ZEROTH-ORDER SHAPE OPTIMIZATION UTILIZING A LEARNING CLASSIFIER SYSTEM

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Robert A. Richards

August 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sheri D Sheppard

Sheri D. Sheppard, Ph.D. (Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John Koza

John Koza, Ph.D.

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

[illegible]

Mark Cutkosky, Ph.D.

Approved for the University committee on Graduate Studies:

[illegible]

Abstract

A methodology to perform generalized zeroth-order two- and three-dimensional shape optimization utilizing a learning classifier system is developed and applied. To this end, the applicability of machine learning to mechanical engineering is investigated. Specifically, the methodology has the objective of determining the optimal boundary to minimize mass while satisfying constraints on stress and geometry.

Even with the enormous advances in shape optimization no method has proven to be satisfactory across the broad spectrum of optimization problems facing the modern engineer. The methodology developed in this dissertation is based upon a classifier system (CS) and exploits the CS's adaptability and generality. It thereby overcomes many of the limitations of today's conventional shape optimization techniques. A CS learns rules, postulated as if-then statements, in order to improve its performance in an arbitrary environment, (which for this investigation consists of stress and mass information from components). From this input, and from a population of initially randomly generated rules, the classifier system is expected to learn to make the appropriate component shape modifications to reach a minimum mass design while satisfying all stress constraints. The CS learns by utilizing the design improvement success or failure feedback.

Nearly all shape optimization algorithms developed to date depend on sensitivity information in order to function. This research does not present sensitivity information to the classifier system. Thus, the classifier system must not only learn from a clean slate, but confronts the additional challenge of learning without information that most other shape optimization algorithms deem essential. Therefore, the main deliverable is a zeroth-order shape optimization methodology.

After a review of mechanical engineering shape optimization methods, an explanatory presentation of CSs and their underlying genetic algorithm (GA) describes how classifier systems learn from feedback and the GA. With this foundation set, the coupling of the shape optimization domain with the classifier system proceeds to form, the **S**hape o**P**timization via **H**ypothesizing **I**nductive classifier system comple**X** (**SPHINcsX**). The complex learns shape optimization by its application to a suite of sizing optimization problems.

The most tangible artifact of this research is the successful development of the zeroth-order shape optimization complex. The complex proved adept at solving both two- and three-dimensional shape optimization problems. The research also provides a demonstrative example of the power and flexibility of machine learning in general and CSs in particular — how they may be leveraged as tools for mechanical engineering design, and insights into their proper application.

Acknowledgments

I would like to express my deep gratitude to my advisor Professor Sheri Sheppard, who was willing to join me in the ambitious goals of this endeavor. I depended on her erudition and more importantly her keen insight throughout this research. Thanks to Professor John Koza for his invaluable aid, guidance and encouragement during the course of this dissertation. Thanks to Professor Mark Cutkosky for reading and reviewing this dissertation and for his helpful suggestions.

Thank you to my parents, Gillian and Alfonso Richards, for their support and love, as well as their unwavering equanimity. The support of the rest of my immediate and extended family is warmly appreciated.

Deep thanks to my friend Alberto Makino for all his assistance and advice. I greatly appreciate the constancy and support of all my friends who I too many times had to ignore as a sacrifice to my research, these include, Kevin Gonsalves, Donny Hoirup, John Lau, Stergios Marinopoulos, John Marsella, Mike Maynard, Ron McFadden, and Ron Smith. I thank my colleagues, including Jim Widmann, Veejay Mehorta, and Marco Dorigo, for their ideas and feedback.

I thank Randy Melen, Shirley Gruber and Tim Torgenrud and the rest of my colleagues at the Distributed Computing and Communication Systems for their personal support as well as the access to extensive computing resources. I am grateful to Lee Sharpe for the opportunity to use Structural Dynamics Research Corporation's facilities for my research. In addition, I thank Mark Lawry and Thomas Sigafoos for their friendship and support in the acquisition and use of I-DEAS™.

I can not begin to thank Anita Gallegos for her love, patience, and fortitude. Anita has been an inspiration to my research and so many other aspects of my life. I trust that now when she inspires me to verse, I will be able to complete.

To Anita:

Enchafed caritas absentia enkindled

Oblation to orb's autochthons, picayune solace.

Finally, I would like to acknowledge, and apologize, to all those others that have done so much to assist me and to whom I have committed the offense of omitting.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Shape optimization is an emerging part of the mechanical engineering design process, and the tools available to assist the mechanical engineer significantly affect which types of problems are optimized and to what degree. During the past 50 years there has been a major introduction of valuable optimization methodologies into the field of mechanical engineering.

Although the volume of recent research may imply that shape optimization is a modern science, shape optimization has played a pivotal role since antiquity. Almost all design entails optimization, with the demarcation between design and optimization often being blurred. The wheel provides a historical illustration between design and shape optimization. Historical evidence implies that the wheel was first invented in Mesopotamia during the 4th millennium B.C.E. The design and fabrication of the wheel was a monumental design achievement. However, shortly after 2,000 B.C.E. in northern Mesopotamia, central Turkey and northeast Persia, shape optimization improved the wheel considerably — the spoked wheel. How many civilizations may have met their demise because they failed to see the importance of shape optimization, facing battle against spoked chariots, armed themselves with heavy rolling stock, fashioned with solid wheels?

The range of designs and pace at which shape optimization can be applied has been a function of the tools available to the designer. Just as the abacus provided a significant new tool to designers and others, a myriad of scientific advances including the computer today has facilitated the development of many new shape optimization tools including the one developed in this book.

1.1 Objective & Scope

This dissertation has many interrelated objectives. The most general objectives are:

- to develop a scalable[†] methodology for component optimization, from simple two-dimensional sizing designs to three-dimensional shape designs,
- to investigate machine learning's applicability as a tool for mechanical engineering design.

The particular machine learning technique which will be investigated is the *learning classifier system*. The investigation consists of combining the above two objectives by determining if classifier systems can be used for general shape optimization. Some of the sub-objectives for the system to be 'general' is that the methodology must:

- perform without the benefit of sensitivity information,
- be independent of the analysis tool,
- be virtually independent of the boundary representation.

Therefore the deliverable objective is the development of:

- a methodology and complex which performs zeroth-order shape optimization via a learning classifier system.

Two indirect objectives are: to provide guidelines for interfacing the classifier system domain with the mechanical engineering domain, and to convey any serendipities discovered during this exploration.

The scope of the research will be limited as follows:

- optimization will be on isotropic components acted upon by loads, and in static equilibrium due to a set of fixtures, or constraints,
- the design modifications will occur by modifying the boundaries of the component; no new boundaries such as voids will be introduced,
- the objective will be to minimize the component's mass,
- constraints will be placed on the maximum stress allowed.

[†] See *Appendix D, Glossary* for precise definition of *scalable* with regard to this research.

1.2 Motivation

Although a plethora of optimization tools have been developed and enormous progress has been accomplished, many, including Haftka and Grandhi [1986], note that no one method has proven to be satisfactory across the broad spectrum of optimization problems faced by the modern engineer. Various methods, such as dynamic programming, guarantee the solution in their realm of applicability, but computational requirements render them useless for large problems. Other methods, such as calculus-based gradient search procedures, converge nicely, but are limited to problems possessing properties such as continuity and unimodality. Expert systems show promise in that they aid engineers' reasoning via rules, however, the determination of the appropriate rules has proven an arduous and tedious undertaking.

Shape optimization research provides quantifiable benefit to society, for each extra kilogram in, for example, a train, plane, or automobile requires extra resources to build, extra energy to move, supplementary processing to recycle or additional space to dispose. Obviously, motivation exists for further development of efficient optimization methods that are applicable over a broad spectrum of design problems. In addition, shape optimization provides a pragmatic exercise to test the more general goal of investigating machine learning's applicability as a tool for mechanical engineering design.

1.3 Background & Approach

A myriad of techniques were explored, as part of this research, for the salient features desired to excel in the realm of mechanical engineering and shape optimization, while overcoming limitations inherent to calculus-based methods. The explored techniques were drawn from many fields including, mathematics, computer science, operations research, and psychology, and may be categorized as:

- artificial intelligence & machine learning,[†]

[†] For this research, *artificial intelligence* subsumes deductive methodologies while *machine learning* subsumes inductive methodologies.

- enumeration & random,
- nature analogy,
- hybrid & other.

All of these techniques have been exploited already in the mechanical engineering literature. For example, artificial intelligence in general, and expert systems in particular, has seen wide application in design by such researchers as Sriram et al., [1989], Gero [1990], Stolfo [1984], Prabhakar [1994], and Takeda et al., [1990]. Applications have ranged from the design of large structures (Adeli & Balasubramanyam [1988]), to rapid prototyping (Glasgow & Graham [1988]), to the development of an expert system for the optimum selection of filler metals for welds developed by the Colorado School of Mines (Jones & Turpin, [1986]). Other areas of Artificial Intelligence have been applied to design, such as Information Processing Theory by Goel and Pirolli [1989]. Nature analogy techniques, have received attention, albeit less, by researchers such as Coyne [1990] who have employed neural networks in design reasoning.

An overriding issue when performing the literature review for this research was the need for *flexibility* and *adaptability* in the technique chosen in order to perform generalized shape optimization. Generalized shape optimization entails covering a wide scope of problems and performing efficient and effective optimization. Because of machine learning's learning/induction capability, adaptability is an inherent quality; hence machine learning deserved further investigation. Before machine learning could be considered further, a machine learning technique had to be found that was flexible enough to interface naturally to the shape optimization domain. Furthermore, if engineers are expected to accept the results, it would be beneficial if they could see and understand how the results were obtained, and not have to lay blind trust in a black box.

A machine learning technique which shows great promise and as will be shown has excelled is the learning classifier system (further discussed in Chapter 3). It realizes the adaptability criterion by, in one respect, being a self-teaching expert system. That is, a classifier system learns the rules necessary to operate efficiently in an environment, (in this study the environment is: shape optimization). Once it has learned, the engineer could view it as an expert system with the chain of rules used to perform the optimization acting as an explanation facility. Finally, the classifier system is amenable to being interfaced to the shape optimization domain.

The approach then is to take the general shape optimization problem and interface it with a relatively simple classifier system. The classifier system can potentially overcome many of the limitations of current methods due to its generality, thereby not depending on the problem to be optimized possessing certain properties (as is the case in many present approaches). The system will take an initial design, calculate the stress properties and apply a set of rules (classifiers) to generate a new design. The rules attempt to match patterns of stress that occur on and interior to the modifiable boundaries. This iterative process will continue until an optimum design is created. The evolving design's changes in merit will be used as a feedback to the classifier system. Utilizing the feedback and a genetic algorithm, good rules will gain strength and breed with other good rules in an attempt to evolve better rules.

Many of the issues that need to be addressed to successfully apply classifier systems to shape optimization are determined by judgments made regarding the information flow between the design domain and the classifier system. The flexibility of the classifier system allows the information flow to be constructed independent of the design's complexity. This allows for the developments to scale up to larger and more complex problems with no change required in the basic procedure. This cannot be said of most other techniques used in shape optimization.

Although classifier systems and their underlying genetic algorithm are not common in the mechanical engineering field, genetic algorithms have already proven their worth in other real world applications. For example, General Electric has used a proprietary genetic algorithm and expert system package called "Engineous" to help design the engine for the Boeing 777, and a superconducting generator (Ashley [1992]). Other applications include designing optimum welds (Deb [1990]), and optimum paths in spacecraft rendezvous (Freeman, et al. [1990]).

The application of genetic algorithms directly, (i.e., without a classifier system framework), to shape optimization problems has been investigated by others (e.g., Dhingra [1990]). The genetic algorithm evolves a solution to one problem but then must start from scratch on any new problem. Since genetic algorithms work with a population, which in this case consists of designs, it is obvious that a multitude of different designs would need to be investigated. Even investigators who see this area as fruitful are realistic, as Jensen [1992] states euphemistically, "The efficiency of the genetic algorithm approach for structural design is not overly encouraging", or as Punch et al. [1994] admits "the use of genetic algorithms for optimal design of structures requires that hundreds or even millions of possible designs be analyzed".

As a mathematician with no knowledge of engineering is likely to perform sub-par to an engineer in the application of mathematics to engineering problems, one needs engineering acumen to successfully apply and interface classifier systems to engineering shape improvement or any other engineering problem. This point does not always express itself in the reading, but is abundantly clear when one attempts the application of classifier systems to the engineering realm, as in this research.

1.4 Deliverables & Benefits

This research will provide a myriad of benefits to the mechanical engineer, many of the benefits will have tangible incarnations in the form of software systems. A major benefit of this research is the development of a shape optimization methodology capable of:

- determining global optimal shapes,
- exhibiting stable performance,
- handling two and three dimensional problems,
- performing without the assistance of sensitivity information,

- rivaling the efficiency of state-of-the-art in shape optimization techniques,
- independence from the analysis technique used to determine stresses.

The instantiation of the methodology contributes the major deliverable, the **S**hape o**P**timization via **H**ypothesizing **I**nductive classifier **s**ystem comple**X** (**SPHINcsX**, pronounced *s f ĭ n k s*). The complex is a software package that performs zeroth-order shape optimization by exploiting an inductive classifier system for stress constrained problems.

The shape optimization methodology and its implementation demonstrate the benefit of machine learning as a tool for mechanical engineering design. More specifically this research benefits mechanical engineering by demonstrating:

- machine learning has its role in the mechanical engineer's tool box,
- an example of how to realize the tool's capabilities,
- guidelines for machine learning's application in general, and classifier systems in particular.

Another beneficial artifact is the:

- Learned population of classifiers.

Simply stated, a learned classifier system may be applied as an expert system. The learned population of classifiers is synonymous to the knowledge base in an expert system; and could be used directly in **SPHINcsX** or as the knowledge base in another expert system.

1.5 Outline of Dissertation

Chapter 2 provides a formal review of computer aided shape optimization. This includes the definition of shape optimization and its mathematical representation. Different classes of shape optimization will be reviewed and clarification of this work's scope will be laid out. Criteria necessary to perform shape optimization will be discussed, as well as the evaluation of the optimization's performance. The current state-of-the-art will be presented with associated achievements, performance and limitations.

Chapter 3 provides a pedagogical treatment of learning classifier systems. This chapter is more in-depth because the expected audience of this work is more engineering oriented and familiarity is not presumed in the machine learning domain. Ample references are provided to the interested reader to learn more about topics discussed throughout this dissertation. The classifier is described by breaking it down into its major components. Particular attention is paid to the genetic algorithm which provides much of the learning power to the classifier system. Once the mechanism of the classifier system has been discussed and portrayed graphically, the metrics used to measure its performance are introduced with empirical predictions for the expected time needed for learning.

Chapter 4 describes the integration of shape optimization with classifier systems. The information important enough to be presented to the classifier system is determined. This information is then presented to the classifier system in a form which is meaningful. From the classifier system, an effect is caused. The interplay between the output interface and the shape modification is described. The feedback mechanism is developed so that from the changes made to the design, appropriate information may be presented to the classifier system in order for it to learn.

Chapter 5 details the complete algorithm developed in this study and completes the definition of the **S**hape o**P**timization via **H**ypothesizing **I**nductive **c**lassifier **s**ystem comple**X** (**SPHINcsX**). The chapter sets the classifier system parameters described in Chapter 3, and includes the algorithm for initializing the classifier population. The final major portion of the chapter establishes the learning suite of problems used to mentor SPHINcsX.

Chapter 6 details the mentoring process used to teach SPHINcsX. The mentoring process consists of applying the learning suite established in Chapter 5 to SPHINcsX in learning mode. The chapter monitors the learning performance, ascertaining the criteria for deeming the complex learned.

Chapter 7 demonstrates and evaluates the performance of SPHINcsX. The chapter covers the differences between learning mode and application mode. In application mode, SPHINcsX acts much like an expert system. Next, SPHINcsX in application mode, is applied to a set of designs, all of which have never been seen before. The performance is evaluated and compared to other available techniques, and results.

Chapter 8 summarizes and draws conclusions from this research. An overview is presented and major new knowledge derived from this study is summarized. Then certain unexpected serendipities that arose during the research are described. Some minor limitations and ways in which they could be overcome are covered next. Finally, future directions are suggested and final conclusions are presented.

Appendices provide the complete initial classifier population, the learned population, a detailed version of the algorithms used by SPHINcsX, and a *glossary*.

Chapter 2

Shape Optimization — Background & Setting the Stage

Formal research into the optimization of structures has been attempted since antiquity; evidence of structural optimization in the modern era was first documented in the 17th century by Galileo in his *Treatise*, where the optimum shape of beams was investigated. The following sections will formally define shape optimization and review the state of the art.

2.1 Definitions

There have been many definitions advanced for *optimization theory* and *shape optimization*. A concise definition for optimization theory is given by Beightler et al. [1979] as:

> *Optimization theory encompasses the quantitative study of optima and methods for finding them.*

Another definition presented by Pike [1986] states:

> *The objective [of optimization theory] is to select the best possible decision for a given set of circumstances without having to enumerate all the possibilities.*

Much of the literature uses a definition similar to the following for shape optimization:

> The determination of the boundary form which best meets the design criteria while simultaneously satisfying all design constraints. Shape optimization is generally associated with structural optimization where the objective is to minimize mass with constraints imposed on stress, displacement, buckling and/or natural frequency.

Observe from these definitions that optimization theory has two main thrusts:

1. Determination of the optimum.
2. Methodologies for improvement leading to the optimum.

This, and most other works, deal with developing methodologies for improvement leading to an optimum. This is why the deliverable of this research is not a set of case studies which have been optimized but a technique and complex (as defined in Chapter 5) which can perform optimization on certain classes of problems.

The mathematical form of the shape optimization problem consists of the *objective function* whose minimum value is sought, and *constraints* limiting the range over which the objective function may be minimized. The objective function and the constraints are functions of the design variables, where the design variables consist of those entities which may be modified.

The constraints have two classifications: *explicit* and *implicit.* Explicit constraints can be expressed as a function of the design variables, while implicit constraints can not be expressed explicitly in terms of the design variables.

The mathematical form of the optimization problem is:

$$\begin{aligned} \text{minimize:} \quad & f(x) \\ \text{subject to:} \quad & g_e(x) \leq 0 \\ & g_i(x) \leq 0 \end{aligned} \tag{2.1}$$

where,

x	Vector (x_1, x_2, ..., x_l) of l design variables.
$f(x)$	Objective function.
$g_e(x)$	Vector ($g_e^1(x)$, $g_e^2(x)$, ..., $g_e^m(x)$) of m explicit constraints.
$g_i(x)$	Vector ($g_i^1(x)$, $g_i^2(x)$, ..., $g_i^n(x)$) of n implicit constraints.

2.2 Shape Optimization Classes

There are three distinct classes of shape optimization problems. In order of computational complexity, these are: size, shape, and topological optimization (Sandgren, et al. [1991]).

- *Size optimization* (also called *cross-sectional optimization*) refers to the determination of specific geometric dimensions for a pre-selected design class, such as the thickness of a shell, the size of a truss member or the radius of a circular stress element. This class of problems has been under modern investigation for decades (Schmidt [1960]).
- *Shape optimization* (also called *geometric optimization*) introduces additional design variables which allow for boundary movement. Due to its increased difficulty relative to size optimization, the geometrical changes have historically been limited; however, it has gained importance in the aircraft and automotive industries, as well as others, providing improvements to turbines, airfoil shapes and connecting arms (Ali [1994]). Size optimization is a subset of shape optimization.
- *Topological optimization* involves topological as well as shape and size modifications. Topological modifications deal with assemblies of components. The components in the assembly may be modified and components may be added, deleted or moved in the assembly in the attempt to generate an improved design. Relatively little work has been done in this area (Suzuki & Kikuchi [1991], Gage [1994], Reddy & Cagan [1994]), despite the importance of the concept.

The current investigation considers shape optimization. Although interest in and investigation of shape optimization research has been intensifying during the last few decades, as Sangren et al., [1991] emphasizes, "there is much work to be done before this class of optimization can become an integral part of the design process".

2.3 Criteria for Shape Optimization

The minimum criteria necessary for any optimization process are:

- evaluate objective,
- check constraints.

Though some optimization methods can operate utilizing only these facts, most methods developed to date cannot operate with the minimum criteria. Most need auxiliary information such as:

- derivatives (i.e., sensitivities),

and properties such as:

- continuity of objective function,
- closed form representation of objective function,
- unimodality of the objective function.

The derivatives required consist of the derivative of the objective function and the derivatives of the constraints with respect to the design variables. These derivatives are termed *sensitivities*. The process of determining the sensitivities is termed *sensitivity analysis*. For many problems this auxiliary information is unavailable or expensive to determine. Modern optimization algorithms have become so dependent on auxiliary information that many see some auxiliary information as being part of the minimum criteria. Haftka and Grandhi [1986] epitomize this sentiment:

Design sensitivity analysis, that is the calculation of quantitative information on how the response of a structure is affected by changes in the variables that define its shape, is a fundamental requirement for shape optimization.

This study investigates a technique which can operate with only the minimum criteria. In so doing, it not only surpasses enumeration and random search but competes favorably or exceeds techniques which exploit auxiliary information. Furthermore, the technique studied is flexible enough to be able to exploit auxiliary information to further increase its efficiency. Further background on other optimization methods is provided in Section 2.4.4.

2.4 Process of Shape Optimization

Figure 2.1 provides a simplified view of the shape optimization process. In Figure 2.1 the *Start* block represents an *initial condition* that must be set by human intervention before any automated optimization may proceed. The initial condition starts with a (normally feasible) design. A *feasible design* is one where none of the constraints are violated.

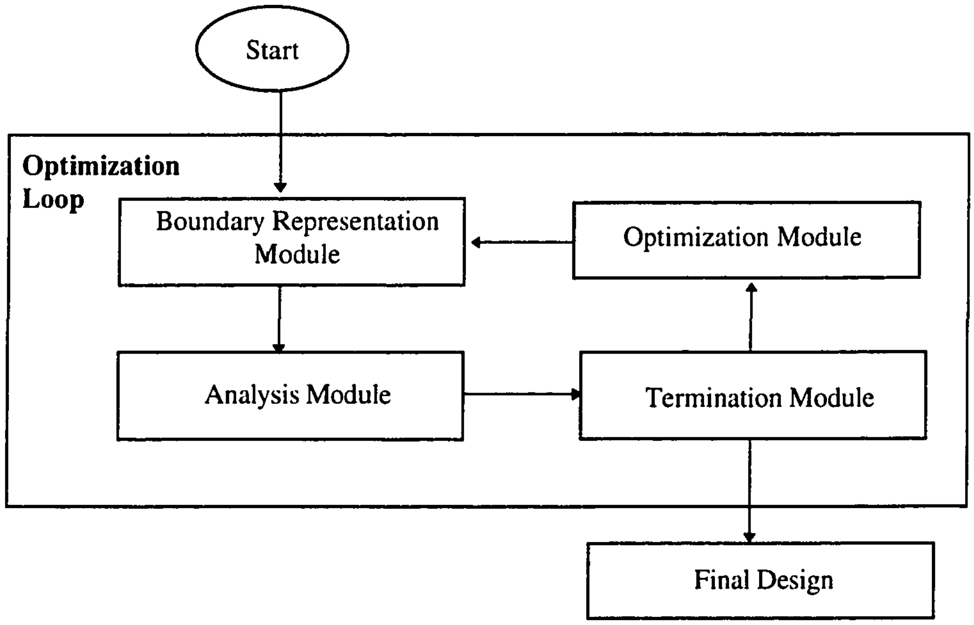


Figure 2.1 ***Shape Optimization Process Overview***

Figure 2.1 shows that there are four major modules in the shape optimization process:

- Boundary Representation,
- Analysis,
- Optimization,
- Termination.

The design needs to be built with at least one degree of freedom, usually more. These degrees of freedom are the design variables; the design variable vector may be as simple as a radius for a rod, or as complex as consisting of many sets of control points which define surface patches of complex surfaces. Therefore, a boundary representation must be defined as part of the design's initial condition. Figure 2.2 provides a more detailed view of the shape optimization process, (note that **X** in the figure is a vector of design variables). The boundary representation module and the design variables are further discussed in Section 2.4.1.

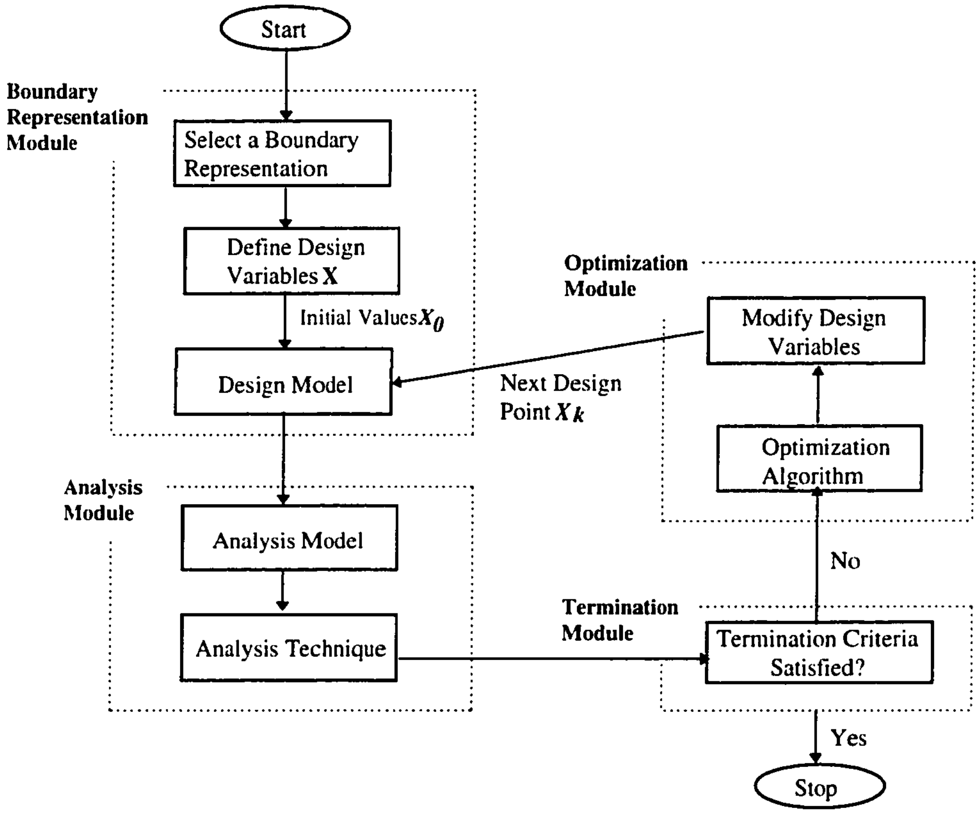


Figure 2.2 ***Optimization Process Major Modules***

After the boundary representation module defines the geometric representation of the design, the analysis module converts the model to an analysis model if the present geometric representation cannot be used directly. The analysis is performed so that the objective function and the implicit constraints may be evaluated. If any auxiliary information is needed by the optimization module, the analysis module must provide this also. This is further discussed in Section 2.4.2.

From the analysis module, the termination module checks to see if any of the termination criteria are met. From here the system branches, stopping if the termination module determines the termination branch should be taken, otherwise the branch is made to the optimization module. The termination module is further discussed in Section 2.4.3 and Chapter 4.

The optimization module determines which design variables to modify and by how much. The optimization module then changes the design variables. These new design variables are subsequently passed to the boundary representation module, completing the loop; this loop is termed an *optimization iteration*. The optimization module is further discussed in Section 2.4.4 and is the main thrust of this dissertation.

2.4.1 Boundary Representation & Design Variables

The optimal shape depends on the boundary representation and design variables selected to represent the modifiable boundaries, which define the *design space*. For example, the optimum shape of the cantilever beam shown in Figure 2.3 is dependent on the boundary representation and the number of design variables.

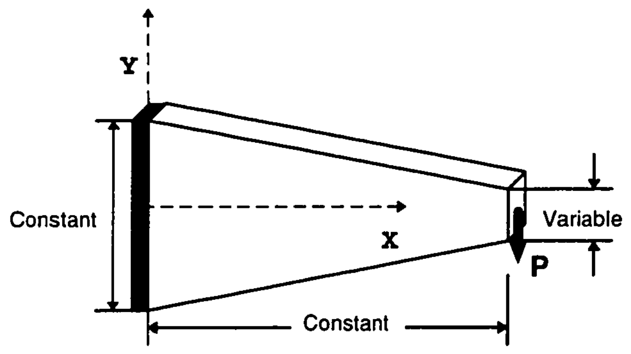


Figure 2.3 *Simple Cantilever Beam*

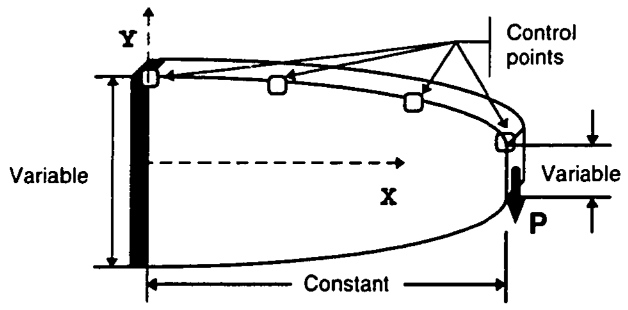


Figure 2.4 *Cantilever Beam with Spline Boundary Representation*

If the cantilever beam was modeled as shown in Figure 2.3 with only one design variable which is the height of the free end, then the optimum would be different than the model shown in Figure 2.4, where the modifiable boundary is represented as a series of cubic splines with four control points defining the curve.

Many research efforts have explored the relation of boundary representation to shape optimization (e.g., Widmann [1994]). Advances in Computer-Aided Engineering (CAE) have seen the decoupling of the analysis model from the boundary representation. Earlier studies set the design variables to the node locations in the finite element model, encountering many limitations (e.g., Fleury & Braibant [1983]), including distorted elements resulting in invalid analysis results.

Although the acceptance of the importance of decoupling the boundary representation from the analysis model is universal, a consensus on the best boundary representation has not been forthcoming. Many representations have been investigated, a non-exhaustive list is shown in Table 2.1.

Table 2.1 ***Boundary Representations***

Boundary Representation	As used by, e.g.
polynomials	Botkin [1982]
trigonometric functions	Yang & Fitzhorn [1992]
splines	Braibant & Fleury [1984]
linear curvature element	Widmann [1994]
intrinsic functions of arc length, and curvature	Hsu [1992]

This study's scope attempts to treat the boundary representation module as a black box; but since the optimization module must interface with the boundary representation module it must be cognizant of the representation used. To this end, an optimization module will be developed that is capable of working with many different boundary representations, however, with the singular restriction that the design variables (control points) are actually on the design boundary. This restriction still allows for most boundary representations including:

- lines, arcs, circles, ellipses,

- cubic splines (where the design variables are the points through which the cubic splines pass),
- surface patches.

The intention is to permit the designer the flexibility to construct the initial design with boundary representations appropriate for the situation, from which the optimization module will use the boundary representations as additional constraints.

2.4.2 Analysis Module

The analysis module's purpose is to determine the values of the:

- objective function,
- implicit constraints,
- auxiliary information (needed by the optimization module).

For a certain category of components these values may be calculated directly via first principles, (e.g., beam theory, elasticity, shell and plate theory). For more complex components, the finite element (FE) method (Zienkiewicz [1980]) provides a generally applicable technique. The boundary element method (Zhao [1991]) is another generally applicable method receiving increasing attention, but has yet to evolve into a commercial contender to the FE method.

Because of the difficulty of interfacing an analysis module and an optimization module, many optimization techniques have been coupled to the analysis module. This is the reason for the historical use of FE nodes as design variables. It is much easier to implement a system that utilizes the same representation for the geometric representation and the analysis representation. In the case of finite element analysis, advances such as, automatic mesh generation, adaptive mesh refinement, error estimates, and geometry-based constraints, have permitted the, albeit difficult, decoupling of the optimization module from the analysis module. Most modern commercial finite-element analysis packages, such as SDRC's I-DEAS™, Ansys Inc.'s ANSYS™, and Rasna's Mechanica™, provide most if not all of these advances. Although the implementation of the conversion from the geometrical representation to the analysis model and the interface between the analysis module and the optimization module may be non-trivial, as is the case in this study; this work will focus on the advancements made in optimization theory which occur in the optimization module.

The analysis module utilized here is envisioned as a black box which will provide the minimum criteria to the optimization module. Therefore, the optimization module will be designed not to be dependent on the technique used by the analysis module. To this end, two types of analysis modules will be used in this research, one will derive its results from closed form solutions while the other will utilize the finite element method.

2.4.3 Termination Module

The termination module checks criteria to determine if the optimization process should be terminated. Three categories classify most termination criteria: optimum, progress, and resource. *Optimum* termination criteria check if the design is optimal and if so, stops. *Progress* termination criteria check if improvement over one or many iterations falls below a threshold, stopping if so. *Resource* termination criteria stop if a resource limitation, such as computing resources, has been exceeded. The termination module terminates the optimization process as soon as *any*, of the termination criteria, is satisfied.

Optimum termination criteria provide the preferred method of ending the optimization process because the final design's degree of optimization is known. Unfortunately, for many design categories an optimum termination criterion is not forthcoming, therefore another termination criterion terminates the optimization process. An example of where an optimum termination criterion can be active is for designs where the optimum occurs when the modifiable boundaries are fully stressed in the optimized configuration. Designs which meet this condition include two-dimensional designs with no internal boundaries (e.g., holes) where the objective is to minimize mass [Hsu, 1992]. The optimization stops when the stresses on all the modifiable boundaries converge to within an allowable deviation, ε, from the fully stressed condition. The allowable deviation, ε, is a user specified unitless parameter. Many studies use a value of 0.05 or less; which means the stresses must converge to within 5% or less of the fully stressed state. Note that ε is zero for the optimal condition.

Progress termination criteria use lack of improvement over one or many iterations as the determining factor for stopping. For methods that guarantee, or have a high probability, of creating monotonically improving designs, or methods where a design change to a worse design is grounds for stopping, the optimization process is terminated when improvement between iterations drops below a threshold, $\varepsilon_{\mathrm{imp}}$. The improvement threshold, $\varepsilon_{\mathrm{imp}}$, is a user specified parameter. For optimization methodologies that search the design space in a manner that allows for new designs which may be worse than the previous design, termination occurs when improvement during the past n_{imp} iterations drops below $\varepsilon_{\mathrm{imp}}$. The number of past iterations, n_{imp}, to consider is normally a user specified parameter.

Resource termination criteria stop the optimization process when a resource limitation has been exceeded. Examples of resource based termination criteria are:

- maximum allowable number of optimization iterations,
- maximum allowable wall-clock time,
- maximum allowable expenditure of computing resources.

2.4.4 Optimization Module

The optimization module's purpose is to make modifications to the design which result in an optimal or improved design. This is accomplished by interfacing with the analysis module to acquire information necessary to proceed through the optimization module. As discussed above, this requires at least the minimum criteria and usually additional auxiliary information.

With all the necessary information at its disposal, the optimization module determines the design variable(s) to modify, and the magnitude of the modification(s). The optimization module then effects these decisions, modifying the design variables.

These new design variables are passed to the boundary representation module where the dimensions are changed and the geometric representation is updated.

A multitude of optimization algorithms may be used in the optimization module. For the purpose of review they are categorized as follows:

- calculus-based,
- artificial intelligence & machine learning,
- enumerative & random,
- nature analogy,
- hybrid & other.

Calculus-based methods have been studied the most. There are two main classes (Pike [1986]): *indirect methods*, and *direct methods*. *Indirect methods* seek local extrema by solving the usually, nonlinear set of equations resulting from setting the gradient of the objective function equal to zero. This is the generalization of the elementary calculus notion of extremal points. *Direct methods* seek local optima by beginning somewhere on the objective function and moving in a direction related to the local function gradient. This is the concept of hill-climbing. Calculus methods are local in scope, require the existence of derivatives and usually continuity and unimodality (of, for example the objective function). Such limitations destroy the technique's utility in many real world optimization scenarios.

These limitations are in no way meant to denigrate the robustness and utility of calculus-based methods. These powerful tools have solved (e.g., Zienkiewicz & Campbell [1973]) and continue to solve complex and important problems (e.g., Widmann [1994]). Most commercial FE software packages which include an optimization module utilize calculus-based methods. The point is to recognize when this tool is appropriate, and when it is not.[†]

[†] It is easy to view all problems as "*calculus nails*" when the only tool at one's disposal is a "*calculus hammer*".

Artificial intelligence and *machine learning* methods provide techniques for software to partially learn and reason. As discussed in Chapter 1, some of the promise of these techniques have born fruit in mechanical engineering, while others from this category such as fuzzy logic (Zadeh [1985]), hypothetical reasoning (Harris [1989]), model-based reasoning (Fulton & Pepe [1990]) need further exploration to determine their utility. With regard to shape optimization, there is nothing inherent that prevents these techniques from succeeding in the shape optimization domain. The classifier system is a machine learning method which utilizes a genetic algorithm, a nature analogy method, in its learning process.

Both *enumerative* and *random* techniques are not restricted to specific problem domains like the calculus methods are. Their major weakness is the same — inefficiency. In addition, an enumeration of all possibilities directly violates Pike's definition of optimization theory which states, "without having to enumerate all the possibilities". Even though guided random search can be orders of magnitude more efficient than enumeration, many search spaces are simply too large to be attacked by these techniques in less than geological time frames.

Nature analogy methods are those which are derived by analogy from some natural phenomenon. These natural phenomena range from the annealing of metals (Kirkpatrick, et al. [1983]) to neural networks (Rumelhart et al. [1986]), to the evolution of species. Some of these methods have been touched upon in Chapter 1; recall that the genetic algorithm utilized by the classifier system is based on a nature analogy. While this study provides the initial classifier system application to mechanical and structural optimization, (raw) genetic algorithms have been applied by Goldberg [1986], Jensen [1992], and others.

There are many *other* optimization methods which do not fall completely into the above categories (e.g., Hsu [1992]), or are a *hybrid* of techniques, such as using artificial intelligence in conjunction with other approaches (e.g., Ashley [1992]).

2.5 Performance Metrics

To quantify the performance of a shape optimization methods the following factors must be considered:

- scope,
- efficiency,
- effectiveness.

Scope entails the spectrum of shape optimization classes which can be handled (by a particular method), such as sizing and shape optimization problems only. Efficiency describes the rate at which improvement occurs relative to the termination point. Effectiveness measures the level of optimization of the best design found. Efficiency and effectiveness are compared graphically in Figure 2.5, Curve A is more efficient than Curve B although they are equally effective after *X* iterations. Of course, for any number of iterations less than *X* the effectiveness of Curve A would be superior to the effectiveness of Curve B.

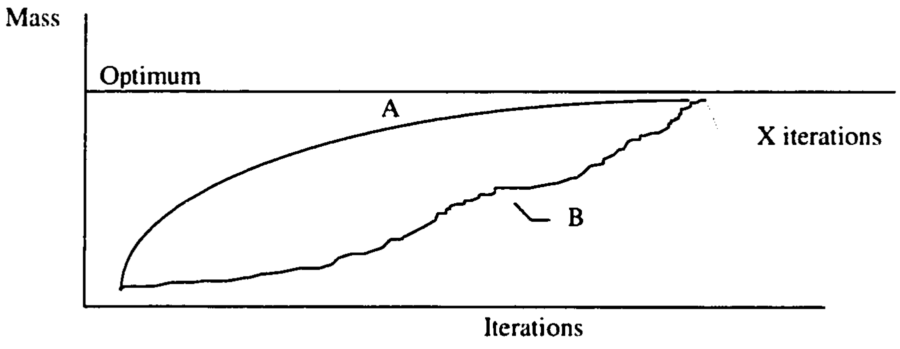


Figure 2.5 ***Optimization Efficiency & Effectiveness***

Therefore the optimization system developed in this research will be compared with other techniques by comparing the scope of this system with other systems. Then the primary performance metric will be the efficiency of this research's optimization system in determining the optimum of benchmark problems, which will be compared directly against the best performances found in the literature. The effectiveness metric will also be compared to the results found in the literature.

Recall that most other techniques (see Section 2.3) benefit from:

- sensitivity information,
- hard coding of the implementor's algorithm.

While the system developed in this dissertation:

- has no sensitivities supplied to it,
- must learn to perform optimization.

Table 2.2 compares the scope and efficiency of the optimization method categories described in Section 2.4.4. The asterisks reveal where this research fulfills the potential of machine learning as a tool for shape optimization.

Table 2.2 ***Scope and Efficiency Ranges of Optimization Method Categories***

Optimization Class	Characteristic	Calculus-based	Artificial Intelligence & Machine Learning	Enumerative & Random	Nature Analogy	Hybrid & Other
Size	Scope	Y	Y	Y	Y	Y
Size	Efficient	Y	Y	N	N	N
Shape (2D)	Scope	Y	Y	Y	Y	Y
Shape (2D)	Efficient	Y	N*	N	N	N
Shape (3D)	Scope	Y	N*	Y	Y	Y
Shape (3D)	Efficient	N	N*	N	N	N
Topological	Scope	N	N	Y	Y	Y
Topological	Efficient	N	N	N	N	N

*** This research converts these to *Y***

Performance of State of the Art

When evaluating the computational costs of most modern optimization systems, the analysis module dominates the expenditure of resources. It is thus considered the base cost and used as a measure of efficiency. This can be extended to comparisons of sensitivity and non-sensitivity based optimization; for example if the determination of the sensitivity information adds an extra 40% to the resources expended in the analysis module then one simply multiplies the iterations required by the sensitivity based optimization system by 1.4 to get the *equivalent analyses* in the non-sensitivity based system.

Few optimization systems cover the intended scope of this study, and those that do suffer debilitating limitations as the following attests. One example is genetic algorithm based systems; Goldberg and Samtani [1986] developed a system to optimize ten-member plane trusses while Jensen [1992] developed a system that can improve a broad range of geometries, both functioned without the benefit of sensitivity information. The drawback of both developments is inefficiency. The examples solved required *thousands* of analyses.

Modern optimization systems with a more restricted scope, which usually depend on sensitivity information, perform orders of magnitude better. However, most of these systems require the initial design to be no more than two times (100%) heavier than the optimum, or rephrased: the optimum will require a 50% reduction in mass or less. In the cases presented by Kothawala et al. [1988] concerning the optimization of an engine piston and an aircraft wing, the optima were 30% and 33% lighter than the original design, respectively. Vanderplaats and Blakely [1989] also assume that, "the initial design is a reasonable one, in which the objective function, mass, is to be reduced by about 20%".

With the above restrictions, many systems have been able to perform optimizations on simple shapes with a few design variables in less than 20 equivalent analyses. Braibant [1986] applied sensitivity based optimization to simple 2-dimensional shapes, completing the optimization in about ten equivalent analyses, with reductions in mass greater than 50%. Vanderplaats and Blakely [1989] processed examples with optima usually achieved at a cost of under 15 equivalent analyses. Belegundu [1993] investigated the optimization of a torque arm. His system found an optimum shape in 17 equivalent analyses with a 40% reduction in weight. Hsu [1992] performed shape optimization on components, including a torque arm, reducing the mass 38% in less than 10 equivalent analyses without any auxiliary information.

2.6 Summary

This chapter provided a formal review of shape optimization. This included definitions of shape optimization and its mathematical representation. Different classes of shape optimization were reviewed and clarification relevant to this work was presented. The chapter discussed criteria necessary to perform shape optimization, as well as the evaluation of the optimization's performance. The chapter closed with a review of the achievements, performance, and limitations of state-of-the-art shape optimization techniques.

Chapter

3

Background — Classifier Systems & Genetic Algorithms

Soon after the advent of the electronic computer, scientists envisioned its potential to exhibit learning behavior. Since the early machine learning work by Samuel [1959], many machine learning systems have been developed. One of these, the learning classifier system, introduced by Holland and Reitman [1978] is a machine learning system which possesses the salient properties needed to learn in the shape optimization domain.

Though a complete review of machine learning field is beyond the scope of this work, two important concepts need to be formalized. In machine learning, the *machine* is a software system running on a computer, while *learning* is analogous to (some subset of) the human learning behavior. Furthermore, behavior is a product of the interaction between an *agent* and its environment, where the *agent* is any entity, (biological, mechanical or otherwise), that can perform actions, intelligent or not. The universe of possible behavioral patterns is therefore determined by the structure and the dynamics of both the agent and the environment, and in particular by the interface between the two (the sensors and the effectors).

Machine learning distinguishes between two important classifications for behavior; one is *stimulus-response* and the other is *dynamic*. Stimulus-response (S-R) behavior is reflex behavior; that is, a stimulus to the agent (via its sensors) causes a direct response by the agent (via its effectors). Dynamic behavior, is more complex with there being an internal state which may perform multiple layers of computation before a response is generated.

Learning has many denotations and connotations; the following two descriptions summarize learning for the purpose of this study:

- An agent (machine learning system) learns (with respect to an environment) if its production of a response alters the state of the environment in such a way that future responses of the same type tend to be better.
- Systems that are capable of making changes to themselves over time with the goal of improving their performance on the tasks confronting them in a particular environment [demonstrate learning]. (Kondratoff & Michalski [1990]).

The rest of this chapter introduces a simple Michigan Approach *Classifier System* (CS), as first described by Holland and Reitman [1978]. The major components of the classifier system are detailed; one of these components includes the *genetic algorithm* (GA). Since the genetic algorithm plays such a vital role in the classifier system's learning ability, the significant aspects of the genetic algorithm are presented. With the classifier system's components introduced, the holistic viewpoint is next presented demonstrating the interplay of the newly described components. With the CS and GA developed, relevant past applications are reviewed. The chapter closes with methods used to evaluate the learning performance of classifier systems. For a more in-depth overview of genetic algorithms and classifier systems the interested reader is directed to Goldberg [1989], and the seminal work by Holland [1975]. Much of the subsequent review is based on these works.

3.1 Introducing the Classifier System

A classifier system (CS) is a machine learning system that learns syntactically simple string rules, called classifiers, as introduced by Holland and Reitman [1978]. These classifiers guide the system's performance in an arbitrary environment. A classifier system derives its name from its ability to learn to classify messages from the environment into general sets and is similar to a control system in many respects. As a control system[†] uses feedback to "control" or "adapt" its output for an environment, a

[†] Dorf [1983] provides an introduction to control systems.

classifier system uses feedback to "teach" or "adapt" its classifiers for an environment. Since most environments are not guaranteed to be static and learning can never be known to be complete, the learning process may never cease.

The classifier system has developed out of the merging of expert systems (as described in, Charniak and McDermott [1985]; Waterman [1986]), and genetic algorithms as originated by Holland [1975]. This synthesis has overcome the main drawback to expert systems; namely, the long task of discovering and inputting rules. Using a genetic algorithm, the CS learns the rules needed to perform in an environment, (in this current study the environment is structural shape optimization).

The development of expert systems has helped advance many fields by having computers reason more like humans. Expert systems allow the computer to use rules. Some rules are concrete facts while others are rules-of-thumb (heuristics) that work in most situations, but the specific rules are still unknown for all situations.

As mentioned above, an obstacle to the wider use of expert systems is the fact that all rules for the expert system must be provided by humans, and therefore have to be collected from literature and interviews. Furthermore, since the rule set is static, the system can never discover if a rule-of-thumb is non-applicable, and should consequently be eliminated or modified. Another conflict occurs when more than one rule may be applicable to a situation; all such conflicts must be foreseen or the system may halt, not knowing how to proceed.

The learning capability of the CS greatly enhances the realization of the expert system's promise. With the classifier system, one may input rules (as with an expert system) or start from random rules, or, as is likely to be done in most real world scenarios, input as many rules as possible and let the classifier system learn new ones and try to improve the entered rules.

A classifier system is much more than a simple expert system that can learn from experience (which in itself is an immense boon). As the rest of this chapter will attest, a classifier system is a general machine learning system applicable to diverse environments, able to learn with incomplete information and classify the environment into hierarchies.

The science of classifier systems is young, however there are two approaches which have developed, the *Michigan Approach* and the *Pitt Approach* (DeJong [1988]). This study employs the Michigan approach because it is the classical approach, having proven itself and undergone more development. This current study also does not posses any of the characteristics where the Pitt approach may prove superior. The following describes the foundation of a Michigan Approach classifier system[†]. A classifier system has three major components:

- Rule and message sub-system,
- Apportionment of credit sub-system,
- Classifier discovery mechanisms (primarily the genetic algorithm).

Figure 3.1 shows the interactions between the classifier system and the environment. The classifier system receives information about the environment, performs internal processing and then effects the environment. It then uses feedback about the effect on the environment to learn from the experience. This arrangement has the classifier system in *learning mode*, because the classifier system is utilizing the feedback to learn from experience. Conversely, if no feedback is provided, the classifier system is in *application mode*. Application mode is utilized after sufficient learning is accomplished. The subsequent discussion up until Section 3.4 deals with the classifier system exclusively in learning mode.

[†] Further reference to *classifier system* will mean a *Michigan approach classifier system*.

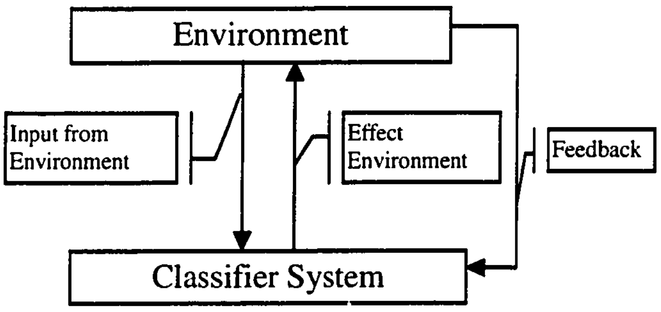


Figure 3.1 ***Interactions between Classifier System and Environment***

Figure 3.2 provides more detail on the classifier system's internal components. In Figure 3.2 the detectors, effectors, and classifier population blocks are part of the rule and message sub-system; the auction and reward/punishment blocks are part of the apportionment of credit sub-system; and the classifier discovery block is part of the classifier discovery mechanisms. The following three sub-sections describe the components in more detail as well as the information flow between the components.

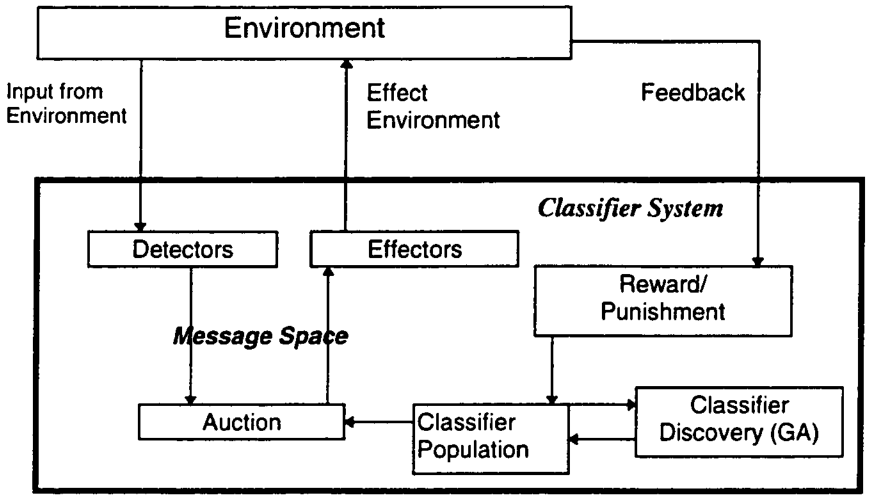


Figure 3.2 ***Classifier System Modules & Interaction with Environment***

3.1.1 Rule and Message Sub-system

Each classifier consists of a rule or conditional statement whose constituents are words drawn from the ternary alphabet (0,1,#). It has one or more words or conditions as the antecedent, an action statement as the consequent, and an associated strength. The rule portion has the template:

$$\text{IF } \langle \text{condition}_1 \rangle \& \langle \text{condition}_2 \rangle \& \ldots \& \langle \text{condition}_N \rangle \text{ THEN } \langle \text{action} \rangle \tag{3.1}$$

where,

<condition> is encoded as a string from the alphabet {0, 1, #}
<action> is encoded as a string from the alphabet {0, 1}.

The "#" symbol acts as a wild card or "don't care" in the condition, matching either a 0 or 1. This allows for more general rules. The more "don't care" symbols the more general the rule. The measure used to quantify this is called: *specificity*. The *specificity* of a classifier is the number of non # symbols in the antecedent. If a classifier's antecedent consists of all # characters then the specificity is zero, if there are no # characters in the antecedent then the specificity is equal to the antecedent's string length.

The strength portion of the classifier gives a measure of the rule's past performance in the environment in which it is learning. That is, the higher a classifier's strength the better it has performed and the more likely it will actually be used when the condition matches an *environmental message* (see Section 3.1.2.1) and to reproduce when the GA is applied (see Section 3.2). The strength values are relative; therefore, a range limit is set. If a strength falls out of this range, the strength value can be set to the closest range extreme to eliminate the range violation.

The messages, generated either from the environment (or from the action of other classifiers), match the condition part of the classifier rule. Therefore, an action is a type of message, with the consequence of an action being the modification of the environment (or the attempted matching with another classifiers in some classifier systems). In this study, classifiers only match messages from the environment and actions generated from classifiers only modify the environment. This restriction makes the classifier system a stimulus-response (S-R) system.

For a condition to match a message, every part of the condition string must match every part of the message string. Therefore the message,

```
011001
```

would match all of the following conditions

```
0110#1
011001
##100#
######
```

as well as others.

To illustrate, Table 3.1 shows samples of strings which are valid forms for classifiers, (with the ":" symbol denoting the break between the antecedent and action, i.e. <antecedent>:<action>), in the first column, and their associated strength in the second column.

Table 3.1 *Samples of Valid Classifiers*

Rule	Strength
011:101	23.2
011001##10#110:11	17.3
10101000110011##100#:11100001	32.9
####:1	7.1
100##00100##0011##:011001	29.0

The alphabet is restricted to allow for the power of genetic algorithms to be applied to the rule set as described below in Section 3.2, Genetic Algorithm. The alphabet in no way restricts the representational capacity of the classifiers.

The form of a rule differs from those normally found in expert systems. In expert systems, the rules often consist of sentences, for example,

IF *there is a stress concentration at a sharp corner,*
THEN *round the corner.*

Such syntax makes it very difficult for a computer system to be able to modify such a rule. Just as text is stored on computer disks as 0's and 1's, any rule can be translated into 0's, 1's, and #'s, so that it is in the form of a classifier. Once translated, rules can be manipulated more easily, and rule discovery and modification can occur.

The messages from the environment, which match the antecedent of the classifiers, are filtered and converted via input sensors. The sensors (called *detectors* in classifier system parlance) discriminantly select certain aspects of the environment to sense and then translate the input to a binary form which can be processed by the classifiers.

The actions of the classifiers modify the environment via the effectors (or output interface) see Figure 3.2. The effectors translate the binary action into a form which is appropriate to modify the environment within an envelope of allowable modifications.

3.1.2 Apportionment of Credit Sub-system

The apportionment of credit sub-system deals with the modifications in strength of classifiers as the classifier system learns (Booker, et al. [1989]). Strength modifications occur via three interrelated mechanisms:

- Auction,
- Reinforcement & punishment,
- Taxation.

As the classifier system receives messages from the environment, all the classifiers which match one (or more) of the messages compete, by submitting a *bid*, in an *auction* to determine a victorious classifier that will effect the environment. Section 3.1.2.1 further discusses the auction. The victorious classifier's modification will be beneficial or detrimental to the environment. With this feedback, the apportionment of credit sub-system appropriately uses *reinforcement & punishment* to increase or decrease the strength of the victorious classifier that caused the modifications. Section 3.1.2.2 details how feedback from the environment is used with reinforcement and punishment. Finally, *taxation* is levied on each classifier per iteration and on each classifier that submits a bid during an auction. Details of and the need for taxation are provided in Section 3.1.2.3.

Computer simulations show that the exact mechanism for the apportionment of credit sub-system is not critical to the learning ability of the CS (e.g., Riolo, [1988]). That is, the apportionment of credit sub-system may have many forms and the CS will still learn, albeit incrementally more efficiently with the apportionment of credit sub-system in some forms than others. This is an example of one of the many CS parameters which needs to be set. The values to which the parameters should be set cover a range, guided by biological analogy and empirical results. Many times the parameters are manipulated during the learning process to determine if such manipulations can enhance learning.

3.1.2.1 Auction: Bidding & Competition

An auction is performed among all the classifiers which have an antecedent that matches at least one of the environmental messages. The classifier system's detectors receive input from the environment and assemble the input into environmental messages (see Chapter 4, Section 4.3). Each classifier attempts to match each environmental message, with each classifier that matches bidding in the auction. Figure 3.3 displays a simplified view of how the auction functions.

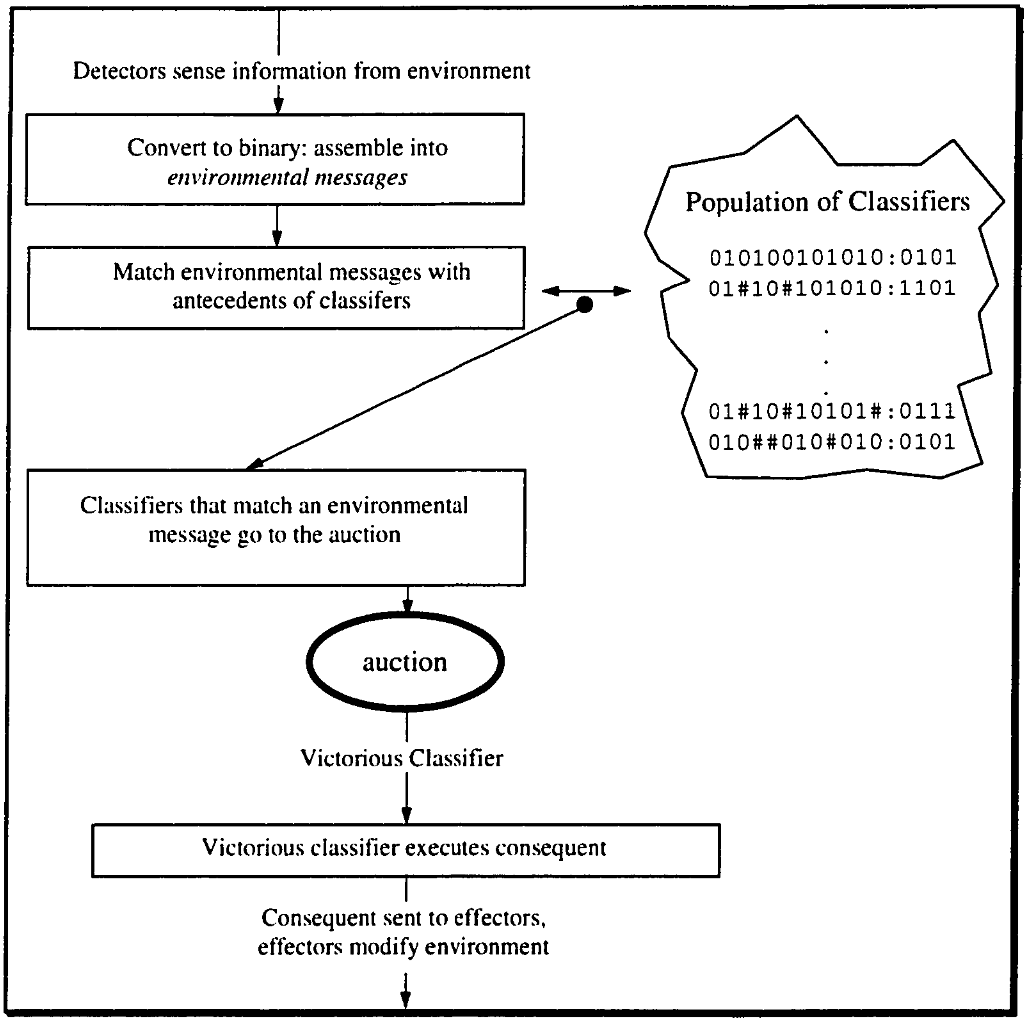
With the matching classifier pool determined, the auction commences. Each classifier participating in the auction submits a bid, the bid is a function of the classifier's strength and specificity. Only the bid of the victorious classifier is paid, so only the victorious classifier has its strength decreased by the amount of its winning bid. The bid of classifier ***i*** at iteration ***t***, $\boldsymbol{B_i(t)}$, is calculated as:

$$B_i(t) = k_0 * (k_1 + k_2 * BidRatio^{BRPow}) * S_i(t) \qquad (3.2)$$

where,

k_0	Classifier Bid Coefficient: positive constant less than unity that acts as an overall risk factor influencing what proportion of a classifier's strength will be bid and possibly lost on a single step.
k_1	Bid Coefficient 1: constant less than unity for non-specificity portion of bid.
k_2	Bid Coefficient 2: constant less than unity for specificity portion

	of bid.
$S_i(t)$	Strength of classifier ***i*** at step ***t***.
BidRatio	Measure of the classifier's normalized specificity. A ***BidRatio*** of 1 means there is just one possible message that matches each condition, while a ***BidRatio*** of zero means the classifier would be matched by any message and the antecedent would consist of all wildcard characters.
BRPow	Parameter controlling the importance of the ***BidRatio*** in determining a classifier's bid (default is 1).



Detectors sense information from environment
Convert to binary: assemble into *environmental messages*
Match environmental messages with antecedents of classifers
Population of Classifiers
010100101010:0101
01#10#101010:1101
.
.
.
01#10#10101#:0111
010##010#010:0101
Classifiers that match an environmental message go to the auction
auction
Victorious Classifier
Victorious classifier executes consequent
Consequent sent to effectors, effectors modify environment

Figure 3.3 ***Auction in Classifier System***

To promote exploration of the classifier space, the bids submitted by each competing classifier in Equation 3.2 are not used directly to determine the auction winner, random noise is added to the auction. Therefore the *effective bid, $eB_i(t)$*, is calculated as the sum of the deterministic bid, $B_i(t)$, and a noise term, $N(\sigma_{bid})$ as shown in Equation 3.3:

$$eB_i(t) = B_i(t) + N(\sigma_{bid}). \qquad (3.3)$$

3.1.2.2 Reinforcement & Punishment

Since the pioneering work on machine learning by Samuel [1959], the *credit assignment* problem (Minsky [1963]) has been known to be a key problem for any learning system in which many interacting parts determine the system's global performance. Credit assignment deals with the problem of deciding, when many parts of a system are active over a period of time (or even at every time step), which of those parts active at some step t contribute to achieving some desired outcome at step $t+n$, for $n > 0$. This is a particularly challenging problem.

To solve the credit assignment problem in classifier systems, the bucket brigade algorithm, as defined by Holland [1986], was developed, and has experienced limited success to date. An alternative and simpler solution (when possible) is the implementation of the CS as a *stimulus-response* (S-R) system — this solution has proven successful as Table 3.5 in Section 3.5 exhibits. A S-R classifier system activates only one classifier each iteration and the activated classifier effects the environment. Therefore the environmental modification can easily be attributed to a single source.

A *trainer* is necessary to determine whether the environmental modification was beneficial or detrimental. Some machine learning systems require a *tutor trainer* which knows the correct or best answer, enabling the system's actual response to be compared with the correct response. Fortunately, a classifier system requires only the more flexible *reinforcement trainer.* Reinforcement learning requires only positive or negative feedback from the reinforcement trainer as a consequence of a response.

When the victorious classifier creates a beneficial effect to the environment, the trainer sends positive feedback causing an increase in the victorious classifier's strength. Conversely, a detrimental effect leads to punishment. Since the victorious classifier's strength decreases when it wins the auction and pays its bid, (as shown in Equation 3.2), punishment occurs implicitly anytime a reward is not provided. In addition, an adjunct strength reduction may occur. If the trainer has the ability to rank environmental effects, then the rewards and punishments can be scaled appropriately.

The strength $S_i(t+1)$ of a classifier ***i*** at the end of iteration ***t*** is:

$$S_i(t+1) = S_i(t) + R_i(t) - B_i(t), \tag{3.4}$$

where,

$S_i(t)$	Strength of classifier ***i*** at beginning of iteration ***t***.
$R_i(t)$	Reward from the environment during iteration ***t***.
$B_i(t)$	Classifier's bid during iteration ***t*** (as defined by Equation 3.2) only paid if victorious.

Again, classifier ***i*** only makes a bid payment if victorious in the auction and effects the environment. The reward factor, $R_i(t)$, is only non-zero if the classifier won the auction on the previous iteration. The reward (or punishment) for the action at iteration ***t*** will not be applied until iteration ***t + 1***. Note that $R_i(t)$ is less that zero for punishment, and greater than zero for reward.

3.1.2.3 Taxes

Taxation occurs to prevent the classifier population from being cluttered with artificially high strength classifiers of little or no utility. There are two types of taxes:

- life tax,
- bid tax.

The *life tax*, Tax_{life}, (also called *head tax*) is a fixed rate tax applied to every classifier on every iteration. The purpose is to reduce the strength of classifiers that rarely or never are matched and therefore provide little or no utility. Non-producing classifier's strengths are slowly decreased, making them candidates for replacement when the classifier discovery mechanisms (primarily the genetic algorithm) creates new classifiers. The *bid tax*, Tax_{bid}, is a fixed-rate tax that is applied to each classifier that bids during an iteration. One reason for a bid tax is to penalize overly general classifiers, i.e., classifiers that bid on every step but perhaps seldom win because they have a low specificity which leads to low bids and so a low chance of winning the auction to post effector messages (Riolo [1988]).

The free-fall half life determines the order of magnitude of the life tax, and is determined by observing that the strengths of inactive (non-matching) classifiers are only reduced by the life tax. Therefore after n iterations of inactivity the strength of an inactive classifier has the strength:

$$S(t+n) = S(t) * (1 - Tax_{life})^{n}. \tag{3.5}$$

The half life, n, of an inactive classifier may then be determined by Equation 3.6,

$$n = \frac{\log(1/2)}{\log(1 - Tax_{life})}. \tag{3.6}$$

If the half-life is specified, the tax rate may be found by Equation 3.7,

$$Tax_{life} = 1 - (1/2)^{(1/n)}. \tag{3.7}$$

As will be discussed in Section 3.1.3, new classifiers are inserted into the population at the average strength of their parents, thus the tax rate must be set to ensure that inactive rules are degraded sufficiently before the application of the genetic algorithm. If this is not done, relatively inactive rules can retain an unrealistically high level of strength and ultimately reach reproduction disproportionately, thereby cluttering future populations with large numbers of overrated inactive rules. However, the tax burden can not be so great that rules which have only remained inactive by chance become so weak that they are essentially eliminated from any auction. Thus the life tax should be set to yield a half life on the order of the application period of the genetic algorithm.

With all the apportionment of credit mechanisms defined, the complete strength equation is shown in Equation 3.8:

$$S_i(t+1) = (1 - Tax_{life})S_i(t) + R_i(t) - B_i(t) - Tax_{bid} * B_i(t). \qquad (3.8)$$

Recall that,

$R_i(t)$	will only be non-zero if classifier **i** won the auction on iteration **t-1**.
$B_i(t)$	is only paid if classifier **i** wins the auction.
$Tax_{bid} * B_i(t)$	is only paid if classifier **i** bids in the auction (irrespective of whether classifier **i** wins the auction or not).

3.1.3 Classifier Discovery Mechanisms

Two classifier discovery mechanisms are implemented in the system utilized in this study:

- Genetic algorithm,
- Triggered cover detector operator.

The foremost operator, the *genetic algorithm* (GA), provides the bulk of the discovery and learning capability found in a classifier system. Discussion of the GA is deferred to Section 3.2 and its sub-sections to provide the coverage due.

The *triggered cover detector operator* (TCDO) is a triggered rule generation mechanism, i.e., a rule generation operator that is only activated (i.e., triggered) when certain conditions occur. In fact, it is triggered whenever the classifier system does not have a classifier which matches (i.e. covers) any environmental message. It responds by producing one new classifier that would be satisfied by an environmental message at step *t* with a condition that matches the unmatched environmental message. The action part is just randomly generated on the {0,1} alphabet.

The TCDO is a special case of a mutation operator (described in Section 3.2) which implements a random walk through the space of possible classifiers. A random walk performs pathetically in the astronomical search space of possible classifiers; however, in conjunction with a GA, a TCDO improves learning relative to the GA being applied alone (Robertson & Riolo [1988]).

Two considerations must be accounted for when determining the initial strength given to a new classifier created by either the TCDO or the GA:

1. The strength should not be too low, otherwise the new classifier will never win an auction and therefore never get a chance to prove itself better (or worse) than existing classifiers.
2. The strength should not be too high, otherwise the new classifiers will be tried too often, overruling existing rules that perform well, and may lead to unstable performance.

Computer simulation studies by Riolo [1988] and others conclude that rules introduced by the TCDO should have the average of the strengths of the classifiers in the population; while the offspring of the GA should have the average strength of the parents.

3.2 Genetic Algorithm

Thomas Malthus' *Essay on the Principles of Populations* in 1798 is one of the earliest known works dealing with the modeling of biological adaptation. However, not until 1865, when Gregor Johann Mendel, an Augustinian monk in a Brno[†] monastery studied the inheritance characteristics of peas, was the catalyst of modern genetics set forth. Unfortunately, this work languished in obscurity until 1900, when it was rediscovered by H. de Vries, C. Correns and E. Teschermak (Doolitle [1986]). Thus, virtually all inheritance and genetic research has occurred in the 20th century.

Most complex organisms evolve by means of two primary processes: natural selection and sexual reproduction. The first determines which members of a population survive to reproduce, and the second ensures mixing and recombination among the genes of their offspring.

A genetic algorithm (GA) is a stochastic search algorithm based on the mechanics of natural selection (Darwin [1897]) and population genetics (Mettler, et al. [1988]). Genetic algorithms are patterned after natural genetic operators that enable biological populations to effectively and robustly adapt to their environment and to changes in their

[†] Located in the present day Czech Republic.

environment. Some of the correspondences between biological genetics and genetic algorithms are shown in Table 3.2.

Table 3.2 ***Biological and Artificial Vernacular Correspondence***

Biological Term	Corresponding Genetic Algorithm Term
chromosome	classifier or string
gene	character or bit
allele	bit value
locus	position

Genetic algorithms, as Goldberg [1989] states and demonstrates, are theoretically and empirically proven to provide robust search in complex spaces. The GA performs its search balancing the need to retain population diversity 'exploration', so that potentially important information is not lost, with the need to focus on fit portions of the population, 'exploitation' (Whitley [1989]). Reproduction in GA theory, as in biology, is defined as the process of producing offspring (Melloni et al. [1979]). However, mating may occur between any two classifiers,[†] as there is no male-female distinction.

Improvements come from trying new, risky things. Because many of the risks fail, exploration involves a period of performance degradation. Deciding to what degree the present should be mortgaged for the future is a classic problem for all systems that adapt and learn. The genetic algorithm's approach to this obstacle is crossover (as discussed below).

In computational terms, genetic algorithms are distinguished from other search methods by the following features:

- A *population of structures*[‡] that can be interpreted as candidate solutions to the given problem.
- The *competitive selection* of structures for reproduction, based on each structure's fitness as a solution to the given problem.

[†] Genetic algorithms can work on other structures besides classifiers, but the review will use classifiers for consistency with the study.

[‡] The structures in this study are the rule portion of the classifiers.

- Idealized *genetic operators* that recombine the selected structures to create new structures for further testing.

The following discusses the operators of a basic genetic algorithm. A mathematical justification for the GA's power is provided through the schema theorem (Holland [1975]). The schema theorem has developed from earlier work by Holland [1968] and is under continuing development. For more information on the schema theorem the interested reader is directed to, in addition to the references already cited, Bethke [1981], Fitzpatrick and Grefenstette [1988]. Koza [1992], and Whitley [1994] provide both theoretical foundations, as well as lucid descriptions of schema and schema theory.

Genetic Algorithm Operators

The basic genetic algorithm operators involved in reproduction are:

- Selection,
- Crossover,
- Mutation.

The placement of these operators in the overall genetic algorithm is shown in Figure 3.4.

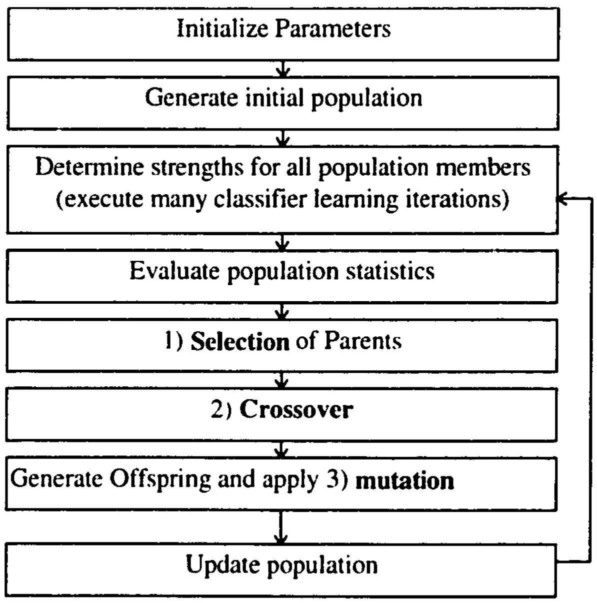

Initialize Parameters
Generate initial population
Determine strengths for all population members (execute many classifier learning iterations)
Evaluate population statistics
1) **Selection** of Parents
2) **Crossover**
Generate Offspring and apply 3) **mutation**
Update population

Figure 3.4 ***Simple Genetic Algorithm Flowchart***

In Figure 3.4 there is a box that reads, 'Determine strength for all population members', in the case of a classifier system this determination can not occur during a single iteration. Classifier systems determine the ranking among the population members via multiple interactions with the environment whereby strength changes occur via the apportionment of credit sub-system of the classifier system. Only after multiple interactions with the environment will the classifier strengths represent a measure of how well the classifier performs in the environment. The number of iterations that occur between each application of the genetic algorithm is called an *epoch*. Therefore in Figure 3.4 each loop represents one epoch.

1) *Selection* deals with the selection of classifiers of the population which will reproduce. The selection algorithm allocates reproductive trials to classifiers as a function of their strength. Some selection strategies are deterministic such as *elitism* where just a certain percentage of the strongest classifiers are selected. However, most research has shown that stochastic selection biased by strength is more productive.

For stochastic selection, the selection probability is proportional to the individual's strength. During selection, high strength classifiers have a greater probability of producing offspring for the next generation than lower strength classifiers. There are many different ways to implement the stochastic selection operator, with most methods which bias selection towards high strength proving successful as Goldberg and Samtani [1986] as well as others have shown.

Fitness proportionate reproduction is a simple rule whereby the probability of reproduction during a given generation is proportional to the fitness of the individual. In this investigation, the probability that a classifier, ***i***, will be selected for mating is given simply by the classifier's strength divided by the total strength of all the classifiers:

$$P_i = \frac{S_i}{\sum_{k=1}^{n} S_k} , \qquad (3.9)$$

where,

P_i	Probability of selection for classifier ***i***.
S_i	Strength of the classifier ***i***.
n	Total number of classifiers.

This gives every member of the population a finite probability of becoming a parent, with stronger classifiers having a better chance.

2) Crossover takes a portion of each parent (as described below) and combines the two portions to create offspring. After selection, the strings are copied into a mating pool and crossover occurs on the copies.

First, panmictic[†] pairs of parents are chosen from the copies in the mating pool. That is, the mate for each individual which was chosen during selection is randomly bred with one of the other classifiers which was chosen during selection. Techniques have been suggested which bias the mate to have certain characteristics but none of these techniques were employed in the current work.

Second, each pair of copies undergoes crossing over as follows: an integer position k along the string is selected uniformly at random on the interval $(1, L\text{-}1)$, where L is the length of the string. Two new strings (classifiers) are created by swapping all characters between positions L and k inclusively.

To visualize how this works, consider two strings ***A*** and ***B*** of length *7* mated at random from the mating pool:

```
A  =  a1 a2 a3 a4 a5 a6 a7
B  =  b1 b2 b3 b4 b5 b6 b7.
```

Consider the random selection of ***k*** is four. The resulting crossover yields two new classifiers ***A'*** and ***B'*** following the partial exchange.

```
A' = b1 b2 b3 b4 a5 a6 a7
B' = a1 a2 a3 a4 b5 b6 b7.
```

[†] Random, see *panmixia* in *Appendix D, Glossary* for precise definition.

The simple crossover described above is a special case of the *n*-point crossover operator. In the *n*-point crossover operator, more than one crossover point is selected and several substrings from each parent are exchanged. This study employs solely the single-point crossover operator.

Although the mechanics of the selection and crossover operators are simple, the biased selection and the structured, though stochastic, information exchange of crossover give genetic algorithms much of their power.

3) *Mutation*, the random alteration of a string position, performs a secondary role in the reproduction process. Mutation is needed to guard against premature convergence, and to guarantee that any location in the search space may be reached. In the classifier's tertiary code, a mutation could change,

```
      0 to a 1 or #;
      1 to a 0 or #;
or    # to a 0 or 1.
```

By itself, mutation is a random walk through the classifier space. The frequency of mutation, by biological analogy and empirical studies, is on the order of one mutation per ten thousand position transfers.

3.3 Replacement & Crowding

Replacement and Crowding handles the introduction of new classifiers into a population and the elimination of classifiers from a population. The classic implementations of classifier systems and genetic algorithms have constant size populations. Therefore for each new individual created, another individual must be eliminated.

An important dynamic of GAs and CSs is the population percentage replaced on each generation. *Generational replacement genetic algorithm* (GRGA) replaces the entire population with each generation; this is the traditional approach of straight genetic algorithms. *Steady state genetic algorithm* (SSGA) replaces only a small portion of the population on each generation. Classifier systems normally use the SSGA approach. This study will not deviate from the norm and uses a SSGA.

With a SSGA approach, the question of which classifiers to replace arises. The senescence of a classifier plays no factor in replacement; a classifier may be eliminated after only one generation or potentially be immortal. While it is logical to replace low strength classifiers, simple replacement of the worst can be improved upon. A crowding mechanism among a low strength sub-population is implemented. The technique is modeled on that by De Jong [1975].

The technique is employed for each new classifier generated for insertion into the population. A *crowding factor* of checks are made to determine which classifier to replace. Each check consists of randomly selecting a *crowding sub-population* from the entire population, then selecting the lowest strength classifier in the sub-population. The selected classifier is added to a pool of replacement candidates. When the crowding factor checks are complete, the pool members are compared to the child and the child replaces the most similar candidate on the basis of similarity count. Similarity count is a simple count of the positions where both the child and candidate are identical. This method is beneficial in that it helps maintain diversity within the population.

After completing the above, each of the offspring is checked to see if it is a twin to any of the other members of the population. This may occur even with the above procedure because the twins may be both offspring. If a twin is found, a mutation is introduced into the lower strength twin, the process is repeated, if necessary, until the individual is unique. A twin provides no benefits and is detrimental because it decreases population diversity.

3.4 Classifier Systems: The Holistic Viewpoint

Now that the components of the classifier system have been introduced, a holistic view may be more fully appreciated. When the classifier system is not learning, it receives information from the environment via the detectors, determines the appropriate classifier to fire, then performs the action prescribed by the fired classifier via the effectors. This arrangement is called *application mode*, and is shown in Figure 3.5.

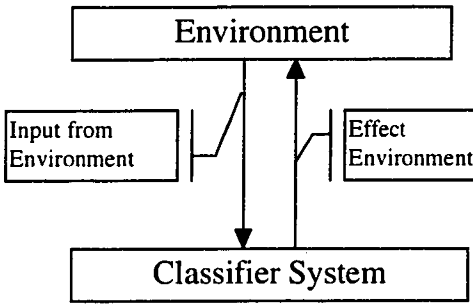


Figure 3.5 ***Classifier System and Environment Interactions: Application Mode***

When learning is occurring, some form of an initial population must be created. As stated, one may commence with many possible initial populations. To fully test the learning ability of the CS a *tabula rasa* is used. Even if a randomly generated initial population is selected, many population parameters still must be set. These include the number of conditions in the antecedent, the word length for each condition and the action and the probability of selecting a # in the randomly generated population. These issues will be further discussed and actual selections made for this study in the next chapter.

The basic interactions between an environment and a classifier system in learning mode as first shown in Figure 3.1, is repeated in Figure 3.6.

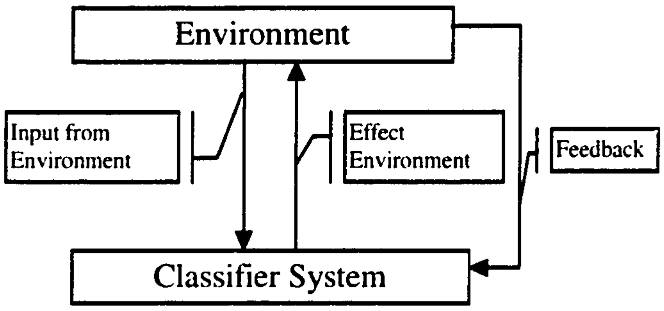


Figure 3.6 ***Classifier System and Environment Interactions: Learning Mode***

Since the initial classifiers are randomly generated, they are most likely of low quality and should be considered nothing more than guesses. The classifier system performs many iterations of interaction with the environment receiving feedback allowing the guesses to be ranked. These iterations consist of the classifier system's *major cycle*; a flowchart of the major cycle is shown in Figure 3.7. The major cycle shown in Figure 3.7 extends the information provided in Figure 3.3, in Section 3.1.2.1. The earlier figure did not include the feedback used by the apportionment of credit sub-system to reward or punish the responsible classifier.

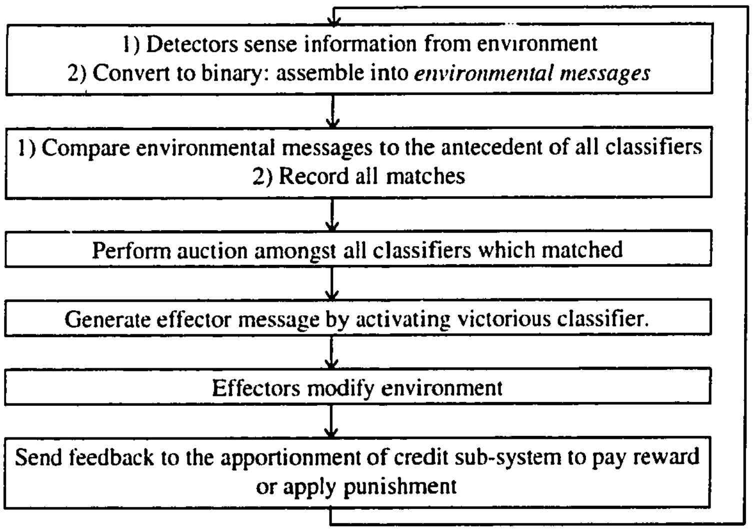

1) Detectors sense information from environment
2) Convert to binary: assemble into *environmental messages*
1) Compare environmental messages to the antecedent of all classifiers
2) Record all matches
Perform auction amongst all classifiers which matched
Generate effector message by activating victorious classifier.
Effectors modify environment
Send feedback to the apportionment of credit sub-system to pay reward or apply punishment

Figure 3.7 ***Classifier System Major Cycle***

After an epoch (of iterations) the genetic algorithm is applied mating the best guesses. As the iterations and epochs increase the quality of the guesses increases. Since general guesses (i.e., classifier with many # symbols) participate in auctions more than specific guesses, the initial learning will find some general guesses which are correct more times than not. With the concept of major cycle and epoch defined, the genetic algorithm flowchart shown in Figure 3.4 in Section 3.2 can be specialized for the classifier system, as shown in Figure 3.8.

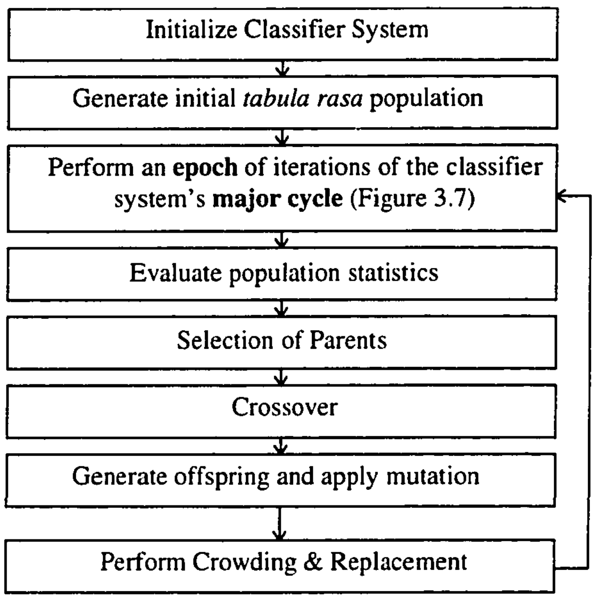

Initialize Classifier System
Generate initial *tabula rasa* population
Perform an **epoch** of iterations of the classifier system's **major cycle** (Figure 3.7)
Evaluate population statistics
Selection of Parents
Crossover
Generate offspring and apply mutation
Perform Crowding & Replacement

Figure 3.8 ***Genetic Algorithm in Classifier System***

With some learning behind it, the population of classifiers may be thought of as a population of hypotheses (Holland [1992]). As always, a hypothesis (classifier) enters the auction when it is pertinent to the situation. A hypothesis' competitiveness is determined by its past performance and its specificity. For the victorious hypothesis, its destiny is tied to the result of its actions. As epochs pass, successful hypotheses will exchange information via the genetic algorithm. These offspring will replace disproved hypotheses with more plausible but untested hypotheses.

Figure 3.9 shows more details of the classifier system's structure, adding detail to Figure 3.1 from Section 3.1.

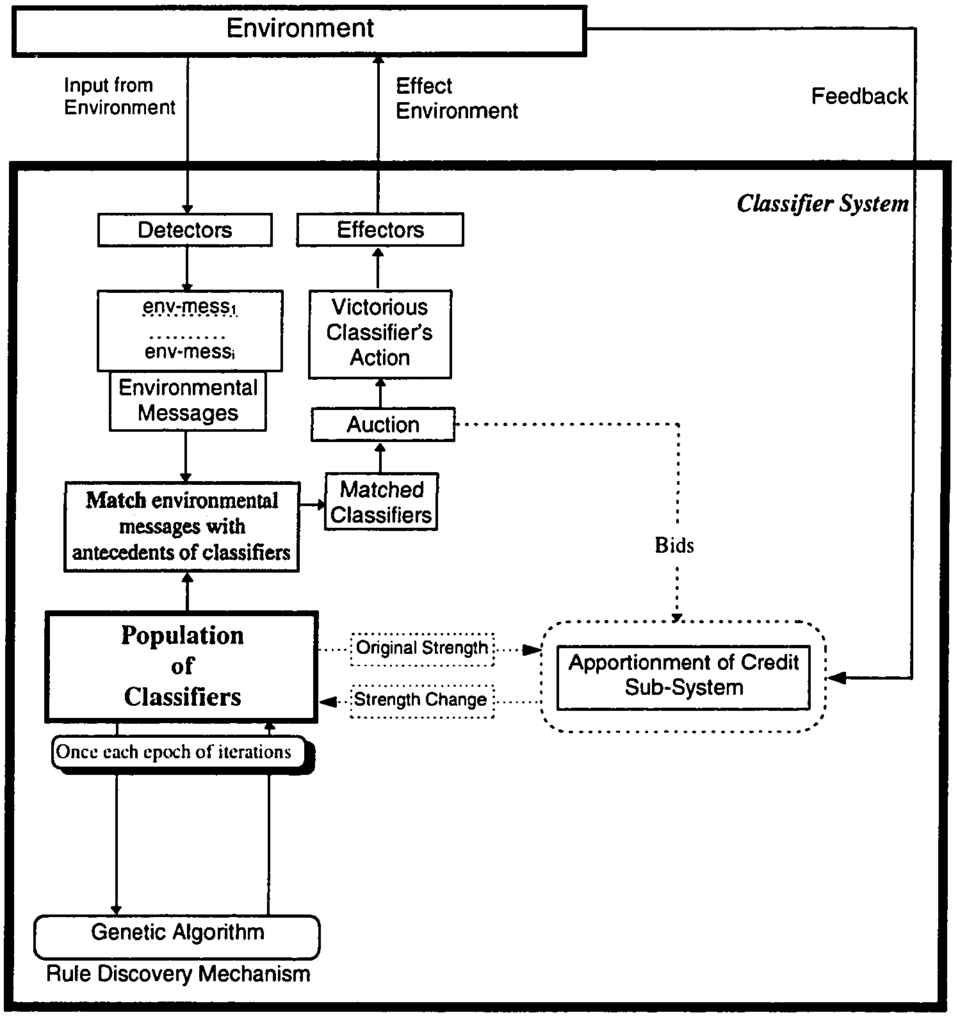


Figure 3.9 ***The Classifier System & Interaction with Environment: Learning Mode***

With more epochs comes the evolution of more specific hypotheses which control behavior in their narrow domains, overriding the more general default rules. This development of general (or default) hypotheses and specific (or exception) hypotheses allows the classifier system to learn gracefully, permitting the handling of novel situations by general hypotheses while providing for exception hypotheses when necessary. This hierarchy of classifiers is known as *default hierarchies* and will be further explored in Section 3.4.1.

As epochs continue and most of the feedback becomes positive, the classifiers may be thought of as more and more validated hypotheses. Furthermore, when the classifier system can pass criteria to be considered learned, the classifiers may be considered heuristics and rules.

Figure 3.10 shows the detailed interactions of the major components of the classifier system and a detailed view of the rule and message sub-system.

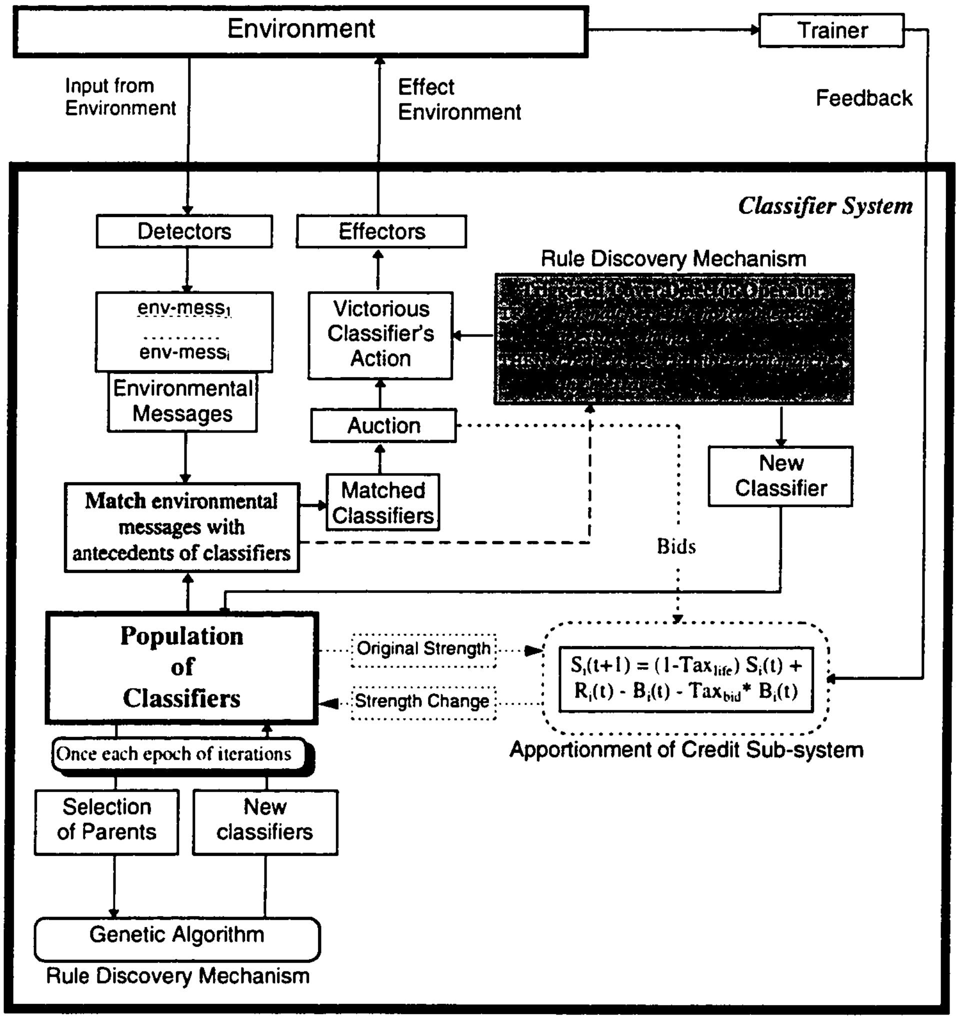


Figure 3.10 ***Detailed Classifier System & Interaction with Environment: Learning Mode***

3.4.1 Default Hierarchies

A default hierarchy is a multi-level structure in which classifiers become more general as the top level is ascended. Each general rule responds to a broad set of environmental messages, so that just a few rules can cover all possible states of the environment. Since a general rule may respond in the same way to many inputs that do not really belong in the same category, it will often err. To correct the mistakes made by the general classifiers, lower level, exception rules evolve in the default hierarchy. The lower level classifiers are more specific than the higher level rules; each exception rule responds to a subset of the situations covered by the more general rule, but it also makes fewer mistakes than the default rules made on those states.

Default hierarchies have several features that make them well suited for learning systems that must build models of very complex domains, they:

- Can be made as error-free as necessary, by adding classifiers to cover exceptions to the top level rules, to cover exceptions to the exceptions, and so on, until the required degree of accuracy is achieved.
- Make it possible for the system to learn gracefully, since adding rules to cover exceptions will not cause the system's performance to change drastically, even when the new rules are incorrect.
- Allow a minuscule population, (as compared to the search space of all possible classifiers), to evolve a population which collectively covers the overall problem space.

Because the antecedent of classifiers can be more or less general (by having more or fewer wildcard # symbols), default hierarchies are defined implicitly by many sets of classifiers. For example, consider the following simple single-condition classifiers of length $l = 3$:

$1\#\#{:}Action_1$
$10\#{:}Action_2$
$101{:}Action_3$.

These classifiers define a simple three-level default hierarchy, in which the first classifier is the most general, covering four messages, the second is an exception to the first, covering two of those four messages, and the third is an exception to both, covering just one message.

3.4.2 Other Mechanisms

The above has described the workings of a simple classifier system and basic genetic algorithm. The discussion also added relevant background to modifications to the rudiments which are used by this study. A variety of other additions and variations to the classifier system have been suggested in the literature. Many of these were investigated but were either found to be ineffectual or found not to be appropriate for this study. Table 3.3 shows a sampling.

Table 3.3 *Classifier System Extensions*

Extension Name	*References*
Implicit Niching	Horn, et al. [1994]
Coverage-base Genetic Induction	Greene & Smith [1994]
Fuzzy Classifier Systems	Valenzuela-Rendón [1991] Parodi & Bonelli [1993]
Using Performance-Based Action Selection	Wilson [1994]
Island model genetic algorithm (IMGA)	Whitley [1993]

3.5 Applications of Classifier Systems & Genetic Algorithms

Despite classifier systems and genetic algorithms' youth, GAs, and CSs to a lesser extent, have seen rapid growth in their application. Genetic Algorithms have found near optimal solutions in a variety of environments (Goldberg [1989]). Table 3.4 presents some GA engineering applications.

Table 3.4 ***Engineering Applications of Genetic Algorithms***

Description	*Reference*
Optimal structures using genetic algorithm include work by Dhingra and Jensen	Dhingra [1990] Jensen [1992]
The flow vectoring of supersonic exhaust nozzles using a genetic algorithm to define optimally shaped contours was investigated by King	King [1991]
Callahan investigated the use of Genetic Algorithms for the strength-to-weight and stiffness-to-weight optimization of Laminates	Callahan [1991]
The application of GA to the designing optimum welds has been investigated by Deb	Deb [1990]
Baffes and Wang have investigated the use of GA in the path planning of a mobile transporter	Baffes and Wang [1988]
General Electric's Engineous helped design the engine for the Boeing 777	Ashley [1992]
VLSI cell placement	Kling [1991]
Design of Air-Injected Hydrocyclone	Karr and Goldberg [1990]
Composite material structures' design and optimization	Punch et al. [1994]
Composite laminate staking sequence optimization for buckling load maximization	Le Riche and Haftka [1993]

Table 3.5 presents some of the more successful classifier system applications. These examples are stimulus-response (S-R) systems, searching the space of possible stimulus-response rules. Except for allocating payoffs directly to the classifiers that produced results, the bucket brigade algorithm as defined by Holland [1986] did not play a role in these systems.

Table 3.5 ***Applications of Classifier System***

Description	*Reference*
Dorigo has developed a robot path planning system utilizing many classifier systems simultaneously.	Dorigo and Sirtori [1991]
Utilized a classifier system to control a simulated creature in a simple two-dimensional environment.	Booker [1982]
Demonstrated the application of a classifier system to the control of gas flow through a national pipeline system.	Goldberg [1983]
Applied classifier systems to learning dynamic planning problems, such as determining plans of movement through artificial environments in search of food.	Roberts [1993]
Used classifier systems to learn to categorize Boolean multiplexer functions.	Wilson [1986]

3.6 Performance Analysis

Classifier system performance divides into two modes,

- Learning mode performance,
- Application mode performance.

The learning mode performance measures how well the classifier system is learning to perform the correct behavior in an environment. The application mode performance measures the performance of the learned classifier system in handling problems from the same domain (but different problems) from which it was taught.

Application mode performance is addressed in Chapter 7, where the application mode performance is measured and compared to the performance of other techniques which solve problems in (a subset of) the environment which the learned classifier can perform.

Pure random search provides a lower bound on the learning mode performance of genetic algorithms and classifier systems; of course, substantial increases in performance over random search must ensue before suggesting that the classifier system is learning.

The evidence for learning is an increase in the classifier system's performance. Therefore, to know that the classifier system is learning the target behavior, various performance metrics are employed.

The simplest measure of learning performance is the ratio of the number of correct responses to the total number of responses produced:

$$P1 = \frac{\textit{Number of correct responses}}{\textit{Total number of responses}}. \tag{3.11}$$

P1 will always be less than or equal to *1*, and is defined as the cumulative measure, and gives an idea of the whole learning process. A local measure portrays the present performance level, and is defined as follows:

$$P2 = \frac{\textit{Number of correct responses during epoch}}{\textit{Epoch length}}. \qquad (3.12)$$

A different type of metric relates to multi-step goals. For example, if the classifier system's actions are each only a single step towards a larger goal, then the number of steps to attain the goal is an important metric. The shape optimization environment is a situation where a classifier system cannot be expected to find the optimal shape in a single design iteration. Thus for multi-step goals the performance metric P3 is defined as:

$$P3 = \text{Number of iterations to attain goal.} \qquad (3.13)$$

All these metrics should show an asymptotic improvement with increasing learning iterations, and thus reveal not only the learning progress but also a point where the diminishing returns are so small that continued learning is not justified. At such a point the learning regime must be deemed to have reached, a learned state, or a point of failure.

Another set of metrics tries to explain some of the internal workings of the classifier system. One is a histogram of the strength distribution for the classifier population. In this chart the classifier strengths are grouped into sets with the following ranges;

0-1, 1-3, 3-5, 5-7, 7-9, 9-11, 11-13, 13-15, 15-17, 17-19, 19-20.

The number of classifiers in each set is plotted as a vertical bar, and the horizontal axis shows the strength from 0 to 20. Initially the histogram will show just one bar with a height equal to the population size because all classifiers are initialized with the same value. Figure 3.11 displays a hypothetical strength histogram.

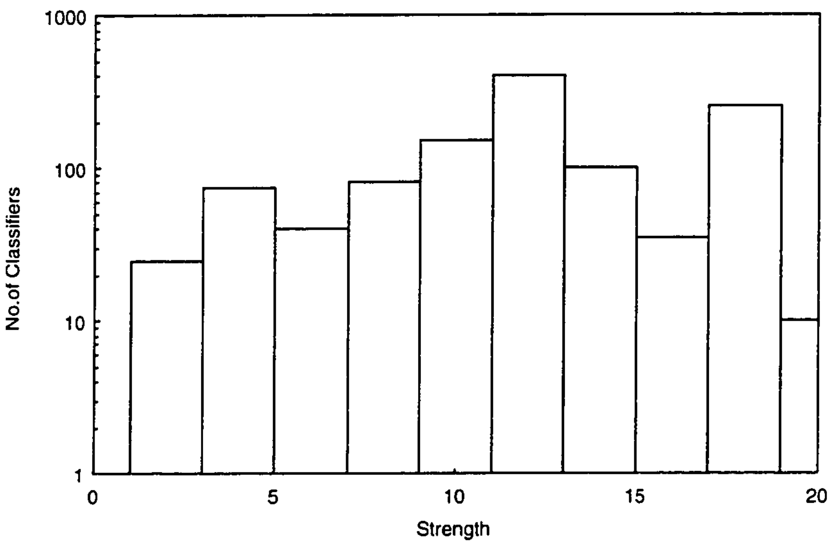


Figure 3.11 ***Strength Histogram Illustration***

The other metric in this set depicts the default hierarchies in the population. The specificity of a classifier is equivalent to its level in the default hierarchies. Figure 3.12 displays a default hierarchy chart of a population of 1000 classifiers. The classifier strengths in this illustration range from 0 to 20, and there are 60 levels of specificity. The strengths can be any real number between 0 and 20 inclusive, while a classifier's specificity can have only a certain number of fixed values. In this illustration there are 61 such values, a level 0 specificity in Figure 3.12 represents having an antecedent consisting of all # symbols, and a level 60 specificity has no # symbols in the antecedent. The overall chart reveals the level of specificity diversity in the population, while the region with strengths greater than the population mean represents the default hierarchies of the best classifiers.

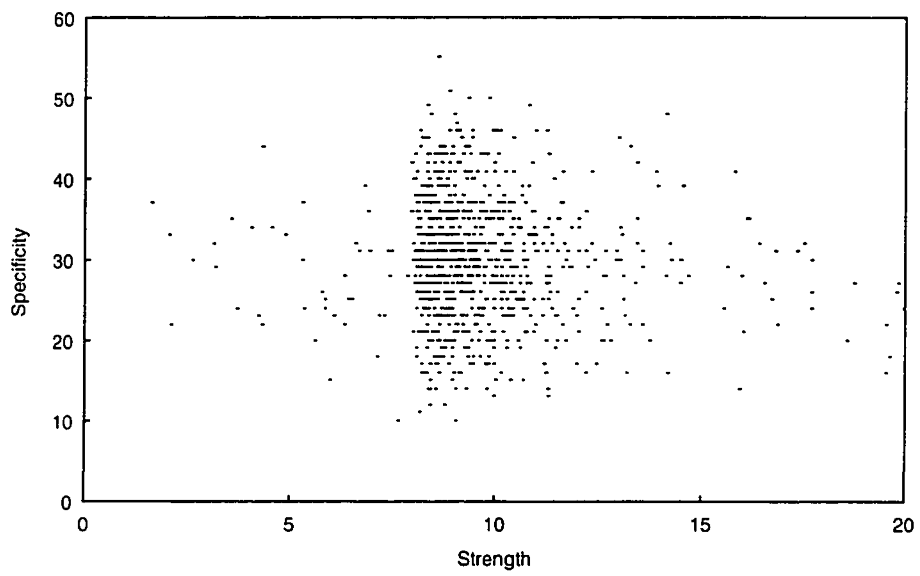


Figure 3.12 ***Example Default Hierarchy Snapshot at Learning Iteration*** **t**

A different set of metrics deals with taking a closer look at the multiple step goal attainment performance (see *P3* above) of the classifier system at various stages of learning. For these metrics, a property describing the state of the environment is charted versus the iterations needed to solve a multi-iteration goal. Figure 3.13 provides an example, showing a chart with two properties that describe the state of a system which requires multiple steps to reach a desired goal. In this example, the system may be thought of as already having *Y* learning iterations behind it when this new goal is presented to the classifier system. The iterations in Figure 3.13 represent those required to accomplish the goal. This illustration took 55 iterations for the goal to be satisfied, since learning continues with this problem, these 55 iterations occur on the learning iterations *Y+1* through *Y+55*. These metrics measure the efficiency and effectiveness as described in Chapter 2, Section 2.5 at different stages of learning.

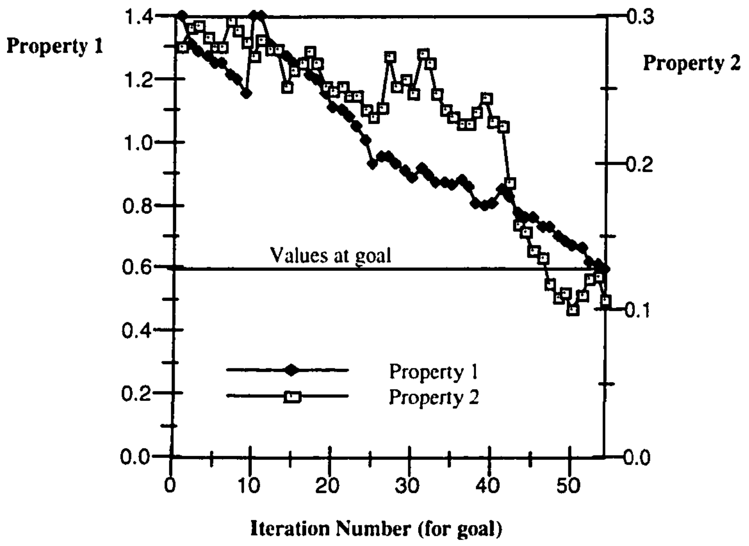


Figure 3.13 ***Example Multi-iteration Goal Plot from Learning Iteration Y to Y+55***

Learning Times

The main drawback for the classifier system is the computational resources needed for the system to learn. The results of this study demonstrate that these computational needs are not restrictive for shape optimization. Furthermore, the CS benefits greatly from where these computational costs derive as the following attests. First, the training is off-line, that is, training is performed not when real problems are being solved but prior on a suite of appropriate examples. Second, in industry one would probably obtain a pre-trained system, thus avoiding the computational and time costs of the training. The consumer then would perceive the acquisition as that of an expert system.

Since the learning process is extensive, historical precedent provides beneficial cues to gauge what duration of learning should be expected, and to know when to concede if learning does not manifest itself. For example, the problems attacked by Riolo [1988] took between 50,000 and 100,000 cycles before achievement of adequate performance. "Alecsys", which taught a simulated and real robot to find benefits and avoid dangers (Dorigo & Schnepf [1991]) needed training on the order of 100,000 to 1 million cycles before the system was considered learned. The current investigation's environment possesses properties which should prove easier to solve than the examples cited, therefore, the learning is expected to require less than 100,000 iterations for significant manifestations of learning to evolve. Furthermore, if measurable learning has not appeared by 500,000 iterations, serious reservations and re-evaluation are in order!

Recall that the genetic algorithm can solve the problems from the environment this study's classifier system is intended to (see Chapter 2, Section 2.5.1). However, as shown, the GA approach can take the same magnitude of iterations to solve just one problem. Furthermore, the GA does not retain any knowledge from the effort, so when a new problem is solved, the solution time is the same as if no prior solves had occurred. Therefore, the classifier system shows the promise of learning to solve problems in this environment in the same magnitude of iterations as the genetic algorithm takes to solve just one problem! Therefore, if the learned classifier system can then solve additional problems in any significantly diminished number of iterations, the overhead of the learning will be justified and the benefit of applying a GA directly will be nil.

3.7 Summary

This chapter provided a pedagogical treatment of the Michigan approach classifier system. Due to the complex nature of the classifier system the review dealt most heavily with the aspects which have the most direct impact on this study. The genetic algorithm discussion emphasized how the algorithm is applied in conjunction with a classifier system. With the mechanisms of the classifier system discussed, some applications were reviewed and the metrics used to measure its performance were introduced with empirical predictions for the expected learning requirements. Table 3.6 summarizes the particulars of the classifier system mentioned in regard to this study.

Table 3.6 ***Classifier System Specifics for this Research***

CS approach	Michigan
Behavior classification	Stimulus-Response (no bucket brigade)
Environment	Structural Shape optimization with stress constraints
Selection	Fitness proportionate reproduction
Pairing of parents	Panmictic
Crossover	Single point crossover
Initial population	*Tabula rasa*
Population size	Static
Replacement & Crowding	Steady state genetic algorithm
Expected learning iterations	< 100,000

Chapter

4

Interfacing Shape Optimization with Classifier Systems

The previous chapter identified that the classifier system (CS) always needs customization for the particular environment to which it is interfacing. This chapter demonstrates how the shape optimization environment can be interfaced with the classifier system tool. The necessary interplay between the CS and the environment dominates the customization. This interplay occurs via three interfaces, *detector*, *effector*, and *feedback*, as summarized in Table 4.1.

Table 4.1 ***Interfaces Overview***

Interface Name	Abbreviation	Summary
Detector	DI	Senses information important to the classifier system from the environment
Effector	EI	Effects environment as prescribed by the classifier system
Feedback	FI	Provides supplemental information needed by apportionment of credit sub-system

The decisions made concerning the interfaces determine many non-interface classifier system customizations. The classifier system by definition encompasses the three interfaces. The phrase *Classifier System Proper* (CSP) is defined as the classifier system absent the interfaces.

Before proceeding with the details of the interfaces, a digression regarding the CS paradigm is appropriate. For engineers to use a classifier system, an adjustment or departure from conventional problem solving doctrine is needed. Classifier systems are quite different from the conventional search methods (as shown in Chapter 3) encountered in engineering optimization in at least the following ways. They:

- work with a coding of the (design) variables, not the variables themselves,
- search from a population of rules, not from a single rule,
- learn from experience,
- use probabilistic operators to guide their search. (By contrast, most common engineering search schemes are deterministic in nature.)

As a point of departure, gradient search may be modeled as a classifier system with one rule encoding the gradient search technique. If a set of rules was then created, one being gradient search and the others being randomly generated, it would be possible to train the system to perform more efficiently or to perform where gradient search could not. In this investigation, the classifier system is taught from a random set of rules in an environment in which gradient search could not perform because no derivative information is provided.

A few global aspects of the CS and its internal workings should be remembered as more details are presented. Recall from Chapter 1, Section 1.3 that the CS, in this study, performs pattern matching against patterns of stress that occur on and interior to the modifiable boundaries, then modifies the boundary in an effort to improve the design. A *classifier* is one such pattern matching rule that also includes a property called strength that provides a means of ranking amongst other classifiers. The total population of classifiers is fixed in size and usually numbers in the hundreds or thousands.

Chapter 2 provided a background on the general problem of shape optimization. The current chapter defines the scope of shape optimization problems over which this study is concerned. With the shape optimization scope set, the environment that the classifier system interacts with will be defined and the interfaces can be established.

4.1 Shape Optimization Scope

For this study, the scope is limited to shape optimization problems where the objective is to minimize the mass of a component with constraints on the maximum allowable von Mises stress. The mass minimization can occur only via the modification of present boundaries. Boundaries may be deemed fixed or modifiable, furthermore the modifiable boundaries may be further constrained by their representation. This scope is further clarified below.

Recall from Chapter 2 that the optimization problem may be formalized as:

$$\begin{aligned} &\textit{minimize:} && f(x) \\ &\text{subject to:} && \mathbf{g}_e(\mathbf{x}) \le 0 \\ & && \mathbf{g}_i(\mathbf{x}) \le 0 \end{aligned} \tag{4.1}$$

where for this study,

x	Vector of design variables which define modifiable surface(s).
$f(x)$	Mass of the design as a function of the design variables.
$g_e(x)$	Vector of explicit constraints consisting of limits for each design variable's dimension.
$g_i(x)$	Vector of implicit constraints including: 1 . Maximum allowable von Mises stress & allowable error. 2 . Restrictions on design variable allowable movement directions. 3 . Minimum rate of improvement. 4 . Maximum allowable iterations for convergence.

The objective is to minimize the mass of the design by manipulation of the boundary. The representation of the boundary and design variables was discussed in Chapter 2, Section 2.4.1. The explicit constraints consist of the bounds placed on the design variables, if any. The control points defining the boundary are the design variables. The major implicit constraint is the maximum allowable von Mises stress, since the optimization performed in this study is relative to stress. The other implicit constraints deal with the termination criteria. One criterion is that the process must terminate if all the design's stresses are within a certain percentage of the maximum allowable. Another states that if the incremental improvement of the design is falling below a certain threshold then optimization must cease. Finally, a constraint is placed on the maximum number of iterations that may occur before terminating the optimization. There are also many non-formalized constraints including, limitations of the analysis technique accuracy, boundary representation limitations and statistical variation in material properties.

This study's scope does not cover the boundary representation module; but since the optimization module must interface with the boundary representation module it must be cognizant of the representation used. To this end the optimization module developed works with many different boundary representations, however, the optimization module will be restricted to working with boundary representations where the design variables are actually on the design boundary. This restriction still allows for most boundary representations including:

- lines, arcs, circles, ellipses,
- cubic splines (where the design variables are the points through which the cubic splines pass),
- surface patches.

The intention is to permit the designer the flexibility needed to construct the initial design with boundary representations appropriate for the situation, from which the optimization module will use the boundary representations as additional constraints.

4.2 Interfaces Overview

The detectors (Chapter 3, Section 3.1.1) provide component state information to the classifier system proper, the effectors (Chapter 3, Section 3.1.1) provide the means for the CSP to modify the component, and the feedback interface provides the apportionment of credit sub-system (Chapter 3, Section 3.1.2) of the CSP with information (about the component's change of state) needed for learning. Figure 4.1 displays a simplified view of the interfaces.

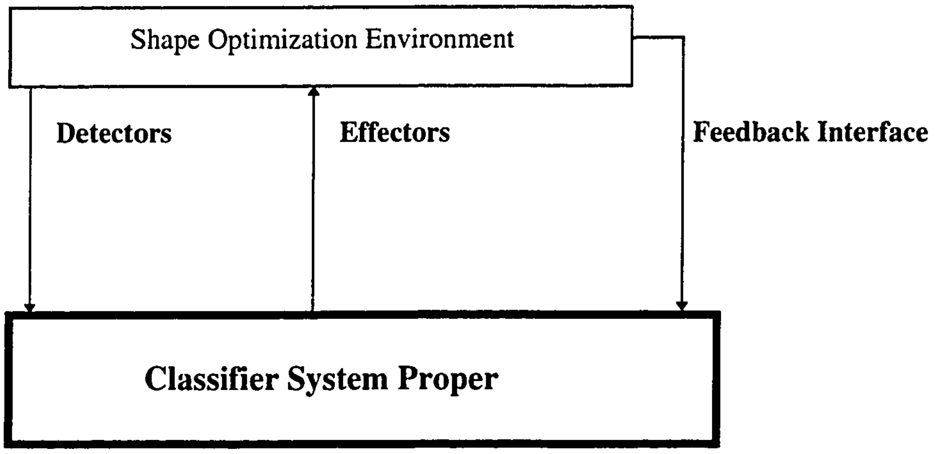


Figure 4.1 ***Interfaces Flowchart***

Again, for shape optimization, the environment consists of one solid component under static loading, the optimization of which may be summarized as:

> a search through the space of determining the best boundary enclosing the material which best meets the design criteria while simultaneously satisfying all the design constraints.

With this in mind, the representation scheme implemented in this investigation modifies the boundary of the design in its quest for improvement.

The detectors filter the available information and process the stress information at control points and at points near each control point; this is fully covered in Section 4.3. The detectors convert the stresses to a binary representation, and formats them into environmental messages. An *Environmental Message* (EM) format corresponds to the format expected by the classifier's antecedent. The detectors create one environmental message per control point.

The effectors receive the action from the CSP, create a modification vector from the binary action, and finally affect (i.e., modify) the component. The component is modified via the modification vector moving one (or more) control point(s), which redefines the component's boundary. Section 4.4 fully covers the effector interface.

The feedback interface collects additional information required by the classifier system proper's AOC module. The AOC module collects the information it may later need from the detectors, however, if additional information is required the feedback interface collects it. In this investigation, the detectors collect all the requisite stress information but ignore mass information, thus the feedback interface collects the required mass information. Section 4.5 covers the feedback interface in detail and its relation to the AOC module. Figure 4.2 shows in more detail the information flow between the environment and the classifier system proper.

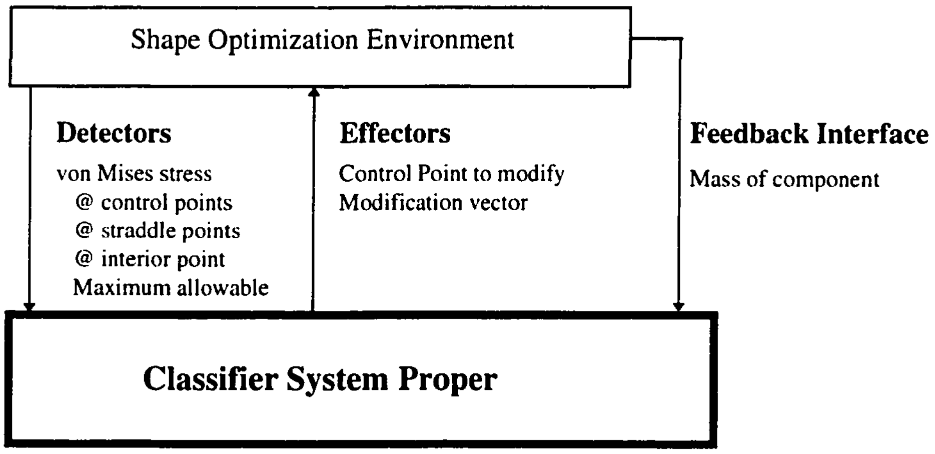


Figure 4.2 *Interfaces Flowchart for Shape Optimization Environment*

4.3 Detectors

The detectors filter all the information produced by the environment, sensing only the subset deemed important by the classifier system. The issues addressed by the detectors drive many of the choices made regarding the format of the classifiers and other internal characteristics.

The shape optimization environment provides a slew of information to the detectors for potential sensing. Using the example of a finite-element analysis module, information available for sensing may include:

- stresses,

  e.g., von Mises, Principal, ...
  e.g., at control points, at nodes, ...
- strains & displacements,
- mass of structure,
- node & element information,
- sensitivities,

  e.g., of stress, of strain, ...
- deflections.

This study will only utilize the minimum criteria (as presented in Chapter 2, Section 2.3). This eliminates:

- sensitivities,
- element information,
- node information,

from the potentially sensed information. This leaves stress, strain, deflection, and mass information as candidates.

In this study, the *von Mises failure criterion*[†] (Shigley & Mitchell, [1983]) is employed. Therefore, with regard to stresses, only von Mises stresses are of concern. Furthermore, no constraints are placed on displacements so global deflection is not sensed. This narrows the list of possible sensed information to:

- von Mises stresses at particular locations,
- mass of structure.

The remaining von Mises stress issue is: at what locations should von Mises stress information be sensed? Since the effect of the system is to move control points which define the modifiable boundaries, the system should at least sample the von Mises stress at all the control points. Recall that the CS is learning to pattern match, i.e., sense a

[†] Also called the *maximum-distortion-energy failure criterion*.

pattern and then prescribe the correct effect. Therefore a pattern of the stress state in the zone around each control point should be presented to the CS. A simple scaleable arrangement is employed, as discussed below.

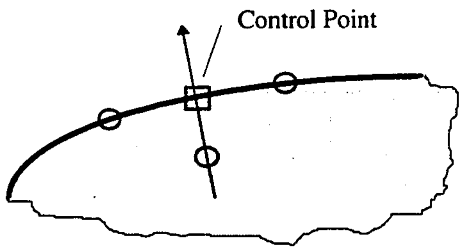


Figure 4.3 ***Control Point with Associated Stress Sensing Locations***

Figure 4.3 illustrates the locations for stress sensing for the two-dimensional case. The sensing occurs at each control point (the box in Figure 4.3), two straddle points (represented as circles on the surface in Figure 4.3), and an interior point (represented as a circle interior to the control point in Figure 4.3). The straddle points are two points, one on each side of the control and lying on the boundary. The interior point is a point interior to the control point lying on the line defined by the surface normal at the control point. Each (non-control) point is one *global element length* (EL) from the control point. For the interior point the distance is measured along the surface normal at the control point. For the straddle points, the distance is measured along the actual boundary geometry. The *global element length* is a user specified parameter, if the finite-element method is used in the analysis module then a global element length is equivalent to the length of the elements defined along the boundary. However, if another method is used, a global element length is still needed, as described below, and should be set to a value appropriate for the model if the finite-element method was being applied.

The classifiers' format is driven by the input to which the classifiers are to match. The classifiers will now see a maximum of four distinct stresses per control point (in the two-dimensional case), as well as the mass. The mass does not appear to play a role in mapping a control point's state to beneficial modifications. Therefore the mass will be ignored by the detectors, (however, the feedback interface does sense the mass).

For scalability from two-dimensional to three-dimensional problems, the straddle point stresses will be compared by the detectors and only the most *prominent* will be passed as a message to the classifier population. *Prominent* in this context means the stress whose value differs greatest from the maximum allowable von Mises stress. The concept of prominent straddle points allows for a seamless extension to three-dimensional cases. In three dimensions the number of straddle points per control point may vary depending on the boundary representation, however, the concept of prominent straddle point leads to only one straddle point being of interest irrespective of how many exist per control point. Algorithm 4.1 presents the algorithm for determining the prominent straddle point stress.

Algorithm 4.1: Prominent Straddle Point Stress Algorithm

```
I.   Define:
          $\sigma_0$             = maximum allowable von Mises stress
          $\sigma_{str}^{k}$     = von Mises stress at straddle point k
          $\sigma_{str}^{pr}$    = von Mises stress of prominent straddle point
          $N_{str}$              = number of straddle points
II.  $\sigma_{str}^{pr} = \sigma_{str}^{1}$
II.  FOR (k = 2 to $N_{str}$)
       IF      $|\sigma_{str}^{k} - \sigma_0| > |\sigma_{str}^{pr} - \sigma_0|$
       THEN    $\sigma_{str}^{pr} = \sigma_{str}^{k}$
```

The above shows that the detectors sense (per control point) the von Mises stresses at the control point, at the interior point, and at each straddle point. From this information the detectors determine the prominent straddle point per control point and then processes the three stresses into a *detector message*; one detector message is created per control point. A *detector message* is a binary string representing sensory input. To create a detector message the environmental input (stresses in this study) must be mapped to a binary alphabet. The detector message format determines the format of the classifiers' antecedent because the detector messages must match the format of the classifiers' antecedent.

Before proceeding further, the format of the detector messages needs to be established. The main consideration is the appropriate granularity; the environment is continuous but the detector messages discretize the input. The level of desired granularity then determines the length of the detector message. However, the detector message can only cover a range. Thus, the range must be established before the granularity can be set. The optimal condition occurs at the maximum allowable von Mises stress, therefore, this is an appropriate value to have as the midpoint of the range. Using a linear mapping, the extremes would then logically be zero stress and twice the maximum allowable von Mises stress. If any stress is encountered greater than twice the maximum von Mises stress, the detectors would convert it to being equal to twice the maximum von Mises stress, creating the maximum detector message. If this range is found to be too restrictive it may always be increased. The *Normalized von Mises Stress* is defined as the von Mises stress mapped to the range as described above and then divided by twice the maximum allowable von Mises stress, resulting in a number between 0 and 1. The optimal normalized von Mises stress occurs at a value of 0.5.

With the range set, the granularity and thus the detector message length can be determined. Recall that a binary string representing a range divides the range into a number of discrete quanta equal to,

$$2^L$$

where L is the string length. The issue is then: how many equal divisions *(2, 4, 8, 16, 32, 64, 128, ...)* of the stress range are needed so that the divisions are discriminating enough for intelligent processing? As with the other aspects of the detector message format, there are not absolute rules to follow; only empirical precedents. Limiting the resulting string length is a significant consideration. The reason is that the CS and GA are searching the space of all possible strings, the longer the string the exponentially larger the search space! Thus it is better to err on the side of having too short a string length, and only increasing the length when necessary.

This study initially tried 64 divisions, resulting in a detector message length of 6 for each of the stress points. This may appear woefully coarse to engineers accustomed to working in a continuous domain. If the divisions are too coarse the system may exhibit oscillatory behavior because the system can not discriminate enough to converge to the point of triggering the convergence termination criterion. Once again, if this occurs a longer bit string can be adopted.

All the important aspects of the detector messages have now been defined, only a bookkeeping decision needs to be set so the detector messages can be assembled into *environmental messages.* An *environmental message* is simply a concatenation of the three detector messages via a defined ordering. The environmental messages are the messages that must agree in format to the classifier antecedents. The ordering of the detector messages in the environmental message is defined as,

Prominent straddle point stress	Control point stress	Interior point stress

The ordering is not important to the operation of the classifier system, only that it is consistent. The detector and environmental messages definitions are now complete. The algorithm for the detectors and the creation of environmental messages is shown in Figure 4.4.

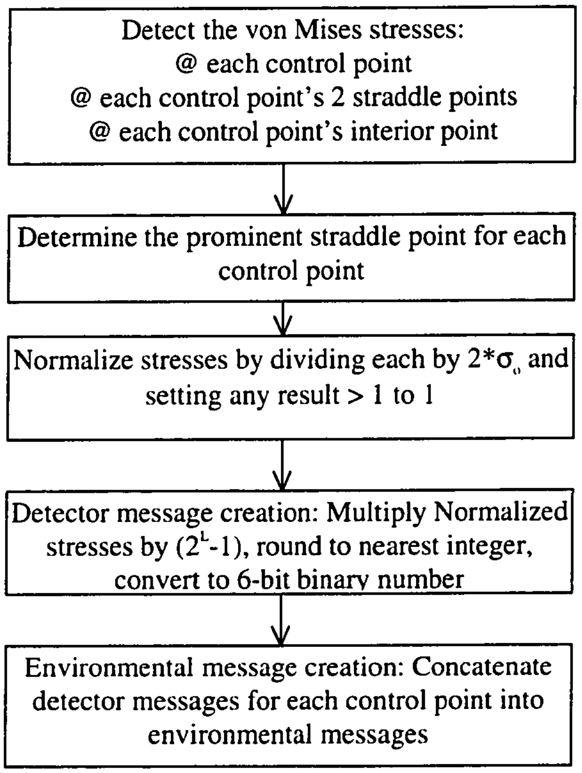

Detect the von Mises stresses:
@ each control point
@ each control point's 2 straddle points
@ each control point's interior point
Determine the prominent straddle point for each control point
Normalize stresses by dividing each by 2*σ_o and setting any result > 1 to 1
Detector message creation: Multiply Normalized stresses by (2^L-1), round to nearest integer, convert to 6-bit binary number
Environmental message creation: Concatenate detector messages for each control point into environmental messages

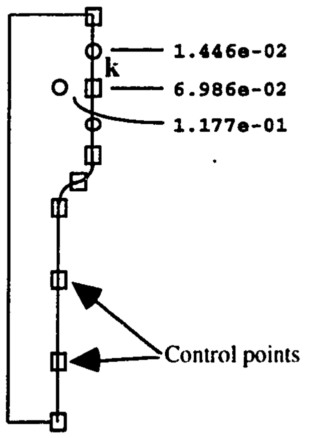
Figure 4.4 ***Environmental Message Creation***

The antecedent (i.e., the *if* portion) of the classifiers is now defined by the requirements of the environmental messages generated from the detector messages. Table 4.2 lists the antecedent properties.

Table 4.2 ***Antecedent Definition***

Antecedent length	18
Alphabet of antecedent	{0,1,#}
Number of conditions	3
Condition 1 length	6
Condition 1 message	Match prominent straddle point's detector message
Condition 2 length	6
Condition 2 message	Match control point's detector message
Condition 3 length	6
Condition 3 message	Match interior point's detector message

Figure 4.5 illustrates the detector interface. The enclosed boundary in Figure 4.5 represents the boundary of a component. The boundary on the right side of the component is modifiable and therefore discretized. The boxes represent the control points defining the boundary. The circles represent the straddle points and the interior point. Moving to the column titled *Normalized von Mises Stress*, the top value represents the stress of the prominent straddle point, the middle value control point **k**, and the bottom the interior point. The next column shows the same stresses converted to detector messages (binary representations). The final column shows the antecedent of a classifier that matches the binary representation. The classifier rule consists of an antecedent having three conditions. The complete antecedent is shown boxed. Notice the antecedent is the concatenation of the three conditions in the right column.



Normalized Von Mises Stress	Binary Representation	Antecedent of a Matching Classifier
1.446e-02	000001	00###1
6.986e-02	001000	0#000#
1.177e-01	000111	#00#11

Complete antecedent of matching classifier:
00###10#000##00#11

Figure 4.5 *Example of a Classifier Matching the Stress Condition at Control Point* **k**

4.4 Effectors

The effectors receive the consequent (i.e., action) of the auction's victorious classifier, translating the binary consequent into a form compatible with the effectors; the effectors then modify the environment (see Chapter 3, Section 3.1). The type of desired environmental effects drive the consequent's format.

The effectors modify control points defining the boundary in the shape optimization environment. Therefore, the consequents must be mapped from a binary representation by the effectors to a modification of the corresponding control point. The control point's modification consists of the distance to move and the direction of the move. For many types of boundary representations, the representation constrains the direction. These include all sizing shape optimization type boundaries such as boundaries defined by a circle, ellipse, thickness, and such. Even more general boundary representations consisting of splines may have constrained directional freedom set by the user. For example, if a spline curve is defined as in Figure 4.6, the dimensions may act as constraints, constricting the control points to moving collinear to the dimension.

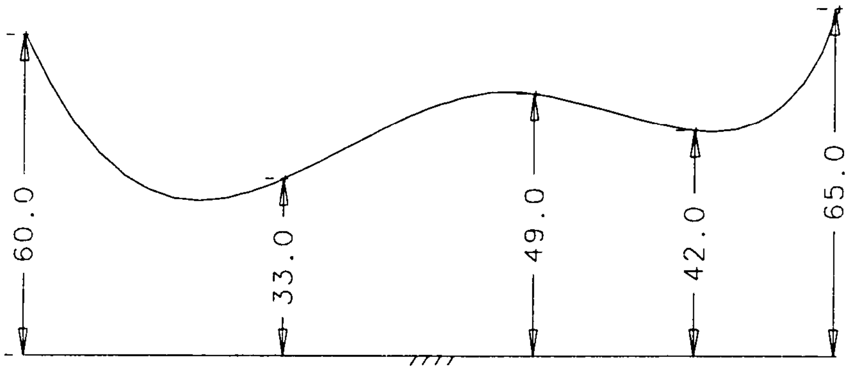


Figure 4.6 ***Spline Curve with Dimension Constrained Control Points***

In the case where a control point is not constrained in its directional freedom; the direction of movement occurs collinear to the surface normal. This discussion shows that the effectors work in conjunction with the boundary representation to determine the direction of movement and therefore the classifier consequent does not factor into the direction determination

The remaining variable for the consequent to control is the control point's move magnitude. The range of possible movements is extensive — the system should be able to optimize components used in structures ranging from bridges to micro-machinery. To design a consequent with enough range and fidelity to optimize such a wide swath necessitates an extended consequent. A rational solution is to have the consequent be a *relative* move magnitude, independent of the scale of the component. As is conventional in optimization systems, *move limits* are imposed on each control point (Fleury [1986]). The move limits define the range used to transform the relative movements to physical movements.

The initial move limit is based on the user defined, global element length (EL). After a control point has been moved three times, approximate functions of stress versus control point location are generated. Algorithm 4.2 defines the move limit algorithm. Now, the purpose of the classifier action or consequent is defined as well as the process followed by the effectors. The consequent or action is, a positive or negative number equal to the control point's relative move magnitude.

Algorithm 4.2: Move Limit Algorithm

```
                    For Control Point Moves 1 & 2

I.    IF     (4 * EL) < (dim_value - lower_limit)/2
      THEN maximum move = 4*EL
      ELSE  maximum move = (dim_value - lower_limit)/2

                 For Control Point Move 3 & Greater

I.    Perform least-squares fit of past iterations, using
            Linear
            Inverse
            Inverse squared
            Inverse cubed
      fitting functions.
II.   Use the best fit as estimate function for stress as a function of the control point
      dimension.
III.  IF     stress < maximum allowable von Mises stress
      THEN maximum move = estimated move to reach three times maximum
                                   allowable
      ELSE  maximum move = three times the estimated move to reach maximum
                                   allowable.
```

The effector interface consists of the following processes:

1. Reception of the victorious classifier's consequent.

2. Conversion of binary consequent to relative move magnitude.
3. Mapping of the relative move magnitude onto the range defined by the move limit, converting it to a physical magnitude.
4. Modification (by the effectors) of the control point's location by the physical magnitude's distance in the direction defined by the constraints of the control point's definition or along the surface normal (if no definition supersedes).

With the purpose of the consequent defined, the length is set to establish the appropriate number of gradations in relative magnitude. For this study the initial number of gradations will be 64. This defines a consequent length of 6, because $2^6 = 64$. Table 4.3 presents the consequent's definition.

Table 4.3 ***Consequent Definition***

Alphabet	{0,1}
Length	6
111111	Largest positive relative magnitude
000000	Largest negative relative magnitude
Positive	Direction away from surface (i.e., along positive surface normal or closest possible via constraint)
Negative	Direction into component (i.e., along negative surface normal or closest possible via constraint)

Table 4.3 shows that the largest positive relative move occurs when the consequent has the value of *111111*, similarly the largest negative relative move occurs when the consequent has the value of *000000*. For the general case, the relative move magnitude is determined by converting the consequent's binary string into its equivalent decimal representation, which will be a number between 0 and 63. The mapping of the relative move magnitude onto the range defined by the maximum move is accomplished as follows:

$$physical\ magnitude = \frac{relative\ move\ magnitude - 31.5}{32} * maximum\ move.$$

Again, the consequent length selection may initially appear too coarse. However as the system nears the optimum, the range defined by the move limits also decreases, resulting in ever finer control point manipulations. In addition, the consequent length can be increased if found unsatisfactory.

Multiple Control Point Modification

Modifications to a design occur after all classifiers that match an environmental message bid for the right to execute its action. When the victorious classifier invokes its action it occurs at the control point that corresponds to the environmental message at which the victorious classifier's bid occurred. This is termed the: *single-control-point modification technique.*

The single-control-point modification technique may be extended into a potential *multiple-control-point modification technique.* The reason the technique is termed potential is that, as will be shown, multiple control point modifications occur only under certain situations. As in the single-control-point modification technique, only **one** classifier wins the auction, and the winner has one control point on the design for which it matched when it generated the winning bid. However, the victorious classifier may have also matched other environmental messages. In the single-control-point modification technique, the control points corresponding to the other environmental messages are left without modification. Now the victorious classifier is permitted to modify all control points for which it matches. This gives the effectors the potential to modify multiple control points in a single design iteration, although this will not always be the case because in many instances only a single point will match the victorious classifier. This study employs the multiple-control-point modification technique exclusively.

Illustration of Multiple Control Point Modification

The following demonstrates a situation where multiple control points are modified by the action of one classifier. In Figure 4.7, three boundary pieces are represented. The pieces could all come from the same modifiable boundary or they could reside on separate boundaries. One useful conception has the uppermost boundary piece taken from a boundary represented by a spline where the control point moves normal to the surface; the middle boundary piece is from an arc, and the lowermost boundary is constrained to remain straight.

Column A shows the normalized von Mises stress with the prominent straddle point first, the control point second, and the interior point third. Column B shows the binary representation of the corresponding normalized von Mises stress. Column C shows the conditions of the classifier that actually won the auction. The antecedent of a classifier consists of 18 fields, the first 6 fields try to match the stress of prominent straddle point, the next 6 fields attempt to match the stress of the control point, and the last 6 fields attempt to match the stress of the interior point. So in Column C the 18 fields of the antecedent of the winning classifier have been broken up into its 3 conditions to illustrate the matching that occurred.

Observing the other stress patterns, it may be noticed that the lowermost control point's stresses also match the winning classifier's antecedent.

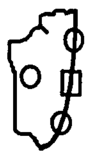
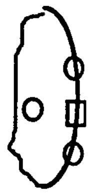
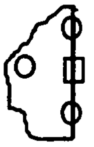
	A. Normalized Von Mises Stress	*B.* Binary Representation	*C.* Conditions of Victorious Classifier
Case 1 	3.342e-02	000010	00001#
	1.171e-02	000001	0#####
	4.564e-02	000011	000#11
Case 2 	7.536e-02	000101	
	3.019e-03	000000	
	6.986e-02	000100	
Case 3 	5.086e-02	000011	00001#
	5.564e-02	000100	0#####
	5.522e-02	000011	000#11

Figure 4.7 ***Multiple Control Point Modification Illustration***

In this illustration, modifications would take place at 2 control points. The full classifier antecedent is:

`00001#0#####000#11.`

Therefore whatever the action of this classifier is, the effectors execute the action at both control points. Since the design modification for every iteration is due to a single classifier; the merit of the design modification can be judged, and feedback can be correctly sent to the classifier which is solely responsible.

4.5 Feedback Interface: Apportionment of Credit

The feedback interface monitors modifications to the environment in order to propagate feedback to the classifier system's apportionment of credit sub-system during learning mode (see Chapter 3, Section 3.1.2). The apportionment of credit sub-system then punishes or rewards the classifier which instigated the environmental modification.

Classifier systems require only punishment/reward type feedback, in contrast to correct answer feedback. Feedback consisting of punishments and rewards can be determined by only knowing if the CS's actions caused a beneficial or detrimental effect. Thus the CS can learn in situations where the correct or perfect modification is not known *a priori*. In addition, any available supplemental information may be used to enhance the learning process, exhibiting yet another example of the classifier system's flexibility.

For the shape optimization environment, the feedback consists of whether the modification resulted in a more optimal or less optimal design. In addition to answering this question, it is not difficult to supply ranking to the feedback. The input interface collects all the requisite stress information but ignores mass information. It is the feedback interface that collects the required mass information. In many cases the change in mass provides a viable ranking of the change when no stress constraint violations occurred pre or post modification. For more complicated situations, other considerations need to be factored.

Recall that for this study's structural design optimization, the objective function minimizes the mass, while constraints include maximum limits on the von Mises stress. For stress constrained optimization problems, the *Total Normalized Stress Error* (TNSE), as defined in the work by Hsu [1992], complements the mass as an evaluator. The TNSE is defined as:

$$TNSE = \frac{\sum_{k=1}^{m} |\sigma_k - \sigma_o| \Delta s_k}{\sum_{k=1}^{m} \Delta s_k}, \tag{4.3}$$

where,

m	Number of control points on a boundary.
σ_k	von Mises stress at control point k.
σ_o	Maximum allowable von Mises stress.
Δs_k	Half the cord length between s_{k+1} and s_{k-1} along the boundary.

The TNSE may be thought of as the entropy of stress, and a necessary (but not sufficient) optimal design condition is one of minimum entropy. Therefore, the change in mass and the change in TNSE per cycle are the primary feedback components.

With knowledge of the changes in stress and mass, and with TNSE defined, a matrix may be set up showing whether the modification proved beneficial or detrimental. This decision matrix is shown in Table 4.4. Further particulars of the application of the Modified Design Decision Matrix to the AOC module are deferred to Chapter 5, Section 5.2.2.

Table 4.4 *Modified Design Decision Matrix*

	Lighter Prev: No viol	Lighter Prev: Violations	Heavier Prev: Violations	Heavier Prev: No viol
TNSE ↓ **No violations**	Beneficial	Beneficial	Beneficial	Detrimental
TNSE ↓ **Violations**	Beneficial	Beneficial	Beneficial	Detrimental
TNSE ↑ **No violations**	Beneficial	Beneficial	Detrimental	Detrimental
TNSE ↑ **Violations**	Detrimental	Detrimental	Detrimental	Detrimental

Legend

Lighter	Mass decreased
Heavier	Mass increased
Prev: No viol	No von Mises stress in the design was greater than the maximum allowable von Mises stress in the previous iteration's design
Prev: Violations	At least one von Mises stress in the design was greater than the maximum allowable von Mises stress in the previous iteration's design
TNSE ↓	The total normalized stress error decreased
TNSE ↑	The total normalized stress error increased
No violations	No von Mises stress in the design is greater than the maximum allowable von Mises stress
Violations	At least one von Mises stress in the design is greater than the maximum allowable von Mises stress

4.6 Interfaces: In-depth Review

With the details of the interfaces defined, a detailed information flow diagram of all the external interactions may be constructed. Figure 4.8 displays the information flow in and out of the CSP categorized by interface.

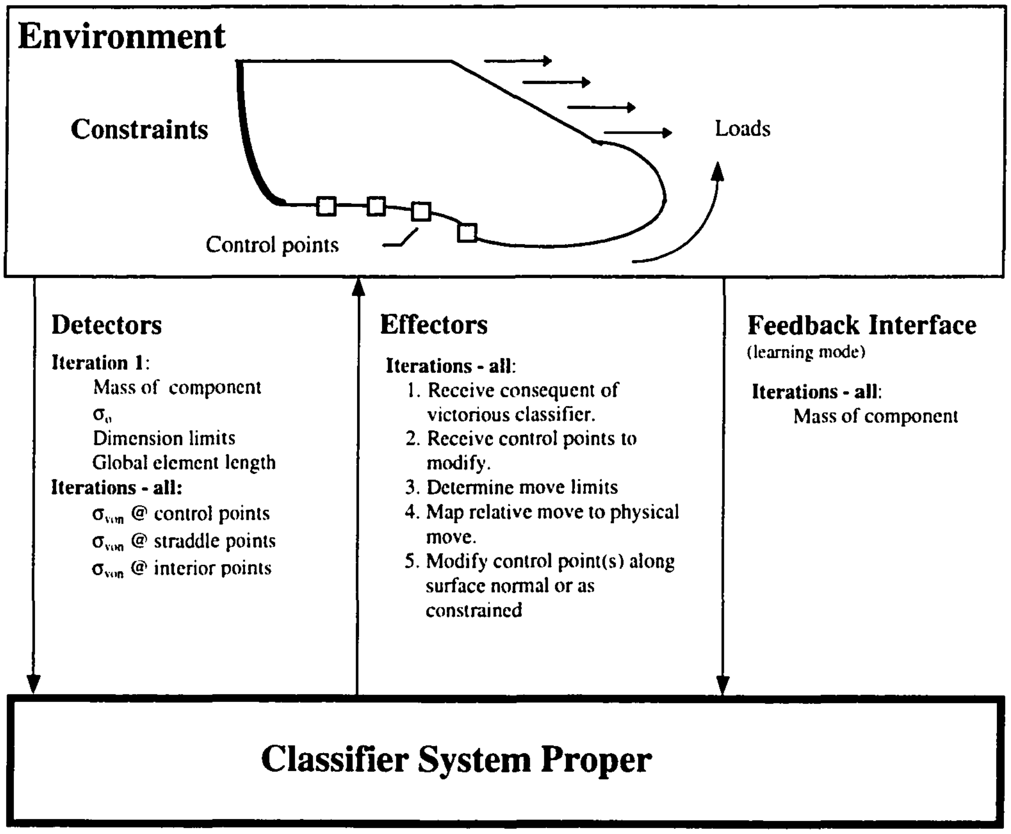


Figure 4.8 *Shape Optimization Environment and Classifier System Proper Interactions*

4.7 Computational Complexity

As discussed in the previous sections, many decisions affected the size of the classifiers. In all cases, the goal of minimizing the resulting classifier length played an influential role in the decisions. This is due to the exponential growth of the search space. Now that the format of the classifiers has been determined to consist of an antecedent of length 18 from a tertiary alphabet and a consequent of length 6 from a binary alphabet, the total classifier search space is:

$$\text{Number of possible classifiers} = 3^{18} * 2^{6}$$
$$> \mathbf{2.479 * 10^{10}}.$$

The classifier system's goal then is to create the best population (on the order of 1,000 classifiers) for mapping the stresses received from the detectors to effects which most effectively and efficiently shape optimize components. The system was expected to do this by only sampling on the order of,

4 E -04 %

of the total space of possible classifiers. The task is daunting, but if accomplished, demonstrates the incredible utility of such a tool to the mechanical engineering community.

4.8 Summary

This chapter began by defining the scope of shape optimization problems this study investigates. After which the bulk of the chapter constructed the mechanisms by which information flows between the shape optimization environment and the classifier system proper. The information flow into, and the types of results desired from the classifier system proper, defined much of the internal characteristics of the classifier system. Table 4.5 summarizes this research's developments to this juncture.

Table 4.5 ***Snapshot of Research Developments***

Objective	Minimize Mass
Scope	Shape optimization (subsumes size optimization). Flexible to a wide range of initial designs, which may be feasible or infeasible.
Efficiency	Competitive with modern optimization techniques when within their scope. Orders of magnitude better than systems which have same scope.
Effectiveness	Optimums as good as any other technique.
Information for Optimization	Minimum criteria, system performs zeroth-order optimization
Use of aux. info	Extendible to exploit auxiliary information
Analysis	Independent of analysis method
Boundary Rep	Flexible to most boundary representations
Algorithm	Learns optimization (no hard coding of algorithm)
Design variables	Control points defining the modifiable boundaries
Failure criterion	von Mises
Failure constraint	Maximum allowable von Mises stress
Design variable constraints	Limits on allowable dimensions (if any) Restrictions on directional freedom of movement
Termination constraints	Percent deviation from optimal stress to be considered optimal Minimum rate of improvement Maximum number of iterations permitted
Stress range	0 to 2 * σ_0: larger stresses mapped to (2 * σ_0)
Detector range	0 to 2 * σ_0: larger stresses mapped to (2 * σ_0)
Detector message format	6 bits
# per control point	3
First	Prominent straddle stress
Second	Control point stress
Third	Interior point stress
Environmental message format	18 bits: Concatenation of detector messages
Specificity	Dependent on # location
User input	Initial design, with design variable constraints Maximum allowable von Mises stress Global element length Termination constraints
Direction of control point movement	Determined by boundary representation or surface point modification normal vector
Antecedent length	18
Alphabet of antecedent	{0,1,#}
Number of conditions	3
Condition 1 length	6
Condition 1 message	Match prominent straddle point's von Mises stress
Condition 2 length	6
Condition 2 message	Match control point's von Mises stress
Condition 3 length	6
Condition 3 message	Match interior point's von Mises stress
Consequent Alphabet	{0,1}
Consequent Length	6
Consequent range	111111 : Largest positive relative magnitude 000000 : Largest negative relative magnitude
Positive	Direction away from surface (i.e., along positive surface normal or closest possible via constraint)
Negative	Direction into component (i.e., along negative surface normal or closest possible via constraint)
Number of possible classifiers	$> 2.4 * 10^{10}$

Chapter

5

SPHINcsX Algorithm & Learning Suite

This chapter details the complete algorithm used by the Shape oPtimization via Hypothesizing Inductive classifier system compleX (SPHINcsX). In order to apply the algorithm in learning mode, a suite of problems is needed. This chapter defines the learning suite of designs used to teach SPHINcsX shape optimization.

5.1 SPHINcsX Algorithm

This section describes the learning algorithm used by SPHINcsX in the shape optimization process. *Algorithm 5.1* provides the basic outline, with references to where details of the steps may be found. Many of the steps were developed in Chapters 3 and 4 and the balance will be discussed in the following subsections. *Algorithm 5.1* complements and clarifies the information flow provided in Figure 3.10 which detailed the classifier system and its interaction with the environment. A more detailed version of the algorithm used by SPHINcsX may be found in Appendix B.

Recall from Chapter 3, Section 3.1 that the CS has two major *modes* in its application to most problems. These are the learning mode and application mode. During the learning mode the system learns to operate in the structural component shape optimization environment. After learning, the CS is applied to problems using its learned rules similar to an expert system. The application mode is a subset of the learning mode. *Algorithm 5.1* shows SPHINcsX with the classifier system in learning mode.

Algorithm 5.1: SPHINcsX Algorithm in Learning Mode

```
            Initialization, Termination & Genetic Algorithm Modules
I.      Initialization of Classifier System                     (Chapter 5, Section 5.2)
II.     Initialization of a design to be optimized              (Chapter 5, Section 5.3)
III.    Increment: problem iteration and learning iteration.
IV.     IF learning termination criteria met, terminate.                     (Chapter 6)
V.      IF epoch completed; apply genetic algorithm.
VI.     Continue to Optimization/Learning Loop.

                        Optimization/Learning Loop
I.      Analysis Module             Iterations all              (Chapter 2, Section 2.4.2)
II.A.   Feedback Interface          Iterations $i > 1$          (Chapter 4, Section 4.5)
        Read mass of design at iteration $i$
   B.   Apportionment of Credit     Iterations $i > 1$
        Determine if the design has improved or deteriorated, reward or punish.
III.A.  Detector Interface          Iteration: 1                (Chapter 4, Section 4.3)
         Read:  Mass, Maximum allowable von Mises stress, Limits on dimensions
                Global element length
   B.   Detector Interface          Iterations: all
         Read:  von Mises stress at: control points, straddle points, interior points
        Create environmental messages
IV.     Auction module              Iterations: all             (Chapter 3, Section 3.1.2.1)
   1)   Match environmental messages with classifiers.
   2)   IF      no classifiers matched in 1), apply the triggered cover detector
                operator, Skip to step IV.5
   3)   Auction: Have all the classifiers that matched compete in an auction to determine
        which one shall be permitted to execute its action.
   4)   Pass the action of the classifier that won the auction to the effectors.
   5)   Record the victorious classifier for this iteration.
V.      Collect taxes               Iterations: all             (Chapter 3, Section 3.1.2.3)
VI.     Effector Interface          Iterations: all             (Chapter 4, Section 4.4)
        Modify all control points which matched the victorious classifier.
VII.    Termination criteria        Iterations: all             (Chapter 4, Section 4.1)
   1)   IF      stresses are within $\varepsilon$ of the optimum, terminate & return to Initialization,
        Termination & Genetic Algorithm Module step II.
   2)   IF      iteration $i$ is greater than a user supplied maximum, terminate & return
                to Initialization, Termination & Genetic Algorithm Module step II.
   3)   IF      none of the above termination criteria are satisfied, continue.
VIII.   Set the active design to the design created in step VI; increment the problem iteration
        and the learning iteration.
        IF      epoch completed; return to Initialization, Termination & Genetic
                Algorithm Modules step IV.
        ELSE    return to step I.
```

5.2 Initialization of Classifier System

Much of the *structure* of the classifier system was defined in Chapter 4, see Table 4.5. The initialization of a classifier system is concerned with the *parameters* of the classifier system. For example, the CS's structure defines the format of the individuals in the population, the size of the population and the initial individuals (which meet the requisite structure) are parameter settings. The following sections describe the initialization of the classifier system in detail.

5.2.1 Initial population

The determination of the best population size is an area that classifier system research has not addressed completely. Results from other studies and simulations (Horn, et al. [1994]) provide some guidelines for reasonable population size settings. The computations required by an oversized population are relatively small compared to the computations needed to evaluate the stresses in the shape optimization environment. For these reasons a population of, 1,000 is employed. The actual classifiers which are used during learning are tracked, so if the system learns while only using a subset of the population then it will be known that the population size is adequate. Furthermore, the population can be reduced to include only active individuals if desired. Contrarily, if all the classifiers are being employed and the learning does not progress to a learned state, the remedy may be to increase the population size.

As stated previously (Chapter 3, Section 3.4) a *tabula rasa* will be used for the initial population. However, a parameter needs to be set which determines the probability of picking a *#*. The population is created by picking a *0*, *1*, or *#* for each position for each classifier's antecedent. The probability of picking a # for each position is determined by the, *# probability parameter*, for each position for which a # is not selected, there is an even chance a 0 or 1 will be picked. The consequent for each classifier is created by selecting a 0 or a 1 for each position with equal probability.

The *# probability parameter* is set to 0.6. The complete initial *tabula rasa* population algorithm is shown in Algorithm 5.2. Since the algorithm uses stochastic operators the algorithm has a non-deterministic result.

Algorithm 5.2: Initial *Tabula Rasa* Population Algorithm

```
                         Main Loop
I.     Set classifier number, j, equal to 0.
II.    Set j = j + 1
III.   Set antecedent position counter, k, equal to 0
IV     Set consequent position counter, l, equal to 0
V.     Continue to Antecedent Loop
VI.    Continue to Consequent Loop
VII.   IF     j > 1,000 population generated
       THEN Stop
       ELSE  Return to step II.

                      Antecedent Loop
I.     Set k = k+1
II.    Flip biased coin with 60% chance of landing heads
III.   IF     coin landed heads
       THEN set position k of antecedent to #
                   Skip to step VI
       ELSE continue
IV.    Again flip unbiased coin with 50% chance of landing heads or tails
V.     IF     coin landed heads
       THEN set position k of antecedent to 0
       ELSE  set position k of antecedent to 1
VI.    IF     k = 18
       THEN Exit Antecedent Loop
       ELSE  Return to step I.

                      Consequent Loop
I.     Set l = l+1
II.    Flip unbiased coin with 50% chance of landing heads or tails
III.   IF     coin landed heads
       THEN set position l of antecedent to 0
       ELSE  set position l of antecedent to 1
IV.    IF     l = 6
       THEN Exit Consequent Loop
       ELSE  Return to step I.
```

5.2.2 Apportionment of Credit Parameters

The feedback structure of the apportionment of credit sub-system was described in Chapter 4, Section 4.4. The discussion presented a decision matrix, Table 4.4, for determining whether a design modification proved beneficial or detrimental. The amount of reward paid to a classifier that causes a beneficial design modification can be scaled since the amount of improvement is quantifiable. Table 5.1 shows the evaluator used to scale the improvement for beneficial modifications. This matrix corresponds to the upper left portion of the matrix shown in Table 5.1. In the case of a detrimental modification an implicit punishment is implemented.

Table 5.1 ***Evaluator for Beneficial Modifications***

	Lighter Prev: No viol	**Lighter Prev: Violations**	**Heavier Prev: Violations**
TNSE ↓ No violations	ΔMass	ΔMass	ΔTNSE
TNSE ↓ Violations	ΔMass	ΔMass	ΔTNSE
TNSE ↑ No violations	ΔMass	ΔMass	

Table 5.1 shows that the distinguishing feature determining the use of ΔMass or ΔTNSE as the evaluator is whether the modified design is lighter or not. Algorithm 5.2.2 presents how the evaluators are utilized to provide reward. In addition the punishment technique is explained.

Algorithm 5.3: Reward and Punishment Algorithm

```
I.   IF    modified design better (see Table 4.4)
     THEN  Continue to Reward Section
     ELSE  Continue to Punishment Section
```

Reward Section

```
I.   IF    modified design lighter
     THEN  Continue to step II
     ELSE  Continue to step IV
II.  Calculate the change in the mass between the modified design and the
     pre-modified design. The change is the ratio of; the difference between the
     pre-modified design's mass ($m_i$) and the modified design's mass ($m_{i+1}$), and the
     pre-modified design's mass, the result is set to ΔMass.
         $\Delta Mass = \frac{m_i - m_{i+1}}{m_i}$
III. Set the Reward to ΔMass multiplied by the Reward Coefficient.
         Reward = ΔMass * (Reward Coefficient)
     Jump to step VI.

IV.  Calculate the change in the TNSE between the modified design and the
     pre-modified design. The change is the ratio of; the difference between the
     pre-modified design's TNSE ($TNSE_i$) and the modified design's TNSE
     ($TNSE_{i+1}$), and the pre-modified design's TNSE, the result is set to ΔTNSE.
         $\Delta TNSE = \frac{TNSE_i - TNSE_{i+1}}{TNSE_i}$

V.   Set the Reward to ΔTNSE multiplied by the Reward Coefficient.
         Reward = ΔTNSE * (Reward Coefficient)
     Continue to step VI.

VI.  Update the Strength of the classifier.
     IF    Strength + Reward < 20
     THEN  Strength = Strength + Reward
     ELSE  Strength = 20
```

Punishment Section

```
I.   No explicit punishment. Classifier is implicitly punished because the classifier's
     winning bid decreased its strength (see Chapter 3, Section 3.1.2.2).
```

During the development of SPHINcsX the *Reward Coefficient* was adjusted and the resulting affects on the learning process were observed. The value of *7.5* proved to provide a good level of positive feedback.

5.2.3 Auction Parameters

Chapter 3, Section 3.1.2.1 provided the structure and equations used by the auction module. Table 5.2 shows the auction parameters used by SPHINcsX. The parameters where determined via guidelines provided by previous studies (e.g., Goldberg [1989]) and experimentation conducted for this study.

Table 5.2 *Auction Parameters*

Classifier Bid Coefficient	k_o	0.1
Bid Coefficient 1	k_1	0.1
Bid Coefficient 2	k_2	0.0833
BidRatio exponent	BRPow	1.0
Standard deviation of noise	σ_{bid}	0.15

5.2.4 Tax Parameters

Chapter 3, Section 3.1.2.3 provided the structure and equations used for taxation. As described there is an algorithmic approach for setting the Life Tax utilizing the free-fall half life, which gives,

$$(1/2)^{(1/n)} = 1 - Tax_{life}$$

$$Tax_{life} = 1 - (1/2)^{(1/n)}$$

The parameter, n, is the epoch length. Section 5.2.5 below shows that this study uses an epoch length of 150 iterations. Using an epoch length of 150 results in a Life Tax of,

$$Tax_{life} = 0.00461.$$

Experiments conducted for this study found that using a bid tax less than the inverse of the number of classifiers in the population provided good results. Since the population contains 1,000 classifiers the bid tax was tested below 0.01, the best results were found to occur using a bid tax of 0.003. The tax parameters are summarized in Table 5.3.

Table 5.3 ***Tax Parameters***

Life Tax (also called Head Tax)	Tax_{life}	0.0046
Bid Tax	Tax_{bid}	0.003

5.2.5 Genetic Algorithm & Triggered Cover Detector Operator Parameters

Chapter 3, Section 3.2 provided the structure and process followed in the Genetic Algorithm module, while Section 3.1.3 described the Triggered Cover Detector Operator (TCDO). In a classifier system, many iterations occur whereby the classifier system performs interactions with the environment between each application of the genetic algorithm; the number of iterations between the genetic algorithm application is termed an *epoch*. The epoch is necessary to create a ranking among the classifiers. The GA's power depends upon the relationship of the classifier strengths corresponding to the steady-state strength ranking. That is, if a population of classifiers was permitted to run without a GA ever being applied, and without a TCDO, eventually a near steady-state condition would result where the strengths would represent the relative merit of each classifier in relation to the rest of the population. The other extreme would be to apply the GA on every iteration, in this case the classifier system would not have the ability to rank the population, thus the strengths would be meaningless. The GA in this scenario would be performing random search. So the goal then is to determine an epoch length providing enough experience to create a valid relative ranking between the classifiers, while not making the epoch length so long as to waste computational resources and therefore not fully exploit the full learning capacity of the genetic algorithm.

Classifier system research has not provided a quantitative method for setting the epoch length, however, as with many classifier system parameters a wide range of epoch lengths will afford satisfactory results. The classifier system literature and experimentation carried out for this study resulted in a suite of parameters for expedient learning. Table 5.4 displays the genetic algorithm and triggered cover detector operator parameters.

Table 5.4 *Genetic Algorithm & Triggered Cover Detector Operator Parameters*

Epoch length (GA period)	150 cycles
Proportion of Population Selected to Breed per Epoch	10%
Probability of Mutation	0.1%
Probability of Crossover	100%
Crowding Factor	100
Crowding Sub-population	150

5.3 Learning Suite

The educational process consists of a progression through a learning suite of designs to be optimized. The learning suite contains a range of designs covering the spectrum from simple one design variable sizing optimization to complex multiple design variable sizing optimization designs. The educational process defines the minimum design space scope for which SPHINcsX is effective. As the educational process continues SPHINcsX may reach a point where further classes of designs cannot be handled. If such a situation occurs it establishes the scope of applicability. However, if such a point is not reached, the scope may be broader than the learning suite's scope. This property provides the justification for a learning suite consisting only of sizing problems, for empirical results showed that SPHINcsX can not distinguish between size and shape optimization problems, thus if a broad range of size optimization problems are used to teach SPHINcsX then the learned SPHINcsX should also be able to optimize shape designs.

The learning suite is broken down into *clusters*; *clusters* are groups of similar problems, that have a common unique characteristic. The clusters composing the learning suite are named as follows ranging from simple to more complex:

- Tension Rod,
- Pressure Vessel,

- Torsion Solid Rod,
- Cantilever Beam,
- Torsion Elliptical Solid Rod,
- Torsion Elliptical Hollow Rod.

The following sections describe each of the clusters and formalize the optimization problems SPHINcsX will use for its learning suite.

5.3.1 Tension Rod

The Tension Rod, labeled *TR*, provides a good basic cluster of problems to commence with because it possesses the following characteristics:

- sizing problem,
- closed form solution (in many incarnations),
- known optimal solution,

which provide for simple design analysis and optimization performance evaluation.

The simplest incarnation of the tension rod design, labeled *TR1*, is shown in Figure 5.1. In this case only one design variable exists, the radius of the tension rod, which is constant through the length of the rod. In this case, the stress (σ) is also constant throughout the length and has the value of,

$$\frac{Force}{Area} \tag{5.1}$$

where,

$$Area = \pi R^2.$$

Since the only design variable is R, the problem is reduced to,

$$\sigma = \left(\frac{F}{\pi}\right)\frac{1}{R^2} \tag{5.2}$$

therefore, the optimal radius may be calculated from,

$$R = \sqrt{\frac{F}{\pi\sigma}}. \tag{5.3}$$

SPHINcsX does not know that such an explicit representation exists for this problem. Since the stress is uniform throughout the tension rod, the control point, straddle point and interior point stresses (see Chapter 4, Section 4.3) are all equal. As shown in Equation 5.2 the tension rod problem exposes SPHINcsX to a situation where the stress is an inverse squared function of the design variable.

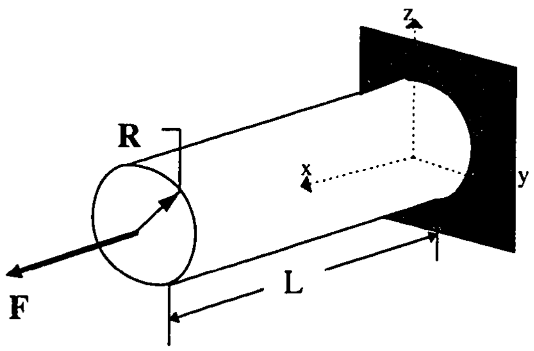


Figure 5.1 *Tension Rod*

Table 5.5 *Tension Rod Problem Parameters*

Name	Label	Value
Force	F	785 Newtons
Length	L	0.1 meters
Maximum Allowable von Mises Stress	σ_o	$10*10^6$ Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.002 meters
Mass	M	πR^2

Table 5.6 *Tension Rod Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Radius	R	$R_{\text{init}}^{\text{min}}$	0.015	ΔR_{init}	0.015	0.030	0.001

Table 5.7 *Tension Rod Stress Characteristics*

Stress Point	σ_x	σ_y	σ_z	von Mises Stress
Control Point 'R' (Rod Exterior)	$\frac{F}{\pi R^2}$	0	0	$\frac{F}{\pi R^2}$
Interior point	$\frac{F}{\pi R^2}$	0	0	$\frac{F}{\pi R^2}$
Prominent straddle point	$\frac{F}{\pi R^2}$	0	0	$\frac{F}{\pi R^2}$

To assist SPHINcsX in learning to handle general problems and not just learn to rote the designs used during the educational process, randomness is added to the initial design configuration. The educational process consists of hundreds of optimizations. To make each design optimization unique, a range of possible initial design variable values is implemented. To illustrate, in TRI the initial radius definition includes a random constituent,

$$R_{\text{init}}^{\text{min}} + \Delta R * [rand(0,1)]$$

where,

$rand(0,1)$	A random number between 0 and 1 inclusive.
ΔR	The maximum of the allowable range.

Therefore, if the random number is *0*, R_{init} will be the initial radius, $R_{\text{init}}^{\text{min}}$, while if the random number is *1* then the initial radius is $R_{\text{init}}^{\text{min}} + \Delta R$. For any other random number between 0 and 1 the initial value of the radius will be,

$$R_{\text{init}}^{\text{min}} < R_{\text{init}} < R_{\text{init}}^{\text{min}} + \Delta R. \tag{5.4}$$

The general case for the tension rod problem, labeled *TRg*, is defined in Figure 5.2. The radius is defined by ***n*** control points, and a smooth transition occurs between the ***n*** control points. For ease of computation the length is defined to be great enough so that at each control point the stresses (straddle, interior & control) are defined by,

$$\sigma = \left(\frac{F}{\pi}\right)\frac{1}{R^2}, \tag{5.5}$$

in effect, ignoring stress concentration affects. In TRg the initial value of n is variable, thus TRg exposes SPHINcsX to a multiple design variable situation and situations where SPHINcsX's multiple control point modification capability may be applied beneficially. When $n = 1$ TRg degrades to TRI.

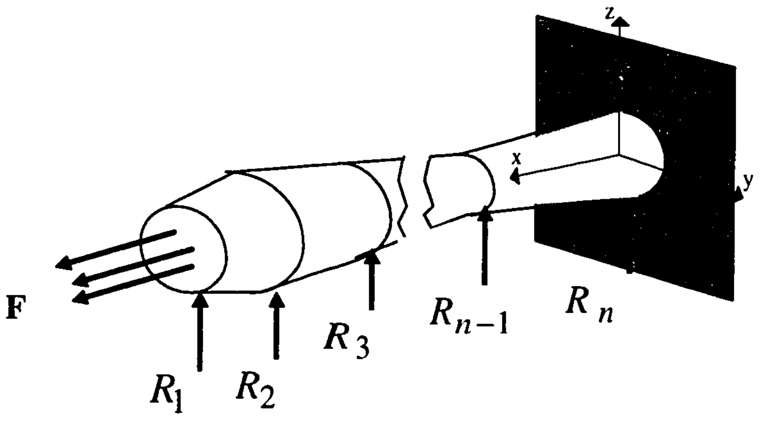


Figure 5.2 *General Tension Rod*

Table 5.8 *General Tension Rod Problem Parameters*

Name	Label	Value
Force	F	785 Newtons
Length	L	Long enough to minimize stress concentrations
Maximum Allowable von Mises Stress	σ_o	$10*10^6$ Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.002 meters
Mass	M	$\pi*(R1^2 + R2^2 + \ldots + Rn^2)$

Table 5.9 *General Tension Rod Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper	Lower
		Label	Value (m)	Label	Value (m)	Limit (m)	Limit (m)
Radius 1	R_1	$R1_{\text{init}}^{\text{min}}$	0.015	$\Delta R1_{\text{init}}$	0.015	0.030	0.001
Radius 2	R_2	$R2_{\text{init}}^{\text{min}}$	0.015	$\Delta R2_{\text{init}}$	0.015	0.030	0.001
...	...	...	...	...	...	...	...
Radius N	R_n	$Rn_{\text{init}}^{\text{min}}$	0.015	ΔRn_{init}	0.015	0.030	0.001

Table 5.10 *General Tension Rod Stress Characteristics*

Stress Point	σ_x	σ_y	σ_z	von Mises Stress
Control Point 'Ri' (Rod Exterior for Radius *i*)	$\frac{F}{\pi Ri^2}$	0	0	$\frac{F}{\pi Ri^2}$
Interior point	$\frac{F}{\pi Ri^2}$	0	0	$\frac{F}{\pi Ri^2}$
Prominent straddle point	$\frac{F}{\pi Ri^2}$	0	0	$\frac{F}{\pi Ri^2}$

5.3.2 Pressure Vessel

The Pressure Vessel, *PV*, provides the second cluster of designs and Figure 5.3 defines the problem. Only the inner radius is a design variable. The pressure vessel possesses the following properties:

- sizing problem,
- closed form solution,
- known optimal solution,
- von Mises stress a complex function of the design variable,
- the pressure vessel is very long, and thus the longitudinal stresses at the point of interest is zero.

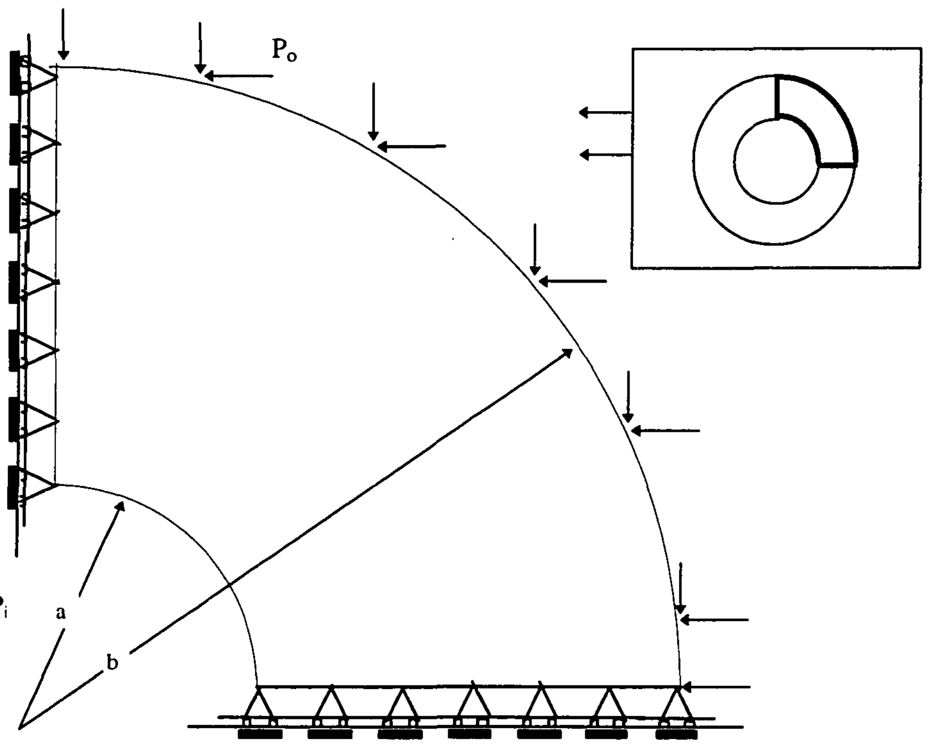


Figure 5.3 *Pressure Vessel: Quarter Model*

Table 5.11 *Pressure Vessel Problem Parameters*

Name	Label	Value
Internal pressure	P_i	0 Pascals
External pressure	P_0	400,000 Pascals
Maximum Allowable von Mises Stress	σ_o	$1*10^6$ Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.005 meters
Mass	M	$\pi*(b^2 - a^2)$
External Radius (fixed)	b	0.25

Table 5.12 *Pressure Vessel Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Inner radius	a	$a_{\text{init}}^{\text{min}}$	0.01	Δa_{init}	0.23	0.24	0.01

Table 5.13 *Pressure Vessel Stress Characteristics*

	σ_t	σ_r	σ_l	σ_{von}
Control point & straddle	$-2P_o\frac{b^2}{b^2-a^2}$	0	0	σ_t
Interior point	$-P_o\frac{b^2\left(1+\frac{a^2}{(b-EL)^2}\right)}{b^2-a^2}$	$-P_o\frac{b^2\left(1-\frac{a^2}{(b-EL)^2}\right)}{b^2-a^2}$	0	$\sqrt{\sigma_t^2-\sigma_t\sigma_r+\sigma_r^2}$

5.3.3 Torsion Solid Rod

The Torsion Solid Rod, labeled *TSR*, provides the third cluster of designs. The torsion solid rod possesses the following features:

- sizing problem,
- closed form solution,
- known optimal solution,
- von Mises stress an inverse cubed function of the design variable.

Figure 5.4 defines the torsion solid rod design optimization problem, where the torsion solid rod always remains circular. The main educational characteristic is SPHINcsX will be taught to handle problems where the von Mises stress varies as an inverse cubed function of the design variable. For the circular torsion solid rod the shear stress, τ, is a function of the torque, T, radius, r, and the polar moment of inertia, J, as calculated by the equation;

$$\tau = \frac{T * r}{J}.$$

Where for a solid round section,

$$J = \frac{\pi * d^4}{32} = \frac{\pi * r^4}{2},$$

and therefore,

$$\tau = \frac{T * r}{\frac{\pi * r^4}{2}} = \frac{2 * T}{\pi r^3}. \qquad (5.6)$$

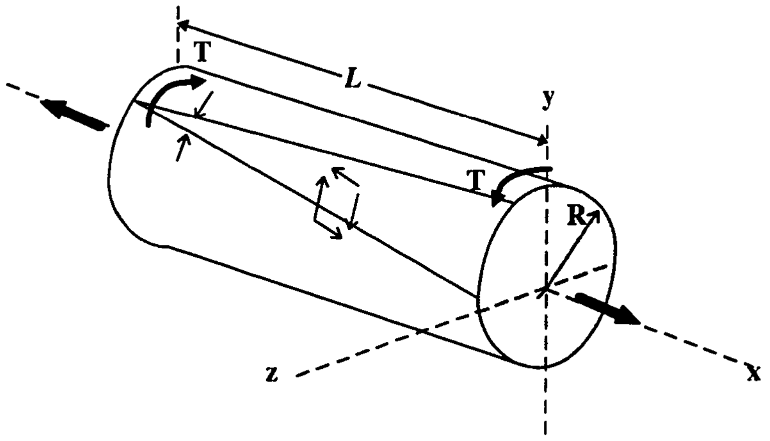


Figure 5.4 ***Torsion Solid Rod***

Table 5.14 ***Torsion Solid Rod Problem Parameters***

Name	Label	Value
Torque	T	1.0 Newton-Meter
Length	L	0.1 meters
Maximum Allowable von Mises Stress	σ_o	$10 * 10^6$ Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.002 meters
Mass	M	πR^2

Table 5.15 *Torsion Solid Rod Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Radius	R	$R_{\text{init}}^{\text{min}}$	0.015	ΔR_{init}	0.015	0.045	0.001

Table 5.16 *Torsion Solid Rod Stress Characteristics*

Stress Point	τ	von Mises Stress
Control Point 'R' (Rod Exterior)	$\frac{2T}{\pi R^3}$	$\sqrt{3\tau^2}$
Interior point	$\frac{2T(R-EL)}{\pi R^4}$	$\sqrt{3\tau^2}$
Prominent straddle point	$\frac{2T}{\pi R^3}$	$\sqrt{3\tau^2}$

5.3.4 Cantilever Beam

The Cantilever Beam, *CB*, provides the fourth cluster of designs. The main educational difference provided is the bending load the cantilever beam is exposed to, other properties of the CB are:

- straddle point stresses are different from the control point stress,
- interior point stress is different from the control point stress,
- stress varies along the boundary,
- location of the maximum stress varies as the design variable varies.

Figure 5.5 shows the cantilever beam design.

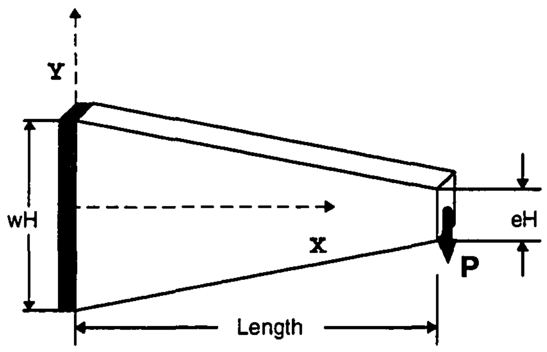


Figure 5.5 *Cantilever Beam*

The problem, as posed has one design variable, *eH*, (*wH* is fixed); therefore the goal is then to size *eH* as small as possible without violating the maximum allowable von Mises stress on the exterior boundary (or internally). The boundaries that are effected by changes in *eH* do not change shape, however the stress is not constant along the boundaries as was the case in the previous clusters. The largest stress always occurs on the top and bottom boundaries. Due to symmetry only the top boundary will be considered. The tensile stress along the top boundary of a cantilever beam loaded by a force, *P*, as shown in Figure 5.5 is given by Equation 5.7.

$$\sigma_x = \frac{P*(L-x)*y}{I}. \tag{5.7}$$

Where y is the distance from the center line to the top boundary at the corresponding value of x, L is the length of the beam and I is the moment of inertia. Assuming a unit thickness for the cantilever beam, the moment of inertia is,

$$I = \frac{2}{3}y^3. \tag{5.8}$$

Substituting Equation 5.8 into Equation 5.7 yields,

$$\sigma_x = \frac{P*(L-x)*y}{\frac{2}{3}y^3} = \frac{3}{2}*P*\frac{L-x}{y^2}. \tag{5.9}$$

The value of y can be written as a function of x as,

$$y = \frac{wH}{2} - \frac{x}{L}\left(\frac{wH - eH}{2}\right). \qquad (5.10)$$

Replacing y in Equation 5.9 yields,

$$\sigma_x = \frac{3}{2} * P * \frac{L - x}{\left[\frac{wH}{2} - \frac{x}{L}\left(\frac{wH - eH}{2}\right)\right]^2}. \qquad (5.11)$$

Figure 5.6 displays a plot of this equation showing stress as a function of x and the free-end height, eH. As Figure 5.6 reveals the stress along the boundary is non-monotonic and the maximum value occurs at different x values depending on the free-end height.

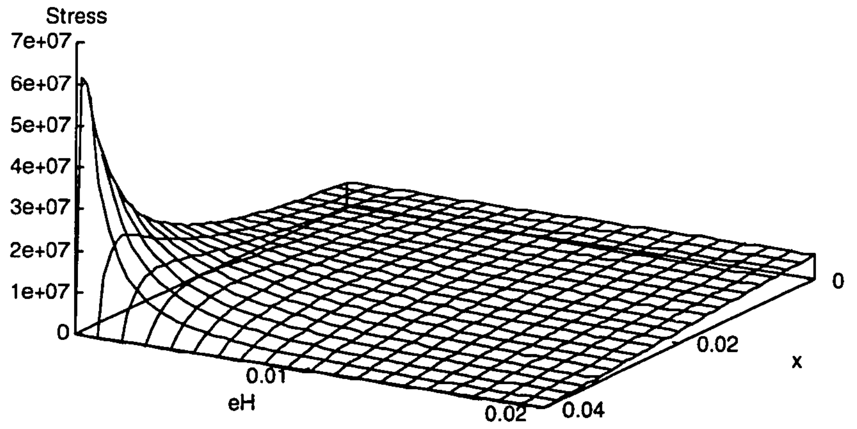


Figure 5.6 ***Cantilever Beam Stress Characteristics***

The maximum stress occurs when the derivative of Equation 5.11 is zero. Differentiating Equation 5.11 yields,

$$\frac{d\sigma_x}{dx} = \frac{3P}{2L} * \frac{(L - x)(wH - eH)}{\left[\frac{wH}{2} - \frac{x}{L}\left(\frac{wH - eH}{2}\right)\right]^3} - \frac{3}{2} * \frac{P}{\left[\frac{wH}{2} - \frac{x}{L}\left(\frac{wH - eH}{2}\right)\right]^2}. \qquad (5.12)$$

Setting the derivative to zero and solving for x yields the location where the maximum stress occurs,

$$x_{\max} = L * \frac{2 * eH - wH}{eH - wH}. \tag{5.13}$$

The value for x_{max} is only between 0 and L for $eH < ½\, wH$. For values of $eH \geq ½\, wH$, the maximum stress occurs at $x = 0$. Substituting the value of x_{max} from Equation 5.13 into Equation 5.10 for y yields the y location for the maximum stress, this simplifies to,

$$y_{\max} = eH. \tag{5.14}$$

Substituting the value of x_{max} from Equation 5.13 into Equation 5.11 yields after some simplification, the maximum stress for $eH < ½\, wH$,

$$\sigma_x^{\max}(eH) = \frac{3}{2} * \frac{L * P}{(wH - eH) * eH}. \tag{5.15}$$

The maximum stress for $eH \geq ½\, wH$ is,

$$\sigma_x^{\max}(eH) = \frac{3}{2} * \frac{L * P}{wH^2}. \tag{5.16}$$

The cantilever beam problem now has an interesting boundary representation. The boundary representation may be thought of as a straight line on the top front edge in Figure 5.6. However, the control point moves along the boundary so as to coincide with the location of the maximum stress. Therefore SPHINcsX must learn under the conditions of this non-conventional boundary representation.

Table 5.17, Table 5.18, and Table 5.19 define the cantilever beam design optimization problem. Table 5.19 includes references to Algorithms 5.4, 5.5 and 5.6.

Table 5.17 *Cantilever Beam Problem Parameters*

Name	Label	Value
Force/thickness	P/t	10,000 Newtons/meter
Length	L	0.04 meters
Height at Wall	wH	0.020 meters
Maximum Allowable von Mises Stress	σ_o	$10*10^6$ Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.001 meters
Mass	M	1/2*(wH + eH) * L

Table 5.18 *Cantilever Beam Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Free End Height	eH	$eH_{init}^{\min}$	0.005	ΔeH_{init}	0.005	0.020	0.0005

Table 5.19 *Cantilever Beam Stress Characteristics*

Stress Point	σ_x	τ	σ_{von}
Control Point 'eH' (Free End)	$\frac{3}{2}*\frac{L*P}{(wH-eH)*eH}$ for $eH<\frac{wH}{2}$ $\frac{6*L*P}{wH^2}$ for $eH\ge\frac{wH}{2}$	0	σ_x
Interior point	$\frac{3}{2}*\frac{L*P}{(wH-eH)*eH^2}(eH-EL)$ for $eH<\frac{wH}{2}$ $\frac{6*L*P}{wH^2}(1-2\frac{EL}{wH})$ for $eH\ge\frac{wH}{2}$	τ from Algorithm 5.4	$\sqrt{\sigma_x^2+3\tau^2}$
Straddle point 1	σ_x^{sp1} from Algorithm 5.5	0	σ_x
Straddle point 2	σ_x^{sp2} from Algorithm 5.6	0	σ_x

Algorithm 5.4: τ **for Interior Point**

```
IF      $y_{max} - EL > 0$
THEN    $\tau = \frac{3}{2} * \frac{P * EL}{(y_{max})^2} * \left(1 + \frac{EL}{2 * y_{max}}\right)$
ELSE    $\tau = \frac{3}{4} * \frac{P}{y_{max}}$
```

Algorithm 5.5: σ_x **for Straddle Point 1**

```
I.    IF      $x_{max} - EL > 0$
      THEN    $x_{sp1} = x_{max} - EL$
      ELSE    $x_{sp1} = 0$
II.   $y_{sp1} = \frac{wH}{2} - \frac{x_{sp1}}{L}\left(\frac{wH - eH}{2}\right)$
III.  $\sigma_x^{sp1} = \frac{3}{2} * P * \frac{L - x_{sp1}}{\left(y_{sp1}\right)^2}$
```

Algorithm 5.6: σ_x **for Straddle Point 2**

```
I.    IF      $x_{max} + EL < L$
      THEN    $x_{sp2} = x_{max} + EL$
      ELSE    $x_{sp2} = L$
II.   $y_{sp2} = \frac{wH}{2} - \frac{x_{sp2}}{L}\left(\frac{wH - eH}{2}\right)$
III.  $\sigma_x^{sp2} = \frac{3}{2} * P * \frac{L - x_{sp2}}{\left(y_{sp2}\right)^2}$
```

5.3.5 Torsion Elliptical Solid Rod

The Torsion Elliptical Solid Rod, *TESR*, provides the fifth cluster of designs. The TESR extends the properties of the TSR by:

- defining the exterior with two design variables.

Figure 5.7 shows the torsion elliptical solid rod design, while Table 5.20 defines the problem parameters. Table 5.21 defines the design variables and Table 5.22 defines the stress characteristics (Roark & Young [1982]) for the torsion elliptical solid rod design optimization problem. The stress calculations for the prominent straddle points and the interior points are only approximate, exposing SPHINcsX to a situation where optimization must be performed with slightly inaccurate stress information.

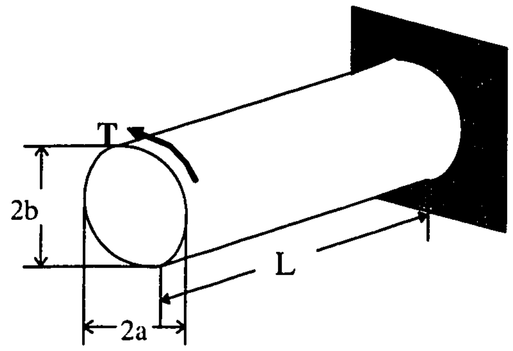


Figure 5.7 *Torsion Elliptical Solid Rod*

Table 5.20 *Torsion Elliptical Solid Rod Problem Parameters*

Name	Label	Value
Torque	T	1.0 Newton-Meter
Length	L	0.1 meters
Maximum Allowable von Mises Stress	σ_o	10 * 10^6 Pascals
Convergence Limit	ε	0.04
Element Length	EL	0.0005 meters
Mass	M	πab

Table 5.21 *Torsion Elliptical Solid Rod Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Major Axis	a	$a_{\text{init}}^{\text{min}}$	0.010	Δa_{init}	0.005	0.015	0.001
Minor Axis	b	$b_{\text{init}}^{\text{max}}$	0.007	Δb_{init}	0.005	0.015	0.001

Table 5.22 *Torsion Elliptical Solid Rod Stress Characteristics*

Stress Point	τ	von Mises Stress
Control Point 'a' (End of major axis)	$\frac{2T}{\pi\ ba^2}$	$\sigma^a_{von} = \sqrt{3\tau^2}$
Interior point	▨	$\frac{a-EL}{a}\sigma^a_{von}$
Prominent straddle point	▨	$\sigma^a_{von} - \left[\frac{EL}{\frac{1}{2}\pi[a+b]}\left(\sigma^a_{von} - \sigma^b_{von}\right)\right]$
Control point 'b' (End of minor axis)	$\frac{2T}{\pi\ ab^2}$	$\sigma^b_{von} = \sqrt{3\tau^2}$
Interior point	▨	$\frac{b-EL}{b}\sigma^b_{von}$
Prominent straddle point	▨	$\sigma^b_{von} - \left[\frac{EL}{\frac{1}{2}\pi[a+b]}\left(\sigma^b_{von} - \sigma^a_{von}\right)\right]$

5.3.6 Torsion Elliptical Hollow Rod

The Torsion Elliptical Hollow Rod, *TEHR*, provides the sixth cluster of designs. The TEHR extends the TESR by:

- defining the interior and exterior boundaries with two design variables each,
- creating a design where local optima exist.

Figure 5.8 shows the torsion elliptical hollow rod design, while Table 5.23 defines the problem parameters, Table 5.24 defines the design variables and Table 5.25 defines the stress characteristics (Roark & Young [1982]) for the torsion elliptical hollow rod design optimization problem. Again the stress calculations for the prominent straddle points and the interior points are only approximate, exposing SPHINcsX to a situation where optimization must be performed with slightly incorrect stress information.

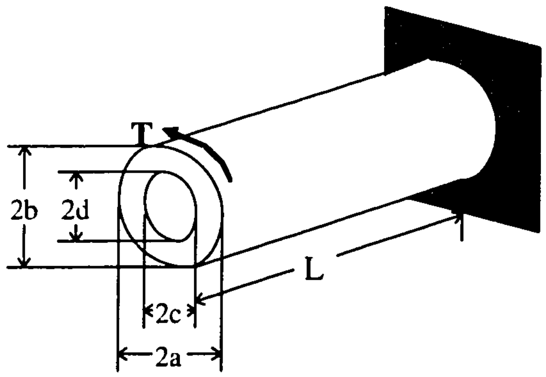


Figure 5.8 *Torsion Elliptical Hollow Rod*

Table 5.23 *Torsion Elliptical Hollow Rod Problem Parameters*

Name	Label	Value
Torque	T	1.0 Newton-Meter
Length	L	0.1 meters
Maximum Allowable von Mises Stress	σ_o	$1*10^6$ Pascals
Convergence Limit	ε	0.05
Element Length	EL	0.0005 meters
Mass	M	π*(a*b - c*d)

Table 5.24 *Torsion Elliptical Hollow Rod Design Variables*

Name	Label	Minimum Initial Value		Range for Initial Value		Upper Limit (m)	Lower Limit (m)
		Label	Value (m)	Label	Value (m)		
Exterior Major Axis	a	$a_{\text{init}}^{\min}$	0.020	Δa_{init}	0.005	0.030	0.015
Exterior Minor Axis	b	$b_{\text{init}}^{\max}$	0.016	Δb_{init}	0.005	0.030	0.015
Interior Major Axis	c	$c_{\text{init}}^{\min}$	0.007	Δc_{init}	0.005	0.015	0.001
Interior Minor Axis	d	$d_{\text{init}}^{\max}$	0.005	Δd_{init}	0.005	0.015	0.001

Table 5.25 *Torsion Elliptical Hollow Rod Stress Characteristics*[†]

Stress Point	τ	von Mises Stress
Control Point 'a'	$\frac{2T}{\pi b a^2\left(1-q^4\right)}$	$\sigma_{von}^{a}=\sqrt{3\tau^2}$
Interior point		$\frac{a-EL}{a}\sigma_{von}^{a}$
Prominent straddle point		$\sigma_{von}^{a}-\left[\frac{EL}{\frac{1}{2}\pi ab}\left(\sigma_{von}^{a}-\sigma_{von}^{b}\right)\right]$
Control Point 'b'	$\frac{2T}{\pi a b^2\left(1-q^4\right)}$	$\sigma_{von}^{b}=\sqrt{3\tau^2}$
Interior point		$\frac{b-EL}{b}\sigma_{von}^{b}$
Prominent straddle point		$\sigma_{von}^{b}-\left[\frac{EL}{\frac{1}{2}\pi ab}\left(\sigma_{von}^{b}-\sigma_{von}^{a}\right)\right]$
Control Point 'c'	$\frac{2Tc}{\pi b a^3\left(1-q^4\right)}$	$\sigma_{von}^{c}=\sqrt{3\tau^2}$
Interior point		$\frac{c-EL}{c}\sigma_{von}^{c}$
Prominent straddle point		$\sigma_{von}^{c}-\left[\frac{EL}{\frac{1}{2}\pi cd}\left(\sigma_{von}^{c}-\sigma_{von}^{d}\right)\right]$
Control Point 'd'	$\frac{2Td}{\pi a b^3(1-q^4)}$	$\sigma_{von}^{d}=\sqrt{3\tau^2}$
Interior point		$\frac{d-EL}{d}\sigma_{von}^{d}$
Prominent straddle point		$\sigma_{von}^{d}-\left[\frac{EL}{\frac{1}{2}\pi cd}\left(\sigma_{von}^{d}-\sigma_{von}^{c}\right)\right]$

† Note: IF $\frac{c}{a}>\frac{a}{b}$, THEN $q=\frac{c}{a}$, ELSE $q=\frac{a}{b}$.

5.4 Summary

This chapter detailed the learning algorithm used by the Shape oPtimization via Hypothesizing Inductive classifier system compleX (SPHINcsX). After which the learning suite of designs used to teach SPHINcsX shape optimization was presented. Chapter 3, Chapter 4 and this chapter fully define the classifier system which is employed by SPHINcsX. Table 5.26 summarizes the classifier system parameters used by SPHINcsX. Table 5.27 summarizes attributes of SPHINcsX, including much of the classifier system's structure.

Table 5.26 ***Compendium of SPHINcsX Parameters***

Reward Coefficient	7.5
Bid Coefficient	0.1
Bid Sigma	0.15
Bid Coefficient 1	0.1
Bid Coefficient 2	0.0833
Effective Bid Coefficient 1	0.1
Effective Bid Coefficient 2	0.0833
Head Tax (also called Life Tax)	0.0028
Bid Tax	0.01
Epoch length (GA period)	250 cycles
Proportion of Population Selected to Breed per Epoch	10%
Probability of Mutation	0.1%
Probability of Crossover	100%
Crowding Factor	100
Crowding Sub-population	150

Table 5.27 ***Compendium of SPHINcsX Attributes***

CS approach	Michigan Approach
Behavior classification	Stimulus-Response (no bucket brigade)
Environment	Structural Shape optimization with stress constraints
Selection	Fitness proportionate reproduction
Pairing of parents	Panmictic
Crossover	Single point crossover
Initial population	*Tabula rasa*
Population size	Static
Replacement & Crowding	Steady state genetic algorithm
Expected learning cycles	< 100,000
Failure criterion	von Mises
Detector range	> 0 and < 2 * σ_0
Detector message format	18 bits long; consisting of 3, 6 bit long messages
Part 1 of message	Prominent straddle stress
Part 2 of message	Control point stress
Part 3 of message	Interior point stress
Stress range	0 to 2 * σ_0 and larger stresses mapped to (2* σ_0)
Specificity	Dependent on # location
User input	Maximum stress Initial design Element length
Direction of control	Determined by boundary representation or surface point modification normal vector
Antecedent length	18
Alphabet of antecedent	{0,1,#}
Number of conditions	3
Condition 1 length	6
Condition 1 message	Match prominent straddle point's von Mises stress
Condition 2 length	6
Condition 2 message	Match control point's von Mises stress
Condition 3 length	6
Condition 3 message	Match interior point's von Mises stress
Consequent Alphabet	{0,1}
Consequent Length	6
Consequent range	111111 : Largest positive relative magnitude 000000 : Largest negative relative magnitude
Positive	Direction away from surface (i.e. along positive surface normal or closest possible via constraint)
Negative	Direction into component (i.e. along negative surface normal or closest possible via constraint)

Chapter

6

Learning Regime for SPHINcsX

This chapter details the mentoring process used to teach SPHINcsX. This process consists of applying the learning suite to SPHINcsX in learning mode. This combination is termed the *learning regime*. The chapter tracks SPHINcsX's learning performance from a *tabula rasa* population to a learned state. Since there is no prior precedent to base performance on, various performance metrics monitor the progress so that criteria for considering a population learned will become self evident.

6.1 Learning Initialization

The SPHINcsX algorithm, as shown in Chapter 5, Algorithm 5.1, begins with the initialization of the classifier system. The classifier system initialization consists of the following:

- Set initial population size & populace,
- Setup Punishment/Reward,
- Set Auction parameters,
- Set taxes,
- Set Genetic Algorithm & Triggered Cover Detector Operator parameters,
- Set learning iteration, *n*, equal to 0,
- Set epoch, *E*, equal to 0.

The SPHINcsX algorithm prescribes all the classifier system initializations except for an actual instantiation of the population of classifiers. The initial populace is constructed as described in Chapter 5, Algorithm 5.2. The *Tabula Rasa Population* used in this work is shown in Appendix A.

Next the initialization of the design to be optimized occurs. A list of the necessary design initializations consists of the following:

- Initial design model,
- Boundary representation definition,
- Global element length,
- Upper & lower limits on dimensions (if any),
- Maximum allowable von Mises stress,
- Set optimization problem iteration, *i*, equal to 0.

The problems in the learning suite, as defined in Chapter 5, Section 3 and the subsequent subsections, include all of these necessary initializations.

The learning regime selects an initial design stochastically from a pool in the learning suite. That is, the learning regime may include provisions so that the selection occurs from a pool which is a subset of the entire learning suite. The selected design is then used in the SPHINcsX algorithm where an attempted optimization proceeds. After the attempted optimization, the process continues with a stochastic selection of another initial design. The optimization is stated as *attempted* because there is a maximum set on the number of optimization iterations that will be allowed before SPHINcsX terminates the optimization process and moves on to another problem. This occurs to prevent SPHINcsX from getting stuck on one particular case.

Initial Design Selection Scheme

The SPHINcsX algorithm does not include the details of selecting the initial designs used for learning. This is the case because of SPHINcsX's flexibility in handling many possible problem suites and still learning efficiently. The initial design selection scheme described here is meant to be but one of many plausible suites.

One of the aspects considered when determining the initial design selection scheme was the need to monitor learning performance throughout the process. To this end, each of the clusters is introduced separately, then after all the clusters have been introduced the initial design selection allows for any initial design to be chosen. Algorithm 6.1 lists the sequence for selecting initial designs.

Algorithm 6.1: Initial Design Selection Scheme

```
I.    For learning iteration 1 through $t_I$
        1. Select the *Tension Rod* cluster
        2. Set the number of design variables to 5
        3. Continue to **Design Variable Initial Value(s) Procedure** (defined below)
II.   For learning iteration $t_I+1$ through $t_{II}$
        1. Select the *Pressure Vessel* cluster
        2. Continue to **Design Variable Initial Value(s) Procedure** (defined below)
III.  For learning iteration $t_{II}+1$ through $t_{III}$
        1. Select the *Torsion Solid Rod* cluster
        2. Continue to **Design Variable Initial Value(s) Procedure** (defined below)
IV.   For learning iteration $t_{III}+1$ through $t_{IV}$
        1. Select the *Cantilever Beam* cluster
        2. Continue to **Design Variable Initial Value(s) Procedure** (defined below)
V.    For learning iteration $t_{IV}+1$ through $t_V$
        1. Select the *Torsion Elliptical Solid Rod* cluster
        2. Continue to **Design Variable Initial Value(s) Procedure** (defined below)
VI.   For learning iteration $t_V+1$ through $t_{VI}$
        1. Select the *Torsion Elliptical Hollow Rod* cluster
        2. Continue to **Design Variable Initial Value(s) Procedure**
VII.  For learning iteration greater than $t_{VI}+1$
        1. Randomly select one of the clusters
        2. Continue to **Design Variable Initial Value(s) Procedure**

                **Design Variable Initial Value(s) Procedure**

      For each design variable
        1. Randomly select a number between 0 & 1 inclusive
        2. Calculate the initial value of the design variable (DV):
           $DV_{\text{init}} = DV_{\text{init}}^{\min} + \Delta DV * rand(0,1)$
```

6.2 Implementation of Learning Regime

With a method now established for initializing the classifier system and for initializing the designs in the learning suite, the learning regime may be implemented.

The extensive computational requirements mandate an automated software implementation.

SPHINcsX's algorithm, in conjunction with the learning suite, is implemented in the SPHINcsX software system. Because of the multi-faceted nature of SPHINcsX, the software implementation consists of many sub-systems including:

- Classifier System,
- Genetic Algorithm,
- Analysis module for closed-form solvable designs,
- Open Link interface to I-DEAS™ for finite element based analysis module,
- Microsoft (MS) Windows graphical user interface front-end,
- Graphics package,
- Graphical classifier viewer.

During SPHINcsX's development, many problems, both shaping and sizing, were employed to test its learning capabilities (Richards [1992]). One result was that *generalized learning could be achieved via either sizing or shaping problems as long as enough different stress patterns were presented.* The learning regime utilizes only problems solvable using closed-formed solutions. The I-DEAS™ analysis module is employed in application mode to test the learned classifier system on more complex geometries.

The learning phase is computationally expensive, requiring many thousands of cycles before valuable learning is expected to be expressed (see Chapter 3, Section 3.6.1). This should be expected by such a general learning technique which is not given domain specific information. SPHINcsX is also handicapped by starting with a random set of classifiers. Note, however, the incredible overall savings in computational resources if SPHINcsX can be mentored to optimize a broad scope of problems (virtually all requiring general analysis techniques like the finite element or boundary element method), utilizing only problems solvable in closed form. It is easy to appreciate the immense savings in resources when a learning iteration only requires the solution of a few equations versus that of performing even a modest finite-element model composed of say 100 nodes. Recall, that after completing learning, the use of SPHINcsX on design optimization problems should be no more computationally expensive than other optimization schemes (if any others are available).

6.3 Learning Performance

The progress of the learning regime is evaluated by monitoring the performance metrics, *P1*, *P2*, *P3* introduced in Chapter 3, Section 3.6. Recall that *P1* is the ratio of the number of correct responses to the total number of responses, while *P2* is the ratio of correct responses during the last epoch to the number of iterations in the epoch, finally *P3* is the number of iterations to reach the optimum. Unfortunately, the field of classifier systems has not developed mechanisms that utilize the performance metrics to declare when the classifier system has learned; the metrics must be monitored for signs or indicators of learning, from which judgments are made. The performance metrics, *P1* and *P2* should show asymptotic improvement with increasing learning iterations, and thus reveal not only the learning progress but also a point where the diminishing returns are so small that continued learning is not justified. The genetic algorithm's purpose is exploration, so the populace after the GA application will most likely be worse because some new members will be provided with artificially high strengths, than the pre-GA populace. Therefore the asymptotic improvements in *P1* and *P2* may not be strictly monotonic.

The learning iterations, (t_I, t_{II}, t_{III}, t_{IV}, t_V, t_{VI}), when new design clusters are introduced in Algorithm 6.1 can not be set *a priori*, but will be determined via the monitoring of the performance metrics. As mentioned above, each introduction of a new design cluster is expected to cause an initial decrease in all the performance metrics.

To set a foundation on performance, the *tabula rasa* population is used to optimize the simplest tension rod, TRI (as defined in Chapter 5, Section 5.3.1), with the initial radius set to $Ri^{\min}_{\text{init}} + \Delta Ri_{\text{init}} = 0.03$ meters (as defined in Table 5.6). A plot of the tension rod's mass and total normalized stress error (TNSE) vs. number of iterations is shown in Figure 6.1. As Figure 6.1 shows, the TNSE never gets close to the optimum, which must be less than ε, which is equal to 0.02 (as set in Table 5.5). The undulations in the mass demonstrate that the modifications are essentially random. An upper limit of 0.3 meters is placed on the radius, which bounds the mass and explains why the mass peaks are all equal. The lower bound on performance is randomness.

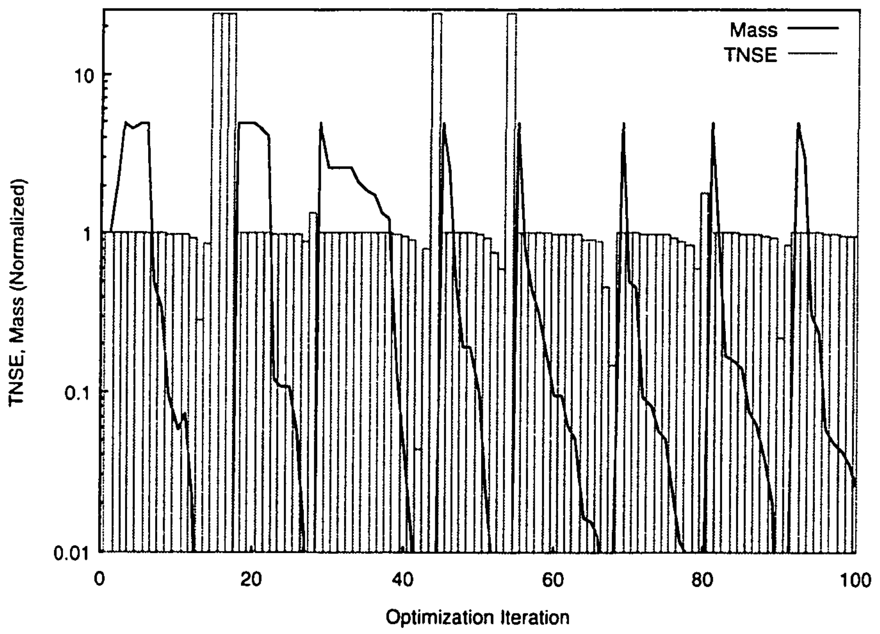


Figure 6.1 ***Performance of Initial Population***

Since all the *tabula rasa* population's members have an initial strength of 10, a strength histogram of the initial population consists of a single bar, as shown in Figure 6.2. The width of the bar is between the strengths of 9 to 11, the histogram groups the strengths into sets with the ranges;

0-1, 1-3, 3-5, 5-7, 7-9, 9-11, 11-13, 13-15, 15-17, 17-19, 19-20,

as described in Chapter 3, Section 3.6.

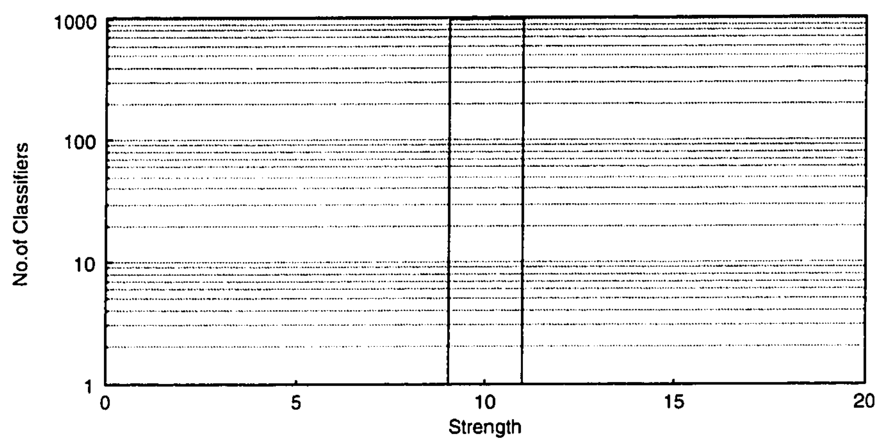


Figure 6.2 *Initial Population Strength Histogram*

The default hierarchy chart for the initial population, shown in Figure 6.3, appears cluttered because all strengths are equal to 10. However, the chart depicts the range of classifier specificities in the initial population.

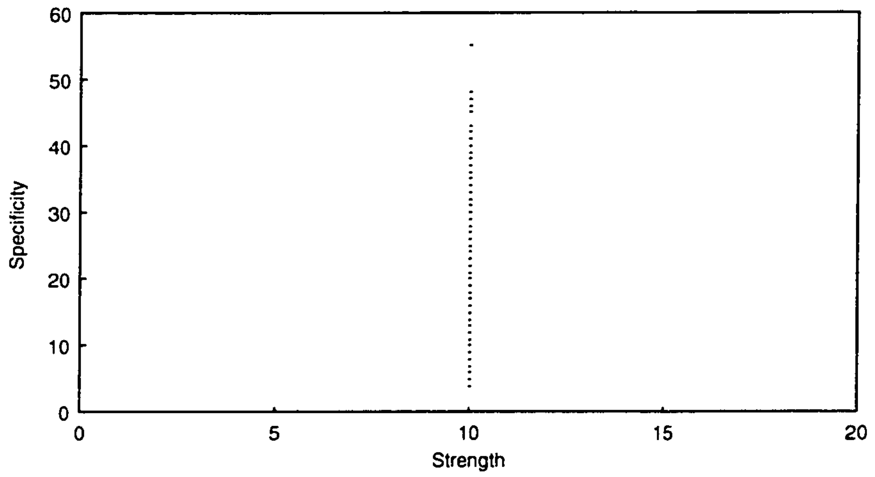


Figure 6.3 *Initial Population Default Hierarchies*

6.3.1 Evolution via Tension Rod Cluster ($0 \leq t \leq 1500$)

Learning commences by selecting a design from the general tension rod cluster, TRg, using 5 different radii. The 5 different radii are stochastically set as described in Chapter 5, Section 5.3.1. The performance above used the simpler case with just 1 radius. For consistency the learning is started again using the tabula rasa population and each design will be created with 5 different radii.

To determine when the productivity of mentoring via the tension rod designs has decreased to a level where the next cluster should be introduced, the performance metrics *P1*, *P2*, and *P3* are monitored. The information provided by the performance metrics must be examined in tandem to assess when learning has occurred to the level where continued learning with the same cluster of designs is inefficient. Figure 6.4 displays the performance metric, *P1*, for learning iteration 0 through 1500.

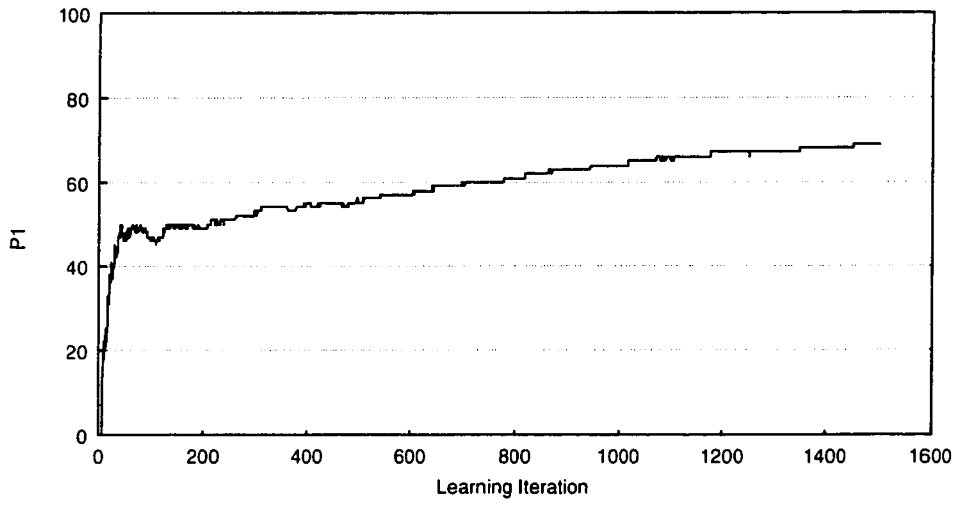


Figure 6.4 ***Tension Rod Cluster Correct Response Rate***

Figure 6.5 displays the performance metric, *P2*, for learning iterations 0 through 1500. The bars represent the percentage of correct answers during the epoch, (an epoch is 150 learning iterations). Note that if a random decision was made on which direction to move a design point, *P2* would be expected to approach 50%. Observe that *P2* is near 50% during the first epoch, an expected result considering the *tabula rasa* population. After six epochs, *P2* has reached nearly a 80% correct response rate. *P2* rate is not expected to ever reach 100% because the GA is applied every epoch and introduces new classifiers, some of which will invariably make incorrect decisions.

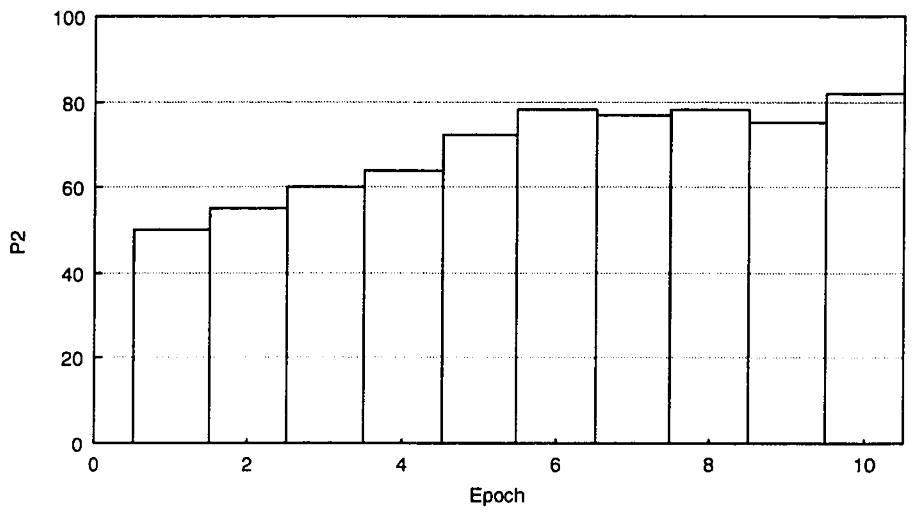


Figure 6.5 ***Tension Rod Cluster Correct Response Rate per Epoch***

Figure 6.6 displays the performance metric, *P3*, for learning iteration 0 through 1500. Recall that the maximum allowable number of iterations for performing the optimization is set to 100. Because of this, and the fact that designs were selected with stochastically varied initial design variables, the graph should be read by observing the density of successful optimizations. Figure 6.6 shows that a successful optimization could not be performed until almost 400 learning iterations occurred. In addition, it took SPHINcsX until over 800 learning iterations before it had learned enough so that it could consistently optimize the tension rod in less than 100 iterations. The variation in the number of iterations to optimize a design is to be expected because each design has different initial values and because performance suffers (probabilistically) initially after the application of the genetic algorithm.

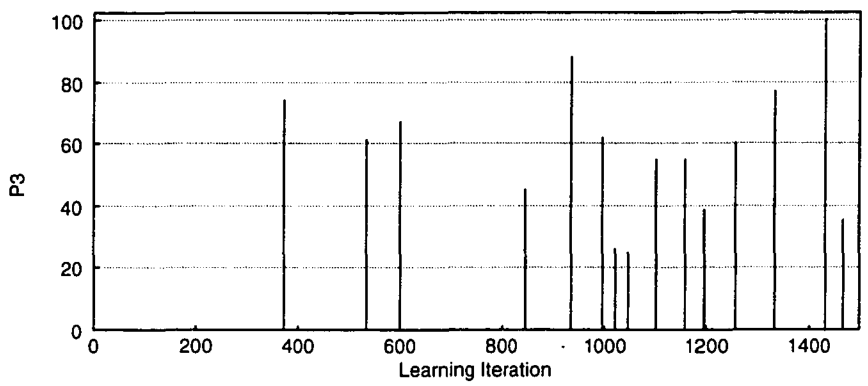


Figure 6.6 *Iterations to Optimum for Tension Rod Designs*

The number of learning iterations required to reach an evolutionary stage where the productivity of mentoring via the tension rod cluster plateaued is not known *a priori*. Since the learning process involves stochastic operations, many simulations were run and the evolution was monitored via the performance metrics to determine the number of learning iterations to perform on the tension rod cluster before transitioning to the next cluster. The figures above show the evolution for the simulation used to create the population which was then used for further learning.

6.3.2 Evolution at 5,000 Learning Iterations: 33 Epochs

After 5,000 learning iterations, the initial design selection scheme has progressed through the TRg cluster, PV cluster, TSR cluster, CB cluster, TESR cluster, and TEHR cluster. Each of the clusters went through an evolution similar to that described for the torsion rod cluster. Figure 6.7 displays the performance metrics *P1* through the entire 5,000 learning iterations, while Figure 6.8 displays *P2* over the same 5,000 learning iterations. The vertical lines in both figures represent the introduction new clusters. The number of learning iterations for each cluster was determined by monitoring the performance metrics. From this monitoring the tension rod cluster was used for learning between iteration 0 through 1,500 as discussed above, next the pressure vessel cluster was used until iteration 2,000. Next the torsion rod cluster was used until iteration 2,800, followed by the cantilever beam until iteration 3,500. The torsion elliptical solid rod was used until iteration 4,400, after which the torsion elliptical hollow rod concluded until iteration 5,000. Observations regarding the learning process as it progressed through the clusters are provided below.

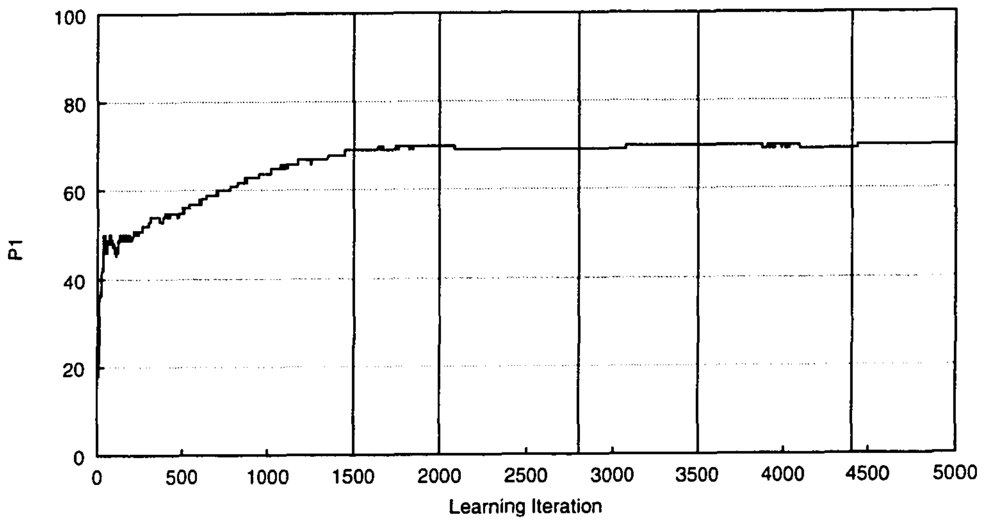


Figure 6.7 ***Correct Response Rate through 5,000 Iterations***

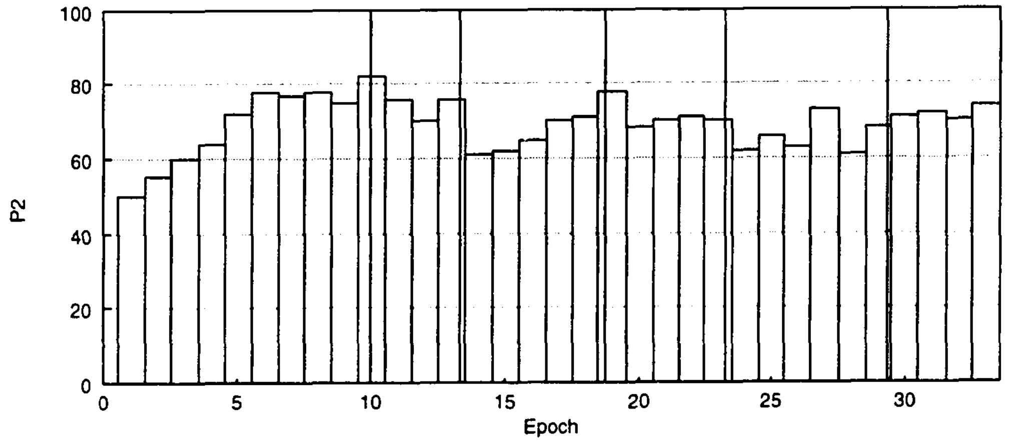


Figure 6.8 ***Correct Response Rate per Epoch through 33 Epochs***

Figure 6.9 displays the performance metric *P3* for the tension rod, pressure vessel, torsion solid rod, cantilever beam, torsion elliptical solid rod and torsion elliptical hollow rod designs. Recall that the maximum allowable number of iterations for performing the optimization is set to 100. Again, the variation in the number of iterations to optimize a design is to be expected because each design has different initial values and performance (probabilistically) suffers initially after the application of the genetic algorithm.

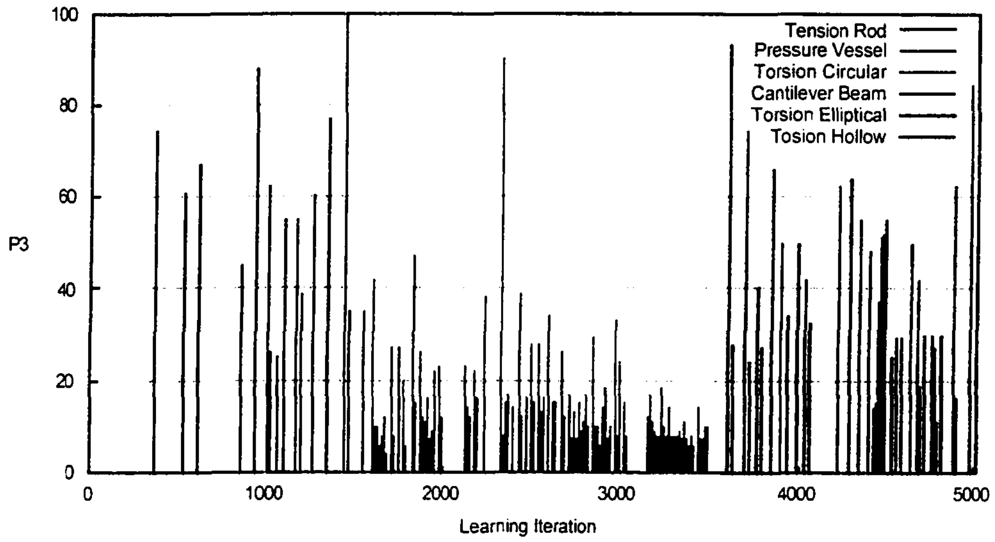


Figure 6.9 ***Iterations to Optimum for Learning Suite Designs***

The second design used in the mentoring process, the pressure vessel, did not appear to present much of a challenge. For even initially, SPHINcsX consistently optimized various initial configurations. Therefore after only 500 learning iterations it appears appropriate to move on to another design.

The third cluster of designs, the torsion rod, provided a greater challenge. By watching the learning, what appeared to occur was that general rules that worked previously for the tension rod and pressure vessel clusters were too general and were not appropriate for the new stress states generated in the torsion rod designs. SPHINcsX eventually culled the inappropriate general classifiers while leaving appropriate general classifiers as well as appropriate specific ones.

The learning prepared SPHINcsX well for the cantilever beam cluster. The problem was always optimized within the 100 iteration limit, except for one case. Even after the conspicuous failure from learning iteration 3,038 to 3,138, SPHINcsX regained its stride, optimizing the rest of the cantilever beam problems in under 20 iterations each. Learning ceased for this design cluster after 700 iterations.

The torsion elliptical solid rod cluster provided SPHINcsX with a multiple control point environment where dimension changes at one design point affected the stress at the other design point. Previous experience proved helpful, as the drop in the performance metric *P2* reveals, dropping to a correct response rate of almost 60%. SPHINcsX's overall performance improved during the exposure to the cluster with *P2* rising back into the 70% correct range by 4,400 learning iterations. With the rising performance SPHINcsX became able to optimize the final set of designs in under 100 iterations.

SPHINcsX dealt with the torsion elliptical hollow rod cluster with remarkable adeptness. Even though each modification at any of the four design points affected the stresses at all the others, SPHINcsX optimized every one of the designs presented to it. Because of this, and because there was no significant drop in *P2*, learning was concluded after 5,000 iterations.

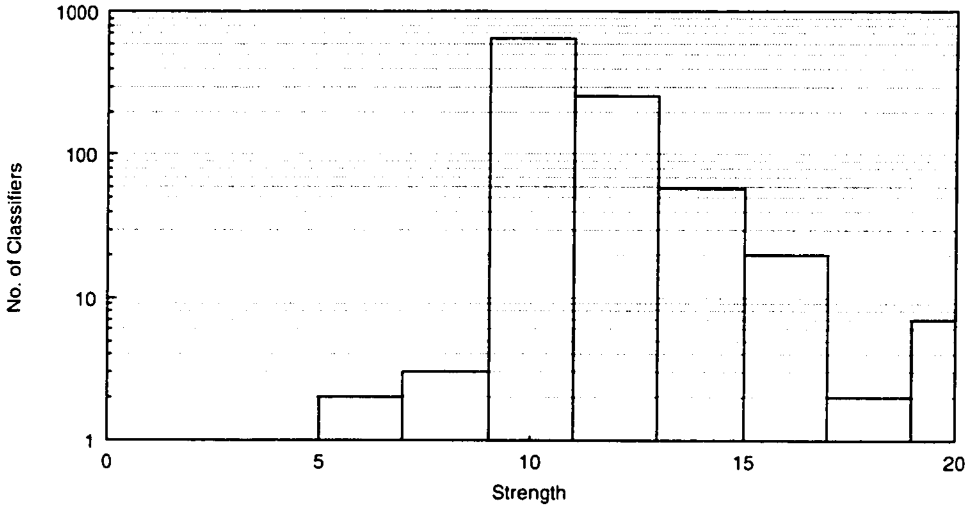


Figure 6.10 ***Strength Histogram after 5,000 Learning Iterations***

Figure 6.10 displays the strength histogram for the classifier population after 5,000 learning iterations. The histogram reveals an important result of the genetic algorithm application. During an epoch, the classifiers which make incorrect decisions have their strength reduced, resulting in strength values which may fall well below the population mean of 10. However after the genetic algorithm is applied, the newly generated classifiers must replace present members in the population so to keep the population size static. The crowding and replacement is performed (as described in Chapter 3, Section 3.3) which virtually eliminates all population members with strengths significantly below 10 at each genetic algorithm application. The histogram in Figure 6.10 may imply, by the number of population members above the mean strength bar, that the productive sub-population consists of approximately 300 classifiers. However it should be recalled that the population is having potentially good classifiers introduced at every genetic algorithm application, which also removes the inferior classifiers; therefore as learning continues the population should consist more and more of good classifiers and the strengths should represent the ranking of these good classifiers.

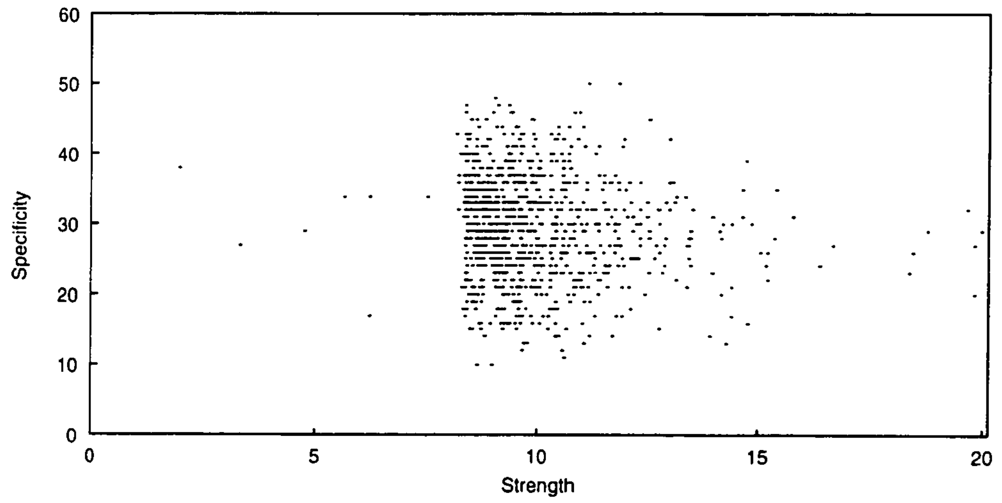


Figure 6.11 ***Default Hierarchy for Population after 5,000 Learning Iterations***

Figure 6.11 displays the default hierarchy plot for the population at this stage, (as contrasted with the initial population's default hierarchy shown in Figure 6.3). The default hierarchies fall in hierarchies of specificity ranging from about 10 for the most general to nearly 50 for the most specific. The default hierarchies have contracted partially compared to the original population which had population members with specificities below 10. This reveals that there is a limit of how general a classifier could be and still cause correct responses.

Figure 6.12 provides a view of the optimization process for one of the stochastically generated initial configurations of the elliptical solid rod under torsion loading, the optimization occurred after approximately 3,800 learning iterations. The mass value is normalized using the initial mass of the design, therefore the mass line provides the fraction of mass the design has as compared to the initial design. The TNSE values vs. optimization iteration number portrays a generally decreasing TNSE condition. A TNSE of zero represents a fully stressed boundary (Hsu [1992]). During learning, the design may become better or worse at any iteration. Regardless of the outcome, the generated (better or worse) design is used for further learning.

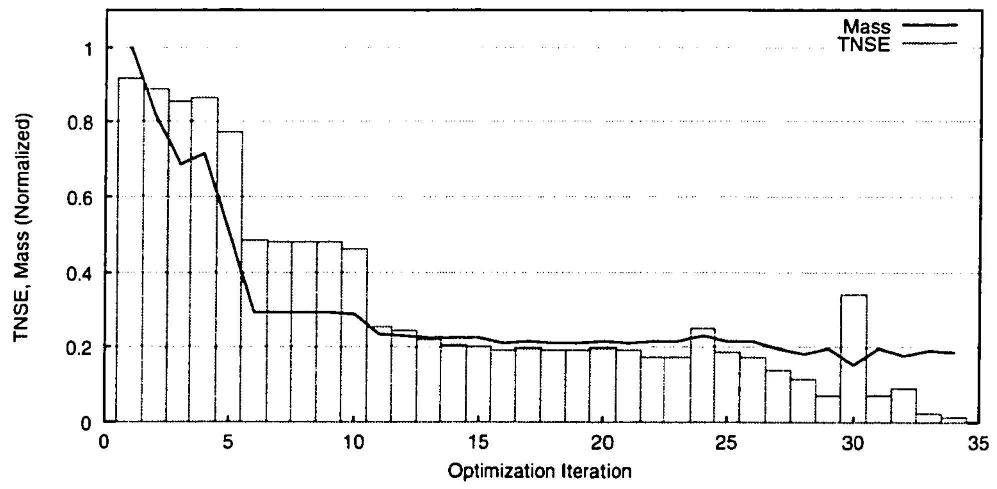


Figure 6.12 ***Optimization of an Torsion Elliptical Rod Problem***

Figure 6.13 displays the values of the design variables vs. optimization iteration for the same problems from the torsion elliptical solid rod cluster. Notice the range of dimension values spanned — the initial design had a major axis value near 0.012 and a minor axis near 0.009. The optimum occurs when both axes equal 0.0048. Therefore, the optimization commenced with an initial design over 5 times heavier than the optimum's. Figure 6.14 juxtaposes the initial design with the optimized design, the optimization transforms a highly overdesigned elliptical rod into an optimal circular rod.

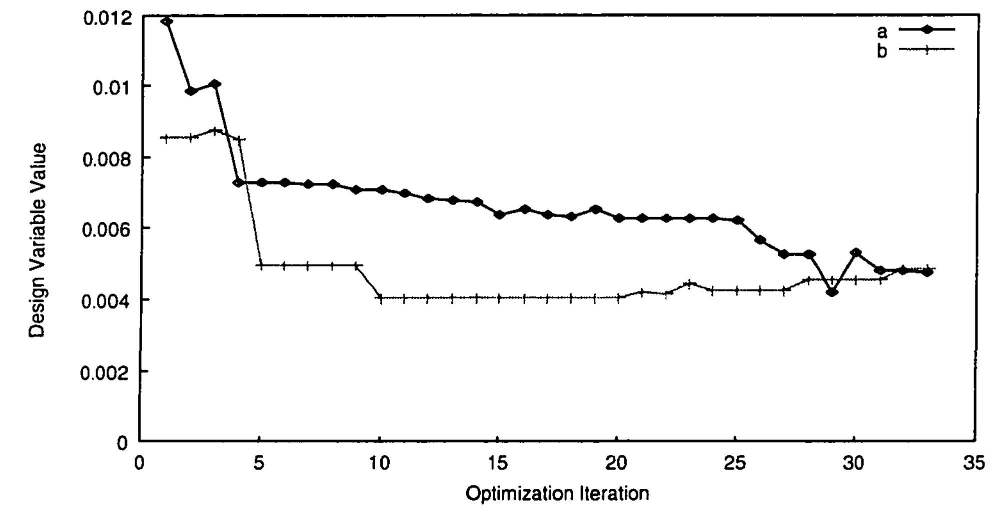


Figure 6.13 ***Dimension Values through the Optimization Process for Torsion Elliptical Rod***

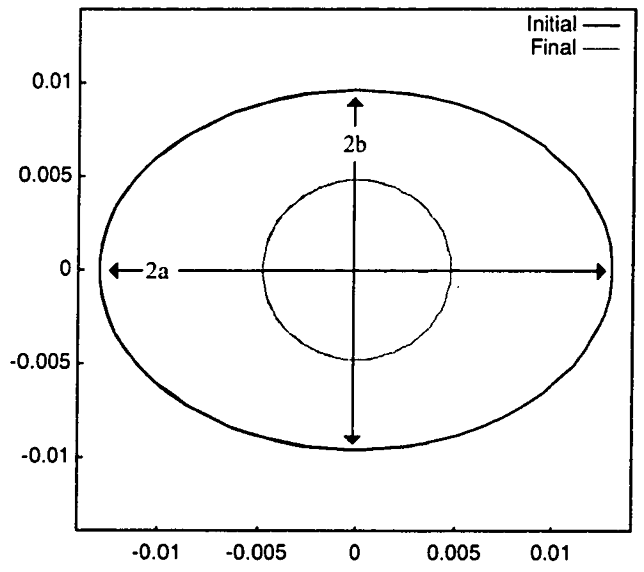


Figure 6.14 ***Initial and Optimized Shape of Torsion Elliptical Rod***

It is little wonder that SPHINcsX was not always able to fully optimize the design within the 100 iteration limitation; it is quite an achievement, actually, that SPHINcsX could optimize some of the initial designs from this cluster. Recall from Chapter 2, Section 2.5.1 that most state-of-the-art optimization methodologies deal with initial designs that are within a factor of two of the optimal mass design.

6.3.3 Learning Supervision & Order Dependencies

The learning described above presents a supervised learning regime with a defined ordering of the design cluster introductions. As mentioned in Section 6.1, SPHINcsX could be trained with many initial design selection schemes. A selection scheme could be devised so that SPHINcsX started by selecting a design from all designs in the learning suite, creating an unsupervised learning regime. In such a case there would be no ordering in the introduction of the design clusters since their introduction would be stochastically decided.

This flexibility in the learning regime is not speculation. During this research many learning regimes were tested, one of which was used in Richards & Sheppard [1992]. Again as mentioned, a major consideration in using the learning regime described in this chapter was the need to monitor learning performance throughout the process.

Just as ranking the feedback can improve the learning performance, supervised learning may prove more efficient than unsupervised learning, however, too much supervision or constraining may also restrict learning. The ordered introduction of design clusters did provide a generally (though not monotonic) increasing level of performance.

It is difficult to track the value of the knowledge gained in the earliest stages of the learning regime because valuable knowledge is usually propagated as valuable sub-units of classifiers (schemata) not as entire classifiers. However, it is interesting to note that out of the population of 1,000 classifiers which existed at the end of the tension rod cluster evolution (learning iteration 1,500); 30 still remained at the completion of the learning regime. Out of these 30, 7 remained above average strength classifiers throughout the learning regime, providing more evidence to support the results above exhibiting the general nature of SPHINcsX's learning.

6.4 Summary

The learning regime used to teach SPHINcsX, applied the learning suite of problems defined in Chapter 5 to SPHINcsX in learning mode. The learning regime presented a stochastically generated initial design from one of the design clusters in the learning suite to SPHINcsX for optimization. SPHINcsX attempted to optimize the design within a maximum of 100 iterations. After a successful optimization or after 100 iterations, whichever came first, another initial design was generated and presented to SPHINcsX. Whenever an epoch (150 learning iterations) occurred, the genetic algorithm was applied providing a mechanism for exploring the space of potentially better hypotheses from the current population. Table 6.1 summarizes some of the aspects of the learning process, showing the ranges of learning iterations applied to each design cluster as well as presenting the level of the performance metrics *P1* and *P2* at the end of each cluster's learning phase. Recall from Section 6.3.1 that the ranges of learning iterations applied to each design cluster were determined empirically by monitoring *P1 P2* and *P3*.

Table 6.1 ***Learning Summary***

Design Cluster	Iteration (Start)	Iteration (End)	P1	P2
TRg (n=5)	1	1,500 (t_I)	69	82
PV	1,501	2,000 (t_{II})	70	76
TSR	2,001	2,800 (t_{III})	69	71
CB	2,801	3,500 (t_{IV})	70	70
TESR	3,501	4,400 (t_V)	69	71
TEHR	4,401	5,000 (t_{VI})	70	74

SPHINcsX rapidly evolved the initial *tabula rasa* population into one that allowed it to perform significantly better than the *tabula rasa* after only a few epochs. Evidence of the generality of the sub-population of above average classifiers (strength > 10) was their low specificity, and more importantly their ability to handle new clusters of problems with significantly better than random performance. Each new cluster of problems exposed SPHINcsX to situations where the above average strength general classifiers could be tested to verify that their generality was appropriate for a wider array of shape optimization situations. In the cases where general classifiers failed in new situations, their strengths waned due to continued incorrect responses or exception classifiers covering the situations where the general classifier would have erred.

The choice of problems to include in the learning suite was influenced by many factors, the primary being the level of problem difficulty and diversity necessary to exercise SPHINcsX into evolving non-problem specific classifiers. As already noted, the learning suite consists of size optimization problems, while an objective is the development of a shape optimization methodology. Only through experience of learning with shape problems (Richards & Sheppard [1992]) and by learning with size problems was it discovered that size problems, if diverse enough, could provide similar levels of learning than their more computationally expensive shape brethren.

The performance levels reached in the learning regime are testament to the learning capability of SPHINcsX and the underlying classifier system. Even though the results of the learning provide ample evidence of SPHINcsX's ability to perform optimization, the learning regime's intention is not proof of success, only a hint. Only by applying SPHINcsX to problems considered more difficult (to humans) and never before seen by SPHINcsX can success or failure be judged. Chapter 7 provides this examination.

Chapter 7 Applications

This chapter tests and applies the now learned SPHINcsX. The chapter opens with a detailed look at the SPHINcsX algorithm in application mode. The major subject matter of this chapter is the application of SPHINcsX (in application mode) to three heretofore unseen problems. That is, to test the generality of SPHINcsX, and its rule set, to solve any problem within a scope, three problems never applied to SPHINcsX in the learning phase are used to test the accomplishment of the learning regime. Accomplishment is measured by determining if SPHINcsX has reached the objective of being able to perform generalized shape optimization on stress constrained designs without auxiliary information.

The learning regime consisted of problems all from the sizing optimization class (see Chapter 2, Section 2.2, Shape Optimization Classes). The three problems used to test SPHINcsX are drawn from the shape optimization class. One of these is a three-dimensional general shape optimization problem intended to provide a rigorous examination. The chapter closes with an review of SPHINcsX's performance, comparing the performance with other modern methods when such are available.

7.1 Application Mode

The classifier system, and hence SPHINcsX, has two major *modes*: the learning mode and application mode, as first described in Chapter 3, Section 3.1. Chapter 6 described the progress of SPHINcsX with its classifier system in learning mode. SPHINcsX learned to operate in the structural component shape optimization environment. In this chapter, SPHINcsX is applied to design optimization problems using its learned rules in application mode. The application mode is a subset of the learning mode; the major difference is that in application mode, the classifier system does not receive feedback from the environment, no strength changes occur, and the genetic algorithm is not applied. *Algorithm 5.1* presented the SPHINcsX algorithm with the classifier system in learning mode. *Algorithm 7.1* provides the basic outline of SPHINcsX in application mode. A more detailed version of the application mode algorithm used by SPHINcsX is provided in Appendix B.

Algorithm 7.1: SPHINcsX Algorithm in Application Mode

```
                 Initialization & Termination Modules
I.    Initialization of Classifier System            (Chapter 5, Section 5.2)
II.   Initialization of a design to be optimized     (Chapter 5, Section 5.3)
III.  Increment                                                   (Chapter 6)
        Set optimization problem iteration, i, equal to i + 1
IV. Continue to Optimization Loop

                          Optimization Loop
I.     Analysis Module:       Iterations: all      (Chapter 2, Section 2.4.2)
II.A.  Detector Interface     Iteration: 1           (Chapter 4, Section 4.3)
        Read:  Mass, Maximum allowable von Mises stress, Limits on dimensions,
               Global element length
  B.   Detector Interface     Iterations: all
        Read:  von Mises stress at: control points, straddle points, interior points
       Convert stresses to their binary representation
       Create environmental messages
III.   Auction module         Iterations: all      (Chapter 3, Section 3.1.2.1)
  1)   Match environmental messages with classifiers.
  2)   IF      no classifiers matched in 1), apply the triggered cover detector
               operator, skip to step III.4
  3)   Auction: Have all the classifiers that matched in 1) compete in an auction to
       determine which one shall be permitted to execute its action.
  4)   Pass the action of the victorious classifier to the effector interface.
IV.    Effector Interface     Iterations: all        (Chapter 4, Section 4.4)
       Modify all control points which matched the victorious classifier.
V.     Termination criteria   Iterations: all        (Chapter 4, Section 4.1)
  1)   IF      all control point stresses are within ε of the optimum
       THEN terminate this design's optimization.
  2)   IF      iteration i is greater than a user supplied maximum (if any)
       THEN    terminate this design's optimization.
  3)   IF      none of the above termination criteria are satisfied
       THEN    continue.
VI.    Set the active design to the design created in step IV
       Set i = i+1
       Return to step I.
```

7.2 Applications

The three design optimization problems presented below provide a test of SPHINcsX's shape optimization performance. Recall, as discussed in Chapter 2, that shape optimization performance consists of:

- scope,
- efficiency,
- effectiveness.

The sub-sections below concern themselves primarily with discovering if the learned SPHINcsX has learned to solve problems within the scope exemplified by the applications, which is two and three-dimensional shape optimization. With the design scope established, Section 7.3 conducts further performance analysis of SPHINcsX's efficiency and effectiveness.

The following provides a brief description of the three test problems:

1. Cantilever beam: Generalized symmetrical cantilever beam design with a modifiable boundary defined by four moveable control points connected by piecewise cubic splines.
2. Torque arm: Symmetrically shaped torque arm under bending and tensile load with a modifiable boundary defined by fixed end conditions and six moveable control points connected by piecewise cubic splines.
3. Spherical pressure vessel: General three-dimensional pressure vessel with a spherical exterior under constant pressure loading and an irregular interior void defined by a three-dimensional surface patch. The three-dimensional surface patch is to be optimized.

As can be seen, the level of difficulty increases from the cantilever beam to the pressure vessel.

7.2.1 Shape Optimization: Cantilever Beam

The problem of optimizing a cantilever beam was used in the learning regime. The learning regime design, however, was simpler than the design considered here. The learning cantilever beam design was a sizing optimization problem where the shape of all the boundaries remained constant (i.e. straight line boundaries remained straight lines).

The goal of the current problem is to find the shape of the lightest cantilever beam of fixed length, *L*, by varying the shape of the boundary between the fixed end and the free end without exceeding the maximum allowable von Mises stress. The beam's shape is invariant through its thickness, *t*. The beam supports a transverse load, *F*, at one end and is fixed to a support at the other end. Figure 7.1 depicts the initial design. Four control points define the modifiable boundary which is composed of piecewise cubic splines. The initial boundary has all control points evenly spaced along a linear boundary between the endpoints. The control points are further constrained to move only along the y-axis, however, this constraint does not restrict the generality of the curve which can be generated. The solution is assumed to be symmetric about the center line along the length of the beam; therefore, the upper and lower boundaries are constrained to be symmetric.

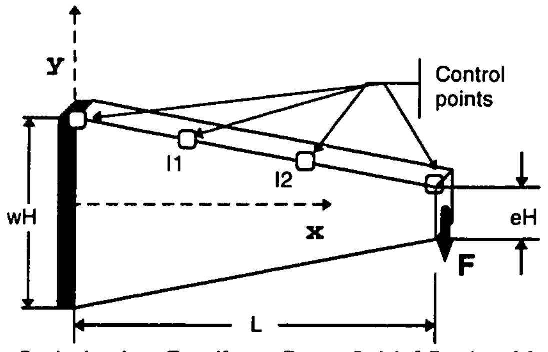


Figure 7.1 ***Shape Optimization Cantilever Beam Initial Design Model***

The load is applied at the free end so that the shearing forces on the free end are distributed according to the same parabolic law as the shear stress, τ, given by beam theory (Timoshenko & Gere [1972]) and shown in Equation 7.1. In addition, the resulting shearing reaction forces at the built-in end are distributed according to the same parabolic law as the shearing stress, given by beam theory; and the intensity of the normal forces at the built-in end is proportional to y, the distance from the neutral axis. The maximum von Mises stress allowed by the material, and an appropriate factor of safety, is

10 MPa. Table 7.1 summarizes the initial design parameters. The initial design has a mass of 58.46 grams (cross-sectional area is, 740.0 mm^2, thickness is 10 mm, and mass density of steel is 0.0079 g/mm^3).

Table 7.1 ***Shape Optimization Cantilever Beam Problem Parameters***

Name	Label	Value
Force/thickness	F/t	10,000 Newtons/m
Length	L	0.04 meters
Height at Wall	wH	0.020 meters (variable)
Height at Free End	eH	0.017 meters (variable)
Maximum Allowable von Mises Stress	σ_0	10*10^6 Pascals
Convergence Limit	ε	0.02
Element Length	EL	0.001 meters
Mass	M	Determined by I-DEAS™

This problem offers the opportunity to compare a discrete shape optimal solution to a known analytical solution. The analytical optimum shape is parabolic and the relationship between the height of the beam (y) and the length (x) is given by Equation 7.1:

$$y = \sqrt{\frac{6*(F/t)*(L-x)}{4*\sigma_o}}, \tag{7.1}$$

where the load per unit width, F/t, equals 10 N/mm; σ_o = 10 MPa; length L = 40 mm. For the parameters considered here, the theoretical minimum mass beam has an area of 413.12 mm^2 (32.64 grams).

SPHINcsX performed the optimization using the I-DEAS™ finite-element analysis module. These results correlated closely to beam theory stress prediction, with the loaded end showing the greatest discrepancy. SPHINcsX produced an optimal discrete solution with a mass of 33.66 grams (426.1 mm^2), a 42% decrease in mass. The discrete solution is only 3.1% more than the analytical solution. SPHINcsX converged to within 2% of a fully stressed boundary after 25 iterations. Only one equivalent finite element analysis was required in each iteration. Recall from Chapter 2, Section 2.5.1 that an equivalent analysis considers the extra overhead of determining auxiliary information such as sensitivities, so if a sensitivity based system solved this problem, each analysis would be greater than one equivalent analysis.

Figure 7.2 compares the final shape with Equation 7.1. The curve generated by SPHINcsX has the same shape as that from the ideal beam theory, differing greatest near the loaded end due to the lower bound placed on the loaded end's height.

Figure 7.3 shows the iteration histories of the performance index TNSE and normalized mass for the beam. Figure 7.4 presents the design variables from the initial design to the optimum, the labels for the control points are as defined in Figure 7.1. It is informative to view Figure 7.3 with Figure 7.4, noting how the TNSE and mass values vary with changes in the design variable values.

As further discussed in Section 7.3, the efficiency of determining the optimum in under 30 iterations is comparable to the state of the art, while the effectiveness of 2% ($\varepsilon = 0.02$) is well within the limits used by most optimization techniques. This initial success betokens the breadth of SPHINcsX's scope.

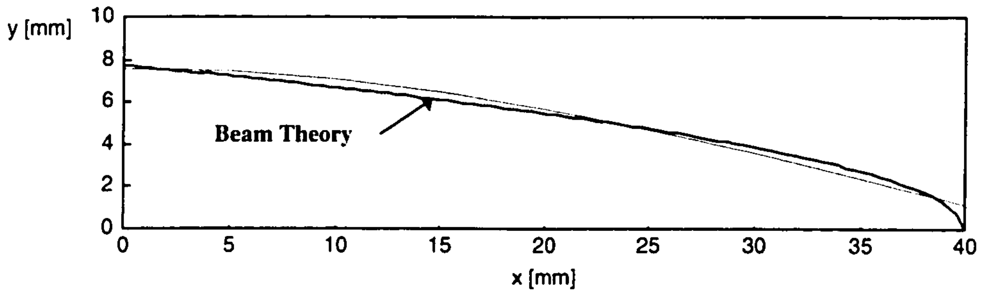


Figure 7.2 ***Cantilever Beam: Comparison of the Final Curve with Beam Theory***

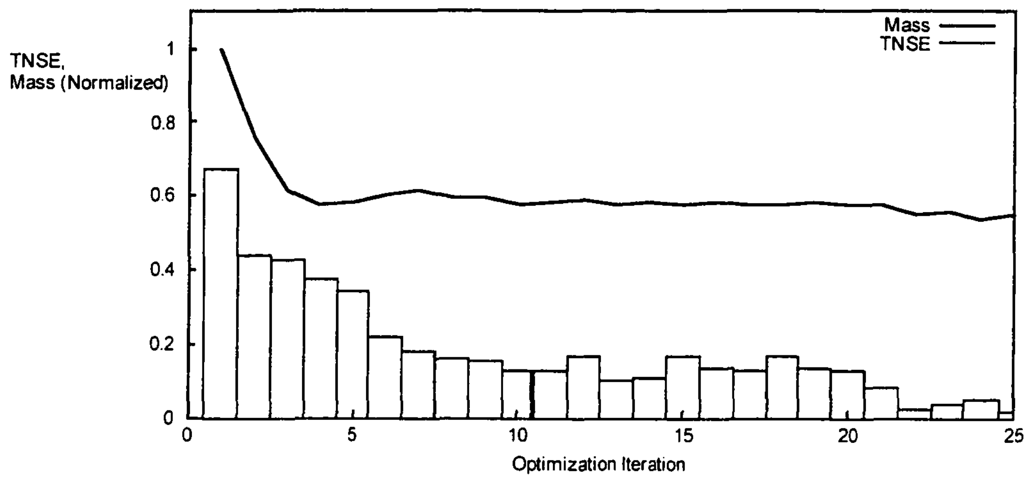


Figure 7.3 ***Optimization of Cantilever Beam***

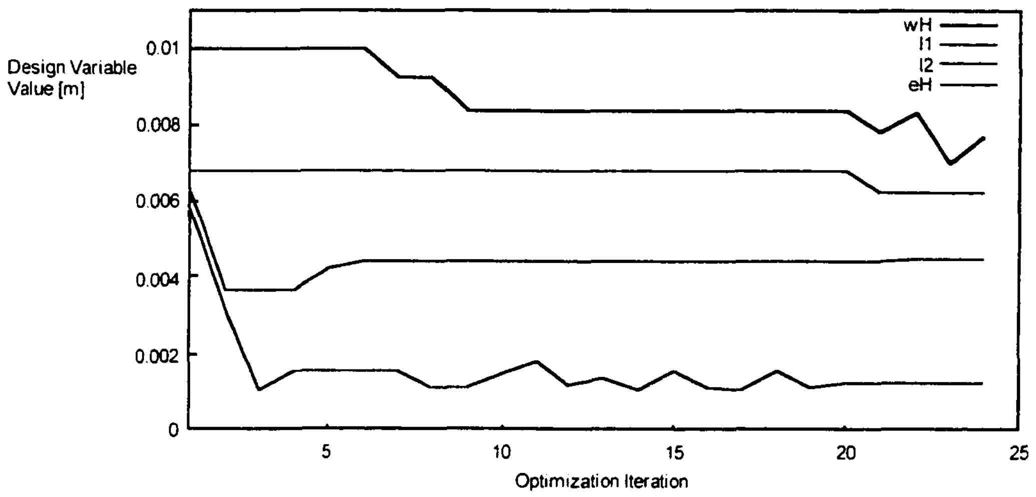


Figure 7.4 ***Cantilever Beam Design Variable History***

7.2.2 Shape Optimization: Torque Arm

Botkin [1982] introduced the torque arm shape optimization problem. Since then many variations have been used to test a multitude of shape optimization methods. A sampling from the literature includes:

- Bennett and Botkin [1985],
- Braibant and Fleury [1984], Braibant and Fleury [1985],
- Yang, Choi and Haug [1985],
- Rajan and Belegundu [1989],
- Hsu [1992],
- Widmann [1994].

The goal of this problem is to find the shape that minimizes the mass of the torque arm by varying only the shape of the boundary between points A and B as shown in Figure 7.5, without exceeding the maximum allowable von Mises stress. The torque arm's shape is invariant through its thickness, *t*, and is loaded by both a transverse load, F_b, and a tensile load, F_t, on the inside of the small hole. In addition, the torque arm is constrained around the circumference of the larger hole located at *x* equal to 0.0416 m. Figure 7.5 depicts the initial design, displaying the fixed dimensions, loading and constraints.

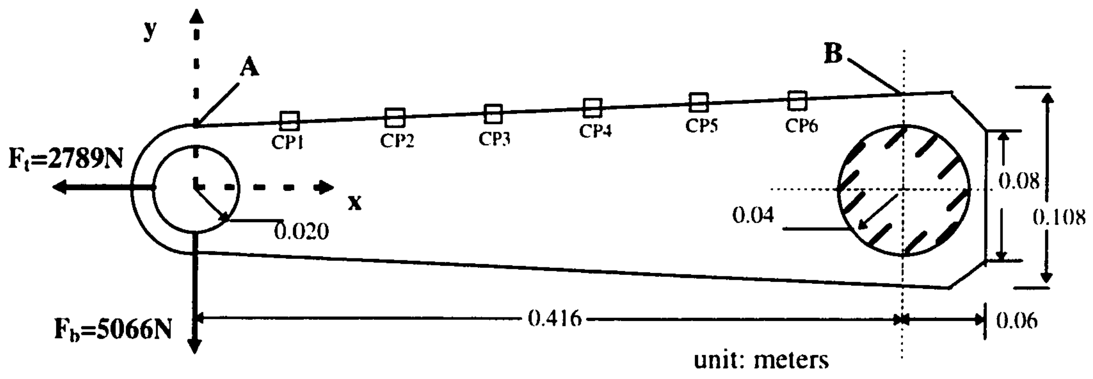


Figure 7.5 ***Initial Configuration and Loading Conditions of the Torque Arm***

The solution is constrained to be symmetric about the center line along the length of the torque arm to account for reversed bending loads. For the loading shown in Figure 7.5, the transverse load places the bottom surface in compression while the tensile load places the bottom surface in tension. In contrast, the top surface experiences tensile stresses as a result of both forces; therefore by superposition, the top surface is the critical surface. Due to the geometric symmetry, and loading conditions, only the top boundary is modeled with design variables. Six control points and the two fixed end points define the top boundary, thus the problem has six design variables. Piecewise cubic splines define the boundary, and the initial boundary has all control points evenly spaced along a linear boundary. The control points, shown in Figure 7.5, are constrained to move only along the y-axis. The thickness of the material is 0.003 m and the maximum allowable von Mises stress is 810 MPa. Table 7.2 summarizes the problem parameters.

Table 7.2 ***Torque Arm Problem Parameters***

Name	Label	Value
Tensile Force	F_t	2789 Newtons
Bending Force	F_b	5066 Newtons
Maximum Allowable von Mises Stress	σ_o	$810*10^6$ Pa
Convergence Limit	ε	0.02
Element Length	EL	0.01 meters
Thickness	t	0.003 meters
Mass	M	Determined by I-DEAS™

Figure 7.6 displays the initial finite element analysis model generated by the I-DEAS™ automatic mesh generator. Point loads of 5066 N in the negative *y* direction and 2789 N in the negative *x* direction are applied to the small hole of the arm as shown. The nodes around the circumference of the large hole are constrained to zero displacement in all six degrees of freedom. The initial analysis model consists of 210 shell quadrilateral type elements. The I-DEAS™ Master Series automatic mesh generator created the mesh. However, before the mesh generation, pseudo-forces are placed at the control points, so the automatic mesh generator creates nodes at those locations.

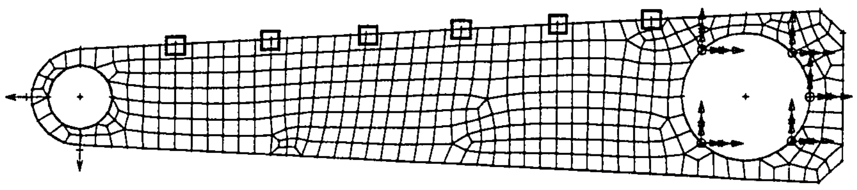


Figure 7.6 ***Initial Analysis Model of the Torque Arm***

SPHINcsX converged to within 2% of a fully stressed boundary after 36 iterations. Only one equivalent finite element analysis was required in each iteration. Figure 7.7 shows the iteration histories of the performance index TNSE and normalized mass for the torque arm. Figure 7.8 presents the design variables showing their progress from the initial design to the optimum. Figure 7.9 shows some of the finite element models during the iterations; the initial mass is, 0.824 kg (mass density of steel is 7,900 kg/m^3). The mass of the final design is 0.540 kg, a 34.5% decrease in mass. This example demonstrated the capability of SPHINcsX in solving a realistic design problem.

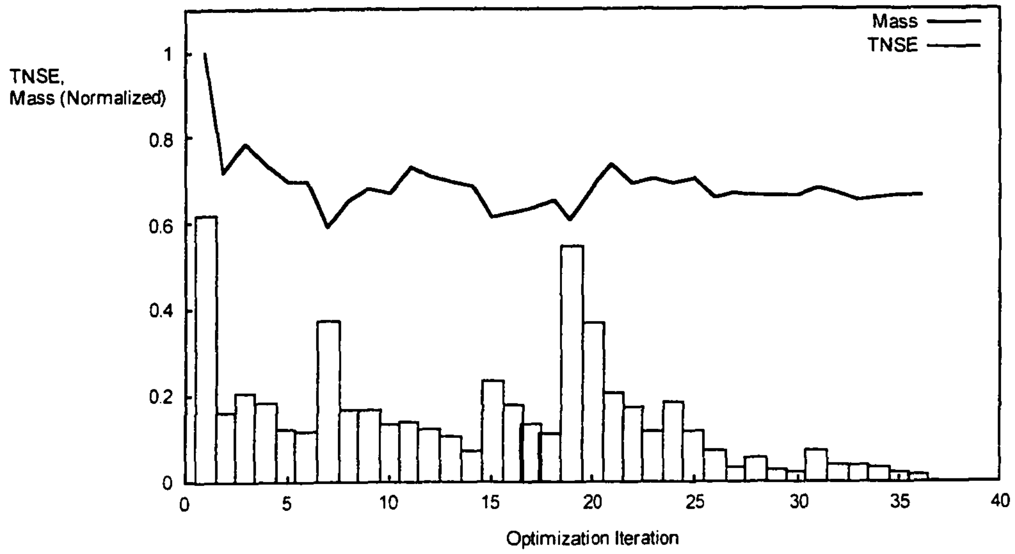


Figure 7.7 *Torque Arm: Iteration History*

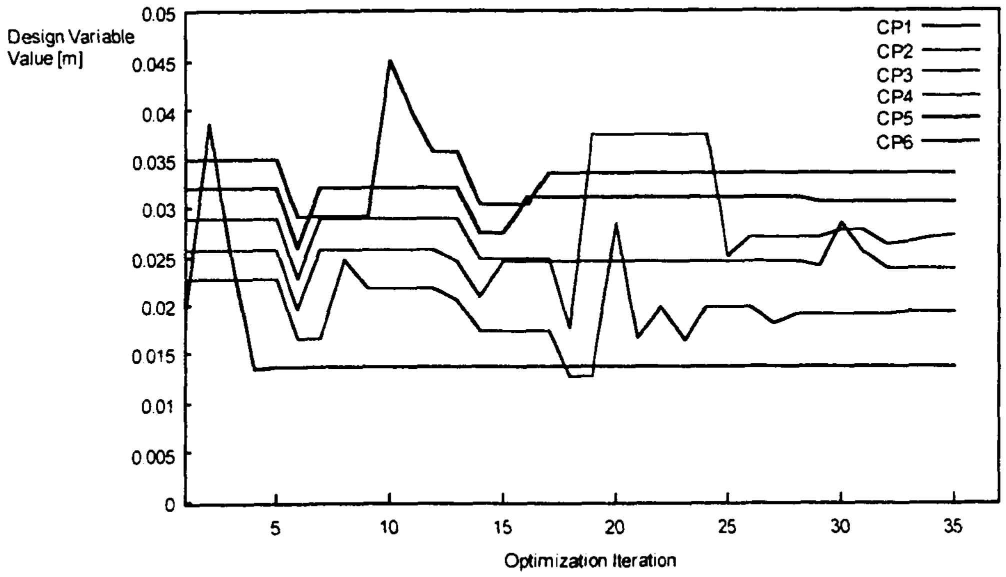
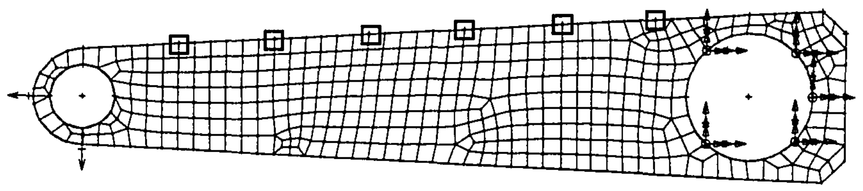
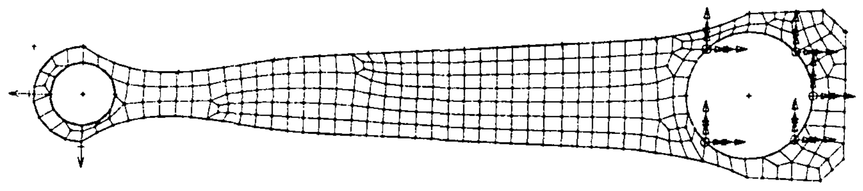


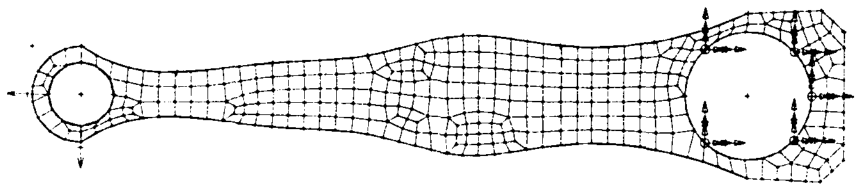
Figure 7.8 *Torque Arm: History of Design Variables*



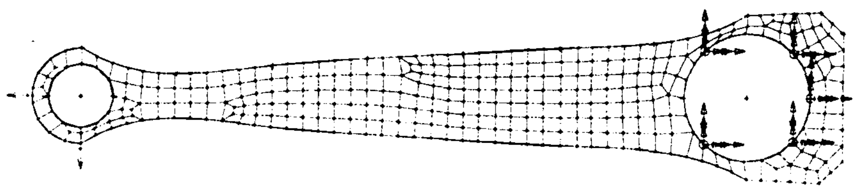
(a) Initial Shape



(b) 12th iteration



(c) 24th iteration



(d) Final shape (36th iteration)

Figure 7.9 *Torque Arm: Finite Element Models*

7.2.3 Three Dimensional Shape Optimization: Pressure Vessel

This section considers the optimization of a three-dimensional pressure vessel. Recall from Chapter 4, Section 4.3 that the prominent straddle point definition included provisions to seamlessly allow for three-dimensional boundaries. Before proceeding, details of how SPHINcsX handles three-dimensional structural component shape optimization are presented. Note that no supplemental education is provided before attempting to apply SPHINcsX to three dimensions. The classifier population that learned via a suite of sizing problems is expected to handle this challenge. Three-dimensional boundary representations compliant with SPHINcsX's restrictions are a direct extension of the two-dimensional boundary representations. Essentially any boundary where the control points lie on the boundary itself is acceptable.

In either the two-dimensional (2-D) case or the three-dimensional (3-D) case, each control point has one associated interior point that is one element length interior to the control point along the surface normal. The number of straddle points per control point varies depending on the boundary representation and the analysis model; however, the concept of prominent straddle point leads to the same environmental message format being constructed by the detectors in both the 2-D and 3-D scenarios. Thus, the structure of the individual classifiers remain the same, as does all of the internal operations of SPHINcsX. Therefore, the only extension to SPHINcsX necessary to move into the 3-D realm is the ability to read multiple straddle point stresses from three-dimensional analysis models.

Figure 7.10 juxtaposes a control point (shown as a box) with its associated interior point and straddle points (shown as circles) for 2-D and 3-D cases. In Figure 7.10, the 3-D case shows the control point, three straddle points and the interior point, although more straddles may be present but not displayed. This display demonstrates that by utilizing the concept of prominent straddle point, the detectors are equally adept at accepting 2-D and 3-D boundary representations.

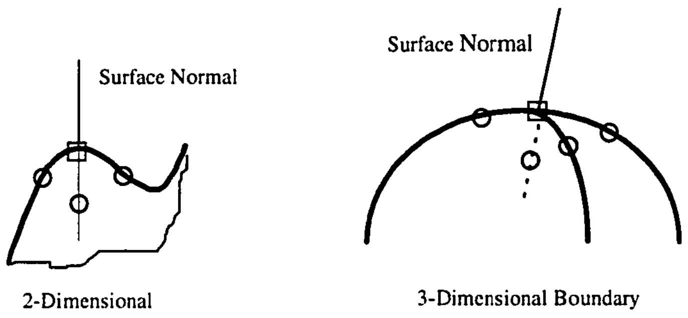


Figure 7.10 ***Control Point Layout on 2-D Boundary and 3-D Surface***

A major influence determining the choice of the first two application problems, the cantilever beam and the torque arm, was that the solution given by SPHINcsX could be compared to other methods in the literature. Both problems have been used as test cases. The cantilever beam provides the added benefit of having a theoretical optimum with which to compare. The torque arm provided a more pragmatic test where even though the theoretical optimum can not be derived analytically, due to previous studies by others, a gauge of success could be measured.

The selection of the 3-D pressure vessel as a test case is mainly derived by the fact that an optimal solution is 'known'. Although an analytically derived optimum may not be forthcoming, an argument based on symmetry reveals that the optimal internal cavity should be spherical and centered about the exterior sphere's center. This symmetry condition decreases the generality and difficulty of the problem, although SPHINcsX knows not of this symmetry.

The optimization of a simple 2-D thick walled pressure vessel was first considered in Chapter 5, Section 5.3.2, where it was used as part of the learning suite. Recall from the problem description that the pressure vessel was a sizing problem where both the internal and external boundaries were constrained to remain circles, and the design variable was the radius of the internal boundary. This application utilizes a spherical pressure vessel, with a fixed spherical exterior loaded by a constant pressure. The initial interior cavity is an irregular non-optimal void.

The goal of this problem is to find the shape of the cavity that minimizes the mass of the pressure vessel without exceeding the maximum allowable von Mises stress. The void is modeled using a variational method (Lange [1994]) to more closely simulate how it might be modeled in practice. The void is modeled by seven layers of design variables, as shown in Figure 7.11, the first and seventh (top and bottom) layers each only have a single point representation, each of the other layers is represented by four control points. Therefore there are a total of 22 control points. The non-end layers' four control points are distributed evenly about the vertical axis but their radial distances from the vertical axis are not equal. So a closed loop fitted to an internal layer's four control points would not result in a circle, but an irregular curve.

The top and bottom points (control points defining layers one and seven) are restricted to move along the vertical axis. The control points in the other five layers can each independently move in the plane defined by the layer in the radial direction. The heights of the control points in layers two and three are variationally determined relative to the height of the control point of layer one, similarly the heights of the control points in layers five and six are variationally determined relative to the height of the control point of layer seven. The height of layer five is considered zero and does not change. The height of all the points in layer two is $^2/_3$ the height of the control point in layer one, similarly the height of all the points in layer three is $^1/_3$ the height of the control point in layer one. For layer five the height is $^1/_3$ the height of layer seven and for layer six the height is $^2/_3$ the height of layer seven.

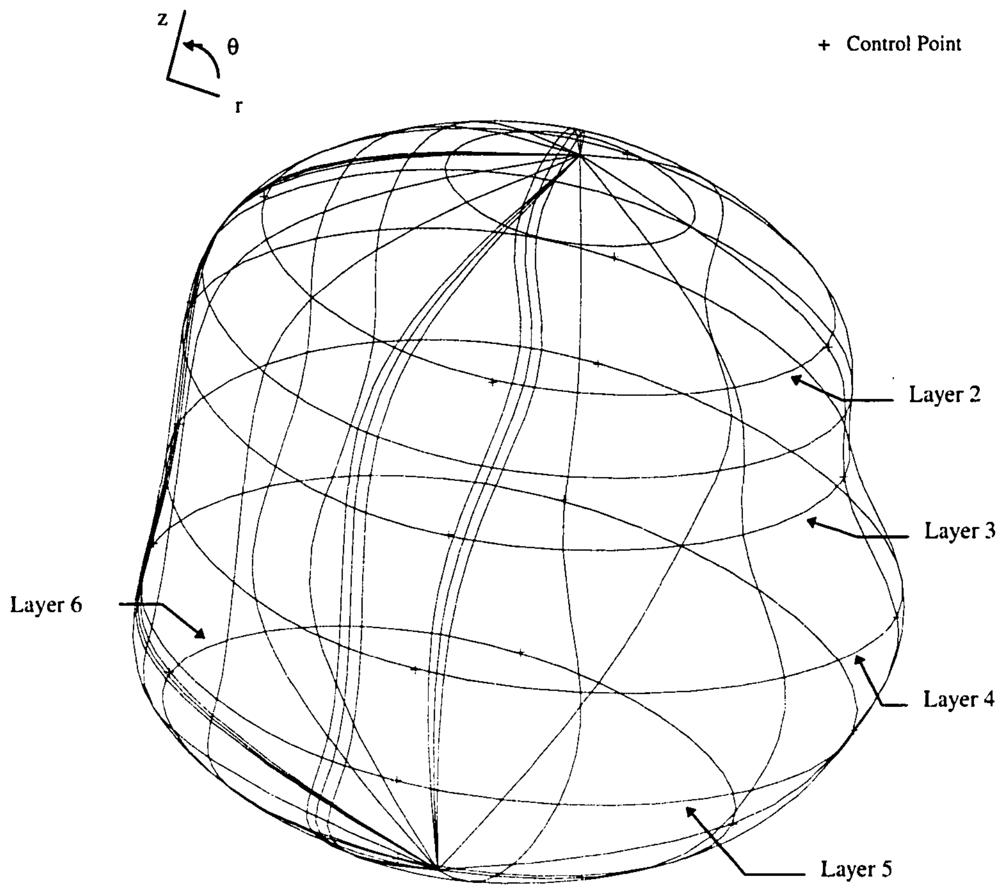


Figure 7.11 ***Initial Cavity of the Spherical Pressure Vessel***

The control points' freedom of movement is only restricted as described above. The boundary is intended to be able to assume virtually any shape and there are no symmetry conditions. Figure 7.11 depicts the initial cavity. The pressure vessel exterior is spherical with a of radius, b, equal to 0.25. The exterior is loaded by a constant pressure, P_o, equal to 500,000 Pascals; and the internal pressure, P_i, is zero. The maximum allowable von Mises stress is 1.0 MPa. The initial design has both under stressed regions and over-stressed regions. This condition is set to verify that SPHINcsX can optimize both feasible and infeasible designs. Table 7.3 summarizes the problem parameters. Table 7.4 provides the initial dimensions for the control points defining the internal cavity.

Table 7.3 *Three-dimensional Pressure Vessel Problem Parameters*

Name	Label	Value
Internal pressure	P_i	0 Pascals
External pressure	P_0	500,000 Pascals
Maximum Allowable von Mises Stress	σ_0	1*10^6 Pa
Convergence Limit	ε	0.01
Element Length	EL	0.005 meters
Mass	M	I-DEAS™ Solid Model
External Radius (fixed)	b	0.25 meters

Table 7.4 *Three-dimensional Pressure Vessel Initial Design Variable Values*

Layer	Number of Control Points	Initial Value of Control Point			
		1	2	3	4
1	1	0.20	▨	▨	▨
2	4	0.14	0.13	0.15	0.145
3	4	0.16	0.17	0.175	0.165
4	4	0.20	0.19	0.17	0.18
5	4	0.19	0.18	0.17	0.16
6	4	0.14	0.13	0.15	0.12
7	1	0.2	▨	▨	▨

One node on the exterior of the model is constrained to zero displacement in all six degrees of freedom. The initial finite-element analysis model consists of 2675 parabolic tetrahedral elements and was created by the I-DEAS™ Master Series automatic mesh generator created the mesh.

SPHINcsX converged to within 1% of the maximum allowable stress over each curve after 51 iterations. Only one equivalent analysis was required in each iteration. Figure 7.12 shows the iteration histories of the performance index TNSE and normalized mass for the pressure vessel. Because of the multiple curves that make up the design, the TNSE values of each curve are shown with line plots instead of bar charts, as was done previously. The figure depicts the overall progress and the success of finding a near optimal solution.

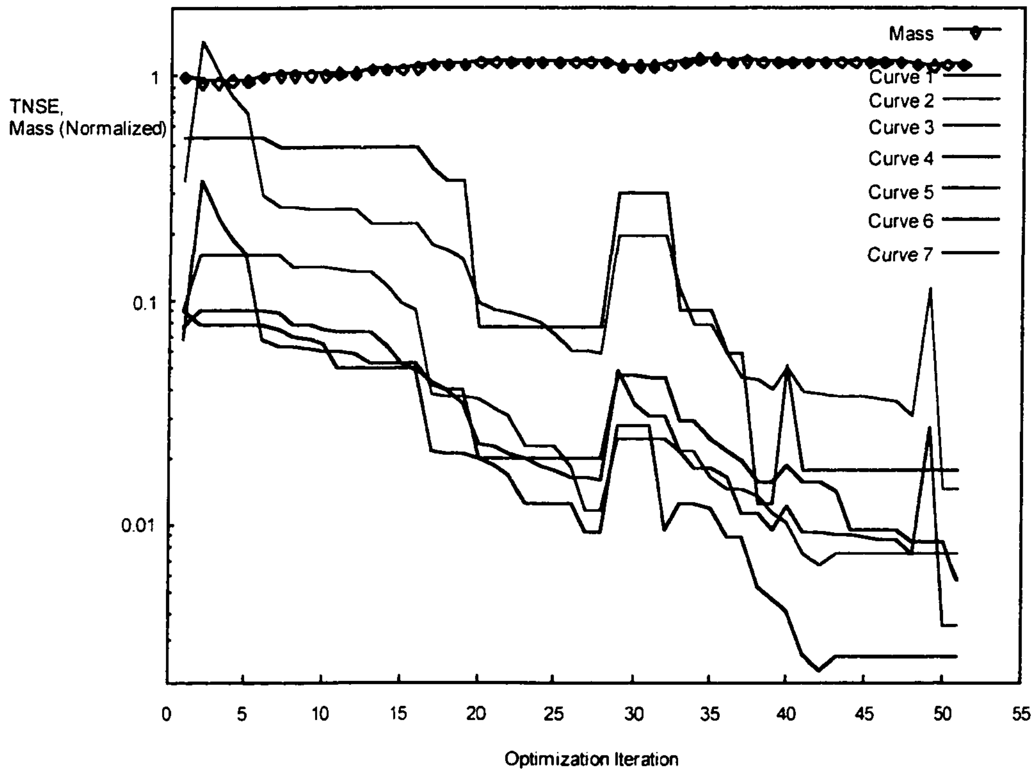


Figure 7.12 *Three Dimensional Pressure Vessel: Iteration History*

Verification of the thick walled pressure vessel solution is accomplished by comparing it to an analytical solution. The verification proceeds under the assumption that for constant pressure loading, the optimal shape of the inside boundary is spherical. For a three-dimensional pressure vessel with a spherical exterior and spherical cavity, the principal stresses on the interior surface are (Roark & Young [1982]):

$$\sigma_1 = \sigma_2 = -\tfrac{3}{2} P_o \frac{b^3}{b^3 - a^3},$$

$$\sigma_3 = 0, \tag{7.2}$$

where b is the outside radius, a is the inside radius, and P_o is the external pressure. The von Mises equivalent stress is the greatest along the inside boundary. Recall, for the design considered here, the maximum allowable von Mises stress is 1.0 MPa. Given that the outside radius equals 0.25 meters, yields an optimal inside radius of 0.1575 meters. The SPHINcsX solutions is approaching the spherical optimal shape. Figure 7.13 shows the points which constitute the locations on the cavity of SPHINcsX's optimal design superimposed over the sphere of the analytical optimum cavity, with the view chosen for it shows the largest amount of error. This figure reveals that almost all the points lie close to the optimal solution with no large discrepancies, displaying that SPHINcsX has nearly discovered the symmetry of the optimal solution without *a priori* knowledge that any symmetry existed.

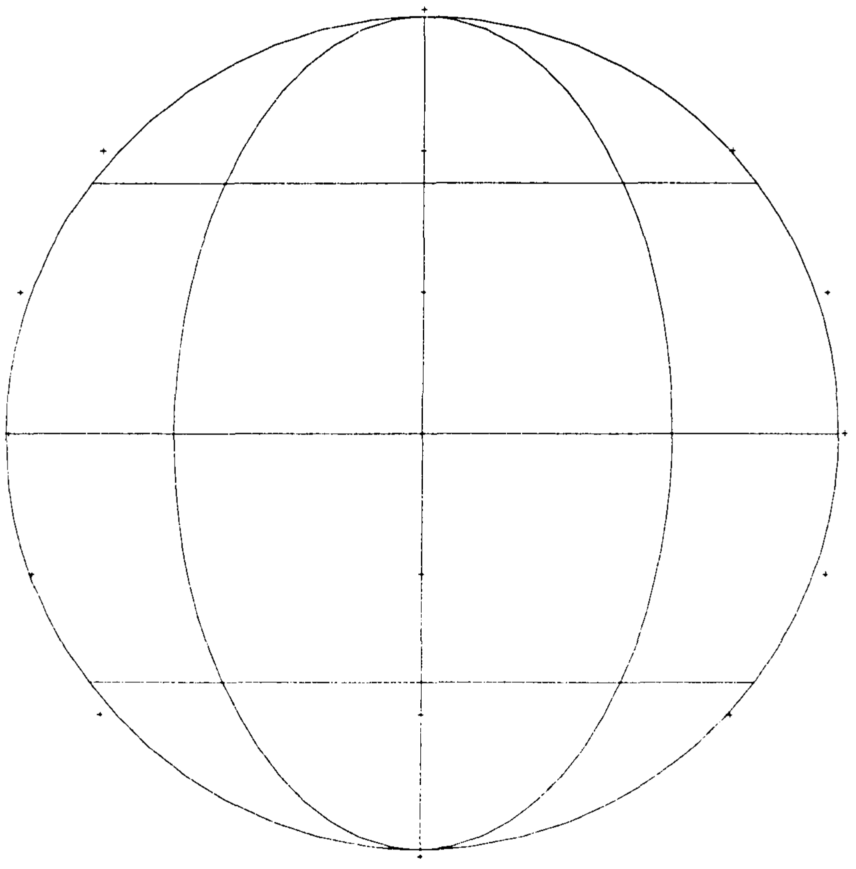


Figure 7.13 ***Comparison of SPHINcsX Solution with Analytical Optimum for Internal Void***

Another plot which shows the progress of the optimization process is one that shows the mean radius and the standard deviation from the mean as optimization iterations progress, this plot is shown in Figure 7.14. The mean radius had an initial value of 0.19 meters and converged to 0.159 meters. This converged mean is 1.5 mm (< 1%) greater than the optimal mean radius. This three-dimensional pressure vessel problem has convincingly demonstrated the ability of SPHINcsX in solving three-dimensional shape optimization design problems.

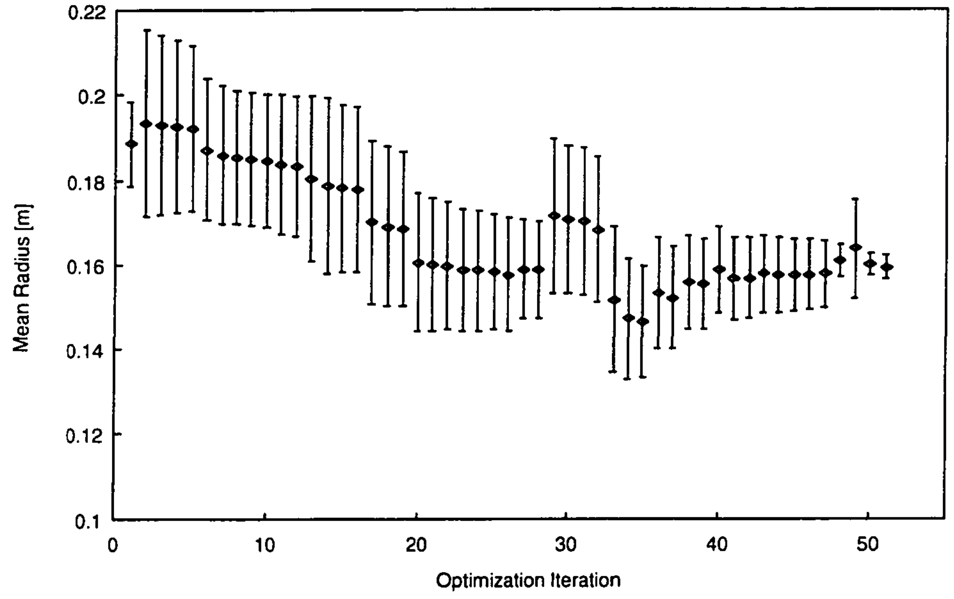


Figure 7.14 ***Mean Radius and Standard Deviation for 3-D Pressure Vessel***

7.3 Performance Analysis

The previous section demonstrated that two- and three-dimensional shape optimization are within SPHINcsX's scope. The efficiency and effectiveness were also discussed and displayed graphically. This section compares SPHINcsX's efficiency and effectiveness with other state-of-the-art methods.

Considering first other methods which cover SPHINcsX's scope, recall from Chapter 2, Section 2.5.1 that few exist and those that do suffer debilitating limitations. For example, genetic algorithm based systems (see Jensen [1992]) can handle all problems in SPHINcsX's scope also without auxiliary information but the solution times would be (conservatively) in the thousands of iterations. SPHINcsX's computational requirements should be drastically less in 3-D than techniques which depend on sensitivity information, but successful applications in the literature are scarce (Kodiyalam et al. [1992]) and applying commercial solutions is restricted by the fact they can not work with general shape representations as input to their optimization modules. The literature does provide some 3-D sizing optimization examples, Yang [1989], Botkin & Yang [1991], and Noel et al. [1995]. An illustration of the considerable cost of performing 3-D optimization utilizing sensitivities is provided by Chargin et al. [1991] where, with 23 design variables, the sensitivity calculation per analysis required over six times the computational overhead as the underlying analysis.

Modern optimization methods provide more competition to SPHINcsX in the realm of two-dimensional shape optimization. Again, recall from Chapter 2, Section 2.5.1 that many systems have been able to perform optimizations on simple shapes with a few design variables in the order of 20 equivalent analyses where the optimum is on the order of 33% lighter than the original design (e.g., Kothawala et al. [1988], Vanderplaats & Blakely [1989]). For the case of the cantilever beam, SPHINcsX proved to be competitive, optimizing in 25 iterations with an associated decrease of 44% in mass.

For the torque arm SPHINcsX determined the optimum in 36 iterations. This is competitive with other methods, however, many specialized methods are much more in their element with this problem and perform considerably better. Belegundu [1993] found an optimum shape of a similar torque arm design in 17 equivalent analyses with a 40% reduction in weight using a sensitivity based method, while Hsu [1992] performed shape optimization on a nearly equivalent design reducing the mass 38% in less than 10 equivalent analyses without any auxiliary information using the curvature function method. Note at the other end of the performance spectrum, Yang and Fitzhorn [1992] have applied the genetic algorithm approach to the two-dimensional torque-arm design with the optimum being found after over 100,000 analyses.

7.4 Summary

Observing how SPHINcsX operates, the generality of the complex may be appreciated. SPHINcsX does not notice differences between size and shape optimization problems. SPHINcsX's computational requirements increase modestly with regard to the number of design variables. As the number of design variables increases, SPHINcsX, operating with only the minimum criteria, requires only one equivalent analysis solve per design change regardless of the number of design variables. Using the multiple-control point modification technique, as defined in Chapter 4, Section 4.4.1, allows for many design variables to be modified per iteration in an intelligent manner.

Since one analysis is required per iteration, irrespective of the number of design variables, the efficiency relative to conventional optimization increases significantly as the number of design variables increases. The three-dimensional spherical pressure vessel evidenced this point, even with 22 design variables the complex optimized the problem in under 55 iterations. Assuming that the sensitivity based system used by Chargin et al. [1991] could solve the problem, it would have to reach the optimum in 8 iterations to rival the efficiency of SPHINcsX, due to the sensitivity calculation overhead. SPHINcsX has demonstrated its latitude in handling problems with various boundary representations and working with different analysis modules, providing efficient optimization for situations where the only other option is to use techniques which require thousands of analyses. The methodology and complex thus broaden the design latitude afforded the engineer.

Chapter 8

Summary & Conclusions

This research resulted from the need to discover an efficient generally applicable scalable shape optimization methodology. In the search for such a methodology, non-calculus-based search and optimization techniques were explored to determine if any could provide results superior to the state of the art. Machine learning in general and the classifier system in particular appeared to possess the desired features. The results presented here demonstrate that the classifier system indeed can be a valuable tool in helping to solve shape optimization problems, and its generality portents its value in other mechanical engineering problems. The rest of this chapter summarizes and draws conclusions from this study. After an overview is presented, major new knowledge derived from this study is summarized. Then certain unexpected serendipities that arose during the research are described. Some minor limitations and ways in which they could be ameliorated are next covered. Finally, future research directions and final conclusions are presented.

A major limitation of calculus-based shape optimization techniques is the calculation of sensitivity information. The computational overhead of calculating the sensitivities is a major consideration and increases dramatically as the number of design variables increases. One of the results of this research, SPHINcsX, learned to perform general shape optimization and did so without the benefit of sensitivity information or any other auxiliary information. Furthermore all the learning occurred via exposure only to two-dimensional sizing optimization problems. It is a further credit to the methodology that not only does it not need sensitivities but it learned without knowing anything about the initial shape, the representation of the curve under modification, the concept of stress and strain or optimization. Essentially, SPHINcsX learned to recognize patterns of stress and execute appropriate action. That is, the mapping between a control point's stress pattern and beneficial control point modification has been discovered.

Since the method does not require sensitivity calculation, the system is independent of analysis technique, be it constitutive equations, computer simulations, or empirical derivation. For most modern industrial problems, the analysis is derived from commercial simulation software where modifications to the software to implement sensitivity analysis are not practical. Zeroth-order methods are easier to exploit in conjunction with commercial simulation software as the integration of SPHINcsX with I-DEAS™ attests.

The results illustrate the computational efficiency of SPHINcsX. The computational effort, after learning, is small relative to that of methods using sensitivity analysis. The pressure vessel application demonstrated how the computational effort increased less than linearly with the increase in design variables. Note that to maintain the same resolution of boundary description when moving from two-dimensions to three dimensions, the number of design variables is squared. Thus sensitivity based methods, which are already computationally constrained by non-trivial two-dimensional structures, face a daunting task when three-dimensional components are encountered.

8.1 New Knowledge

This study has created a significant body of new knowledge, both in the focus of component shape optimization and in the more general realms of mechanical engineering and classifier systems.

The knowledge derived from this research related to component shape optimization is the development of the shape optimization via hypothesizing inductive classifier system complex methodology and software system. Salient characteristics of this methodology include:

- stable performance,
- scope covering two- and three-dimensional shape optimization,

- needs only minimum criteria (zeroth order),
- surpasses or rivals the efficiency of state of the art,
- independent of analysis technique,
- compatible with most boundary representations.

Associated with SPHINcsX and a body of knowledge which SPHINcsX learned itself is the resultant:

- learned population of classifiers.

SPHINcsX evolved this population, which can perform two and three-dimensional shape optimization, by using only a diverse suite of sizing problems.

More general knowledge derived from the research and applicable to the wider domain of mechanical engineering includes:

- demonstrating machine learning's applicability as a tool for mechanical engineering design,
- providing a complete example in order to illustrate how to apply machine learning in general and classifier systems in particular to mechanical engineering,
- illustrating the issues related to interfacing the classifier system domain with the mechanical engineering domain, and the determination of appropriate values for the myriad of classifier system parameters.

The last bullet is important because engineering judgment is critical in many aspects of the classifier system application. If the domains are not interfaced correctly or inappropriate parameters are used failure in application will ensue. For a new tool, such as the classifier system, there is high risk of inappropriate maligning by early adopters who fail to apply the tool properly, which could result in other researchers not applying the tool because of perceived inadequacies. Some of the critical factors include the form of a classifier; how long of a condition is required, should the condition consist of one part or many parts, and what form should the action take? These choices are in large part driven by considerations related to what is important enough to be represented and what actions should be performed. A balance is needed between the power of CS to solve problems in a generalized sense and the need to provide the CS with enough information in order to learn. Of course, all the domain dependent information available can be supplied to the CS, and it may learn faster, but then the CS will be trapped in a narrow domain of applicability. This may be suitable for many applications and is not discouraged. Furthermore, the feedback process requires judgment so the merit of changes are properly handled to provide appropriate reward or punishment.

Since the research was not intended to advance the understanding of classifier systems, the insights in this domain are less significant. Observations from this research regarding classifier systems include:

- Another successful application of the CS without the bucket brigade algorithm. This study and the results of many others (both engineering and non-engineering) show that the CS is a powerful tool when applied as a stimulus-response agent.
- Much can be gleaned from the CS learning by the use of strength histogram plots and default hierarchy plots which previously had not been utilized in the classifier system literature.

8.2 Serendipities

The research reported here did not, of course, progress in a nice linear fashion. There were many fits and starts, avenues explored then abandoned, and innumerable other explorations which are the natural result of the quest for new knowledge. Some of the explorations which may not have been fully realized in this work have significant merit on their own. Some of the more tangible serendipities of this research are mentioned below.

One of the avenues explored would have utilized simulated annealing[†] instead of a classifier system. Simulated annealing provides a general optimization technique; however, the efficiency leaves much to be desired. A property that plays a key role in the

[†] *Simulated annealing* is a stochastic search technique founded on the analogy of metal annealing. Beneficial as well as detrimental modifications are permitted with the magnitude and likelihood of a detrimental effect decreasing as the 'temperature' decreases with increasing iterations.

efficiency of simulated annealing is the annealing schedule. If an annealing schedule could be derived to produce highly optimal results with relative efficiency, then simulated annealing could remain as a candidate for use in this study. The possibility resulted in the derivation of an optimal annealing schedule (Richards [1990]). Even by utilizing the optimal annealing schedule, simulated annealing proved too inefficient for shape optimization, requiring the same order of magnitude of analyses as genetic algorithm methods.

This research utilized I-DEAS™ extensively, unfortunately I-DEAS™ and other general purpose engineering software are not designed to interface with third-party optimization. To circumvent this discrepancy, extensive use was made of the I-DEAS™ Application Programmers Interface (API) termed Open Architecture. A component of Open Architecture is the I-DEAS™ programming language called Open Language. After leveraging Open Language extensively and being hindered by the poor documentation it became apparent that others could benefit from supplemental documentation and training. The result is a tutorial (Richards [1994]) which has been sold internationally, and a tutorial which has been taught at the annual I-DEAS™ user's group conference.

In order to display the classifiers in a more intelligible manner, a viewer was created to translate the alphabet of classifier systems to bar charts. The viewer was not directly necessary to conduct the research but was a helpful aid in explaining the meanings of the classifiers. The viewer has been distributed to other classifier system researchers.

8.3 Limitations

The limitations under which this research was conducted include:

- Optimization performed on isotropic components acted upon by forces, or loads, in static equilibrium.
- The design modifications will occur by modifying the boundaries of the component; no new boundaries such as voids will be introduced.
- The objective will be to minimize the component's mass.

- Constraints will be placed on the maximum stress allowed.
- Acceptable boundary representations must have boundary defining control points physically located on the boundary.

There is nothing related to the methodology developed in this research which prevents any of the present limitations from being overcome. All the limitations except for the boundary representation limitation would cause a change in the environment or the actions performed. Therefore, these limitations would require modification to the interfaces and supplemental training, however the overall methodology would remain the same.

In the case of the boundary representation limitation, the translation of non-compatible boundary representations to compatible boundary representations should not be a daunting task. Once converted to a compatible representation SPHINcsX would carry out the optimization with the geometry in the compatible representation. Once the optimization was complete, the resultant design would be converted back to the user's original boundary representation.

Displacement Constraint

The extension to displacement constrained designs provides an illustration of modifications required to overcome current limitations. While stress is a more local property (to the design variable), displacement is a global property. For example, looking at the cantilever beam shown in Figure 8.15, the region from the load application to the free end does not experience stress, however, the displacement increases to a maximum at the free end. So for any design variable in the non-stressed region, an attempt to decrease the displacement at the said design variable by changing the design variable's value would be futile.

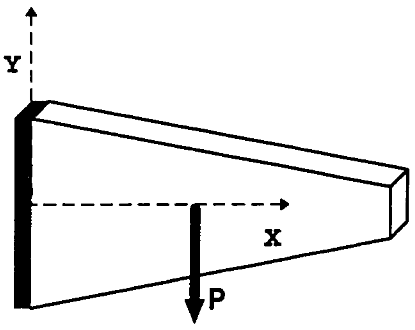


Figure 8.15 *Simple Cantilever Beam under Mid-span Point Load*

Therefore for displacement constrained designs it is not enough to detect displacements at (and near) the design variable locations. So, at least, stress information must be detected in addition to deflection information. A first attempt at extending SPHINcsX to handle displacement constrained optimization would be to add one more detector message per control point, consisting of displacement information. This would necessitate a modification of the classifiers to process the added displacement information. If this addition proved sufficient, the extended SPHINcsX could be trained to solve both stress and displacement constrained optimization designs.

8.4 Future Applications & Closure

Since the application of classifier systems to mechanical engineering is an embryonic field, there are many divergent paths that one could try to learn more about how best to leverage this partnership. The evolution of classifier systems is itself in its infancy. As research into ways of improving the CS's learning capabilities are discovered, those who use the tool will reap the benefits.

One could argue that the CS is also computationally expensive due to the massive training necessary. There are high computational costs, but here again the CS benefits greatly from where those computational costs derive. First of all, the training is off-line. That is, training is performed beforehand. In industry, one would obtain a pre-trained system, thus avoiding the computational and time costs of the learning phase.

Furthermore, there is no reason that the CS would not originally have good rules fed into it by experienced engineers giving it a much higher base from which to learn.

This research's developments tested the classifier system's mettle, however, now that the CS's utility has been shown, avenues are available to enhance the overall optimization process. That is, an optimization methodology could be developed to symbiotically conjoin the salient features of SPHINcsX with other proven techniques. For example, smooth transitions and non-wavy boundaries are usually desirable characteristics for optimal designs, therefore design modifications could be biased or constrained towards creating such characteristics. Numerous other modifications could be evaluated to enhance the present complex.

This investigation has shown that the classifier system has a place in the mechanical engineer's toolbox. Due to the power and flexibility of this tool, it is foreseen that the mechanical engineer will find a growing number of applications for which it may be leveraged.

Appendix A
Tabula Rasa Population

Note: All strengths are equal to 10.

```
#01##011#01###1#1#:100010
0##1###0##110##11#:000011
#0##1#01#0###10###:000101
#01#00###0#0#0#111:011010
#01#1####1######1#:111000
1#11###0#1#00#011#:010011
00#00#0######000##:100001
#01###10110###1#0#:001110
#1#1####0##0##0##0:000001
1####0#011#0#0##1#:101111
##01#0####0###1#0#:000101
##00###1101######0:001110
1#1##00##1#01##010:101000
10#1#0#0#0110##1#1:101011
10#11011#####1##1#:011111
#1#10####0#######0:100110
#1#####01111010011:101001
0##1##1##00#1##1##:011100
#1######1#########:001101
1##1#10####1###100:111010
010###0####0##01##:110001
10#0##11#1##1#####:011100
1###1###0001####11:111010
0##00#1111#0#110##:000101
1#########0#101###:101011
1##11#####01###0##:100011
##101####101##1###:000011
00#011#0###0####1#:001111
##00##1##0####0###:000000
##100##0#########0:100111
11#00#100##0#00#11:110001
00#11#0##1#00###0#:111001
#1#0#0000##0######:001010
0####1#10##11#0###:011010
#111##00#00######1:001011
#####0##1#####0###:011111
00##0#########100#:100010
##0##00#00####0#1#:001011
#01#1000#0###0###1:010110
#1#0#10#010#1##1##:100000
11#1###0#011#1###0:111111
0#####0#10######00:100101
####1#10##0###0#11:100000
#1##0#0#######01##:010111
##0#0#######1#101#:101111
#000###0#1#0#0100#:011001
100##1#0#0#####0#1:001000
01####0##11##1####:110010
10####0#011####0#0:110100
0#0011##1##100##1#:000001
##0######0#1###1#0:101110
0#00#01#10#1#####1:000001
01##0#####01####1#:110111
#0#####000##1##1##:000010
01##1#0##01#1###0#:110001
000##0#111#0####00:001001

##101#01#1##0#1###:110001
##1###10##0#01#1#1:110000
111######1#10####0:000101
10#0###10#1####1#0:110111
##000#00##111#0###:111100
000####11#01##1#10:000010
######0##0#######0:101001
#1##1#######0001##:000101
00####111##1101###:000010
####00#101#1#1#11#:010000
1###1#0##0#1#0#11#:011010
11##0###1##11##100:110101
####1#######1#0###:101011
###0###0#11##0#0##:010010
11#00#01##11#####1:010001
1#1####11###11###0:110011
0###00####0######1:111000
00101#0#0#00##1#1#:000011
##########0##1##10:011111
##1#11000######1##:101011
##0011#1#0########:101100
#1##10##011###0##0:110101
#####001#10##01###:010010
#1###0#0##0#####0#:110010
0#####1##1101###00:011111
0#0##0##011#######:111100
###0##0##1##101###:110111
#1#100#0#110#10###:110111
0#10#0#1#1###00#0#:000010
0####1###00101#1##:001011
#########01100#11#:010110
1##1#1##1###01#1##:100111
#01##10#1###0#####:111101
11#1#00##1#10#1101:110001
0####1##000#0####1:100100
#0##1##########0##:111000
1#0##00#11######1#:000100
1##111101#1##10##1:011000
##0#111#1###0#1##1:000100
#10#0#0##0####11##:011100
##0####1##00#1##01:101111
1#####0#####010##1:111111
####0##1#1#11###0#:011101
00#00#1#10##00##0#:010011
##0##11#1001######:001100
11##0##########0##:101011
##1####1###1#0#1#0:111100
##0#0######001#11#:101111
10100#0#1####1#10#:101111
#00#####1######0#0:111100
##0##1##100#11###0:011111
1#0###01##1##1#0##:110010
#########01##1##0#:010110
#####0#####1####11:110010
1##1#000#1100####1:111011
010#01#1#1#####1##:110110

0#####0##110#11001:100010
1####11#001####00#:111101
#0#1#0####0###1#01:110100
10##0#01##1#0#1###:001000
######1#0#####01#1:000011
####01#0#1#0######:010111
##0#0######1#0#101:001001
00#0#10##0##0###1#:111110
###0#0#1#1#0#####1:001110
00#####1###0##11##:001001
###0##1#0#######10:010111
######00#11####1##:111111
0##########01##1##:111000
1011#0##1##0#1#1##:011111
###1#1#######10#1#:011110
#01####00#0#00##11:011100
1######00#0#1#0###:010000
0##1#001##########:101010
#1##11#####0##1010:101001
#1##111##0###10##0:101000
#0#10##00100#0#0#0:100011
100###1##0##11###0:001001
###01####1#010#0#0:000111
###0#011#1###011#0:011100
#0########00#11###:001000
######01###0##10##:101111
1011#########01#1#:011100
#0#001######01#10#:100011
##1###1011#0#####1:110110
1#1#0###010######1:111011
00110111#00#######:000001
#0#01#00##01001#1#:111010
1#01#####10##1###0:110001
#0#####0##1####00#:110001
0##0#11######000##:011111
##0###00011##1####:010001
#001#0###1##0###0#:100100
#1####1###0#11##1#:000011
1#01000######1####:111101
####1##11######1#1:011101
0#001#1#010#10####:010111
##100000###0##0#0#:101011
100########0##111#:110010
########0####0##0#:001111
1##11#1###1####0##:111001
#10######1#10##0##:001111
0#00010####1##1##0:101001
###1#0##1######0#1:000101
#0#1##01###1##0###:110000
#1##00###01##1#0##:000001
###0####0#10#####0:111010
###0#1######1#0##1:100000
0###0#0#1#0011####:111100
11#11####1#1#00#1#:100110
0##0#000###011110#:000101
##000#0#11###1#10#:101100
```

```
0###1#1#001###1###:101100
#0#1#0##00######01:000001
1##1#1#######1##0#:001110
1###1###000#1#####:111111
#1#####1#101#####1:100101
#0#####0##00##1###:001110
###0011#00010#####:111110
#001##0#1#0####001:110100
1##010###1##0###01:101110
###0###1#######0##:101100
11##101########1#1:111101
0###0##1#0###1#01#:110101
#00#00#00#11####00:111111
####10#0#0###1#00#:010101
#01#1###0#0#######:010110
1#0#1#001##0##0###:000111
#0#1#1#0##0#11101#:001011
###00#1#0##0#01###:100100
##1####1###1###010:000000
0###101#####00#000:001000
01#######000011###:001111
01####0###0######1:011101
1###1#1####0##0#00:111001
##00#1##000###0#00:000101
0#10###0###01011##:000010
#1101#####1###110#:010100
#####0#01##1#0##1#:111000
#0001#1010###01##0:101100
##1#0######01#1#01:101001
1###1#0#0#0111#1#1:110100
1##101##1####1#1##:010001
##1#0#100#110#####:110100
###1###1#1##10####:010010
#0###1#111#1##1##0:100010
###1######1#1#010#:110010
#011#0###10#0#####:000110
1##11#1#0####010#1:110010
#11######00#1110##:010001
11##0110#0#1##0###:001001
0##0#1#1#0###1#10#:010010
0#10##1#0###1#0100:010000
1#1#1##10####101#0:010011
01##011#01011##1##:011111
1#####01#1##1#00#1:010011
#1##0###1##00#0#1#:110101
0#0#0#0#1######1##:011000
##00######01#0##1#:010100
0#1#1######1#####1:101111
#0###1#01##11#1###:101110
#####1#0##0#0011##:011000
01##101#1#1##1110#:111010
##0##0########0###:100110
#11##1##0#11#11###:000111
#10###1#11##11##1#:000111
##01#0#1#0#1####1#:110110
1##1##10#11###1##0:100100
##0#0##00#011#####:111111
#00######001###0##:000000
##11###1#11011#0#0:101000
#0#11#1##0#110###1:000100
00#####110########:100100
######1011##1##0##:010110
0###00#####0##0##0:010000
#0#0###0#111010#10:001100
##1#1#######1#01##0:111110
1##1##01##1#####1#:011111
#01100######01#0##:100101
01#######10#01####:001101
1011#1001####10##1:010110
0#####1###1#110##1:110100
##1001###11#####10:010100
#10##11#1###11#100:110111
1###0#11#0#######1:110010
```

```
00###1#####1010###:001011
10###1#10#######0#:101000
##1#1##0##0#####0#:010101
0####0####0#001##0:001100
1#01#######110####:101000
##1#1###1##1##1#10:110010
111#01###0#0#####1:101000
1##01#1#000####0##:101001
011001###00#######:100011
0##00#########0#011:110100
########101##11###:100010
0###1#000#11###0#0:101101
#11#101##1##01##0#:000101
#######11####00#10:100101
#1#1#####11##1#1##:010010
##1##0###0###101#1:011010
00##10####11#01###:011111
###01#01100#00#1#0:000011
01#######1111#0##1:000010
##0#10###0##0##01#:001001
##1#001#00#00##111:000001
111###########10###:010010
##00##0#1110##0#1#:010011
0#0###1#01#0#0##0#:100010
0###0#1#00###0##11:001111
#0###00##0#111##0#:000101
10#0##0##0#1##1##1:101001
###0####01##0####1:101101
######1#0001###10#:110110
####10###00#####1#:110011
##1#010###1##1#011:100011
0#0##0##01110###0#:010000
1###0###1#10##010#:001110
0#0#1###000##100##:101010
##01#1#0#010#1##0#:111100
#1#######0####01##:011000
#001##0#1#1###1###:010110
0#####00####0#00##:111101
11#0#01##0###1#001:110100
#110101####1##1###:100111
11#11#0##########10:011101
1#0###0####1##1#11:111011
##0#####0#1#101##1:000110
011#####0##111####:011001
11##0#####0##0####:001011
##1###1#01#100##11:101011
###1##11#####0####:011001
###111#0###0##0###:010010
####0##1##00#1##0#:100000
##0##1#0#0#010#0#1:111011
#0#1#1##101####1#1:111101
#1##001101##0##0#0:010110
#0###0####1##1#10#:001001
0#00##1#1######0#1:010110
#1#10#0#11110#11#1:110100
100#0###1######11#:011100
##0####0###00#####:000001
###1#01###11#1#1##:101111
0#0#####001#000100:110101
#1#1####0#1#####11:101101
0##11#1#0#1##00#0#:011001
###0#1#1####0#0#0#:010101
#1#####01#0#1#0010:000111
#1##010##0#10010#0:010011
11##0#0##10#1##10#:001111
C###00###110######:011001
#1#######1#01##0##:100011
101100#1#0##1##010:110001
#01#####00######00:011101
1###0####1##1#0#1#:101011
#0##0#########01##11:010111
#11##0###0##01###0:000000
1##1##10#00#01#1#0:100111
```

```
#0###1#010#1######:010011
#####0#101#1######:010100
0000##0#1###110#10:010001
10##1##110####1###:011010
#1#1#10###0##1#1##:110100
##11101###011###0#:100101
#0#1##00#110#10110:001000
00#001#00##11####0:101010
001010#1####0##1#1:110001
###0###0#0#1#0#1##:000100
######1###0###1###:000100
######11#1###0####:000001
####1######10#0110:010001
###1#0###0###0###0:110111
#####1101#1###01#0:000011
10#######1#11##1#1:000010
01###0##0######1#1:101110
1######11#########1:010001
#####01#0#0100#100:100101
#01##1#0#01##1##0#:001101
0#1###1#01#0#010#0:111101
#00###1#100011###:010001
##0####0##1#######:000111
1##10#0#####100#10:101000
10#1##00###1#0#101:111000
##0#0##0##0#0011##:010101
#10###0011###10##:011011
0#1##0#1#####1#0##:011001
####1#0###1###1###:100001
11###00###1##0##10:100011
#1##1##1####1#####:100110
1###0#10####0#####:000011
#0#000##1#########:101011
001###1##10101####:111000
#01####0###000#1#0:110001
###10#########1##00:011101
0##11##10#1###0##0:100110
#0101##1#11#0#1##0:011011
1##1#####0100#####:110101
1##1##1010#0##1##0:010001
1#######10#0#1#1#0:101000
01##00##1#0100000#:010100
0###0###10##0#1###:100010
#####10#101#0##0#1:010100
##0#0#0111#10#0#0#:110001
######0###0##1#0#0:001000
#0#1#0#0##1#11####:111111
1#####1#0##1#####1:100101
00#0#1#00##0##110#:101100
1#0###0##00#####1#:010100
101##0#####0##0###:000001
0###01###1#11#####:010011
#100#0#1##1##0110#:011000
0##11001#####0####:111110
####00###1###1#1##:011111
#1##0##0###100##0#:011000
#1#1##01#0#11##101:010011
1#11001#########0##:100101
111##0##########0#0:100111
#1#1##001##0##1###:111110
###010#00####0#0#0:101011
1##00000#111##100#:010011
#11#1##1###110#1##:010001
##1#0##1###1#010#1:010101
11#0####00##1#####:101111
1#1###1#11#0###010:011100
#####0#1##000#01#0:001100
#1###0110##1##100#:011011
#01##1###1#1###1##:100001
######11###10#####:000011
##1#1#0###########1:100100
#1#0#0#11###1#1100:000110
0110#1011100###1##:011000
```

```
########01#01##11#:011001
0##0#1010#011#001#:010010
#01#100#0#####1#0#:010111
#1#0#10###0##0##0#:110111
010##00001000#0###:111000
#11##0#####0#1##1#:111001
110##0#0##0##1###1:010000
0##0#1#1#11001##11:001010
##01###00#1#10#1##:011100
##0#00####1#1#0###:000101
###0#10#0#0#1#####:110110
##001###11##100###:010001
#1#01110##########:111011
#00####11###01##0#:111010
#0###11######11111:000001
00#10####1#####0##:111100
#1###1001#####0#0#:010110
1#####1#00#0##1###:100011
0#####00##0###1###:101001
#00##01######1##1#:111001
1#0###1##01#10##11:011100
1#######0010110#1#:101011
##1##010###0##0###:000101
10#1###1#####0#01#:100100
#####1#00#0###11##:111001
#10#1###10001###10:101010
###0##11#0#####0#0:001110
#1#0100#10#001###1:111000
##11###1###1##0#01:110111
##1##1##01###0#0#1:110000
#1#0##1#1###1##00#:110100
1##11###11#0010###:101110
1###0####100#110##:011011
10#0######0011111#:111110
11###011##01#0#1#0:001111
0##011##0######0#1:000110
110##0##1#1#####11:011001
01#########10###1#:011010
1###0010#1#110#0##:010010
10##0#00#001###101:111100
###100##0#001#0###:010001
####1######01111#1:001000
#0000#0##1##1#0###:111110
###1###10#########:111000
1###01###1#1###0#0:101010
1#11#0#1####10#0##:101111
####00111#00##1#1#:001010
01#1#1#010####1###:110110
##1#####1##1000##0:000111
#11#10###10##010##:010100
#####11######0####:001111
####1#0##0###01#11:010110
##1#00#100######0#:111010
##1###0##0#####0##:110010
010#0####1##01##11:110101
0##1#0010###1##010:011110
#1#0#0#1###1#01##0:101010
###0#0##1####0###1:010001
#01##01###1#######:011010
0#1#00##1#0####01#:000001
#1#####0##0##11###:110110
0####0####1###00##:101001
###1###1##0#01#0##:011010
01100#01#00##0##0#:010010
##0110#1####1##0##:100010
1#0########0###1#0:110011
11#01####001###0##:100001
#0#####1##1#######:111100
#######0#10#0##111:101011
###0###0##########:100011
##10######0#1#1##1:100011
############0#1#00:100111
1#1#1#####0011###0:001101
```

```
0#0##1001###110010:101100
#0##0###0##0###001:010100
11##1#000#####10###:111010
#1###1#0##1##0#11#:001011
110#####10#######1:010000
11###0###010####1#:101111
#1##1##10#10#1###0:110110
##0##0###0001##1##:111001
#0#101#0#1###00#0#:111011
####0##0#010######:001011
010#0##0#10###00##:000110
1#1###11##1#1#0#1#:101111
##1#1###1#1#0#10#0:101011
#####0##1####1#11#:101111
#0#1####110##10##0:011011
###10#0100###0####:111110
0##1###1##0####1##:100010
1#####00000#11#10#:011001
######11###01##1##:011000
#10###############:100000
11000#########1####:110111
###0##11#0###0#00#:010001
000#########1000#1#:111110
#0111####11#0####0:001101
0#10####1#11##1###:101001
01#1#1#1#101##01#0:101010
##0#1##0######0#11:101100
0###000###011#0#0#:001100
###1#######1##001##:001101
1###00#10####1####:111001
1###010#1#####1##0:101001
00###0##########1##:110101
0###0#1#00####0100:001010
###101#01#####0##1:000011
##00#0#######1#1##:001101
00###00#11#####1##:100010
110###01###1###0#:101101
##100#0#########1##:100111
#1##1##01###1###1#:111101
#####01100#01#001#:111110
#1#1######1###1##1#:001011
##0#0##########1010:000100
0#0#11#1###100#0#0:010011
0#1##00#####0101##:100000
####10###000##0###:101110
#0####1#1#1#0#####:010111
#####0#00##1#10###:001000
##11###0##01#0#0#1:111010
1####1#####0##1##0:111011
1######0#####0##1#:111110
####11#1#00#####00:111100
#100#########0##1###:101001
##010####1#0#011##:001001
###0###10#111#####:000001
#111##11##0##1010#:101100
#101####1##1###110:011001
####100#01#0#####1:100101
10#100#0##1#0#00#1:100100
1#0####0#####0##1#:010001
110######1#0#10##1:101101
#100#1001#111##1#1:000111
1#####1#00##0111#:111010
##01###1###11####0:110010
##########0#0#1##1:000100
####00#######0#0##1:111001
1##01#1########1#01:100101
##0##0##0###00###1:100011
0#0########10110##:011000
##0000#0####1#00##:100100
0#########0##10#1###:001101
#1####0###0##1#####:101111
1#####0#10#01010#1:111111
###100##1#011#0###:010000
```

```
100001#0010####01#:111011
11####000#000##000:001100
#1##1#01########01:110001
#0######0####0#0###:101101
0#0##11#0#000##00#:000011
##01###############:110101
00####101#111###01:100000
#1###1###101#1####:010110
#0#0###00##0001###:111110
1##101#100001#1##1:110110
10011010#####0####:100010
####0#1##101##001#:001100
#1#0#########00##10:011111
####00#000#01#1###:100010
####1##1##11011#10:111110
#1#0##1######1#0##:101111
#######0#11##0#110:000100
####00#######0##10#:010001
###1###0#0#########:101100
10#01#1#10###1####:101110
1#010#0#####11##11:110011
#####001##10#1####:001111
###11##0##1####0##:110001
#10#1##0#101##0#0#:000100
1#######0##101#0##:101111
###1#11#0######0###:010011
##0##000#11#001#1#:011111
0#1011####1#1##0#1:110111
####1###01#1#01###:011001
##11###0#####00###:011011
##11#011#0#00#1###:110001
###1###01##########:000110
#00#1###1#01######:100000
###0##1###1100####:110101
1#0##10###00###0#1:100010
#0####1#01#1######:100000
######0##1####11#1:001011
0#00#0000####1#0#1:000010
1####1#0##0##01###:110111
##0#1##11######0##:001011
######0#1#00#00##1:101101
##10####0##0##1###:111001
0#1###1#####00##10:011100
####01#0####00011#:000001
###0###############:110111
00##01#######0##0#:011110
#00#00###1##1##0##:101011
111###1##1#01###00:101000
#0111#1###0##1###0:111100
##0#11##1####01#1#:011010
###010#0###1##00#0:100101
##0####01####1##11:011011
###1##0###1####000:010001
00#1#######0##01##:101001
1##1##1##1##0#0###:011000
#0#########10#0##1:010000
0#1#1#############0:100111
###10#1##000####0#:001011
###11###001#0##1#1:000100
##10###1##00#00##1:111111
100#####1#0###10##:001111
##0#01###0#0#01100:110011
1###0#1#010#0###1#:100101
0#100#####010#111#:101010
##0##0#######0111##:100100
###0########1##0#0#:101101
#0##11###11##1###0:001010
1##1##11#0011###01:011001
#######0#10####10#:001100
##0#01##0##0#11#00:010101
#0##0#11###10#1###:010100
######01###1###0##:010100
#101#10#0#1#0#01#1:101110
```

```
##11###11##11###10:001100
0###1##0000###11#0:001111
10#0#0##111#1#001#:101001
###1####0###0#111#:101011
#####100#0##0###1#:110100
##1100#011000110#0:110110
#000#01####1#0##1#:000010
1##0##101#01###001:111010
######1#01###001##:111100
#1###11#001##00###:010101
#01#11#1##0##0##1#:100111
0#0###0#0#####0###:110101
#10###0######0##1#:010110
#10###0##00#10011#:110100
##0##101##1##00###:001010
#000##1#1000###1#0:010111
1#####1###0##0#0##:011010
###011##0##1##10##:001010
########1######01#:011010
1#####10####1#0011:010011
#1####0##1#1#10###:100111
#0##100##00####1#1:001110
0#1#1#01000#101###:100100
##0#1#####001#1###:101011
###0#10#0#01##1#01:010010
0###11##0#1####10#:011110
01#1#0##1#10####1#:001100
1#####1#####1#####:010011
0#10##1#0#011##110:111011
11#1####111##0111#:110100
#####0###0########:110101
#######1####1##1##:111000
###1#00#1##1##1#1#:001010
1###0#########1##0:011100
##1#1#11#00####100:101011
1#1#011##111#0####:111010
11####1#0##1##1###:111111
10#11#1###0##01###:011010
11###001010###1#01:000001
##110##1##1#0###0#:111000
0#0#####1###11##1#:010110
#1###010#00#01#0##:001101
10#1####1##1##1###:011100
##01#####0##10###1:101111
#111#0#101####0###:011001
01#0####0####00###:101011
##01##00####1####0:010101
##0#1#######10#01#:111110
#0####0#11######00:111000
##01##0####10##0##:001110
######0###1#######:111001
#1##11#10###10#1#0:111010
100##00#0#1#1100##:010110
00#1#101#1##1##1##:111010
###1#1#1#1##0#1###:100011
1##0##00######0###:101110
#00######10#######:100111
#00##1##0##0######:001100
##00##0###########:011010
#0#0#0###11#0#1#1#:000100
#11#1##0####10#0#0:100110
11#######1##110#1#:010111
#########1######10:000101
#1#0#####1010#0#0#:101000
0#####0#11#0##1#11:100010
#011#100#010#0##11:100011
#0#11###10##01####:000110
##1#100#####0#0###:010110
##000##10#10#00##1:100001
####00####1#10#001:110000
#0#0#10#0###000###:000011
#####1##0##0#0##0#:001111
##0##0#########00#:000010
```

```
##11#00#0#0111#1##:001100
1##1#1##1#1#101100:100100
#1#1#01##00#10001#:011000
1###0#11##0#01#0##:110010
1##10##0#11#######:101011
#0###1#####10111##:011111
10##0####1#11#11##:000011
######1##1####1###:100101
#111#########1##0##:101100
#0###00###00##10##:100110
101####1####00#1##:001111
##0#00###01##0#1##:011000
010##1##1###01##0#:001000
0##01#######11##0##:111001
###0010##00#####1#:101100
###011###0#00#0###:100010
1####111#01#0##1#0:101111
##0##01####1#1#1#1:010001
#1###1###110#1#0#0:010110
1#########0###0#1#1:011101
#0#1##0#11#01###01:101101
010##11#0###1####1:001000
#10#####00#1#011#1:100010
###0##1#1##00###0#:000001
0###0###110#11##0#:000111
#0###1####0#001###:100010
###0##1###1#0#000#:010010
#01##11#0#1#1100#0:000000
#11#0##01#11######:010110
##101#11###0##0##0:000010
##1#0####1#0####0#:011011
#0#0##00###11##1#0:110000
0#001#00##010##100:111000
#1#####1#####1##01:100111
10#10#00###1##00##:111010
1100###0##1##1##1:101001
##0######0###000#1:000001
#0##01####010####:101001
00110####1###0110#:001001
1#10##########0#0#0:011001
######0##0111####0:001101
01#######11###100#:110001
1###00#1##0#01##1#:011011
0101###001##1100#:111111
01#1#1##10#0#####0:010001
1#111#1##0###10000:000001
##01##0##0#1#11###:101011
#######1#########1#:010110
##0###0##0###111#0:001110
111#11####0###0#10:010000
#1#####01#0#11#10#:000100
0##01#001#1####1#0:000110
10#0#####1##0#1##0:100011
0#10##1##1#1#00###:010100
#1####1#1##0####1#:001011
###0###0#####01110:001101
##0#01##1#1###1#1#:000001
##1#1###11#0##0#01:000000
0#1001##01#01#1#0#:100001
0#0001#11##1###0##:000110
###1###011010##1##:011011
##00##1###########1:101101
##0#0##00##0#1####:001011
##01#1##01111#0###:110111
####111###01##10##:111011
#1#011#1#1##000#0#:110111
##0##1###0##0##00#:011101
###00#01#1####1###:011001
0#011#01####0##0##:101010
1####100##1###010:011111
#1#10##0##1##0###0:111111
#1#0#0##10########:100010
##10#1#0#######1##:100010
```

```
#1#0#1#00##111#1##:100000
###101#000#1#10#0#:111101
######0###0#1#####:101001
##0##11#011######1:000000
####0#######1#0#11:111011
###0#0###0######0#:101100
##101101#1###0####:111110
#01##00######11#01:111100
#0#0#####001####0#:110001
0########1#10##1##:000111
##0#0##0010####011:010101
00#####0##0#0#10#0:100101
#111##1##0##1####1:101011
###011######11##01:110111
#####1#1##1000####:100011
0##1########1##1#0:010111
###0##1#10##0###11:000000
101#####1######0##:111100
00###00####0#110#1:110100
####00#####1###0#1:101000
#1100#10#00###1#0#:100000
#01#1#0##000####0#:110000
1#11#1##0##1#10###:010000
1#01#######0#0##0#:100111
0#1##10##0####0##0:100010
#11#10011#1001###1:000110
##00001#0####1##0#:110000
#####1##01#1######:010010
##1###10##10#1#011:101001
1##1##0##01##0##0#:111011
#####10###0##1##1#:101101
10#00#10####101#1#:100010
1###1##1#########01:100111
#0#00#####0######0#:000110
01#10#001#####0#0#:001111
#######0###1######:110110
01#101####1######0:100001
#0###0###0##0#00#1:101110
1111###0##0##10#0#:110001
##0##0100#111#0#01:010110
0##01##00#1####00#:011011
0###10#11#0##00#0#:010011
01#00#11####1#0###:001111
10#1##10#####101##:111010
##1#11###0######0#:110101
#1####1####0###1##:100010
1###01##01#0#0##11:101110
####1#001#0#1#0###:000111
##1##10#####01####:000001
##0###0##01#######:001100
10#0####0#1#0#0100:100001
#1###11###111011##:111011
1#10#0##0110##0##1:110100
#1##1###1#01110#0#:101111
10##011011#####1##:110111
0##1##0100#####0#0:010110
####01#10#1#####0#:001110
#####10#0#11100##0:010011
#101#0#000######1#:111000
#0#01#01##1##0####:100111
###1####00#1#0##0#:011100
#1#00#######001###:101111
1#1#1##1##0##1###:001000
####0########1##1#:110111
0#011#110##0###101:011100
000#1######01##0##:011010
1#0#11###10######0:000101
111##1###########0:000010
#0#####01#####10##:101010
01######0#1#0#1#00:101101
0#####1111#01##101:001100
#0##010#####0#1###:011010
1###1###0#1#00####:111010
```

```
#111##1####0###00#:100111
01##1#1##0###11101:000000
0#0##############0:010001
0###0###1#0#110##1:010001
##1############01#:001100
0#0#######0#1100#0:110110
1#0####1##01100#1#:101101
####0######0#11#1#:100100
#1#0####1###010##1:111001
1####01###10011#0#:001101
#11#1####0#####100:101010
#0#0####10#0#00###:110000
#0#01##1##0###0#10:100011
00#####011#####01#:011110
#01#####1####0##11:001011
#000##0#0##0#1##1#:101010
###11#011#1#######:010011
0#0#0#01#001####00:000010
#1#####0##0#01####:010010
#1#0011##########0:000010
###0#0#1001#0#####:100100
#####0###1#####0##:110101
#1##1####0#1#####1:001100
0##00##1#10#1#####:011011
###110##0#1###11##:110000
#111###00##0###10#:100110
#0#01#0##00000####:101110
0#####1#1##001100#:110100
0101##10########0#:101001
#011#########000##:110101
0########1##01##0#:000101
##1##10###01###1##:100110
###00#0######1####:000101
101##1###0######01:110011
###111101#100#1##1:101111
11#11#0##01###0###:001000
######1#1####011#0:110111
######1###########:110100
#######0##00###0##:010111
#0#0#######1###100:101011
11####00##001#011#:110110
111#0##1##00#01##1:111001
01#0#0##01#0##1###:100001
#####00##00#11#10#:100001
10####0#####100101:100001
0#0##1#1##1#00##1#:011100
0#####11#1##1###10:010001
##0###1011#####10#:101101
#1####0#0#1####1#1:100111
0111###1####10#0##:100111
011######101#1#0##:001001
1#1###0##0###1#0##:000000
0#01#1#####1###0##:101010
##01#####0###1#0##:101100
1#111#1#####1#0#0#:001011
#######0#0######00:111011
10#0#0##1###0##0#1:010000
########10#1101#1#:100111
0####101#1##10###1:101101
#1#1##0###1#0#0#11:101001
01###1##00##00##1#:011010
######1#1#0#0#0#11:011001
######11#0##11####:011001
#10####1####011##0:011010
##########10#10#01:110100
######0#01#00###0#:011101
1##0#######10#####:000001
#0#010#00##1#1#1##:000101
1#10###00###100###:001101
##10#0####00##0###:001110
##0#####0#1###10#1:100101
1#1###11101#####11:010001
####1#0##11####0#0:100000
```

```
01###0####0###1#0#:011101
1###010####001###1:010110
0#0###1##10##1##1#:111001
###1#1###10#0####1:100100
00#######00##11###:011001
001##1#1#1##1####1:000011
#10#0#0###0####10#:001011
###1#0####1#11#0#1:000101
#1###1#11##0#0####:101011
###1#100####011#10:010000
#0##10#1##1#####1#:101001
##1#111100##00####:111000
#0###1010###110###:100101
###1#####01##0111#:000111
11110###01#1#10###:111001
###100#1##0####10#:101001
11##1#01###1#10#10:100111
0######01#1#01##1#:000111
0#####01##1##01###:000010
###1#0###1###100##:000001
##0#101##1#####10#:101101
########1#0#0#0#0#:010011
###01###1##1##1#01:011100
#00##0###0#01#10##:111010
###1#1#011###1####:100001
00#1#0#####1#####1:000010
1#1#####011##0#0##:000111
##11####0##0#10###:011001
#1####1##01#######:011000
11##00###00###1#0#:101011
#01##01###1#0##11#:001101
1#########11####01:111111
0##01#001#10######:100101
###1#10###1###10#0:110101
#####11#1100#0##0#:110100
#1##0###0###101##1:111010
0######1#00#11#111:110011
#0#0##10000##110##:100000
#1#1#####0###0#0##:100000
#10#####0###0#1#1#:100111
1#1#####1##0#0##1#:011011
#0#0#####0##011###:001010
1###1110##00#0####:110010
00##1######0####1#:110000
0#010##1#####10#10:010001
#0#1######1#1#00#0:101111
#0##010#01#0#1#0##:100001
###0#11#0###0#00##:001011
#0#1###10#01###11#:011001
####0###0#######11:011110
1#0#10###1#0#1####:111001
###1##1######1####:011010
10##1#11#0##00####:000010
0001##1######01###:100011
#1#0#1##100####1##:100001
##011#1##1##0#00##:101010
#1#0####1#####1#11:011110
10##10####0#0##1#1:000011
####0#1#0#0#1###00:010100
##1101#101#0######:111001
0#1###10####10#0#1:001111
######0#####0#1###:100111
000##0##01######0#:101001
0###01######0#1#1#:010010
###1###0##1#######:010011
0#1#01##1#0#######:001111
1#0#10##0#0110####:001110
##000010#11#0#####:110101
#0#0#011##11#0###1:001010
#0##0#000###0##11#:010010
###01###11#0##0#0#:000101
1#00###1#0#1#0##1#:101011
10##0#0##01##1#0##:001010
```

```
#0#000#1#00####10#:110111
#1######011#1###01:101111
#01##1####11000#0#:011101
0#####1#01####1###:011100
#100#0###0##1#####:110010
#0##01#0######0##1:001111
####1#010#1#01####:101100
1##10#11001001l###:100001
#0#1#10########11#:010001
#1#####01##00001#1:110010
0#1####110#0#110#1:101110
####11##011####0#0:011010
###1#######10####0:100011
#011###1101#11####:111011
##0#1#0#0#1#10####:010000
11#1#####1110#0##0:110011
##0#######1101#0##1:001111
0100#0#0#1#######0:101100
#1###001#10####110:001100
####0#1####0##1010:100111
#####00##01101110#:100000
01###011###011###1:011011
###0######1##0#1#1:000001
##0####1101####11#:011101
##0##1#####01#1#0#:101110
##1#01#1####10##10:001110
####011#####1#####:001011
#1##0#0####0######:011101
1###1011###00###00:000100
```

Appendix B

Detailed SPHINcsX Algorithm

Algorithm B.1: SPHINcsX Algorithm in Learning Mode

Initialization, Termination & Genetic Algorithm Modules

```
I.   Initialization of Classifier System                    (Chapter 5, Section 5.2)
       • Set Initial population size & populace
       • Setup Punishment/Reward
       • Set Auction parameters
       • Set taxes
       • Set Genetic Algorithm & Triggered Cover Detector Operator parameters
       • Set learning iteration, $n$, equal to 0
       • Set epoch count, $E$, equal to 0

II.  Initialization of a design to be optimized             (Chapter 5, Section 5.3)
       • Initial Design model
       • Boundary representation definition
       • Global element length
       • Upper & Lower limits on dimensions (if any)
       • Maximum allowable von Mises Stress
       • Set optimization problem iteration, $i$, equal to 0

III. Increment
       • Set optimization problem iteration, $i$, equal to $i + 1$
       • Set learning iteration, $n$, equal to $n + 1$

IV.  IF    learning termination criteria met,                         (Chapter 6)
     THEN  terminate
     ELSE  continue

V.   IF    epoch completed (i.e. n mod Epoch length = 0)
     THEN  apply genetic algorithm to population
           set epoch count, $E$, equal to $E + 1$
           continue
     ELSE  continue
```

VI. Continue to **Optimization/Learning Loop**

Optimization/Learning Loop

I. Analysis Module — Iterations all — (Chapter 2, Section 2.4.2)
Analyze the structure and determine the von Mises stress values at the control points, straddle points and interior points on the modifiable boundaries of the component. Also determine the mass.

II.A. Feedback Interface — Iterations $i > 1$ — (Chapter 4, Section 4.5)
Read mass of design at iteration i

B. Apportionment of Credit — Iterations $i > 1$
Compare mass, stresses, and TNSE at iteration i & i-1. Determine if the design has improved or deteriorated, then appropriately reward or punish the one classifier that caused the most recent modification.

III.A. Detector Interface — Iteration: 1 — (Chapter 4, Section 4.3)

Read:	Mass of component
	Maximum allowable von Mises stress
	Limits on dimensions
	Global element length

B. Detector Interface — Iterations: all

Read:	von Mises stress at control points
	von Mises stress at straddle points
	von Mises stress at interior points

Convert stresses to their binary representation
Create environmental messages

IV. Auction module — Iterations: all — (Chapter 3, Section 3.1.2.1)

1) Check each classifier, determining if it matches any environmental messages. As the classifier may match more than one, each match is recorded.
2) IF no classifiers matched in 1)
THEN apply the *triggered cover detector operator* (Chapter 3, Section 3.1.3)
    Generate a classifier which matches one of the environmental messages. Set the generated classifier as the victorious classifier.
    Skip to step IV.5
ELSE continue
3) Have all the classifiers that matched in 1) compete in an auction to determine which one shall be permitted to execute its action. Factors that influence the winner include the classifier's strength and how well it matched the environmental message (i.e. the more "#" symbols in a classifier the more general it is and the more poorly it matched the stress set).
4) Pass the action of the classifier that won the auction to the effectors. This classifier is termed: victorious classifier.
5) Record which classifier is the victorious classifier for this iteration, for later use by the apportionment of credit sub-system.

```
V.     Collect taxes                Iterations: all      (Chapter 3, Section 3.1.2.3)
VI.    Effector Interface           Iterations: all         (Chapter 4, Section 4.4)
       Create modification vector magnitude by using the action of the victorious
       classifier and mapping it onto the range defined by the move limits.
       Modify all control points which matched the victorious classifier.
VII.   Termination criteria         Iterations: all         (Chapter 4, Section 4.1)
  1)   IF     all control point stresses are within $\varepsilon$ of the optimum
       THEN   terminate this design's optimization.
              Return to **Initialization, Termination & Genetic Algorithm
              Module** *step II* to commence optimization of another design.
  2)   IF     iteration $i$ is greater than a user supplied maximum (if any)
       THEN   terminate this design's optimization.
              Return to **Initialization, Termination & Genetic Algorithm
              Module** *step II* to commence optimization of another design.
  3)   IF     none of the above termination criteria are satisfied
       THEN   continue.
VIII.  Set the active design to the design created in step VI
       Set $i = i+1$
       Set learning  iteration, $n$, equal to $n + 1$
       IF     epoch completed (i.e. $n$ mod Epoch length = 0)
       THEN   Return to **Initialization, Termination & Genetic Algorithm
              Modules** *step IV.*
       ELSE   return to step I.
```

Algorithm B.2: SPHINcsX Algorithm in Application Mode

Initialization & Termination Modules

I. Initialization of Classifier System (Chapter 5, Section 5.2)
- Set Learned population
- Set Auction parameters

II. Initialization of a design to be optimized (Chapter 5, Section 5.3)
- Initial Design model
- Boundary representation definition
- Global element length
- Upper & Lower limits on dimensions (if any)
- Maximum allowable von Mises Stress
- Set optimization problem iteration, *i*, equal to 0

III. Increment (Chapter 6)
- Set optimization problem iteration, *i*, equal to *i + 1*

IV. Continue to **Optimization Loop**

Optimization Loop

```
I.     Analysis Module: Iterations all                    (Chapter 2, Section 2.4.2)
       Analyze the structure and determine the von Mises stress values at the control
       points, straddle points and interior points on the modifiable boundaries of the
       component. Also determine the mass.
II.A.  Detector Interface          Iteration: 1           (Chapter 4, Section 4.3)
        Read:                      Mass of component
                                   Maximum allowable von Mises stress
                                   Limits on dimensions
                                   Global element length
   B.  Detector Interface          Iterations: all
        Read:                      von Mises stress at control points
                                   von Mises stress at straddle points
                                   von Mises stress at interior points
       Convert stresses to their binary representation
       Create environmental messages
III.   Auction module         Iterations: all         (Chapter 3, Section 3.1.2.1)
  1)   Check each classifier, determining if it matches any environmental messages.
       As the classifier may match more than one, each match is recorded.
  2)   IF      no classifiers matched in 1)
```

```
         THEN apply the *triggered cover detector operator*   (Chapter 3, Section 3.1.3)
                  Generate a classifier which matches one of the environmental
                  messages. Set the generated classifier as the victorious classifier.
                  Skip to step III.4
         ELSE continue
    3)   Have all the classifiers that matched in 1) compete in an auction to determine
         which one shall be permitted to execute its action.
    4)   Pass the action of the victorious classifier to the effector interface.
  IV.    Effector Interface            Iterations: all            (Chapter 4, Section 4.4)
         Create modification vector magnitude by using the action of the victorious
         classifier and mapping it onto the range defined by the move limits.
         Modify all control points which matched the victorious classifier.
  V.     Termination criteria          Iterations: all            (Chapter 4, Section 4.1)
    1)   IF      all control point stresses are within $\varepsilon$ of the optimum
         THEN                          terminate this design's optimization.
    2)   IF      iteration i is greater than a user supplied maximum (if any)
         THEN                          terminate this design's optimization.
    3)   IF      none of the above termination criteria are satisfied
         THEN continue.
  VI.    Set the active design to the design created in step IV
         Set $i = i+1$
         Return to step I.
```

Appendix C

Learned Population

```
#1#0##1001###00##0:001101   8.96
#1001#1#0###1####1:010101  12.00
0##0011#10####10#1:001010  11.36
###01#1#10####10#1:101111  17.67
1#11#000##1010####:000110   8.70
#####1######1#10#1:101111   8.40
###0##11##1#11####:100101   8.42
0###0001#01#11##0#:010000   8.96
1###1##1#00######0:111001   8.68
1##0##11#00####0#1:110000   8.18
0#####010#0##00##0:011011   8.41
1###000#1####0####:001000   8.33
###1#####110111###:111000  15.53
1#1#1##1#1###0##10:101101  10.74
1#1#1#0#00#1##0011:110000   9.89
##0#0#1#10#1###1##:000110   8.67
#1111#1###0#00#1##:101011  10.89
#####1#1#11010###1:111111  10.06
##1011###1#11#1###:010010  11.88
1##1#11001##01####:110001  13.93
0#####101###0#####:001100  10.30
0#####01####0#####:011110   6.06
01##0#01#1##0##1##:011001   6.86
00#000######1#1###:110010   8.38
10#0#1##001#0#0111:011100   7.98
###1##1101##1001##:001000   8.81
0#11##11#10#10#001:000101   8.14
#00111#1####1#0011:110000   8.03
#####11#101####1#0:101000   9.58
#10001##########1#0:001100   9.23
0####1#######1#1##0:110000   8.70
10#######1#0#011##:101010  16.56
1###000#####1#00#0:101100   9.32
1##11#####1#1#10#1:110111  18.72
#####11#0######0##:100110  15.96
###0#1######1#1###:110010   9.80
#1000#10#####11#0#:001010  10.31
11##0####11010#1#0:110101   9.11
1###1####10#1####1:101111   8.76
1##0##11#00######0:111001   8.47
#1000#0##1##00#00#:000010   9.70
##1#1#######1##01#:011010   8.42
0#####10#0010#11##:001100   9.37
##11####001#01##1#:111001   8.03
#1001#1#0##0######:101111  10.04
000###10#0##0###0#:000110   8.59
011##########00#000:001101   9.46
10#######01010#1#0:110101  16.41
1######1#10#1010#1:111001   9.03
###0##11#00####0#1:110000   9.56
11#######110111###:011000   9.39
#01###010#0##00##0:010110   9.61
##1111#1####111##1:110101  16.83
111#0#10####11##0#:101000  10.40
##0#0#10###0###0#0:101101  19.52
#01###010##11###0#:001111   8.53
#10#0###0#101#0#1#:001111   9.18
1##1#1##10#1###0#1:010000   9.21
0##1###111###0#001:011011   8.19
011##01#####1#1###:110010   9.40
1#0#0101##0####01#:010001   6.93
#0#0#11##1#1#######:101001   8.65
1###000#1#10##000#:001011   9.33
0#000#1#00#########:001100  10.27
0##0###1###01#1000:001100   9.77
0#11##1#####11##0#:001000   9.30
###1#####11010###0:001000   9.07
####0#####10#1#####:101011  11.26
#01######11010###0:011101   8.46
####1001####1#01#0:101100   9.70
1##01#1#10###0#000:000111   9.18
#1111#########1#1###:100011   8.70
#01###010#0###1##0:101100   8.35
```

```
10#1#00#101#####1#0:001100  10.92
11011000#0###1#1#0:011101   9.24
#01#00100######0##:000110  14.29
1#01##1#10011##0#1:010011   8.67
###11#0###011001##:011000   8.43
0#1#1#1#0#1#00#0##:001000   8.48
01#0###0#1##11##0#:001000   8.78
1###0#00#01#######:001111   9.41
1###000#1#101###0#:001111   8.70
1#11#00#01##00#00#:000010   9.63
1#####1##0###10#0#:101010  19.78
0###0#11###01#1#10:101111   8.76
0##0#0##010#1#11#0:001000  10.06
01#0###0#01#0#0111:111100   7.97
1#11#0000######0#1:101111   9.23
#0###1######1#10##:011000   8.67
1#11#001####111##1:010101  10.25
##011#1#10####10#1:001111  12.39
##01###1####1#001#:100101   8.00
00#####1#10#1010#1:011001   9.40
####0####10#1####0:110101  11.28
#1#0#####01010#1#1:011100   8.88
####0####10#1#11#0:101000   9.98
####0#10###0#0#000:100111  16.72
0#11##10#000####01:001000   8.43
11#01#00###11#####:101001   8.13
#1001#1#0###1##110:000101  10.05
#1111#######1#1###:101010  10.65
####0####10#1####1:101111   8.53
11011000#0###1#1##:001011   8.02
####000#101##11#11:010000   7.95
#100#11###0##00#1#:011001   9.14
10##0###110##1#10#:010011  10.92
#100#11#####1####0:101100   9.42
###10001#0110010##:101011   9.77
#100##1##0##11###0:001101   8.64
#0######01##1#1###:010010   8.83
#0011#1#110#1##010:110000   8.30
1#00#1##011011##1#:111001   8.73
1##011#1###011#11#:011101  12.02
1#00#1##011011##1#:011001   8.50
#####01#####1#1###:010010   9.99
##11#####00##11#0##:000011   8.70
11######0###1#####:001000  10.91
00####0##1#####000:000100   8.22
#001###0#10#010###:001010  10.70
##0##0##0###10###0:011101   9.33
010###1##0##0#1###:010010   8.42
###1#####110111###:011000   9.99
11#01##11#0#######:111100   8.10
1#0#0#1##1#01##0#0:011101   9.91
1#1#1###0##011#0##:011100   8.58
#1####011###0#####:011011   5.74
1##00#01####000###:101001  10.71
01####1#0#1#00#000:001101   9.46
###0##11#00#######:010011   9.98
#11#1010#0###001#0:000111   8.18
1###0#00#01####1#0:001100   8.74
01#0#0##0#101#0#1#:001111   8.87
111###0#10#1#11##0:001101   8.30
11#1###1001010#1##:100001   8.77
#11####10##10#1#01:001010   8.22
10##0000#00#1####0:110000   8.88
10011#1#10#1###11#:100001   8.19
##0###010##110#01#:110011   8.64
##1##1###01011#11#:111101   8.11
####100##1##00#00#:000010  11.30
001#0010###01#####:001011  10.18
0#1#1#1##1011####1:010101  11.51
0#1#0#1#1#0##0#000:101101   8.50
01#011####0#110#0#:001010  10.07
####1##1#01100 10#0:011101   8.44
#01#0010##1#1#0#11:010001   8.22
```

```
1##11#0###011#1###:001110   8.63
#####1#01#####0#00:011010   8.53
11#######0###10##0:111000  12.57
1####11#0######0##:100110  18.53
#10#1##010#11###00:101100   9.43
0##10#0#1###0#011#:010011   9.96
11##########011####:001111   8.48
0#011000#0###1#1#0:111101   8.41
011##########1#0##1:010111   8.54
######010#0##00##0:010110  10.01
000#0#1#1#1011##1#:010010   8.90
1###0#1#1#0##1011#:111001  11.18
#01#####10#01####1:001001   8.81
1###0#1#10#1###0#1:010000   8.56
0##0#11#0#######0##:000110   9.66
00##0#0#0##01#0#1#:010010  10.94
1#111###0###1#####:011010   8.88
1##1######0####10#1:010111   9.97
####0####10#1####0:000101   9.03
1###0001#00##00#1#:111001   9.70
1#####1##0###101#1:100011  14.14
0#####1000100#1##:001011  12.33
10011#1#10#1###11#:001000   8.77
0##01#1#10#####101:010111   9.03
0##0###1#####00001:100101   8.30
011###########1#####:110010   9.25
10##1#10#01#110#0#:001010   9.11
###1####00##11##1#:111001   8.80
##011#1#10####10#1:001010  14.52
0#10###1001001##00:101101   9.97
#0#0#101##0##00##0:101001   9.97
00##########01##0#0:110000  10.97
101#0###011#1#101#:010011  13.88
111#100##1001#0101:101010   9.40
#0##100[illegible]1#####:101000   8.43
##0##0#01#1#000###:111111   9.75
#0####10#0##01####:100010  13.00
##0###1###0111##0#:001100   8.14
000#0#1#1#1011##1#:011001   8.48
01000####100##0##0:011110   8.22
###1######11010###1:010001   9.92
######1##0###10#0#:101010  12.63
00#000######1#1###:010010   8.54
##0#######11#1#11#0:101000   8.50
##1#1#########1#01#0:001100   8.87
01#1#01##10#1#####:001011   8.24
1##1#11001##01####:101000  14.54
1###0#1#10##1#01#0:101100  10.54
1##1#1####1#01##1#0:001000   9.33
11#1###1001010#1##:001000   9.35
####01##011010###1:111111   9.41
#01###010##11##01#:011000   8.35
###1##1001###00##0:001011  10.67
0###0101#####1######:001000   8.06
11#0##0#11#0##000#:001011   8.29
1##0##11#00####0#1:110000  15.63
0##0###1#####00001:011011   8.88
#01#1#10###0###0##:010010   9.30
#1############11#111:111100  16.86
#0#01#00###11######:001001   8.01
1#01##1#####01####1:101001  10.23
1###0#1#10#11###01:110001  13.59
1##1#11001######0##:000110   8.74
0#0#1#00###11######:001001   8.20
1####0####00##11##1#:111001   8.34
10##0#00#1#####0###:001001   9.10
0###0#1001#01####1:011110   8.08
#01#0010#####0#10#1:001111  10.18
##000#1##0####10#1:110111   8.86
#0#0#11##1#1#1#100:011010   9.83
0#10######11010##0#:011000   8.44
####0#11001001##01:010111   9.68
##1#1######1#111101:000001   8.57
```

```
##0######11#11##1#:111001    9.02
000001#######00110:110000    9.30
#0#011#101##1##01#:010010    8.42
1##1#1#1001#1###00:001101    8.11
#1#000#011#100###0:001100    9.04
#01###01#0##1####1:101111    8.62
0#1#1#1##10#1#####:101011   10.15
10##1#10####0##00#:100011   11.50
0####1######1#1###:110010    9.46
0#####11##10100##1:010001    8.26
110#01#1#00#1#0011:110000   13.27
1#######11##1#0###:010101    9.25
##1010######1##1#1:111010    8.40
1#01##0#0##01#0#1#:010010    9.76
0#00#####10#111###:011000    8.51
1#11#00000#11##01#:001001    8.30
0#####0#0##1#####:001011    9.43
####0#1#10#11###01:010001   10.67
0###0001#0#1###1#1:011010    8.15
1###0#0#0###11##1#:111001   10.96
11##0#0#0##011##0#:001101    8.38
10######10####10#1:101111   12.18
#01###010#0##00#1#:011001    9.61
#1#0#11##10###01#1:101101   10.43
10##000#101####1#0:001100   10.88
#####1#1#10#1#11#0:001000   11.26
1###000011001#####:011000    8.97
##11##1001###00##0:001101    8.10
1###1##1#00####1##:100110    8.56
1##1#1###1#0##0000:101000    8.92
1##00#1##1#0##1100:001111    8.07
0####1##1#101###01:110001   10.20
#11#0###0###1#11#0:101000    8.33
######1##0###10#0#:110000    9.93
####0#11001001##00:001001   10.40
###011#1#####11##1:010111    8.20
#0#00101####1#####:001001    8.50
#10001#########1#0:001100    9.25
##1###0#00#1##0011:110000    9.06
110100##0##1#0###0:001011    8.67
00#000######1#1#0#:101100    8.88
0##01#1#10#####101:101111    8.32
###11#0###011#1###:001110    8.48
10###########1#1###:110010   16.03
1#0#0#10###0#0#000:100111   16.17
0#10###10###1#####:001000    8.30
101#0###01#100#001:101101   13.46
###11#0###01101###:010010    9.09
0##00#10#1####1###:010010    9.45
#####1#1#10#1#11#0:001000   10.27
##00##1##1#1###1##:100110    8.35
#10#1##1##0##0011#:111011    9.58
1##10#0#####10##01:110001   11.23
#0##011##01010#1#0:010101    9.74
1###0#####10#111###:110000   12.68
101#0#1#10#1###1##:100110    9.51
011#####10#01#####:101011    8.26
#11#0###0###1#####:000011    8.58
###0##11#01000#1##:101011    8.06
1##1###11#0#######:011100   10.53
0#1#0#1#1#0##1011#:111001    8.37
11#1#####11#0##0#1:001010    8.08
0#####01#1##01####:010101    7.43
#0#######01#01####1:101001    9.40
1##1#0100###1#001#:000101    9.29
01#####011###0#####:010110    3.96
##############1#0##1:010111    8.14
1###0#####10#111###:111000   16.00
####100##1##1#####:000110    9.43
1#1#1#1##10#1#####:101011    9.07
1#0#0#00#1#####1#0:101100    9.43
####10010#0#######:001111    5.56
##11#####1101##01#:010001   10.75
1##00#01####1#10#1:010111    9.02
1###0#####10#1#11#0:101000   10.58
1#0#0#00#1####0###:001001    9.09
11########0###10##0:010000    9.31
##0#######11010###0:101111    8.33
11#0#1##001#0#0111:111100    8.78
1##10#11###01#1#10:001111    8.04
#1##########01##0#0:010000    9.09
####0#####11010#1#0:010101    9 62
0#0##########1#0##1:010111    8.28
1#########0###1#####:001000    8.99
```

```
###1##1001##01####:101000   14.24
#0####1#10#1000#10:011101    8.85
#100#11#####1#1###:010010    9.72
#1#0#11##00#1#11#0:101000    9.62
1###0#1#10#1###000:101101   13.12
#1###1##01##1#10#1:110111    8.32
1#111###0###1#####:001000   10.13
11#0#####01000#00#:100010    8.93
###11#0#####1#001#:000101    8.30
#01##0#0#10#11##0#:010000    8.57
#01#0###011011####:011000    9.90
#####11#101####1#0:101100   11.69
0#000#1#00######0#:001010   11.70
1011#00#1#0##0##00:110011    8.58
0###0001#01#11##0#:000011    8.84
##11##001####1#0##:011100    8.26
1##1###11##0##0###:101011    8.86
###1####00#010###1:111111    9.16
1########11010####:100110    8.52
####1##1#01100100#:111000    8.94
1#11#0000######0##:000110    9.83
#11#0#1#1#0##1011#:111001    8.83
0####1######1#1###:110001    9.36
###111#010#11###00:101100    9.58
0##0011#10#####01#:001110    7.97
01#01##1#00##1#10#:010011    8.82
#10000#011#1###0#1:110000   11.34
##1#####1#00#1#0#01:010001    8.38
0#####010##110####:000010    8.67
##1010######1##1#1:110010    8.39
0##0#1#000#11##01#:011000    8.36
0####1##0##0#01#01:111110    8.80
1###0#01##101###10:001001    9.40
1###0#1#101#111101:000001    8.15
1##1#11001##01##0#:101100    8.82
1##01#1#10#####101:110111    8.62
#0#0#11##1#1###11#:000001    8.32
1###000#####1#1###:010010    9.39
0#####0#10#1#111#1:011001    8.25
####0#####10#1####1:011001    8.36
1########1101#1##1:110101    8.70
0#10##1#10#1###0#0:011101    8.19
#0###11##1#1###11#:000001    8.33
1#01##1#####01####1:001001   10.07
1#####101###0#####:001100    9.15
#0######00###1#1###:010010    8.04
###1######110111###:100110    9.69
1##01#00###11#####:101001   11.86
1#11#00000#11#0011:010000   10.79
0#####0101###01#1#:101100    9.30
11#11######1#1#10#1:110111   11.66
#100#11##01010#1#0:010101    8.64
0#1#1#1#0#111#1#00:101101    8.47
##11#####0##01#0##1:010000    8.34
11#010#011#100###0:001100    7.94
######1##0111##11#:001101    8.97
#11#0###0###1#####:001011    8.43
1#####10#1####00#1:001111    7.40
1###000##1#1#0###0:101100    8.84
###1#01######1#1###:110010   11.22
11#0##0#1#####01#1:000110    8.35
01#0#####0#1#00#000:100100    9.30
0##1#010#####01#1##:000001    9.93
1#111###0###1#####:101000    8.34
#0#0#101###0#01#01:001001    9.24
11########1101##01#:010001   10.14
#####11##0#0110#0#:001010    9.12
011######10#01#####:001011    8.18
###0#0##0#101#0#11:010001    8.70
###01####0###1#####:111010   10.36
##0#0#10#####11##0#:101000   12.93
1#0#0#01##101#111#:000000    9.53
##0########1#1#101#:010011    9.61
11#########1#1#10#1:110111    8.94
00############1#####:000110    8.83
#01###0##1##1#####:001110    9.60
###10001#0110010##:101101    9.36
10#0######10###10#1:110111    9.12
0###000#10101###0#:111100    9.31
#0#00#01#####1#01#0:001100    9.30
###011#1###011#11#:011101    8.38
11011000#0###1#1#0:101000    9.24
1####0#00#01#####0#0:010000    8.65
0###0#####0#10##1###:010010    8.96
```

```
##11###00#1#11####:000101    8.06
#100#####10#111###:011000   10.65
0#1######1101##01#:010001   11.26
##011#####1#1#10#1:010111    9.79
1011#0#0###1#1#100:011010    8.32
0#1#1#1##10#1#####:101101    9.74
##1111#00######0##:100110   11.63
1##1#1###1#1#1###1:110101   13.75
0##10#11##001#1#10:001111    8.04
1##1#1100#1#00#000:001101    8.91
0#1#1#1#0#1#1#10#1:110111    8.23
1#####10#0##00####:101101    4.39
0#####101####1#0##:011100    9.03
00####10001000#1##:001011   10.96
1#11#0000###1#1###:110010    8.96
###0##11#00####1#1:011010    8.64
0##0011#101#######:101111    8.74
0###011#101####1#0:101000    9.30
0#######0##01#0#1#:010010    8.37
01#011####0#1#1#10:101111    9.11
#####01#####1#1###:010010   11.20
00#######11010####:000110    9.03
#01#1#10###0##0##1:010101    6.32
#1000###110###01#0:101100    9.21
1###0#1#100###01#1:101101   12.21
1###000####1#1###1:110101   13.58
1##00#01####1#01#0:001100   10.30
#0011#1#1101######:101100    8.42
1#1#1#1#0####10#0#:010000    8.89
110#00###00######1:110101   10.40
##0#1#100###01#1##:100101   12.01
10#0##11#00####0#1:010000    8.04
###11#0###011001##:111011    8.60
1##0#1######1#1###:110010   12.56
###1######1101##01#:010001   10.28
##0##0######1#1###:010010    8.94
0##00#10#1#######0:001100    9.16
11#0#01#####1#1###:110010   11.76
###1######110111#11:101011    8.44
001#0010####0#10#1:101111    8.31
01#1####001001##00:101001    8.97
1####00#0#111#1#00:101101    8.12
001#####0##100#001:001101    9.50
1#####10001000#1##:101011   15.83
1###0#0001001##00:101001    9.19
#10#1##0####110#0#:001010    8.93
1#11##11#00####0#1:110000   10.43
11##1##0###1#1###1:010101   10.93
##0#0#10####00####:101010   14.67
10##000001##01##00:001000    9.94
1##1####101####1#0:001000    9.82
#0#0#101##0##00##0:101101    9.04
11#0#1##011010###0:011101    8.59
1##1#1##10#1###0#1:010000    8.07
#0#0#1#00##011#11#:011101    8.56
1#1#1##11####1011#:110101   11.52
##1####1#001001#0##:000011    8.96
0#####010##110#01#:010011    9.17
#100#11#0#######0##:000110    8.52
00###########01##0#0:110000    8.61
1#1#1####0##011#0##:101111    8.08
1##1#01001##0#0000:001000    9.30
######1#10#1000#10:111101    9.66
1#00#01#####1#1###:110010   13.02
#0#0#1010##01#0#1#:110010    9.39
1##00#11#00###0111:011101   10.15
1###000######11#10:000011    9.36
00##0#1#1#1#1111##:101011    9.76
110100##0###00#00#:000010    8.84
C110011#10#01#0##0:101000    8.83
1#0#0####10#10###1:111111   10.00
##1011###1#1#1####:001011   10.56
1###0#01##101###01:101010    9.79
###1#####1##11##0#:001000    9.33
1###000#######111#1:011001    9.54
11##1##0#######11###:011011    8.43
0##1#0##00#0#1#1#0:101000    8.58
#0#00#01#####1#01#0:101100    8.40
##11######1101######:001011    9.24
#####1#1#10#1#1#0#:001111    9.48
####000#10101###0#:011100    8.64
010###1##0##0#####:001100    8.79
1##10#01#1##100###:001001   11.41
0#101#####1#1#10#1:110111    9.80
```

```
#0###1######1#1###:001001  13.20   1###000######11#10:101101  10.86   ##0######1##11##0#:001000   8.45
1###000#10101###0#:011100   8.32   ##01########10##01:110001   7.97   0##0##00###1##0###:001111   5.81
#01#1#10###0###0##:111001   8.90   ##0###0#10#1#11101:000010  10.40   ##1#1#######1##010:110000   8.61
#0#0#1######1#1###:100011   8.73   #10000######1#####:000110   8.19   ##1#1#######1##01#:010010   8.41
0##0#01#101####11#:111101   8.70   ##0##0#01#####0###:001001   8.64   ##011#1#10##1##010:110000  10.53
1##1#1###1#11#####:001001   9.80   11#1###10#01##000#:011000   8.62   ########01##1#1###:110010   9.74
1#0#0####10#1#11#0:001000  11.01   1##0#0001##1####00:011011   8.25   0##1###1111#1#1#10:001101   8.08
#########11010###0:101000   9.61   ##11####001#00#1##:000100   8.25   #1000####100##0##0:111110   8.30
1###000#1#101##1#0:001000   9.48   #01#####10##1#1###:110010   8.22   10######10####10#1:010010  10.18
1####11####01#####:001011   8.63   0##00#10#1#######1:011111   9.30   0#10##10##101#1##1:110101  10.63
1##10#01#1##10###1:111111   9.12   1###000######11###:011000   8.85   0#1#1#1#0####10#0#:010000   9.06
11#1#####11#00#000:000111   8.81   ####1##1#0110011##:000110   8.19   #1000#01####1#01#0:101100   8.18
1##1#1###1#0##11##:111100   8.08   1##1#01001##0##0##:010010  10.30   11011000#0####0###:011001   8.58
###1##1001###00##0:101101  17.32   ##11####00##11#0##:110000   8.82   011####10##100#001:000101   8.65
#1000001#01#110#0#:001010  14.14   #11#0###0###1####1:010111   8.30   01####1#0#1#00#0##:001000   8.34
1#####1#0#1#00#0##:101000   8.49   0#00#1##011011##1#:111001   8.26   1##0###1##1##0#0##:110110   9.59
##11#####11010###0:011101   8.78   0####01#####1#1###:010010   9.23   #01#0###011010###0:011101   8.85
####0#1#10#1###1#1:111010   8.63   ##11###0#10010#1##:101000   8.11   #01#####0##100#001:101101   8.02
0##0#####11010###0:101111   8.25   1#0#0###00##10#1##:100100   8.03   0####01#####1#1##1:001111  10.30
###01#1#10#####101:110111   8.93   ######00#####0####:001101  11.24   1#0#0#0##10#1####1:101111   9.38
####0####11010####:000110  10.58   10#1####0#0#1#11#0:001000  11.20   ####0#1#10#1###000:101101   9.50
1#11#000##101#1##1:010101   8.61   ###011#1####1#0011:010000   9.90   ###1#1#1#10#1#11#0:001000   9.71
11#01####11010###0:011101  10.57   #01###010#0##00##0:101101  10.87   0#####0#10#1#111#1:101111   8.43
1##0##11#0####00#1:001111   6.66   #10000#0#1##00#00#:000010   8.40   ###1#####11010###0:011101   9.07
##0##00111##00##1#:011001   9.23   #01#0###011#1#101#:010011   8.90   ####011##01010#1#0:101001  10.31
10######0##100#001:101101  13.06   ##1010######1#####:010111   8.24   #####1######1#1##0:101010   8.75
#111###10#1#00#000:011100   8.59   1#####10#1####0###:101001  19.96   ##0##0#0110#111###:011000   9.66
#0##0#0####0#01#01:001001   8.16   #01#0###011010###0:001100   8.49   #1##########11##0#:001000   8.48
1##01#1#10####10#0:101000  17.53   01#0#0##0#101#0#1#:110010   8.61   ####0####10#1####0:101000   8.41
#0##########00####:001010  19.51   ##0######11011###1:110101  13.28   11000###1101011###:111101   8.29
#1#0#11##101011###:011101   9.17   01#011####0#1#11#0:101000   8.93   0##0011#101######0:111001   8.56
##1###0#00#1#11##1:010101   8.64   ##1111#00###00###0:001100  10.34   #11#####10#01#11#0:001000   8.48
###10001#01#1#0#01:011000   8.59   10##0000#0###1#1##:001011   9.88   ##1####1#0010#11#0:101000  10.49
1##1#1###1#0##000#:101011   9.15   #1#0011#101####1#0:001100   8.51   1##0#0001##1####00:010110   8.41
0###0001#01#11#00#:011000   8.06   10#1####110##1#10#:110011  10.97   1#1#1##1#10#1010#1:011001   9.96
##001#00#0##00#00#:010100   6.77   0##0#0##0#101#0#1#:001111  10.43   110#0010####1111#1:101011   8.41
#01#0#11001001##00:101001  10.01   1###0001#01#1101#1:100011   8.94   ##0#######1#1#1###:000010   9.14
##0##1######1#1###:100011   8.93   0010##1####01##0#0:010000   8.78   0##0###1##1#1#10#1:110111   8.48
011###101###0#####:001100   8.80   1##0##11#00####1##:001011  11.25   0###0#10#010##1###:010010   9.23
1#11#000##101#0#1#:101111   8.59   1#####1##0###101#1:100011  11.31   0######1####1#001#:001101   8.34
10#######01010#1#0:110101   9.72   1#01##1#0##0####1#:101000  10.46   #01###010#0##00#1#:001010  11.61
1#0#0#01##101010#1:011001  12.97   1##00#11#00####0#1:010000  12.95   11011000#0###1#1#0:101000   8.93
##11#000##1010####:100110   8.88   0##00#10#1######11:110111   9.32   1#01##1#####0#10#1:010111   8.19
1##0##11#00######0:111001  11.31   #10#0###0#10#0####:011000  11.14   1########1101##01#:110001  13.16
011######1##11##0#:001000   8.41   000#01#10##100#001:000101   8.30   #01###010#1#11#00#:011000   8.84
0#1#1#1#00##1111##:011101   8.60   ##011#1#101#######:001000   9.59   0#11##1########110:001011   9.77
0#1######1101###0#:110000  10.13   0###0#10#0##00#000:001101   8.26   ######1##0###10#0#:101010  12.61
110#01#1#00#1##01#:001001  10.79   #01#0###01101##0#0:010000   8.98   10#0#1##001#0#0#0#:001000   8.30
0##0#1#000#11###01:010001   9.66   #011###0#10#010##0:101010   8.41   #0###11##1####10#1:001111   8.17
11#0########1#0##1:010111   8.20   1#1#1#######1#####:000110   8.55   #01###01011#0##0#1:001010   8.22
11#1###1001001##00:001001   9.80   11#####101##1##01#:010010   8.91   1##1#1###1#1#1#110:100101  12.00
110110#1####1#0011:110000   8.33   ##0######11010###0:001100   9.05   0##0011#101#####01:101000   8.99
#0011#1##1#0#011##:101111  12.18   #####101####1#####:111111   9.30   1###000##1##00#00#:100010   8.04
1##00#01####1#01#0:001100   9.21   0#10###101#0##0000:001000   7.92   #1000#0##1101###01:010000   9.82
###0##100#########:100010   7.09   0######1111#1#1#10:001101   9.22   1##1###11##0##0###:101011  10.25
0###0####10#1#11#0:001000   9.70   ##1#1#010##110####:000010   8.16   #01#0#1#10#1###1##:100110   8.85
0###0#00#0111001##:111011  10.05   1##0##01####1#01#0:001100  13.01   0#1#1###01##00#00#:000010   9.43
#11#0###0###10###0:011101   9.21   #1000###110###01#1:101101   9.17   0#1#######1#1#10#1:110111  10.14
0#101#####1#1#101#:010011   8.57   1##1#1##01###01#1#:001100   8.74   1#011#####1#1#10#1:110111   8.88
1##0#11##1#1###11#:100001   8.50   ##01###1##011001##:111011   8.30   #0##1000####1#####:010000  10.56
0#11##1#####1#1###:110010  11.30   1#####101###0#####:101100  19.81   ##11####00##11##1#:011001   8.35
#01###01####1#1###:010010   8.80   ####1##1#00##1#10#:110011   9.03   101#0#1#100#1#11#0:001000  10.06
1###0#00#0111#1###:101110   8.60   0#####1###0111##0#:101100   8.64   11####1#0#1#01011#:111001   9.73
#10#1##010#11###00:001100  10.14   10#1###0##1#1#0#11:010001   8.60   11#0#1##01##11##0#:001000   9.57
1#000001#01#11##0#:110000  10.41   #01###010##0#01#01:011110   9.04   ###1#1###1#0##0001:001011   8.06
0#101#####1#1##0#1:010000   9.97   ###111#1##0##0011#:111011   9.04   ##1#1#####1#11110#:111101   9.42
1###0001#00##0#001:011011   9.00   1##1#####10#1####0:000101   8.46   #0##0#0#####10##01:110001   8.42
11#0#0##0#101###00:111111   9.03   011#####10#01#####:100001   8.03   1#00#####11010###0:101000  11.23
0#00#1##010##11101:100010   8.31   #0#011#101#1##100#:010110   8.37   0#####101######01#:001110   8.79
0###1#111#000#0#0#:010000   9.83   ####0####10#1010##:000011  10.49   1##1#1#1001001##00:001101  10.63
####0####10#1#11#0:101000  12.03   10#1####110##1#10#:010011  10.97   ##0#######1010###0:110001  11.46
#0#00#01####1#01#0:001100   8.81   1##1#1######1#11#0:001100   9.66   ##0##0#0#10#11##0#:110000   8.96
011#####10#01#####:110011   9.59   #0011#1#10####10#1:001111   9.90   ##11######1#00#000:001101   8.30
10######0#1#1#101#:010011  13.46   1##1#1###1#1#1#000:101000  11.13   1011#0#0###1#1#1##:101001   8.49
0##1#010####0#####:001010   8.64   #01#0010###01#####:101011  17.68   1#####1#0#111#1#00:001101   8.48
###01#110#0##00##0:010110   9.52   #0##0#1#0#1#1#10#1:010111   8.79   ###0##11#00####0#1:010000  11.98
1#00#####110111###:000110  11.22   1###000#####1#1###:010010   9.95   #1#0##1001##01####:001000  11.60
####0####10#1#11#0:101000   9.18   #10001#########1#1:001101   8.31   0##0#1#0000###1##0:001100   8.35
##0######11011#11#:011101   9.36   #11#0#1#1#1011##1#:011001   8.31   1##1#0######1#####:001000  12.18
10#####1011011#11#:011101   9.10   ##11#####110100##0:001001  10.36   ##11###0#10#010##0:000011   8.51
#####11#101#1#1###:110010  11.50   #0##0#00#0111001##:111011   8.50   ######00#####0####:001101  14.18
11#1###10#01##000#:011000   8.93   011####10##11#1###:010010  10.43   10#1####110##1#10#:010011  10.97
#####11##0####10#1:010111   9.82   1#11#001####111##1:010101  11.63   ##0#0010##1#1#0#11:010001   8.57
##11#000###0##1100:101111   8.54   ##11###1####1#001#:001101  10.75   1##1#1#1001#1###01:001010   8.33
```

```
1#####1#0#1#01011#:111001   8.50
#100#####10#1#####:101011   9.47
01111#01#0111#1###:010010   9.03
0###0#100#0#######:001111   8.59
00##########1#####:101000   9.86
1###0101####1#####:001000   8.04
##0###0#11#0##000#:001001   8.15
###1##1001##01####:001000  10.24
###0#1#00#11#01#0#:011101   8.40
#100#####10#111###:111000   8.94
0#10###1001001##00:001001  10.89
1#1#1##1#1###0####:111000  13.12
1##1#11#00##11#0##:000011   8.25
##0##0#01#1#000###:111111   8.59
11#1#1######1##0#1:110000   8.98
1###000#1#101##01#:111000   8.53
####0#11#0111#1##0:001000   9.19
1###000#####1#0011:010000   9.40
#0##1######111###:101000  10.71
##0##0#011001#####:011000   8.56
11#01####11010###0:101111   8.61
1#11#00001##01####:010001  11.26
1##00#01####1#10#1:110111   8.66
##1####1#00#11##0#:101101   8.81
10#0#######0#01000:101000   8.73
1#####10#00######0:111001   7.14
1##1#1##1#0#######:001111   8.30
##111###0###1#####:001000   8.93
01#1##1##0##11###0:001101   8.13
0######0#10#1#11#0:101000   9.90
0#####101###0##1##:001011   8.15
0##0##1##0###10#0#:011001  11.62
101#01#1#01#00#000:000111   8.19
#11#0##01#1#000###:011111   9.49
10#####1011011####:101100  10.32
##0#001#110##1#10#:101000   8.62
####0####10#111###:111000  12.70
#0#011###110111###:011000   8.91
0#######0##01#0#1#:101111   8.58
110100##0##1#0###0:101100   9.62
011#########1#####:010000   8.30
0#10###1001001###0:111001   8.00
###01#1#10####10#0:101000   8.22
#10000#0#1##011#10:001100   8.67
0##1#0##00#0#1#1##:101011   8.92
##1#100##1001#0#1#:010010   8.68
#11#0#1#1#0#1#1###:110010   8.30
1#############0###:001001   7.63
#########1101##01#:010001  12.46
#11#0#1#1#0##11101:100010   8.37
1##1#1###11010####:100110   8.51
00##0#0#0##01#0##1:011111  10.14
#####1##1#101##01:110001    9.95
0###0001#01#1#0#01:011000   8.39
#010#####10#1#####:101011   9.18
##0##1#01#1101##0#:010000   8.39
11##0####10#1####0:101101   8.49
######1##0###10###:101100  19.61
#11#####10#01#####:101011   8.64
#01#0###011010####:001011   8.37
##0##1######1#1###:010010  10.65
1###0#1#10#1###1#1:111010   8.76
0#####101###0##1##:001101   9.11
0####01#####1#1###:110010  10.41
0##0##1##0###10#0#:010011  10.70
1##10#01#1##1#11#0:101000  10.06
##1011###1#1#1###1:010101  12.82
##01###1####1#001#:101101   9.28
##0#####1###011####:101111  8.16
0#######010011#100:011010   8.65
0#101#####1#1#10##:111000   9.58
##1011###1#1#11#10:001111  10.37
0#1011######1#1###:010010   9.61
1#00#1##01101#11#0:001000   8.00
###011#1####1#0011:110000   8.03
#01###01001010#1#0:010101   9.14
10########1#0#011##:001111 14.52
01#00#01####1#10#1:010111   8.49
1##1#110##101#1##1:010101  11.26
10#0#1##001#0#0#01:101111   8.06
101#01#1#01#00#1##:000100   8.07
11#10#01####1#10#1:010111   9.30
000#01#######1#####:001000  8.65
11000###110#110#0#:001010  11.23
```

```
0##0011#101####1##:100110   8.43
#0#0#101##0##00##0:110101  10.31
0##0#1#0##101#0#1#:001111   8.30
1###0#00#0111#1###:010010   8.37
11#######1101##01#:010010   9.36
######1#10#1###000:001101  11.51
1##00#01####000###:101000   8.23
1##1####110##00001:011011  10.40
###01#1#10####10#1:011001   9.42
11#01#00###1##0###:101111   8.97
##11####00##11###0:001101   9.61
0#1###010#0##00##0:010110   8.48
1##1#10#1#0###10#1:110111   8.34
111#0###0##0###1#1:100011   8.87
#####1#1#10#1#####:001011  11.99
1##1#1###1#0##1100:001111   8.21
11#1#1##0##01#0#1#:010010   8.43
#01#0###011010###0:010001   8.18
1##00#11001001##0#:100100   8.24
#0#000#011#0##0#1#:010011   8.18
0#1###01####0##1##:011100   5.25
1###1##11#0##11#0#:110000   8.78
111#100##1001#0110:001001   9.02
#01#1#10###0##0000:101000   8.29
#11#0###0###1#####:101011   9.63
###1#####1##11##0#:001000  10.27
1##0011#10####10#1:001010  10.05
##101#00#100####01:110001   8.40
11#1########11##01#:011000  8.22
1###000#####1#0011:010111   9.48
1###000####1###0#0:110000   9.90
1##10#01#1###0##1#:011000   8.76
##0##0#01#1#000###:001000   8.90
0#000001#01#1101#1:000011   8.83
001#0010####1#0110:101001   8.25
1#1#1##1#1###11###:111000  10.01
01111#01#0110010##:101101   8.59
1#1#1###0##011##00:010011  10.19
11#1####0#1#00#000:100100  10.08
###1#####11011#111:100001   9.42
#0##0#0#0##01#0110:101001   8.68
11#1##1101##1001##:101000   9.86
1##0##1#10#1###1##:010111   8.10
1###0###101##00110:010000   8.43
0##0#0##011010###1:011111   8.85
##0##0######1#1##0:101010   8.60
####0#1##0####0111:111101   9.96
##11##101###0#####:001100  12.63
1############1#####:010000  8.39
1###0####1#1###1##:000110  10.03
0#11##1#########1#0:101100 11.47
#0##0#0#0##01#0#1#:110010   9.50
10##000001##01##00:001000  10.11
1##1#1###1#0##0001:001011   8.49
1##1#11001##0##0#1:101111  10.18
01####10#1####0###:001000   8.65
1###0#1#10#11###01:101000  12.68
1##00#01####1#10#1:001000   8.27
#11#01######1#####:001000   8.48
0#10###10#1#00#000:101000   8.49
10#0#####100110#0#:101010   8.39
0#0#0010#####1#1##:001011   8.88
#0011#1#1101##1###:010010   8.06
0##0##1##0###10#0#:001010  13.57
0#####010#0##00##0:110110   8.60
##########11010####:100110  8.16
##0#######11#0#10#0:101000  9.18
10##000001##01####:010010   8.88
#0###1#######1#10##:001000  8.03
#01###01#0##1#####:100110   9.23
#1000001#01#1101#1:000011   9.33
####011##01010#1#0:010101  10.65
0###0#01####1#01#0:101100   8.50
#0###1###1#1#1###1:010101  11.23
0###1#010###01####:011001  10.37
#0011#1#1101#####0:000010   8.00
1#####1#0#1#00#000:001101   8.61
1#1#1##1#1##1#0#1#:101111  11.14
#0#10001#011001101:100010   8.30
##1010#######1#####:011010  8.31
######1##0###1011#:011101  10.32
101011###1#1#1###1:010101  10.26
0#0##1#01#110#0#01:011000   9.06
##1#0#1#100###01#1:001101   9.49
```

```
0#####1010##11###0:001101   9.23
01#1##1######0#####:001100  9.23
####0#1##0000#0#0#:101101   9.46
##1##1###01010#1##:001000   8.84
0##001##0#101#0#1#:001111   8.70
#01#1###0#00#1###0:101100   8.37
##11####011##1#0#1:110111   8.13
#010##10###0#0#000:000111  10.05
0#####101###0##1##:101001   8.94
10##000001##01##0#:001100   9.39
1##1#11001##01##0#:011101   9.22
11##1##0#####11###:011000   8.26
1###000####1#1####:011000  10.78
######10#1##11##0#:101000   8.38
011##########1#####:101000  8.98
1##1#1###1#0##0011:110000   8.29
11#1#1#######1##0#0:000110  8.33
1##01#00###0##0011:110000  10.10
00####0##11#1#1#10:001101   8.86
1##1#11001##11##1#:111001   8.59
###1#####1##11##01:101101   8.84
1###00001001#####:011000    8.73
1###000#####1#1###:001000   8.21
##0##0#01#####0#00:111010   8.49
##0##0#0#10#11#01#:110001  10.13
1#1#0010####0##1#0:011001   4.53
###0###0#10#1#11#0:101000  11.23
10#1####1110010##:100011    9.06
#01#0###0#101#0#1#:101111   8.85
##############1#1###:110010 9.02
###1#0#010#01#110#:000100   7.96
##########1#11###01:110001  9.99
####011##01010#1##:001011  11.79
###1###010#01####1:010101  10.82
#01#0#1#0#00#1####:101011   8.60
11###0#010###00#1#:011001   8.35
#0#10001#011001##0:101101   8.72
###1#####11011#11#:111101  10.26
111###0#10#1#1011#:011001   8.37
0#10###10#1#00#000:001101   8.97
0##00#10#1######1#:010010  10.14
#1#########010###0:011101   8.90
10#######10###01#1:101101  17.70
###1######1#00#000:001101   9.07
00#000######1#11##:001010   8.06
#####0##0#101#0#1#:010010   8.57
0####1#######1#1###:010010  8.57
##1010######1#####:101000   8.18
1########10#111###:011000  10.67
#0#0#101##1#1###01:101010   8.30
0#1#1#10#0###1#0##:011100   8.12
###1#####11010###1:110001  10.21
0###0101####1#####:111111   8.90
0##0011#101####1#0:001100   8.12
##11###0#10010###0:101100  10.24
1##1#11001##0##0#1:110010   8.17
011##########00#000:001101  8.24
#010##1####01##0#0:110000   9.23
11#010#011#100###0:010000   7.90
##0##0#01#1#1#1#10:101101   8.19
0####1#######0010##:001101  6.41
10011#1#10####10#1:001111   8.03
1###1##1#1#0#011##:001010  12.24
0#####01010011#100:011010   8.63
0#000#1#10#1###0#1:010000   9.07
1##1####110#101#10:111110   8.88
###0#0##0#101#0#1#:101111   7.25
1##1#10#1#0###10#1:010111   8.66
#1001#1#0###1##1#1:011010   8.08
1#11####011##1#0#1:010111   8.36
1#01##1######1#0110:101001 16.13
0#1#1##00#######0##:000110  9.63
1##1#110#0###1#1##:001011   9.39
#0#011#101#1##100#:101101   9.61
#0#0#101##0#####01#:110001 10.36
01#10#0#1#####0####:011001  7.37
1###1#111#####0111:111101   8.78
1#01##1#1#1#1111##:001011   8.59
1##0#0##011010###1:011111  12.33
1##10#11###01#1#0#:101010  13.46
0##0011#101########:010011  8.21
1##1#####0######0#1:110000 10.15
##101#00##11#1##01:110001   8.36
1##10#11###1#1####:100110   8.44
```

```
##1###0#00#1#11##1:010101  10.29 | #100#11#####1#01##:111000   9.07 | ##0##0#01####0#001:011011   8.30
1#####1##0###10#10:001111  13.43 | ##011#1#101#######:010011  11.36 | 11#####00#1#000#00:001101   8.34
1###0#00#0111001##:111011   8.72 | ##1####1#0010#11##:001100   9.94 | 01#01#1#10####10#1:001111   8.11
##0##1#00##011#11#:111101   8.73 | 0##1##101###0#####:001100   9.22 | 0#####101###0##1##:101101  10.77
1#0######10###01#1:001101  11.49 | 011####10##10#1#01:001010   8.14 | #1#000#011#1###0##:100110  10.34
#11#0###0###1####1:101111   9.52 | 00##0#0#0##01#0#1#:010010   9.30 |
0###0####10#1#11#0:101000   8.48 | 10#1####110##1#10#:101000   8.67 |
00####0##11#1#1##0:001100  10.79 | ##00#######01#1##0:001100   8.65 |
####0####10#1####1:101111   9.25 | 11#0##11#00####0#1:110000  11.86 |
###1####00#010###0:001100   8.94 | 1###0####10#111###:001000  10.56 |
```

Appendix D

Glossary

AI
Abbreviation for 'Artificial Intelligence'.

Agent
Any entity, biological, mechanical or otherwise, that can perform actions, intelligent or not.

Antecedent
The IF portion of an IF-THEN rule. The antecedent (condition) must be satisfied for the consequent (conclusion) to be true.

Application mode performance
Measures the performance of the learned classifier system (now essentially an expert system) in handling problems from the same domain (but different problems) from which it was taught.

Apportionment of credit sub-system
The apportionment of credit sub-system deals with the modifications in strength of classifiers as the classifier system learns.

Artificial intelligence
The field devoted to developing hardware and software that enable a computer to exhibit 'intelligence' as defined and recognized by a consensus of human beings.

ASCII
American Standard Code for Information Interchange. The predominant character set encoding of present-day computers. The modern version uses 7 bits for each character, whereas most earlier codes (including an early version of ASCII) used fewer. This change allowed the inclusion of lowercase letters but not accented letters or any other letter forms not used in English.

Bayesian belief network
An AI technique which provide for reasoning with uncertainty.

B.C.E.
Abbreviation for 'Before the Common Era'.

Bit
Abbreviation for 'Binary digIT'.

Byte
A unit of memory or data equal to the amount used to represent one character; on modern architectures this is usually 8 bits.

CAE
Abbreviation for 'Computer Aided Engineering'.

Calculus-based optimization
An optimization method which depends on (first and/or second) derivatives, and usually continuity and unimodality, (of the objective function).

Consequent
The THEN portion of an IF-THEN rule. The consequent is the conclusion that is true if the antecedent is true.

Constraints
Limitations on the range over which the objective function may be minimized, represented as equality or inequality relations.

Credit assignment
The problem of deciding, when many parts of a system are active over a period of time (or even at every time step), which of those parts active at some step t contribute to achieving some desired outcome at step t+n, for n > 0.

CS
Abbreviation for 'Classifier System'.

Classifier system
1a. A machine learning system that learns syntactically simple string rules, called classifiers, to guide its performance in an arbitrary environment.
1b. Highly parallel, rule-based learning systems designed to continuously build and improve models of their environment based on experience.

Classifier system proper
The classifier system not including the input, output and feedback interfaces.

Conflict resolution
A strategy for determining which rule is activated or 'fired' when the conditions of several rules are satisfied.

Crossover
The exchange of material between two paired classifiers.

Consequent
The THEN portion of an IF-THEN rule.

CSP
Abbreviation for 'Classifier System Proper'.

Default hierarchies
A default hierarchy is a multi-level structure in which classifiers become more general as the top level is ascended. Each general rule responds to a broad set of environmental messages, so that just a few rules can cover all possible states of the environment. Since a general rule may respond in the same way to many inputs that do not really belong in the same category, it will often err. To correct the mistakes made by the general classifiers, lower level, exception rules evolve in the default hierarchy. The lower level classifiers are more specific than the higher level rules: each exception rule responds to a subset of the situations covered by the more general rule, but it also makes fewer mistakes than the default rules made on those states.

Defining length (of schema)
The distance between the outermost defined positions in a schema.

Design
The specification of an artifact that both achieves desired performances and is realizable with high degrees of confidence. (Eastman [1981]).

Design space
The design variables with the associated boundary representation that define the modifiable boundaries over which optimization may be performed.

Design variables
The entities which may be modified during the optimization process. If the values of the design variables are known the design is fully defined.

Domain knowledge
A narrow portion of knowledge that deals with the specific topic of interest.

Entropy
The randomness, or disorder in a system.

Enumerative optimization
A method whereby all possible designs are tested (i.e. enumerated), for continuous problems the technique would discretize the problem and then enumerate.

Environment
The objects and circumstances which define the universe of interactions that the classifier system (or agent) can sense and effect.

Epoch
Number of iterations between the application of the genetic algorithm.
An epoch, (a block of learning cycles) is performed so that the present population of classifiers can be ranked. After an epoch has completed the classifiers are bred via a genetic algorithm to (hopefully) discover a better set of classifiers. After the GA is applied the new population starts another epoch of learning cycles. The entire process is repeated until the population performs to some standard.

Equivalent analyses
This can be extended to comparisons of sensitivity and non-sensitivity based optimization, for example if the determination of the sensitivity information adds an extra 40% to the resources expended in the analysis module than one simply multiplies the iterations required by the sensitivity based optimization system by 1.4 to get the equivalent number of iterations in the non-sensitivity based system.

Expert
A person who is widely recognized as having valuable knowledge in a particular area, and who has demonstrated ability to deal with a particular task or problem much more efficiently than most people.

Expert system
Computer software that can solve a narrowly defined set of problems using information and reasoning techniques normally associated with a human expert. A computer system that performs at or near the level of a human expert in a particular field of endeavor.

Explanation facility
A utility that gives the user access to the reasoning behind a given conclusion.

Feasible design
A design where none of the constraints are violated.

Feedback mechanism
Provides the system with information about how well it is performing in the task domain.

Fitness proportionate reproduction
A simple rule whereby the probability of reproduction during a given generation is proportional to the fitness of the individual.

Fuzzy logic
The process of reaching conclusions based on information and facts that are not 100 percent certain.

GA
Abbreviation for 'Genetic Algorithm'.

Genetic algorithm
1a. A search algorithm based on the mechanics of natural selection and natural genetics.
1b. An iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each generation the structures in the current population are rated for their effectiveness as solutions, and on the basis of these evaluations, a new population of candidate structures is formed using specific 'genetic operators' such as reproduction, crossover, and mutation.

Generation
Same as *Epoch*.

Generational replacement genetic algorithm
Generational replacement genetic algorithm replaces the entire population with each generation.

Global Optimum
The best value of the objective function possible which satisfies all constraints.

GRGA
Abbreviation for 'Generational Replacement Genetic Algorithm'.

Heuristic
A guideline or rule of thumb that is normally effective in dealing with a given situation.

Hypothetical reasoning
An AI inferencing strategy used to balance countervailing factors in order to determine the best solution in complex cases.

IMGA
Abbreviation for 'Island Model Genetic Algorithm'.

Induction
Inference of a rule from particular experiences.

Infeasible design
A design where one or more of the constraints are violated.

Inference engine
Algorithm used to determine valid conclusions within the knowledge base, given certain information.

Input interface
Provides the classifier system with information about the environment, by adding detector messages to the message list. Each detector message is a binary string of length *L*.

Island model genetic algorithm
A genetic algorithm where multiple independent sub-populations each run mostly as islands or independent from the other sub-populations, with occasional migration of fit members between the sub-populations. The migrations promote the sharing of fit schemata, increasing global fitness and maintaining diversity.

Island model of population genetics
Model where separate and isolated sub-populations evolve virtually independently with small amounts of interactions between the sub-populations.

Knowledge
The facts and relationships that a computer program must have in order to perform in an intelligent manner.

Knowledge base
The set of rules contained in an expert system which constitutes the expert system's knowledge.

Knowledge engineering
The process of extracting an expert's knowledge on a particular subject and representing it in a form that is easily accessible to a non-expert.

Learning mode performance
Measures how well the classifier system is learning to perform the correct behavior in an environment.

LCS
Abbreviation for 'Learning Classifier System'.

Local Optima
Feasible locations in the design space where small changes to any of the design variables will result in a worse value for the objective function or an infeasible design.

Mating
The pairing of individuals for reproduction. Assortative mating is a type of mating which is not random but involves individuals of specific characteristics, in contrast to panmictic mating where pairing is random.

Meta rules
Rules which contain information to help the inference engine make the best use of other rules. Meta rules control the usage of certain groups of rules in certain situations.

Model-based reasoning
A type of expert system which contains a model simulating the structure and function of the domain of interest.

Mutation
1. A random alteration of a position in a classifier.
2. The process in which a gene undergoes a permanent, heritable, structural change.

Natural Selection
The selection of members of a population to reproduce, the selection is biased by the ability of the member to perform well in the member's environment.

Optimization
The act or process of making something as fully functional or effective as possible.

Order (of schema)
The order is the number of defined positions (with **1**'s and **0**'s).

Output interface
Provides the method for the classifier system to effect the environment.

Panmixia
Indiscriminate, unrestricted mating. Random mating within a breeding population. Mating without the influence of natural selection.

Production rules
IF-THEN rules in a specific format.

Production system
A rule-based system containing IF-THEN statements as its database. In addition, the system contains some inference mechanism that chooses a sequence of rules to be enacted in order to reach some objective.

Prominent straddle point
The straddle point whose stress value differs greatest from the maximum allowable von Mises objective.

Random mating
Mating without regard to the genetic constitution of the mate.

Random optimization
Methods which take samples of the search space in a pseudo-random manner, usually employing some heuristic sampling techniques to improve efficiency.

Replacement and crowding
Handles the introduction of new individuals into a population and the elimination of individuals from a population.

Reproduction
The process of producing offspring by sexual or asexual means.

Rule
A method of knowledge representation characterized by an IF-THEN format. The conclusions are considered true if the conditions are true. They may contain Boolean logic. A statement consisting of two parts, antecedent and consequent. The antecedent consists of one or more IF clauses, and establishes conditions that must be met if the consequent part of the rule is to be activated. The consequent is composed of the actions ofconclusions that result.

Scalable (Scalability)

The ability of a solution to some problem to work when the difficulty of the problem increases. For this research the scalability is related to the mechanical engineering problem of component optimization, the methodology is scalable for it can be adapted from two-dimensional sizing designs to three-dimensional shape designs with no change in the methodology.

Schema (pl. schemata)
Sub-string similarity templates, i.e. a template describing a sub-string which has the same value at the same locations.

Selection
The process of selecting the classifiers of the population which will reproduce.

Sensitivities
The derivatives of the objective function and constraints with respect to the design variables.

Sensitivity analysis
The process of calculating the sensitivities.

Similarity count
A count of the positions where both the child and candidate are identical.

Simple fact
A fact that can only have two values; true or false, on or off, one or zero.

SSGA
Abbreviation for 'Steady State Genetic Algorithm'.

Steady state genetic algorithm
Replaces only a small portion of the population on each generation.

Stochastic
Including probability or chance.

SUS
Abbreviation for 'Stochastic Universal Selection'.

Tabula Rasa
Erased tablet (Latin).

Triggered cover detector operator
A rule generation operator that is only activated when certain conditions occur.

Unimodal
Possessing a single maximum or minimum (in an interval).

Zeroth-order
An (optimization) method which does not use derivatives (gradient information).

REFERENCES

Adeli, H. and K.V. Balasubramanyam, "A Novel Approach to Expert Systems for Design of Large Structures", AI Magazine, Vol. 9, No. 4, Winter 1988, pp. 54-63.

Ali, Hosam, "Optimization for Finite Element Applications", *Mechanical Engineering*, Vol. 116, No. 12, December 1994, pp. 68-70.

Ashley, Steven, "Engineous Explores the Design Space", *Mechanical Engineering*, February 1992, pp. 49-52.

Baffes, Paul and Lui Wang, "Mobile Transporter Path Planning Using a Genetic Algorithm Approach", *SPIE's Cambridge Symposium on Advances in Intelligent Robotics Systems*, 1988.

Beightler, C.S., D.T. Phillips and D.J. Wilde, *Foundations of Optimization*, Prentice-Hall, Englewood Cliffs, 1979.

Belegundu, Ashok D., "Optimizing the Shapes of Mechanical Components", *Mechanical Engineering*, January 1993, pp. 46-48.

Bethke, A.D., *Genetic Algorithms as Function Optimizers*, Ph.D. Dissertation. University of Michigan, Dissertations Abstracts International 41(9), 3503B, (University Microfilms No. 8106101), 1981.

Booker, Lashon B., *Intelligent Behavior as an Adaptation to the Task Environment*, Ph.D. Dissertation, University of Michigan, 1982.

Booker, Lashon B., D.E. Goldberg, and J.H. Holland, "Classifier Systems and Genetic Algorithms", *Artificial Intelligence*, Vol. 40, 1989, pp. 235-282.

Botkin, M.E., "Shape Optimization of Plate and Shell Structures", *AIAA Journal*, Vol. 20, No. 2, February 1982, pp. 268-273.

Botkin, M.E. and R. J. Yang, "Three-Dimensional Shape Optimization with Substructuring", *AIAA Journal*. March 1991, pp. 486-488.

Braibant, V., "Shape Sensitivity by Finite Elements", *Journal of Structural Mechanics*, 14(2), 1986, pp. 209-228.

Braibant, V. and C. Fleury, "Shape Optimal Design Using B-Spline", *Computer Methods in Applied Mechanics and Engineering*, Vol. 44, 1984, pp. 247-267.

Callahan, Kelvin J., *Strength-to-Weight and Stiffness-to-Weight Optimization of Laminates Using a Genetic Algorithm*, M.S. Thesis, Department of Aerospace Engineering, University of Alabama, Tuscaloosa, Alabama, 1991.

Chargin, Mladin K., Ingo Raasch, Ralph Bruns, and Dawson Deuermeyer. "General Shape Optimization Capability", *Finite Elements in Analysis and Design*, Vol. 7, 1991, pp. 343-354.

Charniak, E. and D. McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley Reading, MA, 1985.

Coyne, R.D., "Design Reasoning Without Explanations", *AI Magazine*, Vol. 11, No. 4, Winter 1990, pp. 72-80.

Darwin, Charles Robert, *The Origin of Species by Means of Natural Selection*, D. Appleton and Company, NY, 1897.

Deb, K., "Optimal Design of a Class of Welded Structures via Genetic Algorithm", *AIAA-990-1179-CP, Proceedings of the 31st Structures, Structural Dynamics and Materials Conference*, April 1990, pp. 444-453.

DeJong, Kenneth A., *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Dissertation (C.C.S.), University of Michigan, 1975.

DeJong, Kenneth A., Using Genetic Algorithms to Learn Task Programs: the Pitt Approach", *Machine Learning*, 1988.

Dhingra, Anoop K., *A Unified Approach to Multiple Objective Design of Engineering Systems*, Ph.D. Dissertation, Purdue University, Lafayette, IN, 1990.

Doolitle, D.P., *Population Genetics: Basic Principles*, Springer-Verlag, Berlin, 1986.

Dorf, R.C., *Modern Control Systems, 3rd Edition*, Addison-Wesley, Reading, MA., 1983.

Dorigo, Marco and Enrico Sirtori, "Alecsys: A Parallel Laboratory for Learning Classifier Systems", *Proceedings of Fourth International Conference on Genetic Algorithms* - July 13-16, 1991 - San Diego - California, Morgan Kaufmann , San Mateo, CA.

Dorigo, Marco and Uwe Schnepf, "Organisation of Robot Behaviour Through Genetic Learning Processes", *Proceedings of Fifth International Conference on Advanced Robotics* - June 19-22,1991 - Pisa - Italy. IEEE.

Eastman, Charles M., "Recent Developments in Representation in the Science of Design", *Proceedings of the Eighteenth Association for Computing Machinery and Institute of Electronics and Electrical Engineers Design Automation Conference*, 1981, Institute for Electronics and Electrical Engineers, Washington, D.C.

Fitzpatrick, J.M. and Grefenstette, J.J., "Genetic Algorithms in Noisy Environments", *Machine Learning*, 1988, pp. 101-120.

Fleury, C., "Shape Optimal Design by Convex Linearization Method", *Optimum Shape, Automated Structural Design*, 1986.

Fox, R. L., *Optimization Methods for Engineering Design*, Addison-Wesley Publishing, Reading, MA, 1971.

Freeman, L.M., K.K. Kumar, C.L. Karr, and D.L. Meredith, "Tuning Fuzzy Logic Controllers Using Genetic Algorithms: Aerospace Applications", *Sixth Conference on Aerospace Applications of Artificial Intelligence*, Dayton SigArt American Computing Machinery, Dayton, OH, October 1990.

Fulton, Steven L. and Charles O. Pepe, "An Introduction to Model-Based Reasoning", *AI Expert*, January 1990, pp. 48-55.

Gage, Peter, *New Approaches to Optimization in Aerospace Conceptual Design*, Ph.D. Dissertation, Department of Aeronautics and Astronautics, Stanford University, 1994.

Gero, John, "Design Prototypes: A Knowledge Representation Schema for Design", *AI Magazine*, Vol. 11, No. 4, Winter 1990, pp. 26-36.

Glasgow, Barry and Elizabeth Graham, "Rapid Prototyping using Core Knowledge Bases", *AI Expert*, April 1988, pp. 54-63.

Goel, Vinod and Peter Pirollo, "Motivating the Notion of Generic", *AI Magazine*, Vol. 10, No. 1, Spring 1989, pp. 18-36.

Goldberg, David E., *Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning* Ph.D. Dissertation, University of Michigan, 1983.

Goldberg, David E., and M.P. Samanti. "Engineering Optimization via Genetic Algorithm", *Proceedings of the Ninth Conference on Electronic Computation*, IL, 1987, pp. 471-482.

Goldberg, D.D., *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, NY, 1989.

Goldberg, D.D., and M. P. Samtani, "Engineering Optimization via Genetic Algorithm", *Electronic Computation*, 1986, pp. 471-482.

Greene, David Perry, and Stephen F. Smith, "Using Coverage as a Model Building Constraint in Learning Classifier Systems", *Evolutionary Computation*, Vol. 2, No. 1, Spring 1994, pp. 67-91.

Haftka, R.T., and R.V. Grandhi, "Structural Shape Optimization - A Survey", *Computer Methods in Applied Mechanics and Engineering*, Elsevier Science Publishers B.V., North-Holland, 1986, pp. 91-106.

Harris, Larry R., "Hypothetical Reasoning", *AI Expert*, June 1989, pp. 56-59.

Holland, J.H. "Hierarchical Descriptions of Universal Spaces and Adaptive Systems", *Technical report ora projects 01252 and 08226*, University of Michigan, Ann Arbor, MI, 1968.

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

Holland, J.H., "Studies of the Spontaneous Emergence of Self-replicating Systems using Cellular Automata and Formal Grammars", *Lindenmayer, A. & G. Rozenberg (eds.), Automata, Languages, Development*, North-Holland, NY, 1976, pp. 385-404.

Holland, J.H. and J.S. Reitman, "Cognitive Systems Based on Adaptive Algorithms", in *D.A. Waterman and F. Hayes-Roth (eds.), Pattern-Directed Inference Systems*, Academic Press, NY, 1978.

Holland, John H., "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems", *Machine Learning: An Artificial Intelligence Approach, Volume II*, Michalski, Ryszard S., Carbonell, Jamie G., and Mitchell, Tom M. (eds.), Morgan Kaufman Publishers, Inc., Los Altos, CA, 1986.

Holland, John H., "Genetic Algorithms", Scientific American, July 1992, pp. 66-72.

Horn, Jeffrey, David E. Goldberg, and Kalyanmoy Deb, "Implicit Niching in a Learning Classifier System: Nature's Way", *Evolutionary Computation*, Vol. 2, No. 1, Spring 1994, pp. 37-66.

Hsu, Y.L, *Zeroth Order Optimization Methods of Two Dimensional Shape Optimization*, Ph.D. Dissertation, Department of Mechanical Engineering , Stanford University, 1992.

Jensen, Eric Dean, *Topological Structural Design Using Genetic Algorithms*, Ph.D. Dissertation, Purdue University, Lafayette, IN, 1992.

Jones, Jerald E. and Willian Turpin, "Developing an Expert System for Engineering", *Mechanical Engineering*, November 1986, pp. 10-16.

Karr, C.L., and D.E. Goldberg, "Genetic Algorithm Based Design of an Air-Injected Hydrocyclone", *Control 90, Mineral and Metallurgical Processing*, 1990, pp. 265-272.

King, Everett Gordon Jr., *Flow Vectoring of Supersonic Exhaust Nozzles Using a Genetic Algorithm to Define Optimally-Shaped Contours*, M.S. Thesis, Department of Aerospace Engineering, University of Alabama, Tuscaloosa, Alabama, 1991.

Kirkpatrick, S., C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, May 13, 1983, pp. 671-680.

Kling, R.M. and P. Banerjee, "Empirical and Theoretical Studies of the Simulated Evolution Method Applied to Standard Cell Placement", *IEEE Transactions on CAD*, Vol. 10, No. 10, October 1991.

Kodiyalam, Srinivas, Virendra Kumar and Peter M. Finnigan, "Constructive Solid Geometry Approach to Three-Dimensional Structural Shape Optimization", *AIAA Journal*, Vol. 30, No. 5, May 1992, pp. 1408-1415.

Kodratoff, Y. and R. Michalski (eds.), *Machine Learning, Volume III*, Chapter 21, Morgan Kaufmann, San Mateo, CA, 1990, pp. 611-638.

Kothawala, Kant S., Sharad V. Belsare, Mostafa Haririan and Juag K. Paeng, "Shaping Up: Optimization of Structural Designs", *Mechanical Engineering*, March 1988, pp. 52-55.

Koza, John R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

Lange, Mary (ed.), Working Smarter: Designing with Engineering Equations, Working Ideas, Milford, Ohio: SDRC, No.3, 1994, pp.1-7.

Le Riche, R. and R.T. Haftka, "Optimization of Laminate Stacking Sequences for Buckling Load Maximization by Genetic Algorithm", *AIAA Journal*, Vol. 31, No 5, pp. 951-956, 1993.

Mettler, Lawrence E., Thomas G. Gregg and Henry E. Schaffer, *Population Genetics and Evolution*, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1988.

Melloni, B.J., G.M. Eisner, and I. Dox, *Melloni's Illustrated Medical Dictionary*, The Williams & Wilkins Company, Baltimore, MD, 1979.

Minsky, Marvin L., "Steps Toward Artificial Intelligence", *Computers and Thought*, Feigenbaum, E.A. and Feldman, J. (eds.), McGraw-Hill, New York, NY, 1963.

Morawski, Paul, "Programming Bayesian Belief Networks", *AI Expert*, August 1989, pp. 74-79.

Noel, F., J.C. Leon and P. Trompette, "Shape Optimization of Three-dimensional Parts based on a Closed Loop between Structure Analysis and Geometric Modeling", *Engineering with Computers*, Vol. 11, 1995, pp. 114-121.

Parodi, A. and P. Boneli, "A New Approach to Fuzzy Classifier Systems", *S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, 1993, pp. 223-230.

Pike, Ralph W., *Optimization for Engineering Systems,* Van Nostrand Reinhold Company, New York, 1986.

Prabhakar, Vallury, *A Knowledge-based Approach to Model Geometry Idealization for Finite Element Analysis,* Ph.D. Dissertation, Department of Mechanical Engineering, Stanford University, 1994.

Punch, William F., Ronald C. Averill, Erik D. Goodman, Shyh-Chang Lin, and Ying Ding, "Design using Genetic Algorithms—Some Results for Composite Material Structures", *IEEE Expert,* (in press).

Reddy, Giridhar and Jonathan Cagan, "Improved Shape Annealing Method for Truss Topology Generation", *Design Theory and Methodology '94,* The American Society of Mechanical Engineers, 1994.

Richards, Robert A., "An Efficient Algorithm for Annealing Schedules in Boltzmann Machines", *Proceedings ofthe International Joint Conference on Neural Networks,* Lawrence Erlbaum Associates, 1990.

Richards, Robert A. and Sheri D. Sheppard, "Learning Classifier Systems in Design Optimization", *Design Theory and Methodology '92,* The American Society of Mechanical Engineers, 1992.

Richards, Robert A., *Macro Language & Program File QuickStart Guide,* Tensor Laboratories, Stanford, CA 1994.

Riolo, Rick L., *Empirical Studies of Default Hierarchies and Sequences of Rules in Learning Classifier Systems,* Ph.D. Dissertation, Computer Science and Engineering Department, University of Michigan, 1988.

Roark, Raymond J. and Warren C. Young, *Formulas for Stress and Strain, Fifth Edition,* McGraw-Hill Book Company, New York, 1982.

Roberts, G.R. "Dynamic planning for Classifier Systems", *S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, 1993, pp. 231-237.

Robertson, George G. and Rick L. Riolo, "A Tale of Two Classifier Systems", *Machine Learning*, 1988.

Rumelhart, David E., and James McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.

Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers", *IBM Journal of Research and Development*, 3, 1959, pp. 211-232.

Sandgren, E., E. Jensen and J.W. Welton, "Topological Design of Structural Components Using Genetic Optimization Methods", *Proceedings of the 1990 Winter Annual Meeting of the American Society of Mechanical Engineers*, Dallas, TX, 1991, pp. 31-43.

Schmidt, L.A., "Structural Design by Symmetric Synthesis", *Proceedings of the Second ASCE Conference on Electronic Computation*, Pittsburgh, PA, 1960, pp. 105-122.

Shigley, Joseph E, and Larry D. Mitchell. *Mechanical Engineering Design*, Fourth Edition, McGraw-Hill Book Company, New York, 1983.

Sriram, Duvvuru, George Stephanolpoulos, Robert Logcher, David Gossard, Nicolas Groleau, David Serrano and Dundee Navinchandra, "Knowledge-Based System Applications in Engineering Design: Research at MIT", *AI Magazine*, Vol. 10, No. 3, Fall 1989, pp. 79-96.

Stolfo, S.J., "Is CAD/CAM Ready for AI?", *Proceedings of the 1984 Pressure Vessels and Piping Conference and Exhibition*, San Antonio, Texas, June 17-21, 1984, ASME, NY.

Suzuki, K., and N. Kikuchi, "A Homogenization Method for Shape and Topology Optimization of a Linear Elastic Structure", *Computer Methods in Applied Mechanics and Engineering*, 1991.

Takeda, Hideaki, Paul Veerkamp, Tetsuo Tomiyama, and Hiroyuki Yoshikawa "Modeling Design Processes", *AI Magazine*, Vol. 11, No. 4, Winter 1990, pp. 37-48.

Timoshenko, S.P. and James M. Gere, *Mechanics of Materials*, Van Nostrand Reinhold Company, NY, 1972.

Valenzuela-Rendon, M., "The Fuzzy Classifier System: A Classifier System for Continuously Varying Variables", R. Belew & L. Booker (eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 346-353.

Vanderplaats, Garret N, and Kenneth D. Blakely, "The Best and the Lightest", *Mechanical Engineering*, February 1989, pp. 56-62.

Waterman, D.A., *A Guide to Expert Systems,* Addison-Wesley Publishing Company, Reading, MA., 1986.

Whitley, D., "The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best", J. Schaffer (eds.), *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, 1989, pp. 116-121.

Whitley, D., "An Executable Model of a Simple Genetic Algorithm", D. Whitley (ed.), *Foundations of Genetic Algorithms II,* Morgan Kaufmann, San Mateo, CA, 1993, pp. 45-62.

Whitley, D., "A Genetic Algorithm Tutorial", *Statistics and Computing,* 4, 1994, pp. 65-85.

Widmann, James Michael, *Structural Shape Optimization with Intrinsic Geometry and Adaptive Model Refinement,* Ph.D. Dissertation, Mechanical Engineering Department, Stanford University, 1994.

Wilson, Stewart W., "Classifier System Learning of a Boolean Function", Research Memo RIS No. 27r, The Rowland Institute of Science, Cambridge MA, 1986.

Wilson, Stewart W., "ZCS: A Zeroth Level Classifier System", *Evolutionary Computation,* Vol. 2, No. 1, Spring 1994, pp. 1-18.

Yang, Ching-Yu, and Patrick A. Fitzhorn, "A Functional Design Model for Shape Optimization", 1992, pp. 755-760.

Yang, R. J., "A Three-Dimensional Shape Optimization System — SHOP3D", *Computers & Structures,* Vol. 31. No. 6, 1989, pp. 881-890.

Zadeh, Lofti A., "Syllogistic Reasoning in Fuzzy Logic and its Application to Usuality and Reasoning with Dispositions", *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-15, No. 6, November/December 1985, pp. 745-763.

Zhao, Z., "Shape Design Sensitivity Analysis and Optimization Using the Boundary Element Method", Springer-Verlag, Berlin, 1991.

Zienkiewicz, O. C., and J.S. Campbell, "Shape Optimization and Sequential Linear Programming", Gallagher, R.H. and Zienkiewicz, O.C. (eds.), *Optimal Structural Design,* Wiley, NY, 1973.

Zienkiewicz, O. C., The Finite Element Method in Structural and Continuum Mechanics, McGraw Hill, NY, 1980, p. 8.