

6.1.1 Hierarchical Topology Optimization

The goal of solving large scale, 3-dimensional topology optimization problems more efficiently is obtained through the successful use of hierarchical data structures, *octrees*. This interaction required extension of binary algorithms to address grayscale

data, the use of image processing, and efficient solution to a highly constrained set of equilibrium equations. Lastly, individual components were successfully integrated to produce an efficient strategy for large scale, 3-dimensional topology optimization problems. For most classes of problems, hierarchical topology optimization produces similar results to standard topology optimization procedures using a fraction of the required CPU time and comparable in-core memory requirements. These general trends are seen in the examples provided. The method is most successful in problems where optimal shapes consist of *thick* members. The only known difficulties occur when addressing problems that possess an optimal shape consisting of a large number of *thin* regions.

6.1.2 Spatial Moment Constrained Topology Optimization

The goal of providing shape control to topology optimization problems is achieved through the use of scalar shape descriptors, *spatial moments.* A new topology optimization problem formulation is developed that incorporates spatial moment constraints. These constraints are enforced to guarantee optimal shapes that possess desired attributes based on the constrained moments. The solution algorithm provides the ability of additional control of the optimal shape as demonstrated in illustrative examples. The main advantages of spatial moment constrained topology optimization are the additional control of the optimal shape and the flexibility of the types of constraints that may be placed on a given problem. The disadvantages are the difficulty of selecting appropriate solution parameters, α_p and r_p, and the actual moment

constraint values that produced reasonable results in a stable manner.

6.2 Areas Of Future Work

As mentioned previously, the thesis focuses on the development of two new solution strategies. Since this thesis provides only a first attempt in solving the problems of interest, much future work is possible.

6.2.1 Hierarchical Topology Optimization

Although the use of hierarchical topology optimizations is very successful, additional work in the following areas is suggested.

- Examine the use of alternative merge criteria in the octree encoding algorithm to provide more efficient discretization of a given shape.

- Implement more efficient solvers for the highly constrained finite element problem used to guarantee compatibility in the hierarchical meshes.

- Examine other hierarchical data structures that may more efficiently develop finite element discretizations for shapes with a large number of thin regions.

6.2.2 Spatial Moment Constrained Topology Optimization

In the area of spatial moment constrained topology optimization, additional work in the following areas is suggested.

- Develop an algorithm to determine reasonable values for the penalty terms α_p and r_p.

- Examine alternative methods for modeling constraints as penalty terms, such as barrier methods.

- Examine the introduction of sequential non-linear programming techniques to solve the constrained problems versus optimal conditions algorithms.

Appendices

Appendix A

Solution Of Topology Optimization Problems

A.1 Introduction

This appendix addresses the methods used in solving generalized topology optimization problems. The problem statement for this class of problems using a homogenization methods in 3-dimensions is shown in (A.1).

$$
\begin{array}{ll}
\text{Minimize:} & \int_\Omega < \mathbf{f}, \mathbf{u} > d\mathbf{u} \\
\mathbf{x} & \\
\\
\text{Subject To:} & \int_\Omega \rho(\mathbf{x})\, d\Omega \; - \; \text{Vol} \; \leq \; 0 \\
& -a_e \leq 0 \qquad \text{for } e = 1, \text{N} \\
& a_e - 1 \leq 0 \qquad \text{for } e = 1, \text{N} \\
& -b_e \leq 0 \qquad \text{for } e = 1, \text{N} \\
& b_e - 1 \leq 0 \qquad \text{for } e = 1, \text{N} \\
& -c_e \leq 0 \qquad \text{for } e = 1, \text{N} \\
& c_e - 1 \leq 0 \qquad \text{for } e = 1, \text{N} \\
& \mathbf{K}(\mathbf{x})\mathbf{u} \; - \; \mathbf{f} \; = \; \mathbf{0} \\
\\
\text{Where,} & \mathbf{x} \; = \; (a_1, b_1, c_1, \psi_1, \phi_1, \theta_1, \ldots, \\
& \quad = \; a_N, b_N, c_N, \psi_N, \phi_N, \theta_N) \\
& \rho_e \; = \; 1 \; - \; (1 - a_e)(1 - b_e)(1 - c_e) \\
& N \; = \; \text{The number of finite elements.}
\end{array}
\tag{A.1}
$$

A.2 Optimality Conditions

An optimum criteria based algorithm is used to solve the optimization problem associated with (A.1). In order to use this algorithm, the conditions for which an optimum solution must satisfy must be developed. Using standard optimization procedures, these conditions may be determined by examining the stationary condition of the Lagrangian for (A.1). In this case, the Lagrangian may be expressed as

$$
\begin{aligned}
\mathcal{L} \; = \; \int_\Omega < \mathbf{f}, \mathbf{u} > d\mathbf{u} \; + \; & \Lambda \left[\int_\Omega \rho \, d\Omega \; - \; Vol \right] \\
& + \; \sum^N \alpha_e^+ (a_e \; - \; 1) \; + \; \sum^N \alpha_e^- (-a_e) \\
& + \; \sum^N \beta_e^+ (b_e \; - \; 1) \; + \; \sum^N \beta_e^- (-b_e) \\
& + \; \sum^N \gamma_e^+ (c_e \; - \; 1) \; + \; \sum^N \gamma_e^- (-c_e) \\
& - \; \mu^T [\mathbf{f} \; - \; \mathbf{K}\mathbf{u}]
\end{aligned}
\tag{A.2}
$$

where, Λ represents the Lagrangian multiplier for the volume constraint and μ contains the Lagrangian multipliers for the equilibrium equation constraint. Also, $\alpha_e^+, \beta_e^+, \gamma_e^+, \alpha_e^-, \beta_e^-$, and γ_e^- represent the Lagrangian multipliers for the upper and lower constraints on the design variables. In discrete form, the Lagrangian may be represented as

$$\begin{aligned} \mathcal{L}_D \;=\; \mathbf{f}^T\mathbf{u} \;+\; \Lambda\left[\sum^N \rho_e v_e \;-\; Vol\right] \\ +\; \sum^N \alpha_e^+(a_e \;-\; 1) \;+\; \sum^N \alpha_e^-(-a_e) \\ +\; \sum^N \beta_e^+(b_e \;-\; 1) \;+\; \sum^N \beta_e^-(-b_e) \\ +\; \sum^N \gamma_e^+(c_e \;-\; 1) \;+\; \sum^N \gamma_e^-(-c_e) \\ -\; \mu^T[\mathbf{f} \;-\; \mathbf{K}\mathbf{u}] \end{aligned} \tag{A.3}$$

The optimality conditions require stationary conditions for the Lagrangian with respect the design variables and the displacement vector. To satisfy the optimality conditions, the following relationships must hold.

$$\frac{\partial \mathcal{L}_D}{\partial \mathbf{u}} \;=\; \mathbf{0} \tag{A.4}$$

$$\frac{\partial \mathcal{L}_D}{\partial a_e} \;=\; \mathbf{0}; \quad \frac{\partial \mathcal{L}_D}{\partial b_e} \;=\; \mathbf{0}; \quad \frac{\partial \mathcal{L}_D}{\partial c_e} \;=\; \mathbf{0}; \qquad \text{for } e = 1, \text{N} \tag{A.5}$$

Re-writing $\mathbf{f}^T\mathbf{u}$ as $\mathbf{u}^T\mathbf{K}\mathbf{u}$ the stationary points with respect to the displacement vector, $\mathbf{u}$, may be expressed as

$$\frac{\partial \mathcal{L}_D}{\partial \mathbf{u}} = 2\mathbf{u}^T\mathbf{K} + \mu^T\mathbf{K} = \mathbf{0} \tag{A.6}$$

This implies,

$$\mu = -2\mathbf{u} \tag{A.7}$$

Examining the optimality conditions with respect to the cell variables, a_e, b_e, and c_e, the following relationships may be developed.

$$\frac{\partial \mathcal{L}_D}{\partial a_e} = \Lambda \frac{\partial \rho_e}{\partial a_e} v_e + \alpha_e^+ - \alpha_e^- - \mathbf{u}^T \frac{\partial \mathbf{K}}{\partial a_e}\mathbf{u} = 0 \qquad \text{for } e = 1,\text{N} \tag{A.8}$$

$$\frac{\partial \mathcal{L}_D}{\partial b_e} = \Lambda \frac{\partial \rho_e}{\partial b_e} v_e + \beta_e^+ - \beta_e^- - \mathbf{u}^T \frac{\partial \mathbf{K}}{\partial b_e}\mathbf{u} = 0 \qquad \text{for } e = 1,\text{N} \tag{A.9}$$

$$\frac{\partial \mathcal{L}_D}{\partial c_e} = \Lambda \frac{\partial \rho_e}{\partial c_e} v_e + \gamma_e^+ - \gamma_e^- - \mathbf{u}^T \frac{\partial \mathbf{K}}{\partial c_e}\mathbf{u} = 0 \qquad \text{for } e = 1,\text{N} \tag{A.10}$$

From these developments, an algorithm for solving the generalized topology optimization problem may be developed.

A.3 Optimality Conditions Algorithm

Figure A.1 displays a flow chart for an optimal conditions algorithm applied to generalized topology optimization. Below is a brief description of each major step performed in the algorithm.

- Finite Element Model - The finite element description of the problem domain is provided including all boundary and loading conditions.

- Shape Optimization Control - Control parameters for the topology optimization are prescribed.

- Initialization - Here, the initial design for the optimization is set.

- Material Model - Computation of the effective material properties for each element is performed.

- Finite Element Analysis - Finite element analysis is performed for the current set of material properties.

- Mutual Energy Analysis - All gradients needed to examine the optimality conditions are computed.

- Angle Update - The orientation of the microcell for each design element is aligned according to the stress information from the finite element analysis.

- Update Density, a, b, & c - A new design vector is computed using the optimality conditions and the following relationship

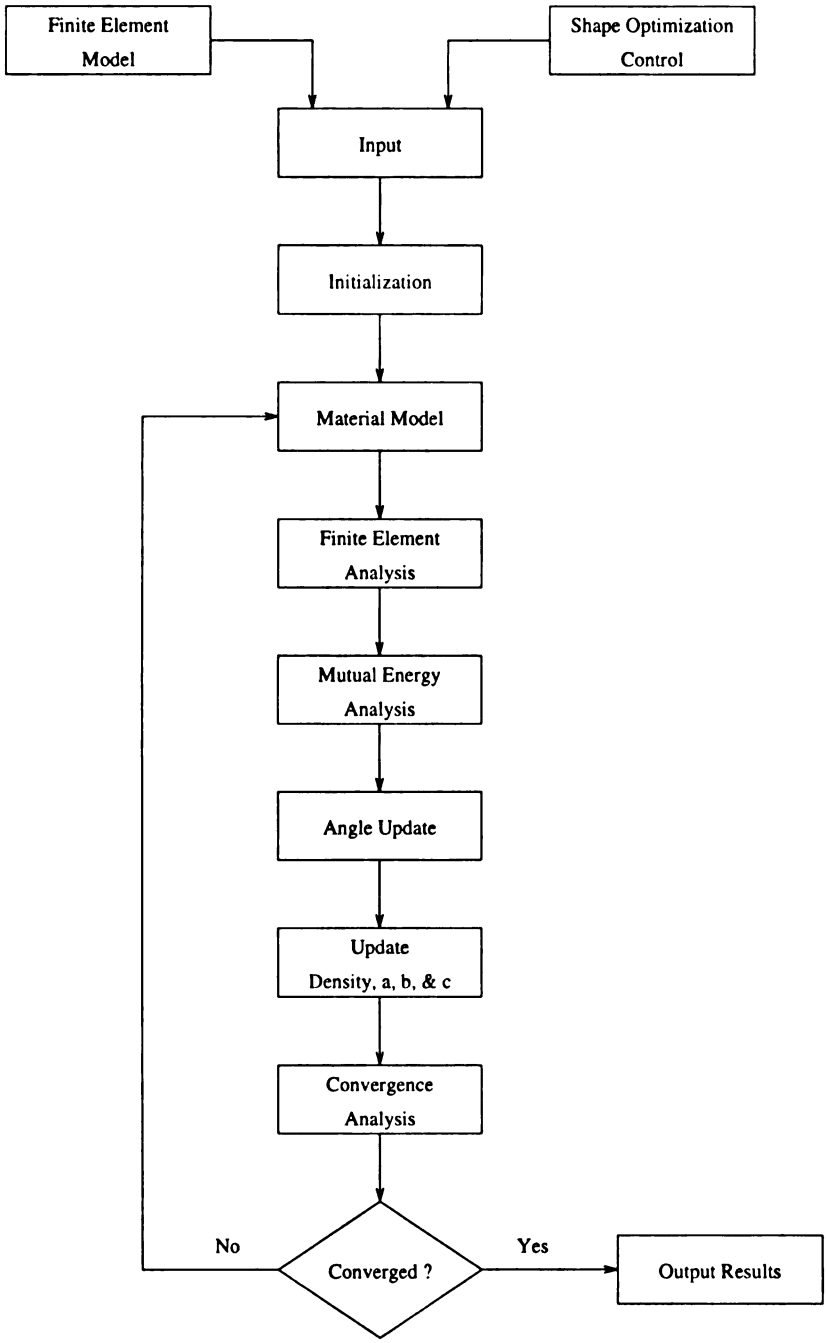

Figure A.1: Optimality conditions algorithm flow chart

$$x_i^{(k+1)} = \left[\frac{\mathbf{u}^T \frac{\partial \mathbf{K}}{\partial x_i} \mathbf{u}}{\Lambda \frac{\partial \rho_e}{\partial e_i}} \right]^{\eta} x_i^{(k)} \tag{A.11}$$

where, η is a parameter used to control the rate of convergence.

- Convergence Analysis - Here, a design is declared optimal if the average compliance has stabilized.

Bibliography

Bibliography

[1] G. Allaire, E. Bonnetier, G. Francfort, and F. Jouve. Shape optimization by the homogenization method. Technical report, Centre de Mathematiques Appliquees, 1995.

[2] M. Bendsoe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Computational Methods in Applied Mechanics and Engineering*, 71:197–224, 1988.

[3] C. R. Giardina and E. R. Dougherty. *Morphological Methods in Image and Signal Processing.* Prentice Hall, 1988.

[4] R. B. Haber, C. S. Jog, and M. P. Bendsoe. Variable-topology shape optimization with a control on perimeter. In *Proceedings Of The 20th ASME Design Automation Conference*, 1994.

[5] R. Haftka and Z. Gurdal. *Solid Mechanics And Its Applications.* Kluwer Academic Publishers, 1993.

[6] G. DeRose Jr. and A. Diaz. Hierarchical solution of large-scale three-dimensional topology optimization problems. In *Proceedings Of The 1996 ASME Design Engineering Technical Conference and Computer in Engineering Conference*, To appear.

[7] R. V. Kohn and G. Strang. Optimal design and relaxation of variational problems, I. *Communications on Pure and Applied Mathematics*, 39(1):113–137, 1986.

[8] R. V. Kohn and G. Strang. Optimal design and relaxation of variational problems, II. *Communications on Pure and Applied Mathematics*, 39(2):139–182, 1986.

[9] R. V. Kohn and G. Strang. Optimal design and relaxation of variational problems, III. *Communications on Pure and Applied Mathematics*, 39(3):353–377, 1986.

[10] M. Mantyla. *An Introduction to Solid Modeling.* Computer Science Press, 1988.

[11] F. Murat and L. Tartar. Calcul des variations et homogeneisation. *Coll de al Dir. des Etudes et Recherches de Electricite de France*, pages 319–370, 1985.

[12] W. Pratt. *Digital Image Processing.* John Wiley And Sons, Inc., 1991.

[13] W. Press, S. Teukosky, W. Vetterling, and B Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[14] H. Samet. *Applications of Spatial Data Structures.* Addison-Wesley Publishing Company, 1990.

[15] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Publishing Company, 1990.

[16] Ole Sigmund. *Design Of Material Structures Using Topology Optimization.* PhD thesis, Technical University Of Denmark, 1994.

[17] K. Suzuki and N. Kikuchi. Shape and topology optimization for generalized layout problems using the homogenization method. *Comp. Meth. Appl. Mechs. Engng.*, 93:291–318, 1991.

[18] K. Suzuki and N. Kikuchi. Shape and topology optimization in three-dimensional solid-structures. Technical report, University of Michigan, USA, 1991.

[19] F. Tomita and S. Tsuji. *Computer Analysis of Visual Textures.* Kluwer Academic Publishers, 1990.

[20] J. M. Winget and T. J. R. Hughes. Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. *Computer Methods In Applied Mechanics And Engineering*, 52:711–815, 1985.