AN ABSTRACT OF THE THESIS OF

Giuseppe Cerbone for the degree of Doctor of Philosophy in Computer Science presented on April 13, 1992.

Title: Machine Learning in Engineering: Techniques to Speed Up Numerical Optimization

Redacted for Privacy

Abstract approved: ______________________________

Thomas G. Dietterich

Redacted for Privacy

____

Paul Cull

Abstract

Many important application problems in engineering can be formalized as non-linear optimization tasks. However, numerical methods for solving such problems are brittle and do not scale well. For example, these methods depend critically on choosing a good starting point from which to perform the optimization search. In high-dimensional spaces, numerical methods have difficulty finding solutions that are even locally optimal. The objective of this thesis is to demonstrate how machine learning techniques can improve the performance of numerical optimizers and facilitate optimization in engineering design.

The machine learning methods have been tested in the domain of 2-dimensional structural design, where the goal is to find a truss of minimum weight that bears a set of fixed loads. Trusses are constructed from pure tension and pure compression members. The difference in the load-bearing properties of tension and compression members causes the gradient of the objective function to be discontinuous, and this prevents the application of powerful gradient-based optimization algorithms in this domain.

In this thesis, the approach to numerical optimization is to find ways of transforming the initial problem into a selected set of subproblems where efficient, gradient-based algorithms can be applied. This is achieved by a three-step "compilation" process.

The first step is to apply speedup learning techniques to partition the overall optimization task into sub-problems for which the gradient is continuous. Then, the second step is to further simplify each sub-problem by using inductive learning techniques to identify regularities and exploit them to reduce the number of independent variables.

Unfortunately, these first two steps have the potential to produce an exponential number of sub-problems. Hence, in the third step, selection rules are derived to identify those sub-problems that are most likely to contain the global optimum. The numerical optimization procedures are only applied to these selected sub-problems.

To identify good sub-problems, a novel ID3-like inductive learning algorithm called UTILITYID3 is applied to a collection of training examples to discover selection rules. These rules analyze the problem statement and identify a small number of sub-problems (typically 3) that are likely to contain the global optimum.

In the domain of 2-dimensional structural design, the combination of these three steps yields a 6-fold speedup in the time required to find an optimal solution. Furthermore, it turns out that this method is less reliant on a good starting point for optimization.

The methods developed in this problem show promise of being applied to a wide range of numerical optimization problems in engineering design.

Machine Learning in Engineering: Techniques to Speed Up Numerical Optimization

by

Giuseppe Cerbone

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed April 13, 1992

Commencement June 14, 1992

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Professor of Computer Science in charge of major

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis presented April 13, 1992

Typed by Giuseppe Cerbone for Giuseppe Cerbone

ACKNOWLEDGEMENTS

Dedication

To my mother **Anna**,

to my father **Roberto** in *memoriam*,

and to *Camila* **Terry**.

Acknowledgements

Thomas G.Dietterich, my advisor, made this work possible. He constantly pushed me to do my best, provided insightful advice and constant assistance, and taught me the invaluable tool of a research methodology. I am in debt to him. Paul Cull helped me to get to Oregon State University and, along with his wife Elaine, provided valuable support at the beginning of this adventure. Moreover, Paul Cull provided *food for thought* in the theoretical analysis of divide-and-conquer. I also wish to thank the other members of my committee. Prasad Tadepalli provided useful feedback on EBL related issues and proof-read most of my papers. Michael Scanlan helped me get "logically" started on the 2-D design domain and kept asking me when I would finish. Gordon Reistad was always available for suggestions as the Graduate School Representative. David Ullman provided encouragement and most of the engineering knowledge and references. John Peterson was available to share his engineering expertise.

Thanks to Tom Dietterich, Peter Friedland, and Rich Keller, I spent the final five months of this project at NASA Ames Research Center as a visiting researcher with the Information Sciences Division.

It is also a pleasure to acknowledge people who made it possible for me to come to the United States from chaotic but beloved Italy. Luigi Maria Ricciardi, my advisor in Italy, put me in contact with Oregon State University. Roberto

Stroffolini encouraged me to go to the States. Both made great efforts in explaining the "advantages" of a PhD. A great part of my accomplishment is due to them.

Many friends in the States helped me through the ordeal. Hussein Al-Muallim provided friendly advice and technical competency. Vijay Srinivasan was very precise in his comments on drafts of this thesis. Nick Flann was a great source of information and provided invaluable feedback on learning issues. Terry Lutkavage sorted through the English in my papers, was of great moral support, and made possible our happy ending. My family provided a *safety net.* My mother Anna, my sister Imma, and my uncle Giuseppe encouraged me throughout.

Finally, I am thankful to the agencies that funded my research. NASA Ames Research Center provided funds under Grant Number NAG 2-630. The Italian National Research Council (CNR) awarded me a fellowship for studies abroad. The Department of Computer Science at Oregon State University awarded me one-half of a Tektronix fellowship and, thanks to George Beekman and to Walter Rudd, hired me as a teaching assistant and as a summer instructor.

Table of Contents

List of Figures

Figure Page

List of Tables

Machine Learning in Engineering:
Techniques to Speed Up Numerical Optimization

Chapter 1

Introduction

The objective of this thesis is to demonstrate how machine learning techniques can improve the performance of numerical optimizers and facilitate optimization in engineering design.

1.1 Motivations

The search for optimality is innate for humans. In the age of space stations and of ever increasing economical concerns, engineers must design efficient artifacts. For example, building a space station will require the transportation of a very large amount of structural material. This must be light enough to be lifted into orbit without exceeding payload capacity, must be cost effective so that the transport is completed in a few launches, and must be strong enough to withstand all stresses in space.

Closer to earth, civil engineers often design bridges that can withstand the weight of vehicles and support structures (trusses) like roofs that do not crash

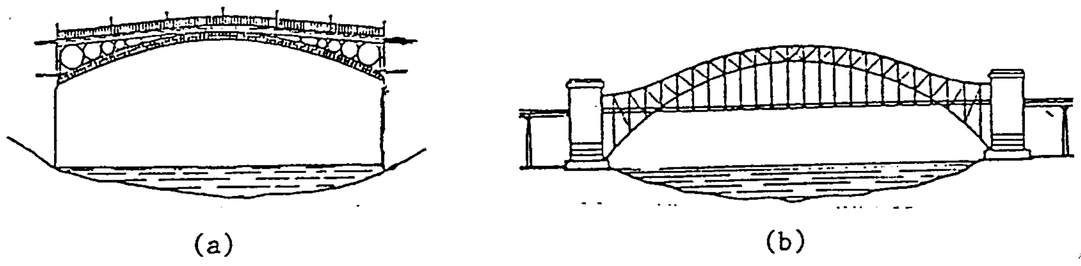


Figure 1. Bridge configurations, (a) Arc, (b) Cable-stayed.

under heavy loads, for instance a snowfall. In addition to these minimal stability requirements, the engineers must take into consideration economic factors which require that the artifact (bridge or truss) must be the lightest and cheapest possible.

The optimality requirements greatly increase the level of complexity in a design task. The constant evolution of shapes and the search for lighter and stronger materials are evident in centuries of efforts by engineers and architects to design optimal artifacts. In order to provide solutions to these problems, computer scientists and engineers are faced with the challenge of automating the overall design phase and incorporating the optimality requirements during the process and not after the artifact has been designed. The goal is the construction of intelligent Computer Aided Design (CAD) systems (Hagen and Tomiyama 1987) to help the engineer build artifacts which are as close to the *best* as possible. However, to do this efficiently, a number of problems with existing techniques must be overcome. First, the *knowledge acquisition* bottleneck (Feigenbaum 1977) prevents the use of traditional software engineering methodologies. Second, current numerical optimization techniques are too brittle and slow to be applicable to many practical tasks. For these reasons, the automation of design has recently been one of the most active fields of research in the Artificial Intelligence (AI) community and especially in the Machine Learning (ML) (Chien et al. 1991) circles. This thesis combines novel ML strategies with traditional symbolic transformation techniques to overcome some of the drawbacks of the software engineering approach and to speed up and increase the reliability of traditional numerical optimizers. Therefore, this work is a contribution to the research on intelligent automated design tools and, to our knowledge, it is one of the first attempts to use ML techniques to aid in optimal design. The thesis also constitutes a step towards our medium to long term goal of constructing truly intelligent and automated CAD tools which output a small set of alternative *near-optimal* designs from a provided specification. Such tools relieve the human designers from the burden of generating alternative designs from the same specifications. This, in turn, allows the engineers to focus on the higher level intellectual activity of refining the array of options generated by the CAD system.

1.2 Optimal Design

Engineering design is a complex activity that comprises two intertwined phases: analysis and synthesis. Analysis is the study of the behaviour of the artifact and, in most engineering tasks, it is a science founded on well-established mathematical techniques. On the contrary, synthesis is the process of creating the artifact and it is still largely considered an art. In a typical design episode, the engineer uses an iterative design-redesign approach. After an initial study of the abstract requirements for the artifact, a *conceptual* design is proposed. This initiates an iterative process during which this design is refined using feedback from the analysis until a satisfactory answer is reached.

Optimality considerations influence all stages of the design process, because a design can be improved in many different ways. For the sake of simplicity, let us assume that our goal is to build a lightweight bridge. Once an initial stable design (see Figure 1a) has been reached, a bridge can be made lighter simply by using a material with the same resistance and lower density. On the other hand, engineers have found that they can greatly decrease the weight of a bridge (and its cost) by using a radically different design which resembles a child's swing and uses light cables rather than heavy columns. Recent cable-stayed bridges (see Figure 1b) are examples of this latter approach. A simple change of materials would not have induced a change in the shape of the bridge. Instead, to obtain the novel design, engineers incorporated the optimality requirement during synthesis.

In this thesis, although we acknowledge the great importance of the science of materials, we simply refer the reader to specialized references (Draffin and Collins 1950) for advances in this field. Instead, we are interested in automating the synthesis of configurations that produce optimal artifacts subject to given evaluation criteria. Further improvement on these configurations can then be obtained by choosing the appropriate materials.

Before we proceed, it is important to point out that, in the opinion of most, absolute optimality is utopian. This becomes especially true in engineering design because of the lack of tools which ensure the global optimality of a solution. In this regard, Vanderplaats (Vanderplaats 1984) states a motto similar to Murphy's law:

> ...*Expectations of achieving the absolute "best" design will invariably lead to "maximum" disappointments.* ...

With this in mind, in the remainder of this chapter we first consider the problems to be overcome in automating optimal design and then briefly survey our solutions.

1.3 Computers in Optimal Design

In most cases, the analytical phase of engineering design is a well-defined (Mittal and Araya 1986) task. On the other hand, the synthesis of the (optimal) artifact is an ill-structured problem (Newell 1966) especially in the early conceptual stages.

The current computer technology to aid engineers in the design process reflects the dichotomy between analysis and synthesis. On the one side, there exist widely used numerical packages that perform analysis using finite element techniques. These analyses are often presented to the user in highly graphical format by complex CAD tools which can visualize the artifact to be analyzed but are not capable of making autonomous design decisions. On the other side, the research to automate the synthesis of artifacts is still far from wide acceptance. This is especially true when the design requirements include optimization. Early work in expert systems for design show that many problems still exist. Furuta, Tu and Yao (1985) review expert systems in design and conclude that:

> *...At present, the computer is a very common tool in structural engineering, but it is mainly used for structural analyses [...]. However, an expert system can provide substantial assistance to more complicated or creative works which are usually not completely or well defined ...*

Later efforts by Duffey and Dixon (1988) with their DOMINIC expert system and by Nevill, G.E. Jr., Garcelon, J.H. et al. (1989) with MOSAIC are a step towards the mechanization of the design synthesis process but ignore optimality requirements. In fact, Duffey and Dixon state that:

> *...The designs produced [by DOMINIC] are generally acceptable, but not necessarily optimal ...*

In building expert systems, optimality requirements add a layer of complexity to the knowledge transfer process. Not only must the expert verbalize rules on how to build an artifact, but the rules must ensure the *best* design. Therefore, one of the main factors for the limited success of expert systems in optimal design is the well-known *knowledge acquisition* (Feigenbaum 1977) bottleneck. In this regard, Duffey and Dixon state:

> *...The human designer, especially in the preliminary or conceptual stages of the design, tends to use knowledge in a way that is difficult to articulate and systematically define, let alone transform into a computational model for automatic design ...*

At present, optimality is introduced in the design phase in a rather asynchronous fashion and only recently (Papalambros and Chirehdast 1990) have researchers built design systems that incorporate optimality requirements in the design-redesign cycle. Typically, engineers cast the optimization problem into a well-defined framework for one of the numerous existing numerical optimization packages (Papalambros and Wilde 1988) and then analyze the results of the numerical optimization either by hand or via CAD systems (Burns 1989). These results are then utilized to modify the design which is then subject to subsequent iterations. This procedure is extremely slow and is prone to errors. These problems prevent the incorporation of optimization techniques into the real-time environment of intelligent CAD systems.

Even when numerical optimizers are used, they are limited by a series of drawbacks (Vanderplaats 1984) which prevent their applicability in real domains. The computational time of the optimization methods increases with the number of independent variables—*dimensionality.* In addition, the increase in dimensionality also triggers a decrease in the ability of the methods to locate global minima. This is a serious flaw in practical applications, since they involve thousands of independent variables. As an example, let us consider the design of a bridge in which each connection point among structural members is a design variable and, consequently, requires at least a pair (x and y coordinates) of independent variables for the numerical optimization task. As we can all notice, there are thousands of such connections in the smallest of bridges, whereas current numerical optimizers can hardly (Pike 1986) handle 50 independent variables.

Furthermore, optimization algorithms have difficulties in dealing with highly discountinuous functions which arise in mechanical engineering design. In these situations, the methods might converge slowly or not converge at all. As shown for instance in Figure 7, practical engineering functions are highly irregular. Robustness is another problem. Seldom do the numerical methods ensure that a solution is a global minimum and the user of the method must restart the program from many different initial points to be confident that the solution is a global one. Obviously, in practical applications this requires an enormous amount of the engineers' time which, we trust, they would rather spend on other tasks.

In conclusion, although there is a substantial need in the engineering community to create systems that can produce optimal designs in real-time, the knowledge acquisition bottleneck and the drawbacks of numerical optimizers prevent the construction of intelligent CAD tools for optimal design. Therefore, researchers are exploring new ways to let the computer acquire (*learn*) knowledge with minimum human interaction and to improve the performance of numerical optimizers. A promising approach to this problem is Machine Learning which is the main focus of our work.

1.4 Machine Learning

A common definition of learning is: "*the acquiring of knowledge or skill*" and Simon (Simon 1983) relates learning and, hence, the acquisition of knowledge to "*changes in a system that enable [it] to do the same task or tasks drawn from the same population more efficiently and more effectively next time.*" But how do these changes occur? How do engineers learn to build artifacts they have never seen before? How do they improve their problem solving skills which allow them to be more proficient over time? Answers to these questions may come from a variety of research activities that explore the human mind both from a physiological and psychological standpoint. However, some useful answers have been forthcoming. Researchers in Artificial Intelligence are building software tools that emulate learning on traditional computers and, more recently, on neural networks (McClelland and Rumelhart 1986).

To simplify the learning task, computer scientists have clearly separated the acquisition of new knowledge from its application during problem solving. During the acquisition phase, the system *learns* the knowledge it needs either to solve problems or to increase the speed at which it derives solutions. The knowledge is then given to the problem solver which, at run time, matches the given task to the appropriate knowledge and applies it to derive a solution. This is the problem solving model we adopt in this thesis. The division between learning and problem solving has, among others, the effect that the knowledge acquisition process can be performed off-line. The model we assume is not atypical. We can argue that most of human learning happens in a similar fashion; for instance, we first go to school and, we hope, later apply the notions and processes we have learned. Amongst learning devices, neural networks are the best example; the weights in the network must be computed (learned) from a set of user-supplied examples before the network can be used to, for instance, classify hand-written characters. Symbolic learning algorithms work in a similar fashion; knowledge is generated (learned) via a "reasoning" mechanism separated from problem solving.

In the problem solving model we have assumed, learning is an off-line process. One first learns and then applies the knowledge to solve problems. This is analogous to the compilation of, say, a C program. The obvious difference being that the output of the compiler is, in most cases, machine readable code that will be later used to solve a problem whereas the output of the learning program is knowledge. In contrast, interpreters of computer programs give the opposite view in which compilation and problem solving are intertwined. One of the main benefits of separating learning and problem solving is that, within certain limits, the time required to learn can be neglected; hence we assume that learning is free of cost. The analogy between learning and compilation is much stronger. In fact, Mostow (1989) defined the term *knowledge compilation* to reflect this similarity.

However, Mostow's definition of knowledge compilation as the *transformation of explicitly represented knowledge about a domain into an efficient algorithm for performing some task in that domain* does not include other learning strategies. Shavlik and Dietterich (1990) fill the gap by suggesting that a system can learn (change) by:

- modifying itself to exploit its knowledge more effectively
- acquiring new knowledge from external sources.

The first strategy is often known as *speedup learning*, *skill acquisition*, or *knowledge compilation*. The second strategy is known as *inductive* or *empirical learning*. In this thesis we show how both learning strategies can be effectively used in a uniform fashion to overcome some of the problems posed by the knowledge acquisition bottleneck and some of the drawbacks of traditional (non-learning) numerical optimizers. Ultimately, we shall see how machine learning techniques help in designing optimal artifacts in the domain of structural design briefly described in the next section.

1.5 Task Description

The task we are using to demonstrate our techniques can be summarized as follows:

Given:
- A 2-dimensional (2-D) region R
- A set of stable points (supports)
- A set of external loads with application points within R

Find:
- Number of connection points and members
- Connectivity of members
- Position of all connection points,

such that the structure has minimum weight and is stable.

Figure 2 shows an example problem in which L is the load and S1 and S2 are two supports. The so-called *topology* is given as a graph structure containing four edges (the members) and four vertices (the load, the two supports, and an intermediate connection point C). The topology does not specify the lengths of the members or the location of C, but it does indicate the number of connection points and how the members are connected. The goal of the design process is to find the best topology and location of the connection points. For this example, the optimal solution is the one with a single connection point and four members connected as shown in Figure 2. In this solution, members E1 and E3 are in tension (they are being "stretched"), while members E2 and E4 are in compression. Tension members will be referred to as "rods" and indicated by thin lines. Compression members will be referred to as "columns" and indicated by thick lines.

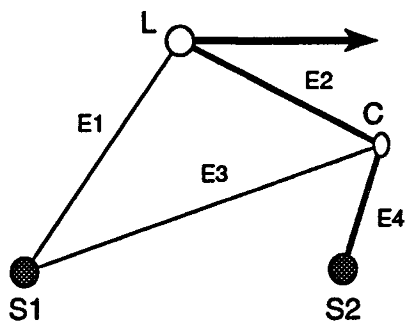


Figure 2. A 2-D structural design problem.

This task is actually only one step in the larger problem of designing good structures. As suggested by Palmer and Sheppard (1970) and Vanderplaats (1984), the strategy is composed of three successive steps. From an AI standpoint, these can be formulated as searches in separate search spaces corresponding to different levels of abstraction of the problem. Two of these spaces are shown in Figure 3. The search starts in the space of topologies to determine the number of connection points and the connectivity of members disregarding the position of the connection points. The topological space is discrete but infinite because one can add connection points *ad infinitum.* This search is purely qualitative and, currently, no automated tools exist[1] to aid the engineer who can only use her/his expertise to

[1]We acknowledge later in this thesis a few efforts, besides ours, to automate the exploratory stage in the skeletal design domain.

navigate in this search space. Once the topology has been chosen, the second step is to determine the locations of the connection points in the 2-D region (and hence the lengths, locations, internal forces, and cross-sectional areas of the members) so as to minimize the weight of the structure disregarding the shape and composition of members. Each point in 2-D is identified by its cartesian coordinates. Thus the cardinality of the search space for the numerical optimizer is $n = 2\ p$, where p is the number of connection points. In theory, this task can be solved using numerical non-linear optimization techniques. In practise, however, the limitations of these techniques prevent their applicability. The focus of this thesis is a series of machine learning methods to overcome some of these limitations. A third and final step in the process optimizes the shapes of the individual members. This can often be accomplished by linear programming and is not further discussed in this thesis.

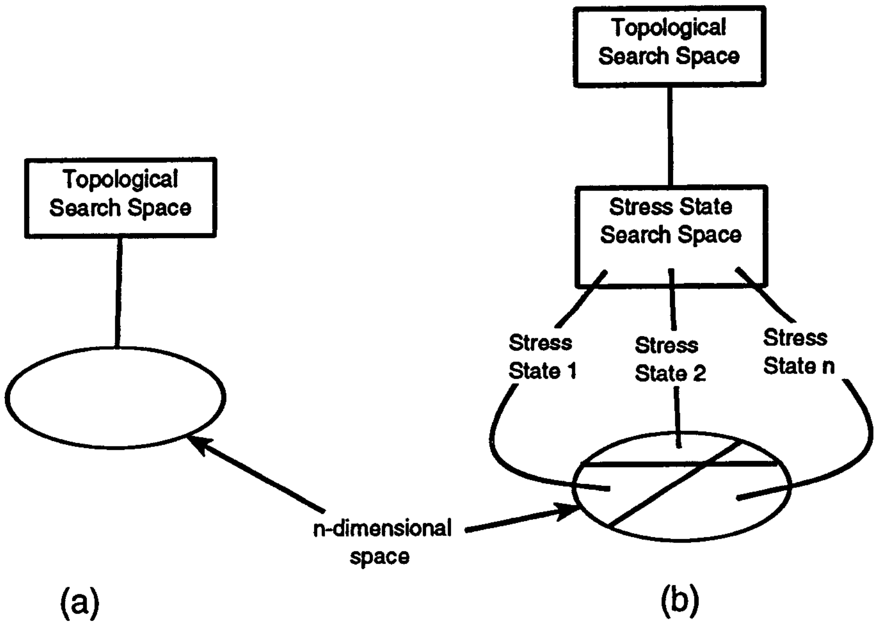


Figure 3. Search spaces for the skeletal design problem.

To provide a tractable testbed for our techniques, we have introduced a series of simplifying assumptions (see Appendix A) which allow us to compute the objective function for the optimal design problem by a three-step process. The first step is to apply the *method of joints* (see Section 2.1.3) to determine the forces operating in each member. Once this is known, the second step is to classify each member as compressive or tensile, which is important, because compressive and tensile members have different densities. The third step is to determine the cross-sectional area of each member. The load that a member can bear is assumed to be linearly proportional to its cross-sectional area. Finally, the weight of each member can be computed as the product of the density of the appropriate material, the length of the member, and the cross-sectional area of the member.

The last two steps can be collapsed into a single parameter c: the ratio of the density per-unit-of-force-borne for compressive members to density per-unit-of-force-borne for tensile members. With this simplification, we can minimize the function

$$Weight = \sum_{\substack{\text{tensile}\\ \text{members}}} \|F_i\|\, l_i + c \sum_{\substack{\text{compressive}\\ \text{members}}} \|F_i\|\, l_i, \tag{1.1}$$

where F_i is the force in member i, and l_i is the length of member i. This $Weight$ function is proportional (see Appendix B) to the actual weight of a structure and represents the objective function that is used throughout this thesis.

There are several reasons why this task is particularly appealing to demonstrate the suitability of machine learning techniques to optimal design. First, given our assumptions, the task is simple enough that it does not require highly specialized engineering knowledge; therefore, it is easily conveyed to most audiences. Second, the algorithm used to analyze the structure (method of joints) is straighforward and yet computationally expensive. Third, to our knowledge, an algorithmic solution for the design of lightweight (optimal) structures does not exist. Fourth, the optimization techniques required to minimize the objective function are non-linear (Friedland 1971) and require a large number of evaluations of the objective function. Fifth, the objective function is non-differentiable. Finally, with the help of an abstraction called *stress state*[2], it is possible to divide the optimization problem into independent subproblems.

Now that we have defined the design task, justified its choice, and formulated the non-linear optimization problem, let us turn to an overview of the solution.

1.6 Overview of Solution

A typical sequence of steps adopted by a problem solver to optimize a design is illustrated by the dashed lines in Figure 4 on the left. First, the engineer uses her/his own knowledge and experience to formulate a numerical task. Second, where possible, numerical optimization techniques are used to produce an optimal solution. Numerical optimization is typically a slow and brittle process. This is due to the fact that most numerical methods are hillclimbers. Thus, their speed is greatly affected by the number of evaluations of the objective function and by the time required for each evaluation. As stated at the beginning of this thesis, one of our goals is to produce fast optimization tasks. In traditional optimization, the engineer bears the burden of improving the performance of an established numerical method. This requires that s/he spends valuable time in finding a suitable starting point and in performing algebraic computations to reformulate the optimization task in way that is suitable to a numerical method. An automated tool must relieve this burden from the human. In our solution, this is accomplished by letting the engineer specify the objective function in an abstract format. However, while the abstract formulation of the problem simplifies the engineer's task, it can greatly slow down the numerical solution. This is because most optimizers are general-purpose and have no knowledge of the problem domain. Therefore, at run time they use the same objective function provided by the engineer for all regions. In addition, it is more likely that abstract formulations are non differentiable. This prevents the use of powerful gradient-based numerical techniques

[2]David G.Ullman is responsible for the name.

– only slower function-based methods are applicable. This is especially true in the design task we are tackling. As an example, the abstract formulation of the weight in Equation 1.1 is non-differentiable, because (a) the expression for forces and lengths are not known before the connection points are located, and (b) forces are included as absolute values.

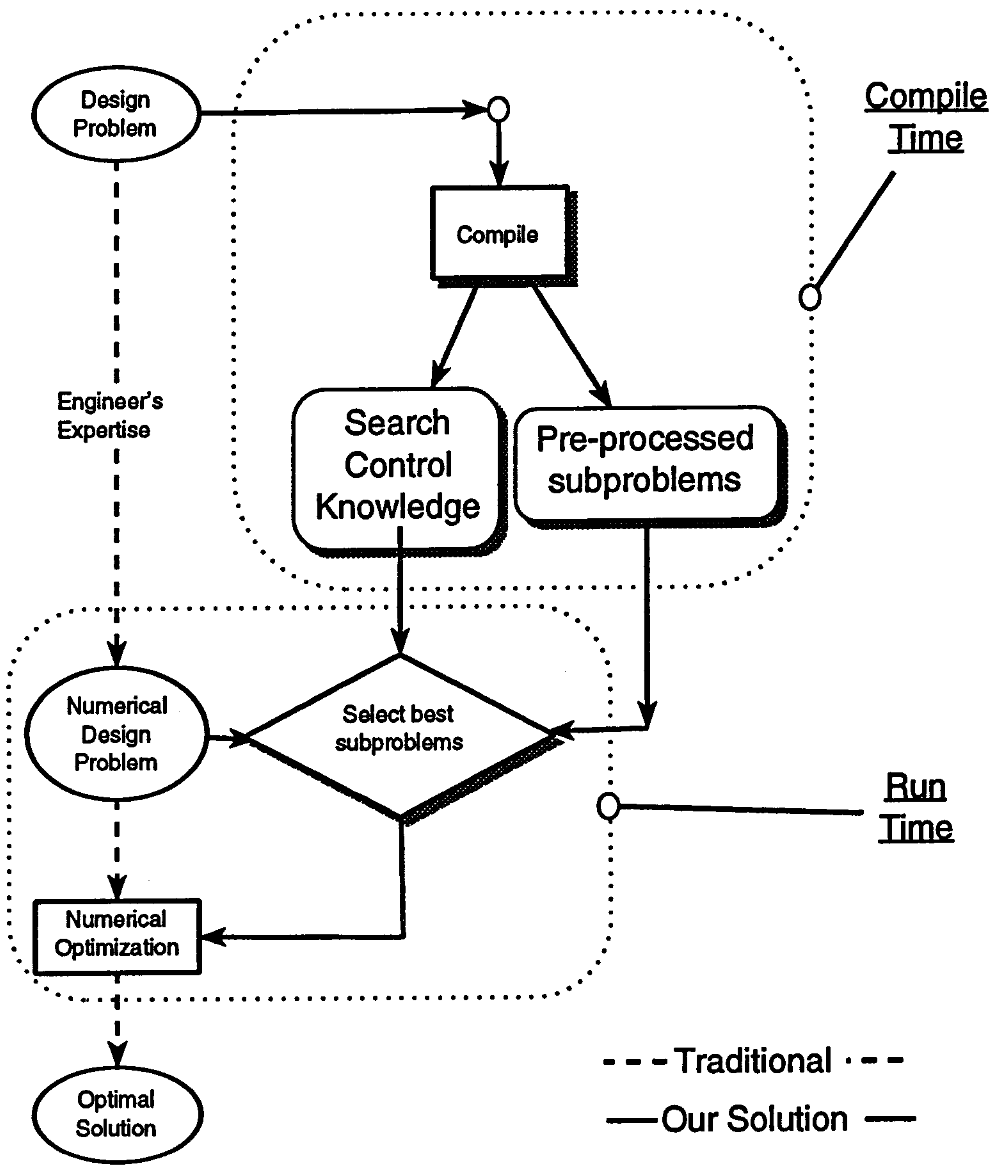


Figure 4. Traditional and new problem solving strategies.

This thesis augments the traditional problem solving schema with the off-line knowledge compilation (or *learning*) stage (Cerbone 1992) illustrated by the solid lines in Figure 4 on the right. The compiler uses a blend of novel and traditional machine learning techniques to increase reliability and speed of the numerical optimization task. These results are accomplished by partitioning at compile time the design problem into subproblems and by deriving:

- Pre-processed objective functions for each subproblem
- Search control knowledge that allows the problem solver to focus only on a few subproblems.

The pre-processed functions contain fewer independent variables and have been greatly simplified. Therefore, they are faster to evaluate. At run time, the problem solver uses the search control knowledge derived during compilation to retrieve a few candidate solutions. Each of these candidates is then given as input to a numerical optimizer. However, in this case, the optimizer is given a simplified objective function. The net result is a speedup of as much as 95% over the run time of the traditional methods and a more reliable numerical optimization process. Being an off-line computation, compilation does not introduce any overhead on the run time operations. In the remainder of this section, we outline the compilation steps.

Figure 5 outlines the compilation strategy. The figure also indicates the methods used at the various stages, and the chapters in this thesis in which they are detailed.

Divide and Conquer. The design problem is partitioned into independent subproblems using a *divide-and-conquer* methodology. As the graph in Figure 6 illustrates, once the number of loads and supports has been fixed, the design task is partitioned according to topologies and stress states. These abstract configurations are generated automatically by the compiler. No user interaction is needed. A problem to be overcome with this strategy is that the number of subproblems (stress states) can be exponential. In fact, while Friedland (1971) demonstrated empirically that only a few topologies need be explored, the number of stress states

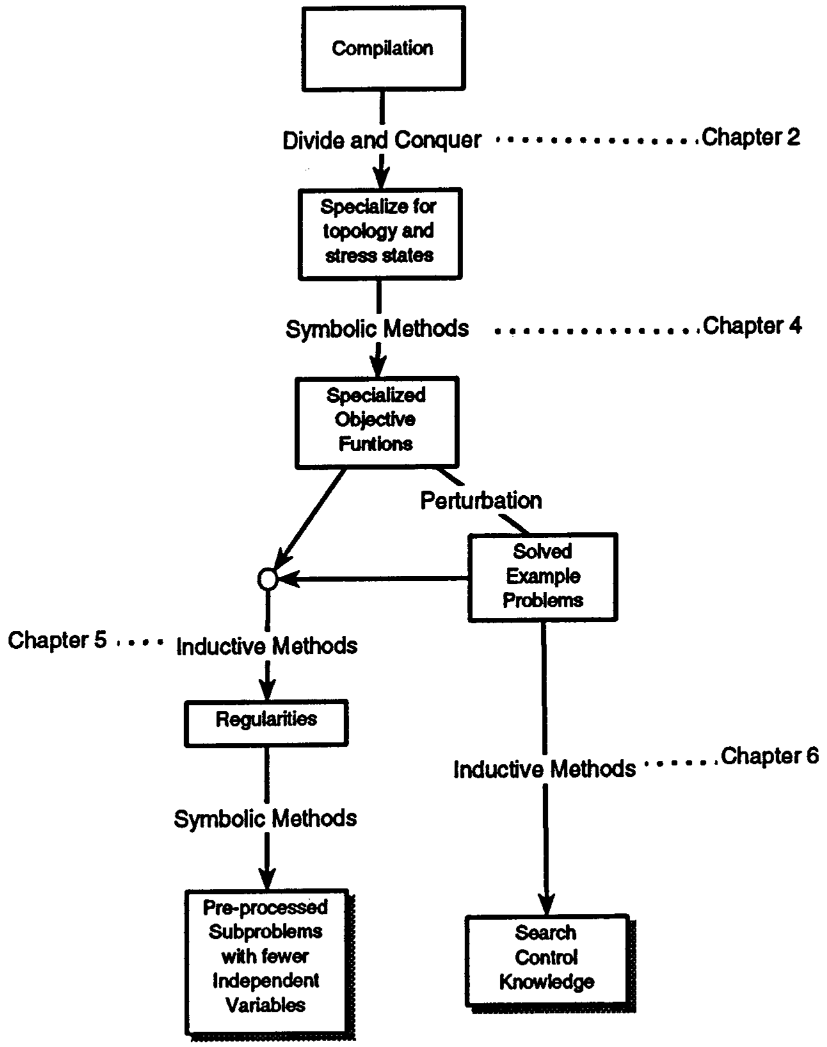


Figure 5. Learning stages and methods.

per topology can be exponential (although most stress states are either unfeasible or need not be explored). Later in this section we shall see how to derive search control knowledge to focus on a few stress states that will likely lead to an optimal solution. Each subproblem generated by this procedure is independent from all others and can be solved on a separate processor. Thus the global minumum is obtained by taking the best solution among all subproblems.

The specialization to a topology and stress state also simplifies the numerical

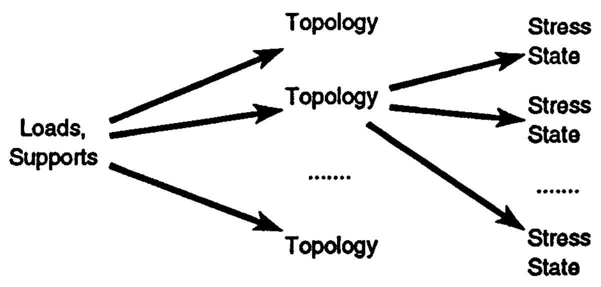


Figure 6. Divide and Conquer applied to the skeletal design task.

optimization task. We conjecture that this is true because the specialization induces a convex region in the n-dimensional space. Figure 7 plots the weight of the structure in Figure 2 as a function of the coordinates (x, y) of the connection point C. As the plot indicates, there are at least three convex regions, one for each stress state. For instance, region R1 in Figure 7 corresponds to a unique topology and stress state. Moreover, all other unimodal regions in the figure correspond to different stress states. Most numerical optimizers are unable to determine a global maximum, because they cannot "jump" from one region to another. On the other hand, when the optimization process is specialized to one stress state, the local minimum will most likely be the global minimum in the region as well. The global minimum for the function as a whole can then be computed as the mimum among all regions. It is also important to notice each objective function restricted to a stress state (or, equivalently, a region in the figure) is differentiable, because the expressions of forces and lengths can be made explicit.

Altough our research was initiated and conducted independently, our divide-and-conquer approach follows the current line of research in building automated scientific assistants (Cook 1991), (Keller 1991). The goal of this research is to relieve the user (practising engineer or scientist) from the low-level algebraic and numerical manipulations to allow her/him to focus on conceptual tasks. In our domain, this means that the engineer need only input the method of joints and the abstractions (topology and stress state) needed to partition the task. In a similar

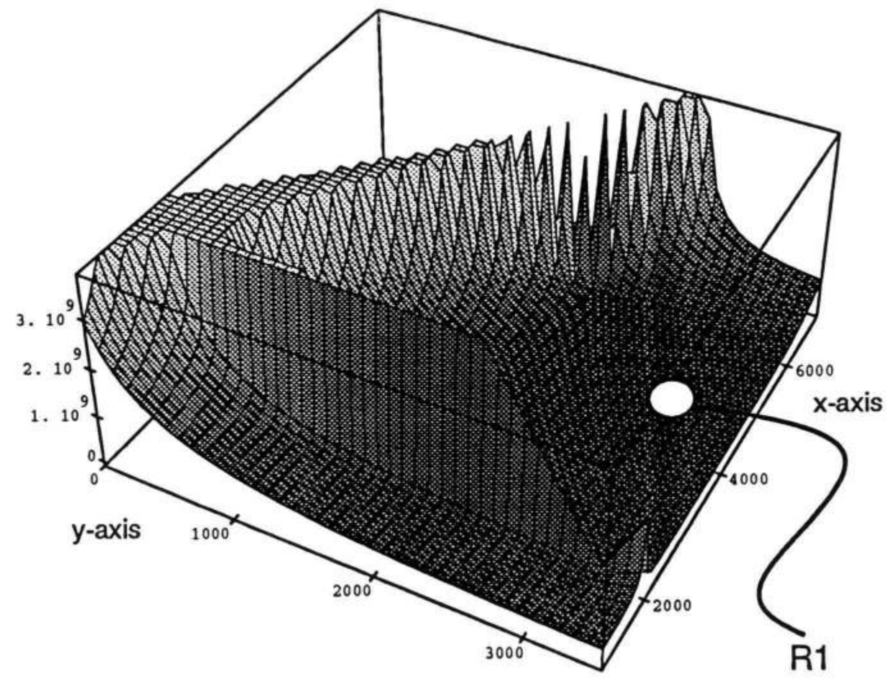
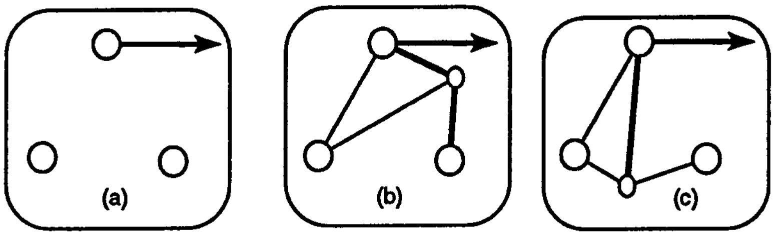


Figure 7. Weight of the structure in Figure 2.

fashion, the ALPAL (Cook 1991) system in use at Lawrence Livermore National Laboratories allows a physicist to input an integro-differential equation and the size of the solution grid –abstraction– and then it specializes the numerical problem to each grid. The Sigma (Keller 1991) project at Nasa Ames Research Center is another significant effort towards scientific modelling assistants.

Symbolic Methods. The second compilation stage in Figure 5 specializes the objective functions from the previous divide and conquer step. During this step, knowledge of the topology and stress state is incorporated into the method of joints and into the objective function. The result is a set (one for each stress state) of objective functions in closed form. Each function is faster to evaluate than the original one and is differentiable; thereby permitting the use of gradient-based run time optimization methods. (Gradient methods typically require fewer evaluations of the objective function to produce a result.) This specialization is performed using traditional compiler optimization techniques such as *partial evaluation* and *loop unrolling* (or *loop unfolding*). Moreover, algebraic simplification is performed by a compiler we have written using the language provided by `Mathematica` (Wolfram 1988), the off-the-shelf symbolic manipulation package.

To visualize this compilation step, the following figure illustrates a design problem (a) and two stress states – (b) and (c)



In this example, the topology (one connection point and the connectivity indicated in the figure) is known and it is incorporated into the objective function. With this knowledge and using loop unrolling, the compiler derives a system of linear equations in symbolic form. The system of equations is the core of the method of joints and the solution is necessary to compute the force in each member. Symbolic algorithms are then used to solve the system of equations and to obtain a closed form expression for the internal forces. Knowledge of the topology also allows the compiler to derive the symbolic expression for the length of each member. Forces and lengths are then substituted into the abstract objective function

$$Weight = \sum_{\substack{\text{tensile}\\ \text{members}}} \|F_i\|\, l_i + c \sum_{\substack{\text{compressive}\\ \text{members}}} \|F_i\|\, l_i.$$

The resulting expression is still not differentiable because the signs of the forces (stress state) are not known. Thus the compiler generates stress states and incorporates them into the function. At this point, another simplification is performed by substituting the givens of the problem (loads and support positions) into the symbolic expressions. Finally, the compiler further simplifies the objective function using partial evaluation techniques. The result is a differentiable mixed symbolic/numeric expression such as

$$\begin{aligned} Weight = & \left(1.14\ 10^{13}x - 5.66\ 10^{9}x^2 + 8.16\ 10^{5}x^3 + 3.28\ 10^{13}y - 3.26\ 10^{9}xy + \right. \\ & \left. 2.44\ 10^{5}x^2y - 6.70\ 10^{9}y^2 + 8.16\ 10^{5}xy^2 + 2.44\ 10^{5}y^3 - 4.08\ 10^{16}\right) / \\ & \left(1.28\ 10^{1}xy - 2.56\ 10^{4}x + 2.56\ 10^{4}y - 6.40\ y^2 - 2.56\ 10^{7}\right). \end{aligned}$$

This closed-form expression is faster to evaluate than the original objective function and it is differentiable. This allows the problem solver to use powerful gradient-based optimization techniques.

Generating example problems. Solved example problems (See Figure 5) are generated automatically by the compiler. Given the specialized objective functions derived at the previous step, the system randomly generates loads and support positions. Each of these problems is then solved off-line and the solution is recorded.

Eliminating independendent variables. The third step in Figure 5 produces a further speedup and increases the reliability of numerical optimizers. This is obtained using inductive methods to decrease the number of independent variables (*dimensionality*) in the numerical optimization problem. The reduction of the number of independent variables is performed at compile time by discovering *regularities* inductively. Regularities are relationships among variables and are used (at compile time) to simplify the optimization problem. In fact, when the optimizer searches within a region, it might turn out to be superfluous to search along all dimensions because there might exist a regularity between, say, coordinates and known quantities (e.g., location or magnitude of loads and location of supports.) Once a regularity is determined, it is incorporated into the objective function. This, in turn, has the effect of eliminating one or more independent variables.

Figure 8 shows an example of a regularity. The angle $\mathtt{C}\ \widehat{\mathtt{S1}}\ \mathtt{S2}$ is one half the an-

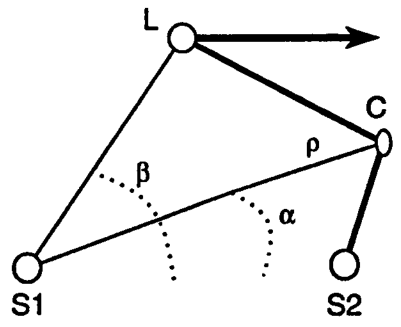


Figure 8. An example of a regularity.

gle L1 $\widehat{\text{S1}}$ S2. The latter is a known quantity because it involves load and support positions. Hence, with a little algebra, we can compute one of the coordinates of C. Using a polar coordinate system, the objective function can then be transformed into an equation whose unknowns are an angle and a distance, instead of x and y – cartesian coordinates. Once the function is in polar coordinates, the regularities among variables are used as constraints and are incorporated into the objective function. The symbolic manipulation routines are then used again to perform algebraic simplifications. The result is an objective function with fewer independent variables. This, in turn, produces an even simpler and faster optimization problem. For instance, the *Weight* shown in the previous expression induces a 2-dimensional optimization task in the variables x and y. After simplification, the number of independent variables is reduced to 1 (e.g., the distance ρ of point C from support S1.)

As reported in Chapter 5, regularities are discovered using a blend of Explanation-Based Learning (EBL) (Ellman 1988) and statistical regression techniques to analyze optimal solutions to design problems.

Learning Search Control Knowledge. Partitioning the optimization task according to topologies and stress states greatly improves the running time of the optimizers on each objective function. On the other hand, the specialization might introduce a large number of candidate solutions; in principle, this number can be exponential in the number of members in the structure. To overcome this problem, we have devised a new inductive learning method (Cerbone and Dietterich 1992a) to prune candidates that do not lead to optimal solutions. This method, along with more traditional machine learning techniques, is fully explained in Chapter 6. As shown in Figure 5, our method learns search control knowledge from solved examples. The learning algorithm outputs decision trees which can then be quickly transformed into IF-THEN-ELSE rules. These rules (or, equivalently, decision trees) map features of the problem to a *set* of stress states for a topology. The features are characteristics of the design task and are computed by a set of procedures given to the learning algorithm. (The automatic derivation of features is a problem for future research.)

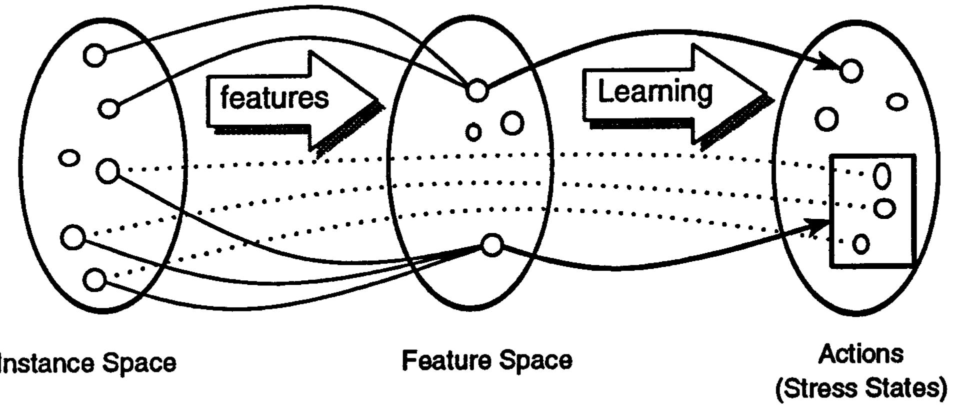


Figure 9. Associating stress states to problem features

Ideally, the learning algorithm should associate a single stress state to each problem description. However, the ability to associate features to a single stress state requires that the features be chosen so that there is a one-to-one association between problem description and solution. This in turn, implies that the stress state selected by the learning algorithm is a solution for all problem instances with the same feature values. This assumption (shown on the top part of Figure 9) is too restrictive in many domains because these features either do not exist or they are too expensive to determine. Instead, as it is shown on the bottom part of Figure 9, in most cases it is possible to associate a set of candidate solutions to feature values. The optimization problem in the skeletal design domain is an example of such a task. In fact, even professional engineers are unable to map problems into a single optimal stress state.

Existing learning algorithms are not designed to learn rules which map problem descriptions into sets of actions. For instance, the ID3 algorithm (Quinlan 1987) can map features into one stress state. In the design domain, the solution proposed by ID3 does not always yield optimal solutions. In fact, as shown in Chapter 6, it can produce solutions that are up to 4 times the global optimum. To overcome this problem, we relax the requirement that the features identify a single stress state and simply require that the learning algorithm chooses a set of stress states.

A trivial solution to this problem would output the set of all possible stress states. This would lead to unacceptable performances at run time, of course. Thus, we also require that the set of stress states minimizes (or maximizes) a given criterion. Our approach is based on the consideration that in practical applications, rather than absolute optimality it is often sufficient to derive a satisfactory solution quickly. Time to obtain a solution and its quality[3] are then combined to define a utility function. This is the criterion that the learning algorithm will maximize to select the set of stress states. Traditional learning algorithms ignore this quality/time tradeoff. In Chapter 6 we formalize this novel learning task and provide UTILITYID3, a new learning algorithm. Estimates of the utility for each stress state are based on the example problems generated by the compiler.

To provide this information, the training examples have a complex format such as

```
N T L   (57 5.61 4.0029) (66 9.0 1.0).
```

This example indicates that we extracted three features whose values are N, T,

[3]In our domain the quality is measured as the ratio between the solution attained by the numerical optimizer and the optimal one.

and L, respectively, from the original design problem. Moreover, two stress states – 57 and 66 – were solved to determine the global maximum. It took 5.61 seconds to solve the first stress state but the solution was about 400% of the optimal one. Given training examples similar to the one shown above, UTILITYID3 outputs a decision tree that maps features into a set of stress states (*actions*) that is consistent with the given examples and that has maximum utility. Consistency ensures that, for each training example, there is a stress state that can solve the problem. Maximum utility introduces the desired time/quality tradeoff. These two requirements make the learning problem $\mathcal{NP}-complete$ (proof by reduction from HITTING-SET.) To overcome this obstacle, we have used Chvatal's approximation algorithm (Chvatal 1979) to determine a solution.

The improved performance of the numerical optimization task makes it much easier to solve the geometrical optimization problem for the design task illustrated in Section 1.5. This, in turn, allows us to propose a new strategy for solving the topological design problem. As observed by Friedland (1971) among others, only a few topologies must be evaluated to design a lightweight skeletal structure. This is because once an initial topology has been chosen, the weight drops by adding a few new connection points but it then stabilizes. With this in mind, we have proposed a selection strategy that generates topologies and, for each of them, computes the optimal geometrical solution. The topology is then chosen as the one that minimizes the best geometries. This process is feasible only because we have been able to drastically improve the performance of the geometrical optimization task. This and other contributions are summarized in the next section.

1.7 Summary of Contributions

The goal of this thesis is to demonstrate how machine learning techniques can be applied to optimal engineering design. This has been accomplished by tackling problems in two different areas:

- Bridging the gaps in the knowledge transfer mechanisms between engineers and expert systems
- Speeding up existing numerical methods.

Table 1.7 illustrates the correspondence between these problems and the machine learning techniques used in their solution. The main contribution of this thesis is to have shown that those ML techniques can be effectively used to overcome the knowledge transfer gap and to increase the efficiency of numerical optimizers.

In our approach, these results required the use of a blend of novel and traditional optimization techniques. First, we have defined a new learning framework which is more appropriate to optimization tasks. This framework involves (a) the requirement that the output of the learning algorithm be a set of alternatives and (b) measures of the cost of obtaining solutions. The learning methods produce search control knowledge for the problem solver. This knowledge is optimal in the sense that the set of proposed alternatives maximizes a given utility function. Within this framework, we have developed and tested several learning algorithms which generate search control knowledge for the problem solver. We found experimentally that one of the algorithms we devised, `UTILITYID3`, outperforms all others in the skeletal design domain. This is a contribution to basic research in machine learning. Second, we have used more traditional compiler optimization techniques in the framework of knowledge compilation and merged them with inductive methods. We have shown that the overall result is a drastic speedup of the numerical optimization techniques.

Our approach opens new research directions into the so far unexplored area of applications of machine learning to numerical optimization. It is our hope that, in the medium-to long-term, our techniques will allow the use of specialized numerical optimizers in real-time applications like intelligent CAD systems.

Table 1. Rows enumerate problems in optimal design. Columns list Machine Learning paradigms. X's indicate the ML paradigm used to solve the problem.

	Knowledge Compilation	*Inductive Learning*
Knowledge Transfer		X
Speedup of Numerical Optimizers	X	X

1.8 Guide to the Thesis

The schema in Figure 10 illustrates the suggested sequence in which this thesis should be read. Hopefully, by now, you would have read Chapter 1, the Intro-

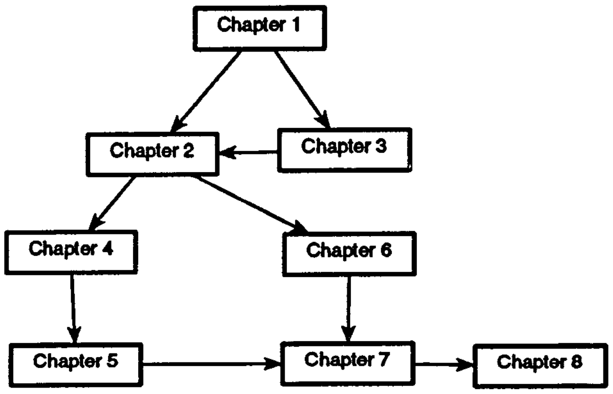


Figure 10. A guide to the thesis

duction, which gives the motivation of this work, briefly surveys the application of computers in a specific area of design, illustrates Machine Learning and some of its techniques, introduces the task we have chosen as our testbed, outlines the solution we have adopted, and lists our contributions to research in machine learning. Chapter 2 details the task used as a testbed for our methods, lists various solutions proposed by engineers, surveys principal numerical optimization methods, and casts the problem into an artificial intelligence search framework. Some of the machine learning techniques that are used in this thesis are presented in Chapter 3 which, for the sake of completeness, also includes a brief digression on why another outstanding technique (EBL) was not used. In Chapter 4 we start presenting the solution we have adopted. In this chapter, we present the techniques used to perform symbolic simplification and the results of our experiments. Chapter 5 illustrates how inductive techniques are used to reduce the number of independent variables in an optimization problem and shows a further speedup over the simplification techniques. Our new learning techniques that associate problems with optimal solutions are presented in Chapter 6 in which we define a new learning framework, illustrate two algorithms to solve the problem, and show the results of the experiments. Finally, the thesis is concluded with Chapter 7 in which we first revisit the engineering task, show the proposed optimization strategy, and speculate on natural extensions of this thesis.

Chapter 2

Structural Design Task

This chapter, along with Appendices A and B, gives the necessary engineering and mathematical background to the problem that we have used as a testbed for our machine learning techniques. Moreover, in observance of the best artificial intelligence traditions, this chapter also shows how to cast the problem into a search framework.

Section 2.1 describes the task and illustrates the associated non-linear optimization problem. Section 2.2 presents some of the approaches taken by the engineering community towards its solution. Section 2.3 presents the numerical techniques used (sometimes) to solve the optimization task and outlines their drawbacks. Finally, Section 2.4 illustrates how the structural design problem can be described as an AI search problem.

2.1 Description

The design of lightweight 2-dimensional (2-D) structures plays a central role in civil and mechanical engineering design, because most typical constructions can be modeled as structural design problems. For example, roofs are made of a series of 2-D panels covered with tiles. Similarly, bridges can be "sliced" and modeled as 2-D panels subject to external forces. Given the importance of the task, much research has been devoted to finding an algorithmic solution to produce (optimal)

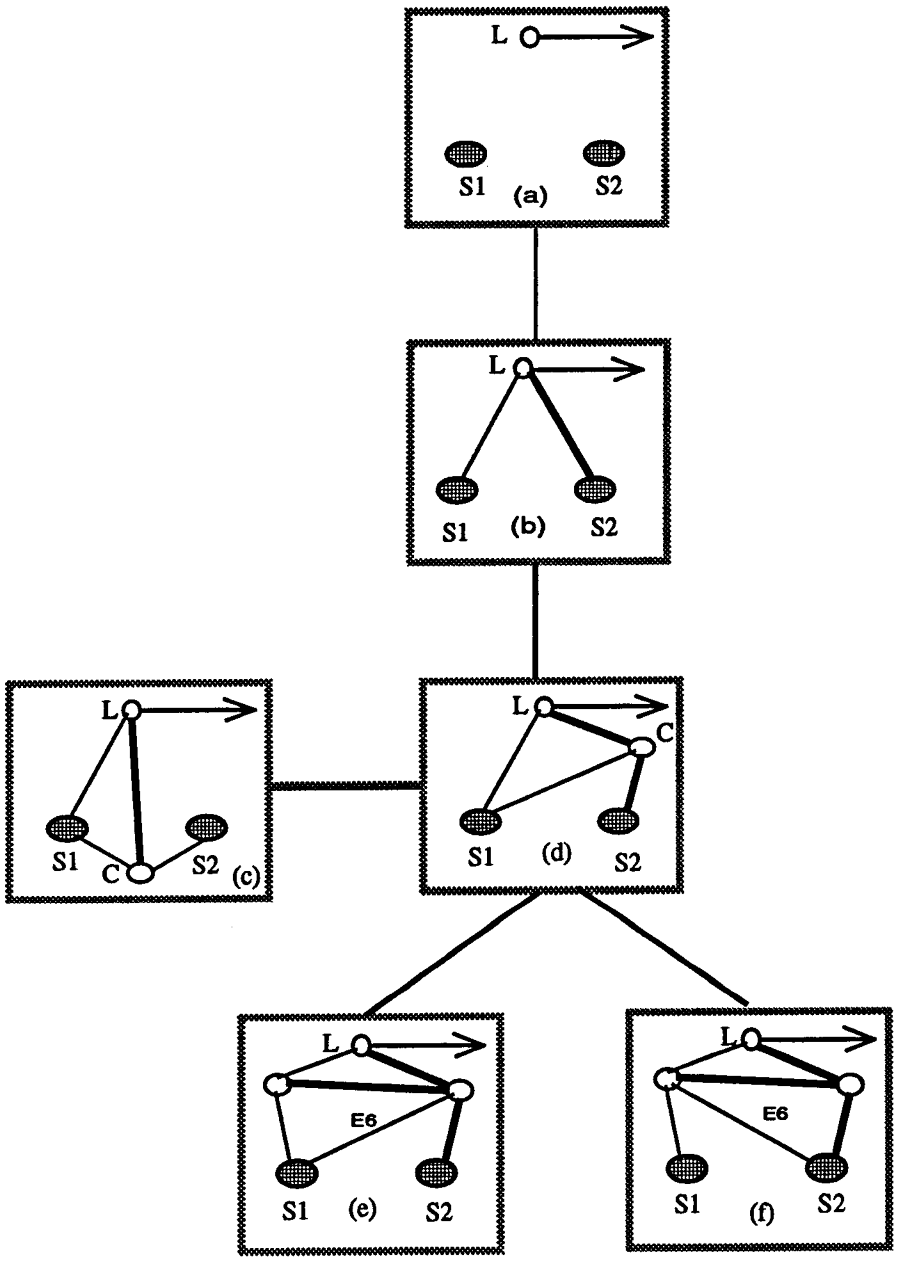


Figure 11. A skeletal design problem and some solutions.

minimum weight structures. Despite the efforts of outstanding researchers since Maxwell (Maxwell 1869), no definite solution is known. Therefore, designing optimal trusses is still very much an art, and the knowledge that leads to optimal designs is often buried in *rules of thumb* or *heuristics* that engineers learn during their studies and years of practice.

In its most general form, the task can be presented as the design of lightweight structures, often called *frames* or *trusses*, to support given external loads using designated stable points. A stable point (*support*) cannot move and ideally can withstand any load. The ground is the most-used support. A *load* is an external force and, in the case of bridges, is a combination of all forces acting on the bridge components; e.g. the weights of the vehicles, the snow that might accumulate on the construction elements, the winds, and so forth. Intuitively, a *stable* truss can be defined as a structure that does not "crash" under the loads imposed on it. Bridges are examples of stable structures.

The weight of an artifact can be decreased in at least two ways. First, the engineer can use lighter material. Second, the shape of the structure can be designed in such a way that the frame requires less material and, hence, is lighter. In this thesis we do not consider the admittedly important advances in the science of materials but, instead, we assume that the materials have been chosen and focus on the synthesis of shapes.

As stated so far, the design task is very general and requires sophisticated analysis techniques to determine the characteristics of the structural elements which are necessary to compute the weight of the structure. Our goal, instead, is to tackle a problem that approximates reality as much as possible and yet provides a manageable task for developing and testing our machine learning techniques. To this end, we have introduced a series of simplifying assumptions (Appendix A) which reduce the optimal design task to that of *skeletal structures* (Topping 1983). While this simplifies the engineering aspects of the task, its main computational charateristics are unaffected: no algorithmic solution is known and the analysis is computationally expensive.

2.1.1 Skeletal Structures

Figure 11a illustrates an example of the simplified optimal design task in which the goal is to build a frame made of *members* (structural elements) connected through frictionless joints (*connection points*) to support an external load L (the arrow), using the supports S1 and S2 (the filled circles). A structural element can be either in tension ("stretched") or in compression. Tension members, or *rods*, are indicated by thin lines. Compression members, or *columns*, are indicated by thick lines. In general, tensile members are lighter than compression members by a factor of approximately 50.

Even with our simplifying assumptions, there is an infinite number of stable frames to support L using S1 and S2, and each of these solutions has a different weight. For instance, as illustrated in Figures 11b, we can use two members that directly connect L to S1 and S2. However, we can also add an arbitrary number of connection points and use a larger number of members to connect the load to the supports through these joints. Figures 11c and 11e show solutions with 1 and 2 connection points, respectively. A second factor influencing the weight is the connectivity of the structural members. Figures 11e and 11f illustrate two solutions which differ only in the way member E6 is connected to its attachment points. In addition to these topological considerations, the weight of the structure also depends on the location of the connection point(s). This leads us to define a new abstraction which is described in the next section.

2.1.2 Stress State

The position of connection point(s) in the 2-D space is an important element determining the overall weight of the structure, because it bears on the type (tensile or compressive) of structural element (or, *member*) needed to connect two points in the chosen configuration. For instance, the trusses in Figures 11c and 11d have one connection point in different locations in the 2-D space. These two solutions have the same connectivity and number of members but, as shown by the thin and thick lines, the truss in 11c uses one column and three rods, while the solution in 11d uses two columns and two rods.

Definition. Let m be the number of members in a topology. The *stress state* is an array of m elements such that the i-th value is $+1$ if the i-th structural member in the topology is tensile and -1 if it is compressive.

As an example, the stress state for the frame in Figure 11d is $(+1, -1, +1, -1)$. The stress state plays a central role in our study, because, as we shall see in Section 2.4, it allows us to decompose the problem space into "well-behaved" regions.

The structural design task (Palmer and Sheppard 1970, Vanderplaats 1984) proceeds in three steps. First, there is a qualitative *topological optimization* problem during which the engineer chooses the number of connection points and the connectivity of the members. Then, during *geometrical optimization*, the location of the connection points is chosen so as to minimize the weight of the structure. The third and final step in the process optimizes the shapes of the individual members. This latter step goes beyond the scope of this thesis and will not be further discussed. After all these factors are established, it is possible to perform the analysis of the structure to determine its weight.

2.1.3 Analysis: Method of Joints

The assumption (see Appendix A) that the skeletal structures we consider are statically determinate greatly simplifies the analysis task. This is because for these structures the topology and the geometry uniquely determine the characteristics of structural members; that is, the internal forces that each member must withstand for the structure to be stable. The computation of the internal forces is the core analytical procedure and it is performed using the *method of joints* (Wang and

Salmon 1984) which is briefly described in the remainder of this section.

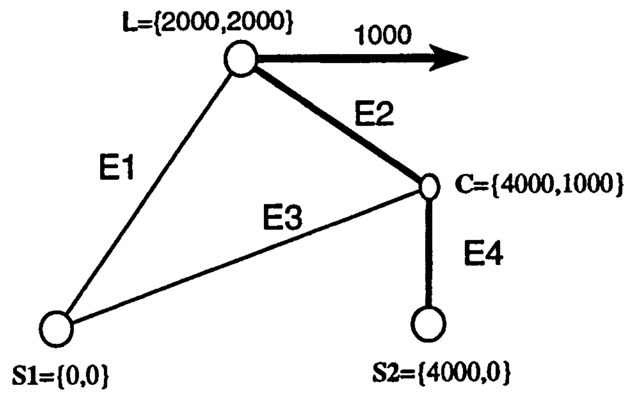


Figure 12. An instance of a skeletal design problem.

The method of joints allows us to write out a system of linear equations to ensure the equilibrium of the structure. This is accomplished by writing as many equations per node as there are *degrees of freedom*; that is, dimensions along which the node can move. Each equation ensures equilibrium by equating the sum of all internal forces along the degree of freedom to the sum of all external forces along the same degree of freedom. Internal forces are exercised by structural members and they can be tensile or compressive. Tensile forces pull on both extremities of the member and tend to stretch it. Compressive forces, instead, push on the extremities of members. External forces are applied to the points and are givens of the problem. For example, in Figure 12 member E1 is subject to tensile forces and member E2 is subject to compressive forces. Point L is subject to an external force of magnitude 1000 units and angle 0. There is no external load applied to the connection point C.

The unknowns in the system of linear equations are the internal forces in each member. For instance, the system in Table 2 has been derived by applying the method of joints to the structure in Figure 12. In this system, the F_i's are the unknown internal forces in each member. The subscript corresponds to the member E_i in the figure. Therefore, the method of joints allows us to compute the force that each member must withstand for the structure to be in equilibrium (stable.) The sign of each solution allows us to specify the characteristic (tensile or compressive) of a member. Positive forces[4] correspond to tensile members. Negative values indicates the need for a compressive element. This distinction is crucial in computing the weight of a structure because of the different densities of compressive and tensile members.

Let us now use the method of joints to derive the system of equations. Given the assumptions in Appendix A, only loads and connection points have freedom of movement. Their degree of freedom is 2, because they can move along both cartesian axes. Hence, for each structural problem one must write $2(l+c)$ equations where l is the number of external loads and c the number of connection points. The assumption that the structure is statically determinate, implies that the matrix of coefficients corresponding to the system of equations, the *statics* or *axial* matrix, is square with dimensions equal to the number of members. The left side of each equation contains one term per structural member that is incident to the node. Once a set of cartesian axes has been fixed, each term contains the unknown component of the force along the degree of freedom. This component is obtained by multiplying the unknown force by the *cosine* of the angle between the member and the frame of reference if the degree of freedom is the x axis. Similarly, if the the degree of freedom is the y axis, the term is obtained by multiplying the force by the *sine* of the same angle. The right hand side of the equation is the x or y component of the vector sum of all external components acting on the node. In matrix form, the vector whose components are the projection of the external loads is called the *load vector.* We notice that elements of the load vector corresponding to connection points are always 0 because no external load is ever applied to these points.

For instance, let us consider the statically determinate structure in Figure 12. The coordinates of each point, and the magnitude and angle of the load are in-

[4]This is a widely accepted convention in engineering.

Table 2. Numerical setup for the method of joints for the example in Figure 12.

$$\begin{aligned}
F_1\, cos(45) + F_2\, cos(-45) &= 1000\, cos(0) \\
F_1\, sin(45) + F_2\, sin(-45) &= 1000\, sin(0) \\
F_2\, cos(-45) + F_3\, cos(22.5) + F_4 cos(90) &= 0 \\
F_2\, sin(-45) + F_3\, sin(22.5) + F_4 sin(90) &= 0
\end{aligned}$$

dicated in the figure. To apply the method of joints we consider one point at a time and write one equation for each degree of freedom of the point. Therefore, for our example, we need only consider points L and C. The first two rows are the equilibrium equations for point L. They ensure stability along the x and y-axis, respectively. The left hand side of the first equation contains one term per member incident to the point. Each term is the component of the internal force of the member along the x-axis. The first term, F_1 $cos(45)$, indicates the projection of the force vector F_1 correspponding to member E1 along the x-axis. The value 45 is the angle of the member with respect to the chosen system of coordinates. This is such that the origin is located in S1 and x-axis goes from the origin to the other support S1; the y-axis is perpendicular to the x-axis and points upwards. The right hand side of the first equation in the table is the component of the force vector corresponding to the external load along the same axis. The second equation in Table 2 ensures stability of the load point L along the y-axis. The remaining two equations are the stability conditions for connection point C and are derived in a similar fashion. It is important to notice that the right hand side in the equations for C is zero because, under our assumptions, no external load is applied to connection points.

The solution to the above system is: $F_1 = 500$, $F_2 = -500$, $F_3 = 707$, and $F_4 = -707$. This indicates that members E1 and E3 are tensile because the sign of the force is positive, while members E2 and E4 are compressive. In addition, the solution also indicates that, for instance, member E2 must withstand a compressive force of 500 units.

To summarize, in applying the method of joints, the position of each connection point, support, and load must be known beforehand. This is necessary to compute the elements of the axial matrix or, correspondingly, the coefficients of the linear equations. Once the axial matrix and the load vector are specified, internal forces can be computed by solving a system of linear equations. This can be easily solved using well known Gaussian elimination algorithms. Having surveyed how to compute the internal forces, let us turn to stating the objective function for the optimization problem.

2.1.4 Objective Function

Under the assumptions given in Appendix A, the objective function used in this study is proportional to the weight of a candidate solution and is usually calculated by a three-step process. The first step is to analyze the structure using the method of joints to compute the internal forces of each structural member. Once this is known, the second step is to determine the characteristics (tensile or compressive) for each member. The third step is to determine the cross-sectional area of each member. The load that a member can bear is assumed to be linearly proportional to its cross-sectional area. Finally, the weight of each member can be computed as the product of the density of the appropriate material, the length of the member, and the cross-sectional area of the member. In Appendix B we show the derivation of the objective function that is used throughout this thesis:

$$Weight = \sum_{\substack{\text{tensile}\\\text{members}}} \|F_i\|\, l_i + c \sum_{\substack{\text{compressive}\\\text{members}}} \|F_i\|\, l_i, \tag{2.2}$$

where F_i is the force in member i, l_i is the length of member i, and c is the ratio of the density per-unit-of-force-borne for compressive members to density per-unit-of-force-borne for tensile members. This function is proportional to the real weight of a structure.

Unless explicitely stated, throughout this thesis we use a cartesian coordinate system. Reich and Fuchs (Reich and Fuchs 1989), studied the effect of the choice of the coordinates system on the optimization methods. They determined that the cartesian coordinate system offers the best tradeoffs in terms of speed and reliabiltiy of the optimization task. The forces (F_i) are transcendental terms in the coordinate system, since they are computed solving a system of linear equations containing trigonometric functions. The lengths (l_i) are algebraic terms, as they represent Euclidean distances between points. The combination of forces and lengths yields a non-linear objective function (see Figure 7.)

The objective function cannot be made explicit in closed form if the coordinates of the connection points are unknown. This is because the internal forces in each member cannot be computed. In turn, this implies that it is not possible to compute which terms in the objective function need to be multiplied by c, the value that assigns different weights to the various members acccording to their characteristics. This latter problem makes it impossible to differentiate the objective function.

2.2 Previous Work in Structural Optimization

This section outlines the three main approaches to the design of lightweight skeletal structures: theoretical, linear and dynamic programming, and shape optimization. Because of its central role in this thesis, a fourth approach, numerical optimization, is discussed in a separate section. The theoretical approaches are non-constructive, because they suggest that a lightweight structure exists but they do not give any algorithm. The linear and dynamic programming approaches have been applied with little success both to the synthesis of topology and geometry and, to our knowledge, they have been abandoned. Nevertheless, they are briefly surveyed, because we strongly believe that they will be useful in the near future. Finally, the shape optimization approaches are the most recent ones and attempt to intertwine the numerical aspect during the synthesis of the structure.

2.2.1 Theoretical Analysis

In 1904, Michell (Michell 1904), *primus inter pares*, attempted to design minimum weight frames. His goal was similar to ours: find the topological configuration of the lightest structure that supports a given set of loads in 2-D. Using Maxwell's result, he derived necessary and sufficient conditions for a structure to attain minimum weight in terms of the strains on each member and their directions. Michell shows that structures made only of compressive or tensile members are optimal and proves the optimality of (a) rods and columns subjected to a single pair of equal and opposite forces, (b) triangular and tetrahedral frames under forces applied at the angles of the geometric figure, and acting along lines which intersect within the figure, and (c) catenaries in general. Michell's results are only of theoretical interest, because they require an enormous number of uneven curved members and, furthermore, there is no constructive algorithm to determine their exact configuration.

2.2.2 Linear and Dynamic Programming

In the 1960's, after Dantzig's results on linear programming (LP) and Bellman's work on dynamic programming (DP), there was a renewed interest in optimal structural design. Researchers applied these techniques to solve the *geometrical* as well as the *topological* optimization problems.

In geometric optimization (Hemp 1973) the unknowns of the primal LP problem are the cross-sectional areas, the weight is the objective function, the stress constraints in each member are the local constraints, and the equilibrium equations are the global constraint. The latter ensures stability of the structure.

LP is then applied to derive cross sectional areas for the individual members. Some members might have a value "close" to zero indicating that they are redundant. Consequently, the *topology* of the given structure can be modified by excluding such members from the final configuration. The optimal structure will then be a subset of the original one. It has been noticed experimentally that in the single loading case (i.e., when only one set of forces is applied over time) the resulting structure is most likely statically determinate.

LP has also been used to solve the topological optimization problem. The basic idea (Hemp 1973) is to superimpose a grid on the region where the structure can be located and consider all the intersections as potential connection points. Loads and supports must be located at some connection points inside the grid. A gigantic LP problem similar to the one described above is then set up. Many of the members will be redundant (i.e., will have zero cross-sectional area) and can be removed from the final configuration.

Although appealing, this approach has several limitations. First, the coarsness of the grid and its shape greatly effect the optimal layout and its weight (Hemp 1973) with finer grids yielding better results. Second, the LP problem for fine grids is extremely costly in terms of computational resources. In fact, if heuristics specific to the problem and various pruning criteria are not used to eliminate superflous members from the grid, the problem is exponential in the number of points in the grid. For instance, for a square grid with n points per side, the linear programming problem can have up to 2^{n^2} variables. It is worth stressing that the heuristics might eliminate important members and, hence, undermine the optimality of the solution.

Dynamic programming was also applied to optimization problem. The space in which the structure could be located, is "sliced" into regions and an optimal substructure of a predetermined topology is identified by formulating a DP problem whose unknowns are the connection points among the various stages of the design. Palmer and Sheppard used this method to design continuous beams (Palmer 1968), 2-D cantilevers (Palmer and Sheppard 1970), and 3-D transmission towers (Sheppard and Palmer 1972).

This approach is very powerful whenever the original design task can be "naturally" divided into designing sub-structures. However, the overall solution must be obtained as sum of optimal solution to sub-problems. The three cases studied by Palmer are of this nature. It is not clear, however, how this technique can used for problems in which the partition is more complicated.

2.2.3 Shape Design

It has always been recognized that shape plays a fundamental role in optimization. Since the early 60's attempts were made at studying the topological properties of structures (Spillers 1963) and at generating topologies by adding and deleting connection points. Although these algorithms could not generate all possible structures, it was shown (Friedland 1971) that the change in topology obtained by increasing the number of connection points and members yields lighter frames. However, after an initial drop in weight with the first few iterations of the algorithm, there is no further significant change (Spillers and Friedland 1972) when additional connection points are introduced. This observation is a crucial one and will be used in this thesis to suggest a strategy to reason at the topological level.

Despite the successes, to our knowledge, only recently Shah and Papalambros have explored again this promising research avenue. Shah (Shah 1988) gives a *shape grammar* capable of generating topologies but gives no insight on how to solve the combinatorial explosion of the generation process. In addition, he does not give any indication on how to improve the numerical optimization techniques that are still needed to optimize the topologies generated by the grammar.

On the other hand, Papalambros and Chirehdast's approach (Papalambros and Chirehdast 1990) seems extremely promising. They propose an Integrated Structural Optimization System (ISOS) which is composed of three parts: initial topology optimization, image enhancement and interpretation, and detailed design. The first phase uses *homogenization* to propose an initial topology. Homogenization starts with a defined region of the space in which the final structure must be confined and a specification of the givens of the problem. As in our problem, these are loads and supports. ISOS then discretizes the region of space and fills it with square cells of "porous" material. A non-linear optimization problem is then set up to minimize an objective function. During optimization, cells are removed or re-sized until a stable and optimal configuration is reached. It must be noticed that this procedure is similar to the mentioned topological optimization using linear programming. One difference being that in linear programming members cannot be re-sized but only removed. The outcome of homogenization is the rough shape of a structure in which the structural members must be recognized. This is accomplished by the second phase which employs machine vision techniques to smooth the "blobs" produced by homogenization and to recognize members. These are then refined and sized during the detailed design phase. One of the results illustrated in the paper, is a solution to a cantilever problem. This "seems" amazingly close to the theoretical lower bound proved by Michell.

However, Papalambros admits a few problems in ISOS. First, ISOS utilizes sophisticated machine vision techniques which are not well-established. Second, ISOS does not speed up the numerical optimization process. On the contrary, it complicates it because of the large number of cells needed to fill fine-grained regions. This is a drawback similar to the one described for the linear optimization approach. Nevertheless, this approach is very promising, because it gives a method to suggest optimal topologies.

We conjecture that the approach that we describe in this thesis is a natural complement to Papalambros' method, because our goal is to speed up numerical optimization via specialization. This is accomplished on a per-topology basis, and each topology could be easily derived using ISOS or any other method.

2.3 Numerical Methods

Perhaps the biggest successes in geometrical optimization have been achieved using non-linear numerical optimization methods. Once the topology (number of connection points and connectivity) is fixed, the geometrical design problem can be cast as a non-linear optimization problem in n-dimensions. For statically determinate structures, each connection point contributes 2 dimensions (its x and y coordinates) to the optimization problem, and the objective function is shown in Section 2.1. Friedland (Friedland 1971) in his PhD thesis gave a numerical optimization schema capable of solving the non-linear problem. Vanderplaats (Vanderplaats 1984) and Papalambros (Papalambros and Wilde 1988), among others, illustrated a series of numerical techniques that have been used to solve the numerical non-linear problem.

However, all researchers agree that numerical techniques are far from being the *panacea* for the skeletal design problem and indicate a list of factors that affect the practical applicability of numerical methods. First, non-linear methods are not easily applied to highly non-linear functions. As shown in Section 2.1, the objective functions that arise in the skeletal design domain have this characteristic. Second, non-linear methods cannot handle a large number of independent variables and typical structural problems involve hundreds of connection points. Third, numerical methods are slow. Fourth, they are not very robust as they may not converge at all and, when they converge, the solutions might depend on the starting point they were given. Finally, numerical methods do not give any insights into the topological optimization problem.

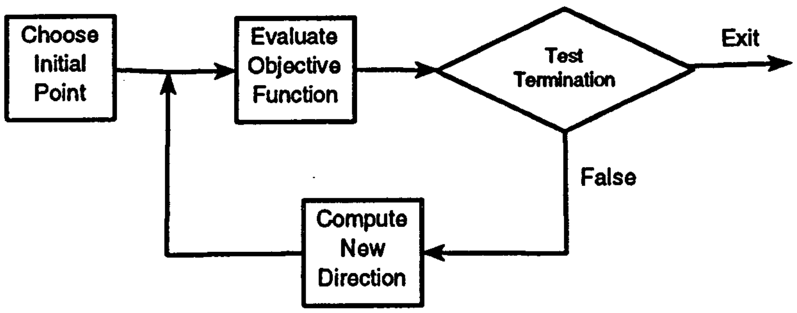


Figure 13. Traditional optimization schema.

Classical optimization textbooks (Vanderplaats 1984, Papalambros and Wilde 1988) present a comprehensive survey of optimization methods and of various techniques for conducting the search for an optimal solution. As shown in Figure 13, this process is iterative and is typical of many domain-independent non-linear optimization methods. Starting at some initial point, the objective function is evaluated and the termination criteria are tested. If the test fails, a new point is generated by taking a step, of some chosen length in some chosen direction, away from the current point. Each point defines a set of values for the independent variables in the objective function. From an AI standpoint, numerical optimizers can be viewed as "smart" hillclimbers. At each iteration they compute the value of a (vector) function which indicates the direction (in n-dimensions) and magnitude of the next step. Most optimization algorithms differ primarily in the criteria used to choose the direction along which to optimize or the length of the steps along a certain direction. For our purposes, we can divide numerical methods into direct and gradient-based depending upon the type of information they use to navigate through the search space.

Direct methods, Powell's (Pike 1986) being the foremost, choose the direction and step size using only evaluations of the objective function. These methods are usually applied when the function is non-differentiable and are very expensive in terms of computational resources. Powell's method, for instance, requires a large number of evaluations of the objective function. This latter observation is crucial to our work, because the weight function in Section 2.1.4 is non-differentiable and it is expensive to compute. Instead, gradient-based methods, like conjugate-gradient descent (Pike 1986), use the partial derivatives of the objective function to choose the new direction of optimization. These numerical methods greatly reduce the number of function evaluations and are faster, because they take larger steps towards the solution at the expense of evaluating the derivatives of the objective function. Usually, gradient-based methods greatly speedup the numerical optimization process. However, for highly non-linear functions, it is possible that the computation of the gradient is more expensive than the computation of the objective function itself. As a result, the optimization process may not show any speedup. One way to overcome this problem is to approximate the gradient with the first few terms of its Taylor series expansion. This is a simple polynomial and is fast to evaluate. As we show in Chapter 4, these approximations can be computed at compile time and used at run time to effectively guide the search. However, before we illustrate it, it is helpful to cast the design problem into the search paradigm.

2.4 Design Task as a Search Problem

The skeletal design task can be cast in a search framework that facilitates (Mittal and Araya 1986) the introduction of AI and ML techniques. To view a problem as search, we must first define the search space; that is, the domain in which a problem solver searches for solutions. For the skeletal design problem, two search spaces can be identified: the space of all topologies and the space of all points in a 2-D region. The space of topologies is infinite but discrete, as it contains all possible structural configurations which can be generated by the following procedure:

```
procedure generateTopologicalSpace
  repeat
      Generate all possible combinations of connections among
          loads, supports, and connection points
      Add one connection point
```

which enumerates the topological space by generating all possible connections among nodes in the structure and adds one extra connection point *ad infinitum.* The diagrams in Figure 11 (except 11c) illustrate a small portion of the topological search space for the 1-load-2-supports problem. Once a topology is fixed, the geometry must be specified. The search space for the geometrical design problem is the continuous $2n$-dimensional space, where n is the number of connection points. This is because a connection point can, in principle, be placed anywhere in the 2-D space. It is the task of the numerical optimizer to search the $2n$ dimensional space to find locations for the connection points that minimize the objective function.

Without further refinements, the search problem is illustrated by the hierarchy of the search spaces imposed by topology and stress state (see Figure 3a.) To find an optimal design, the problem solver must first search through the space of topologies and, then, determine the position of the connection points by searching for a minimum in the $2n$-dimensional space. Currently, these two searches are conducted in an asynchronous fashion. The engineer uses his/her expertise to first determine a topology and then, sometimes, a numerical optimizer is used to search the continuous space.

The stress state partitions the $2n$-dimensional space into smaller regions that are semi-regular (uni-modal). Numerical optimizers perform much better in unimodal search spaces. Moreover, they are particularly efficient when gradient information and specialized functions are provided. Figure 7 shows the region (R1) which corresponds to the structure in Figure 12 for which the stress state is: members E1 and E3 in tension, and members E2 and E4 in compression. We have no formal proof of the conjecture that the regions corresponding to stress states are unimodal but, as we shall see, our experiments strongly support our hypothesis. Therefore, the stress state superimposes an abstract search space on the $2n$-dimensional space. The resulting hierarchy is illustrated in Figure 3b.

With this in mind, for illustrative purposes, we can restate the skeletal design problem in a top-down fashion in which the search happens in three stages and in different spaces. First, the topological space must be searched to determine the number of connection points and the connectivity. Second, the space of stress states is searched to identify the portion(s) of the $2n$-dimensional region which contain the globally optimal solution. Finally, the sub-regions of the $2n$-dimensional space are searched by the numerical optimizer to find the location of the connection points. In practice, however, our solution follows a different order. The space of topologies and stress states are searched first. Then the $2n$-dimensional region is searched to determine the exact configurations of the structures.

As we shall see in Chapters 4, 5, and 6, we have devised machine learning techniques to efficiently search each of these spaces to quickly produce globally optimal designs. However, before we illustrate our solutions, in the next chapter we introduce a few machine learning techniques that have been used in this thesis.

Chapter 3

Machine Learning Techniques

This chapter outlines the traditional machine learning techniques and the experimental methodology employed throughout this thesis.

Section 3.1 introduces the problem of learning search control knowledge. Section 3.2 defines the inductive learning problem and illustrates methods used to learn from examples. Section 3.3 outlines Explanation-Based Learning, one of the most popular speedup learning techniques, and shows why it is not appropriate for our task. Section 3.4 introduces the two knowledge compilation techniques that are employed in this thesis. Finally, Section 3.5 explains the experimental methodology used throughout our work.

3.1 Search Control

Most AI problems involve search control issues: decisions on what to do next in a search space or what operator(s) to apply. Knowledge about search control is useful to the problem solver (human or machine) to improve its run time performance and to solve new problems.

As an example, the search problem for the skeletal design task shown in Section 2.4 would be greatly simplified if we knew the optimal stress state associated with each load and support configuration. In fact, as we demonstrate in the next three chapters, this knowledge can improve the run time performance of the numerical optimizers because it allows a more efficient search in a smaller region.

The ideal solution to the search problem is to provide the problem solver with sufficient and useful (Minton 1990) search control knowledge[5] to navigate efficiently through the search space and to make the correct decision at every step. But, how does the problem solver acquire search control knowledge? People, so-called *experts*, acquire it over time and after long periods of training. The result is search control knowledge in specific fields that allows them to perform tasks they have never encoutered before and improve their performance over time. Expert systems, instead, are traditionally given search control knowledge in the form of rules that *knowledge engineers* formulate after lengthy and costly interviews with human experts.

However, as computers become more sophisticated and can potentially be made to solve problems of increasing complexity, the task of the knowledge engineer becomes ever more difficult. In some cases, it is even practically impossible. For instance, in optimal design it is extremely difficult for an engineer to spell out rules to design optimal frames. In fact, for the skeletal design domain we have interviewed two professional engineers[6] who were given a few optimal design problems. Both engineers derived optimal topological solutions (which our system derives as well) but were unable to indicate general rules that would lead to optimal solutions for similar problems. In addition, when faced with the localization of the connection points to minimize the weight of the frame, the geometrical design task, the engineers were unable to formulate any rule that could be transferred into an expert system. They solved the problem by trying a few points and choosing the minimum. This turned out to be far from the optimal derived by our system.

[5]Control knowledge is useful when the cost of retrieval is negligible with respect to the cost of trying the solution.

[6]We wish to thank Dr.Dave Ullman and Dr.J.Peterson of the Mechanical and Civil Engineering Departments, respectively, at Oregon State University for the time spent during the interviews.

To partially overcome this knowledge acquisition bottleneck (Feigenbaum 1977), researchers are turning towards approaches which let computers automatically acquire (*learn*) knowledge. Machine Learning is the branch of AI that studies, among other things, automated methods to bridge the gap between experts and expert systems. Thus far, two main avenues of research (Shavlik and Dietterich 1990) have been identified: inductive learning and knowledge compilation. Some of the tools used in ML are described in the remaining sections of this chapter.

3.2 Inductive Learning

> "*... The ability of generalizing from examples is one of the hallmarks of intelligence. Our experience of the world is specific, yet we are able to formulate general theories that account for the past and predict the future. Such reasoning is commonly known as induction ...*" (Genesereth and Nilsson 1987)

Having defined (see Section 1.4) learning as the ability to solve tasks more efficiently and more effectively next time, it becomes natural to refer to inductive learning, also called *learning from examples*, as the ability to acquire skills from examples. This can happen either in a controlled environment (*supervised* learning), in which a teacher classifies each example before it is given to the learning algorithm, or in an uncontrolled environment (*unsupervised* learning), in which examples are not classified and the learning algorithm uses its own strategies to create clusters of examples that lead to concepts.

3.2.1 Problem Definition

Let us introduce a formal definition (Bakiri 1991) of the inductive learning task.

Assume that there are n features, $\{a_1a_2 \ldots a_n\}$, characterizing a given domain, and let D_i be the set of allowable values for feature $a_i (= 1, \ldots, n)$. We then define the event space X as:

$$X = D_1 \times D_2 \times \ldots \times D_n.$$

Let there be a mapping function $g : X \longrightarrow V = \{v_1, \ldots, v_C\}$. Hence, V is the set consisting of C elements comprising the output values that the function g is allowed to take.

The inductive learning problem is to find a mapping f which is an approximation of g from a limited number of *training examples.* Each of these is of the form: $(\vec{x}, v)$ where

$\vec{x}$ is an attribute (feature) vector:$\langle x_1 x_2 \ldots x_n \rangle \in X$ and,
$v \in V$.

The learning algorithm takes as input a set of training examples and produces a mapping function f in some representation language, e.g. a decision tree. In this introduction we assume Boolean features; that is, each $x_i \in \{0,1\} (\vec{x} \in \{0,1\}^n)$. Following the above notation, each D_i is the set $\{0,1\}$ and X is $\{0,1\}^n$. Hence, the task can be re-stated as learning the function:

$$f : \{0,1\}^n \longrightarrow V.$$

The set V of allowable values for the function f is often called the set of output classes and each $v \in V$ is the name of an output class. Hence, this inductive learning task can also be referred to as the *multiclass learning* problem.

It is often convenient to give each output value (class name v_i) a unique number $\{1, \ldots, C\}$ that we will refer to as the class number. Since there is a one-to-one correspondence between $\{1, \ldots, C\}$ and $\{v_1, \ldots, v_C\}$, we can learn the equivalent mapping function

$$f : \{0,1\}^n \longrightarrow \{1, \ldots, C\}.$$

Hence, C is also referred to as the number of classes in the domain. Note that $C = 2$ corresponds to the special case of boolean concept learning.

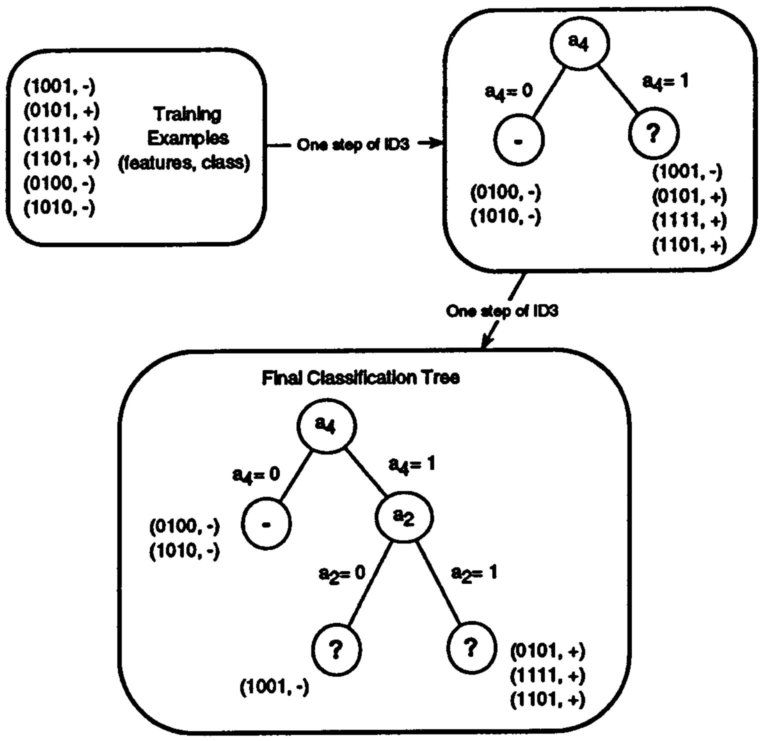


Figure 14. An ID3 example. Building a decision tree from 6 examples and 4 features: $a_1, \ldots, a_4$.

Thus far, ID3 (Quinlan 1986) is the most successful algorithm used to learn from examples. This is briefly described in the next section.

3.2.2 The ID3 Algorithm

ID3 is a learning algorithm of the TDIDT (Top-Down Induction of Decision Trees) family (Quinlan 1986). Given a subset of the learning examples (called the training set), the algorithm constructs a decision tree that can then be employed to classify all the examples of a particular concept. A learning example is a pair: $(\vec{x}, v)$ where $\vec{x}$ is a vector of attributes: $\langle x_1, x_2, \ldots, x_n \rangle$ and v is the class associated with $\vec{x}$. In the general case, the features x_i and the outcome v need not be Boolean. For simplicity, in Table 3 we describe a version of the ID3-algorithm that applies only to binary feature vectors and binary classes.

Table 3. Sketch of the ID3 algorithm

Function build-tree (*training-set*)

INPUT: A *training-set* of m training examples. Each example is a pair: $(\vec{x}_i, v_i)$ where $\vec{x}_i$ is an n dimensional binary attribute (feature) vector: $\langle x_1, x_2, \ldots, x_n \rangle$, $x_j \in \{0, 1\}$ and $v_i \in \{+, -\}$ giving the binary class associated with $\vec{x}_i$. An example is called positive if v_i is $+$, and negative otherwise.

OUTPUT: A (binary) decision tree.

The decision tree is formed recursively as follows:

```
begin {build-tree}
  If the training-set consists only of positive examples then
    output a leaf node marked +.
  else if the training-set consists only of negative examples then
    output a leaf node marked −.
  else
    • Select one of the attributes $a_1, a_2, \ldots, a_n$ to be at the root of the tree.
      (The criterion for the selection will be detailed later.)
      Call that attribute best-a.
    • Divide the training-set to two sets:
        The zero-set containing all examples that have a value of zero for attribute best-a.
        The one-set containing all examples that have a value of one for attribute best-a.
    • Mark attribute best-a as already used.
    • Let:
        zero-set-subtree := build-tree(zero-set)
        one-set-subtree := build-tree(one-set)
      (Note that above are two recursive calls to build-tree)
    • Output a binary tree with best-a as the root,
        the zero-set-subtree as the left subtree, and
        the one-set-subtree as the right subtree.
end. {build-tree}
```

Figure 14 shows how a decision tree is built from a simple training set consisting of 6 examples and 4 attributes: $a_1, \ldots, a_4$.

We will now turn our attention to the criterion for determining which attribute should be tested at the root of a (sub)tree and hence serve as the basis for further splitting the examples reaching that node. This criterion in ID3 is biased to select attributes that will lead to a small decision tree. It is a heuristic, so ID3 is not *guaranteed* to derive the smallest possible decision tree for a given set of training examples.

To detail the attribute selection criterion, we will consider some node in the tree with a set of p positive and n negative training examples reaching that node. The uncertainty in the class value to be assigned for that node is measured by the entropy function:

$$entr(n,p) = -\frac{n}{n+p}\log_2\frac{n}{n+p} - \frac{p}{n+p}\log_2\frac{p}{n+p}$$

The above formula is intuitively appealing, since it assigns a maximum value (1) for class uncertainty when the sample is split evenly between negative and positive examples ($n = p$), and a minimum value (0) for the uncertainty when the sample consists of only one type of examples: either positive ($n = 0$) or negative ($p = 0$).

ID3 selects the feature that provides the most information about the class value, i.e. the one that *minimizes* the uncertainty in the class after the split—calculated as the weighted average of the entropies of the *zero-set* and the *one-set*:

$$unc(a_i) = \frac{n_0 + p_0}{n+p} entr(n_0, p_0) + \frac{n_1 + p_1}{n+p} entr(n_1, p_1)$$

where

n, p	= number of negative, positive examples in the training set reaching the node
a_i	= attribute being considered as a basis for the split
n_0, p_0	= number of negative, positive examples in the zero-set, and
n_1, p_1	= number of negative, positive examples in the one-set.

The uncertainty is calculated for all the attributes (not yet tested on the path from the root to the current node) and the one that minimizes the uncertainty is selected as *best-a.*

3.2.3 Regression

Regression is the analysis of the functional relation f (model) between one ***dependent*** variable y and a set of ***independent*** variables x_i's:

$$y = f(x_1, x_2, \ldots, x_n).$$

The problem is a predictive one. Given a set of numerical values of the n variables one wishes to determine the "best" model that predicts the values of y. Guessing f (*model*) is a difficult problem unless a specific functional form is assumed. Therefore, regression is divided into linear and non-linear. In linear regression with 2 variables the solution is the estimate of the parameters a, b, and c of the equation:

$$y = a + bx_1 + cx_2.$$

These parameters are computed by minimizing the sum of ***least-square*** errors estimates; that is, the sum of the square of the differences between the actual values of the observations and the values forecast by the linear model. This estimation method is called the ***error sum of squares.*** The model that results from regression is often tested for ***goodness of fit*** to the data. This is measured via the variance of the estimated errors. Even when the functional form of the model is known or assumed, it may not be known which independent variables are relevant to a model. Determining the relevant variables is the ***model-selection*** problem.

A powerful off-the-shelf statistical system, SAS (SAS 1989), has been used in our study because it implements some of the most accurate regression procedures in two packages REG and NLIN. The first package performs linear regression and solves the model selection problem using one of nine different procedures. Among them, *forward* and *backward* selection are the foremost. The former starts with a constant model and adds one variable at a time. The latter assumes that initially the model contains all variables and deletes one at a time. In both cases, goodness of fit measures are used to determine which variable should be added/deleted and variables that enter/exit a model are never eliminated/added.

Linear regression is a well understood problem with satisfactory solutions. On the contrary, non-linear regression is at least as difficult as non-linear optimization and it is rarely used. The problem is that the error sum of squares becomes a non-linear optimization problem with the drawbacks we are analyzing in this thesis. The package `NLIN` in `SAS` requires that the functional form of the model be provided and that the model selection problem be solved by the user. These requirements are too restrictive for most real domains.

3.2.4 Discovery

Discovery is one of the most fascinating and yet controversial areas of artificial intelligence. It is fascinating because, according to Simon (Simon 1983), it is the *"finding of new things"* previously unknown to humans. This task closely resembles human creativity and has long (and erroneously) been construed as one of the goals of artificial intelligence. It is controversial because, to date, none of the systems that have been built has fulfilled such a promise. Moreover, it has been argued (Ritchie and Hanna 1984) that the "discoveries" made by existing programs are the result of well-engineered search techniques and heuristics. Nevertheless, we are interested in discovery techniques because they provide a solid background for the heuristic search approaches we use in our reformulation.

Two computer systems among others have contributed to the successes in automatic discovery: AM and BACON. The AM program by Lenat (Lenat 1978) is guided by "interestingness" heuristics in its search of a space of mathematical concepts in number theory. As a result, it re-discovers the concepts of prime numbers and maximally-divisible number – numbers with many divisors. BACON (Langley, Simon et al. 1987) performs a bottom-up search of a space. It starts with a collection of data and derives algebraic laws. One of most interesting parts of BACON is that it is capable of deriving lemmas; that is, intermediate results upon which subsequent discoveries are made. BACON has been successfully used to re-derive Kepler's Third Law and Ohm's Law.

3.3 Explanation-Based Learning

Ellman (1989) presents a complete introduction to Explanation-Based Learning (EBL.) This section briefly surveys the fundamental concepts and components of EBL and argues that drawbacks of this technique affect its applicability to mathematical domains.

EBL can be schematically (Mitchell, Keller and Kedar-Cabelli 1986) defined as

Given:

- Domain Theory (DT) that encodes knowledge about the domain and all operators used during problem solving,
- Target concept describing the concept to be learned,
- Training example, an example of the target concept,
- Operationality criterion, a criterion for the form of the output

Find:

- A generalization of the training example that is a sufficient concept definition for the target concept and that satisfies the operationality criterion.

The key factor in EBL is that it generalizes from the single training example by deriving a proof that the example is an instance of the target concept. Therefore, for EBL to be successful, it must be able to derive a proof.

We argue that in real optimization domains it is rarely possible to derive a complete proof that a solution is an optimum. In theory, one should be able to prove optimality by applying the Karush-Kuhn-Tucker (Pike 1986) condition.

However, in practise these proofs are based on computing Lagrangian multipliers. This is a difficult task for non-linear functions. Agogino and Almagren (1987) built SYMFUN, a symbolic system to reason qualitatively about optimization problems. However, as Agogino points out, the computations are feasible only for very simple functions. In addition, in many cases, the computation of the Lagrangian is purely numerical, and this hardly provides a trace usable by an EBL system. The trace of a numerical optimizer gives little information on the structure of the problem. The extraction of information from a numerical trace is a promising area of research (see Section 7.3.)

Even assuming that it is possible to prove optimality, the proofs would be of little interest. In a mathematical domain, the domain theory will include detailed algebraic operators that allow the system to transform the functions. Thus the proof would contain the concatenation of such rules. The resulting EBL-derived rules would be too detailed to produce any appreciable speedup. An example of this phenomenon is shown in Chapter 4. When the topology is incorporated into the objective function, the speedup is negligeable (58/60 sec.) This is due to the fact that in a complete trace the number of arithmetic operations is greatly increased with respect to the original solution method.

Despite these drawbacks, we have used EBL (see Chapter 5) to eliminate independent variables from an optimization problem. This task is substantially different from solving an optimization problem. The elimination of independent variables required reasoning about geometric entities such as points, lines, and segments. Obviously, such a geometric domain theory can be easily devised and provided to an EBL system. In this task, EBL was employed to provide a generalization mechanism once a proof was generated. The proof was generated using the geometric domain theory and regression techniques that filled gaps in the proofs. These were caused by the inability of the geometric theory to either formulate or to provide justifications for functional or numerical relationships.

3.4 Knowledge Compilation

The Workshop on Knowledge Compilation held in 1986 (Dietterich 1986) was the first attempt to establish knowledge compilation as a methodology. More recently, IEEE Expert (IEEE 1991) has published a special issue on knowledge compilation. This is the first attempt from a magazine with a large circulation to explain this methodology to the larger computer science community.

With a far less ambitious goal, using an approach similar to (Berliner 1989), we have used partial evaluation (Futamura 1971) and unfolding (Burstall and Darlington 1977), two automatic programming techniques, to perform knowledge compilation. These techniques are outlined in the remaining of this section.

3.4.1 Partial Evaluation

Partial evaluation is a simple and yet very powerful concept which was pioneered by Futamura (Futamura 1971) to derive compilers automatically from interpreters. However, it has mostly been used by the compiler optimization researchers to increase the efficiency of existing programs and to generate new programs. This latter aspect is not investigated in this thesis.

In its most general formulation (Beckman et al. 1976), a partial evaluator is a program which takes a procedure $\mathtt{P(x_1,\ x_2,\ \ldots,\ x_n)}$ and m ($\leq n$) expressions $\mathtt{c_1,\ c_2,\ \ldots,\ c_m}$ for the first m of the n variables and returns a specialized version $\mathtt{P'}$ of the procedure such that for all $\mathtt{x}$:

$$\mathtt{P'(x_{m+1},\ x_{m+1},\ \ldots,\ x_n)\ =\ P(c_1,\ \ldots,\ c_m,\ x_{m+1},\ \ldots,\ x_n).}$$

The simple substitution of expressions for variables does not speedup a program. This is especially true in numerical programs. However, when further algebraic simplifications are used to derive simpler expressions, the result can be a much faster program. Consider, for instance, the following algebraic expression: $3\ 2^x+y$. When we substitute the values $x = 5$ and $y = 2$, it takes Mathematica 183.3

milliseconds to evaluate the expression 100 times. However, if the value of x was known in advance, we can partially evaluate the expression w.r.t. x and obtain $3 \cdot 2^5 + y$ which takes Mathematica 156.6 milliseconds to evaluate 100 times. This computation can be made more efficient if we compute $3 \cdot 2^5$ and use $96 + y$ instead. In fact, the evaluation of 100 of these expressions takes 1 millisecond (the timing was done running Mathematica on a SUN Sparc 2 workstation.) In conclusion, partial evaluation alone is not sufficient to speedup programs, but when it is coupled with simplification techniques, it becomes a powerful tool to improve the efficiency of programs.

General purpose partial evaluators are rather complicated to implement. However, the technique described above resembles constant propagation methods used in most optimizing compilers (Aho, Sethi and Ullman 1986). These are much simpler to implement especially with off-the-shelf symbolic manipulation tools like Mathematica (Wolfram 1988). In our implementation we have used Mathematica to perform constant propagation and simplification. A brief overview of Mathematica is given in Appendix C.

The description of partial evaluation presented in this section is by no means complete. In fact, this topic, also known as *mixed* or *partial computation*, has been extensively studied from theoretical (Ershov 1982), practical (Beckman et al. 1976), and artificial intelligence standpoints. This latter aspect is particularly appealing to us because Van Harmelen and Bundy (1988) argue that one particular EBL technique can be viewed as partial evaluation.

3.4.2 Loop Unrolling

Loop unrolling is another transformation technique used in optimizing compilers and can be viewed as yet another application (Ershov 1982) of partial evaluation. The idea is again very simple. If we a have a program with a loop like:

```
Do i = 1 to n; block(i); endDo
```

and at compile time it is known that n = 5, the program can be transformed into a sequence of 5 blocks:

```
block(1); block(2); block(3); block(4); block(5);
```

in which `i` has been instantiated. In addition, if there is no data dependency among the statements in the loop, all n = 5 statements can be executed in parallel. We shall use this technique in the thesis to specialize the given solution method to different topologies.

3.5 Experimental Methodology

All quantitative measurements in this thesis follow closely widely accepted methodologies taught by Dietterich (Dietterich 1987) and suggested by Kibler and Langley (1988) in their paper *Machine Learning as an Experimental Science.*

In this thesis we make two types of claims. First, we claim that the mix of symbolic and inductive methods we have employed produces a speedup in the optimization process. Second, we claim that our novel inductive techniques reduce the size of the search space. The experiments we performed are designed to support these claims. From a methodological standpoint, for each experiment we specify:

- The question to be answered by the experiment
- How the experiment is run
- Interpretation of the results.

However, before we describe our approach, it is important to stress a methodological issue that arises in experiments in inductive learning. As described in Section 3.2, inductive learning algorithms are given a set of *training* examples which they use to produce, say, a decision tree. To test the algorithm at run time, it is common practice to use a *test* set which is separate from the training set. A common procedure to generate the training and test sets is to randomly generate a single set of experiments and then to extract the two disjoint sets using a procedure that performs the partitioning in a random fashion. In the remaining of this section we use hypothetical situations related to our study to illustrate how we have followed these three steps to substantiate the claims.

3.5.1 Speedup Curves

We wish to demonstrate that successive applications of different compilation methods produce a speedup in the optimization process at run time. Therefore, the obvious question to be answered is: "Do the compilation stages produce a speedup in the optimization process?" To answer this question, we compile the optimization procedure with each method and, at each stage, we measure the CPU time it takes to produce a solution. To produce more reliable results, instead of limiting ourselves to one experiment, we perform a series of experiments with each compiled program and compute the average running time. This is taken as the time it takes to produce a solution when the selected compilation stage is applied. The results are then presented in a graphical form similar to the one in Figure 15. In this figure, the compilation methods are reported on the x-axis. The CPU time is reported on the y-axis as a function of the method used during compilation. For instance, it takes 25 seconds when `Method 1` is used. Instead, it takes 6 seconds when `Method 4` is applied. This curve shows an obvious speedup as different and more powerful compilation methods are employed. Moreover, if the various methods represent successive compilation stages, that is they must be applied in sequence, the graph suggests a compile-time to run-time tradeoff. The more time is spent during compilation, the faster the run time process will be. This latter observation can be given during the interpretation of the results. Graphs similar to the one shown in Figure 15 are used in Chapters 4 and 5 to demonstrate how the symbolic and inductive techniques speedup numerical optimizers.

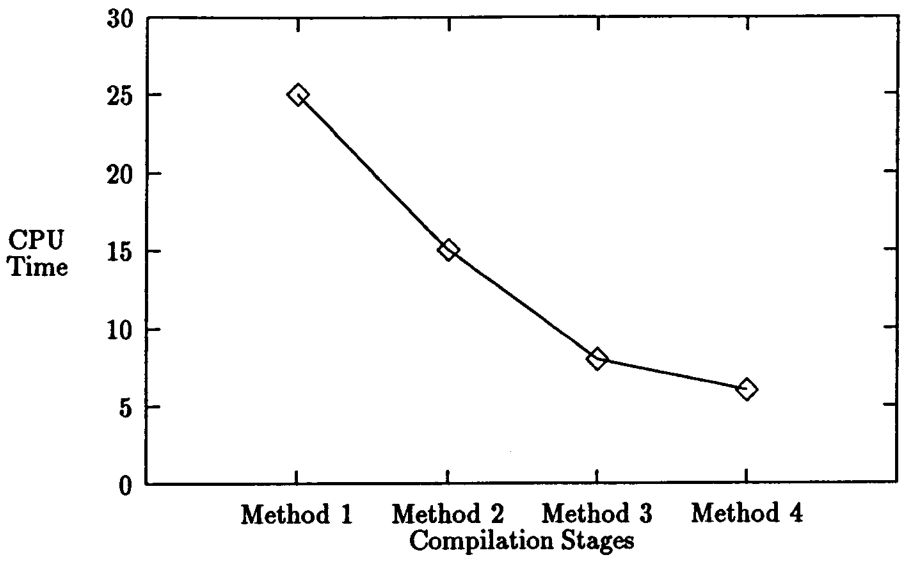


Figure 15. Sample speedup curve.

3.5.2 Learning Curves

The performance of the inductive learning algorithms in producing search control knowledge are shown using learning and tradeoff curves. Learning curves allow us to analyze the performance of the algorithm as more training examples are provided. This permits to answer two questions: (a) How many examples are needed to reach a satisfactory performance? and (b) what is the peak performance of the algorithm?

This latter question assumes that the algorithm is given all available training examples. Having outlined the questions, Table 4 describes the procedure used to produce learning curves. The compile/run time cycle is executed n times with training sets of different sizes. For the sake of exposition, let us assume that the outcome of the run time phase is a measure of correctness of the result; at this point, we are not concerned on the semantics of this measure. In our experiments, we have chosen to increment the size of a training set by a fixed step size until a maximum is reached. Each decision tree produced at compile time by the inductive algorithm is then tested against the test set and correctness of the results

Table 4. Procedure to generate learning curves.

```
procedure learningCurve
  experiments = generateExperiments()
  testSet = randomlyExtractTestSet(experiments)
  trainingSet = randomlyExtractTrainingSet(experiments)
  for *size* from *min* to *max* by *step*
      repeat $n$ times
          currentTrainingSet = randomly choose *size* elements from trainingSet
          decisionTree = run inductive algorithm on *currentTrainingSet*
          run experiments using *decisionTree* on *testSet*
      result = average of the results from previous $n$ runs
      output the pair *(size, result)*
```

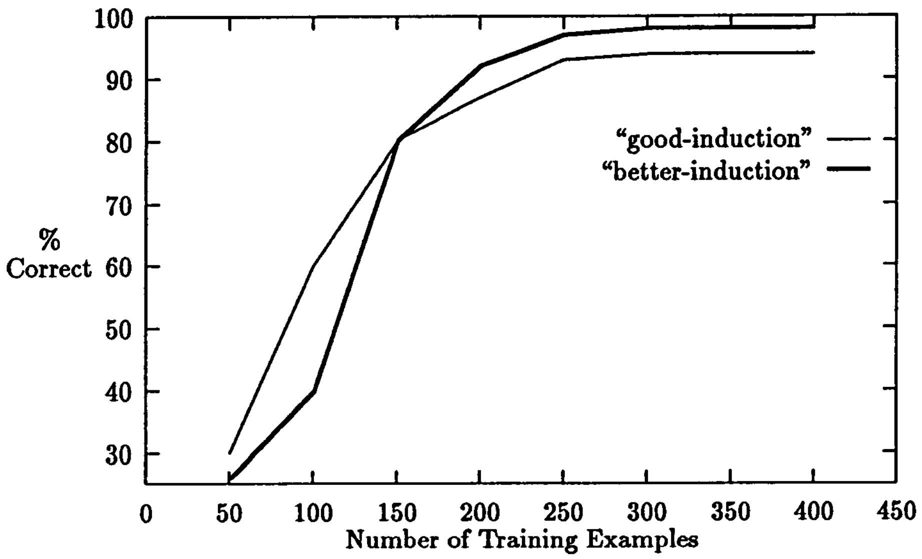


Figure 16. Sample learning curve.

is measured. Again for simplicity, we have kept the test set constant throughout the cycles, while the training set varies. This cycle is repeated $n(=5)$ times to smooth out adverse effects that might arise from the random choice of the training examples. The overall correctness is taken as the average of the individual results over these $n(=5)$ runs. The average correctness and the number of training examples are then reported in a plot similar to the one in Figure 16. Let us first consider the solid line in the figure. The number of training examples is shown on the x-axis, while the correctness is reported on the y-axis. The interpretation of the results is then obvious. The solid line curve indicates that 250 examples are needed to achieve 95% correctness. Moreover, it also indicated that it is superfluous to give more that 250 examples to the inductive algorithm because it has achieved its peak performance. An additional piece of information that can be extracted from the learning curve is a lower bound on the number of examples needed to achieve a satisfactory performance. For instance, in Figure 16, one can see that the algorithm performs poorly with less than 100 examples. It requires 150 examples to achieve 80% correctness. In addition to providing information on a single algorithm, learning curves can be also used to compare the performance of different algorithms. For instance, in Figure 16 we compare the performance of `good-induction` and `better-induction`, two ideal inductive algorithms. The interpretation of the results indicates that the algorithm `good-induction` performs well with fewer examples but, when more than 150 examples are presented, `better-induction` outperforms `good-induction`.

3.5.3 Tradeoff Curves

The last family of experiments we have performed involves tradeoffs between quantities. Suppose that, for each solution, one computes a utility which depends on the time it takes to achieve the solution and the quality of the solution which is represented in terms of its error. A simple utility function can be defined as:

$$utility(time, error) = time + c \times error$$

where c is a constant. The question is then: "What is the tradeoff between *time* and *error* introduced by the learning algorithm?" To answer this question, one can produce data that show how the two quantities vary as the constant c assumes different values. However, as we have seen from the interpretation of the results

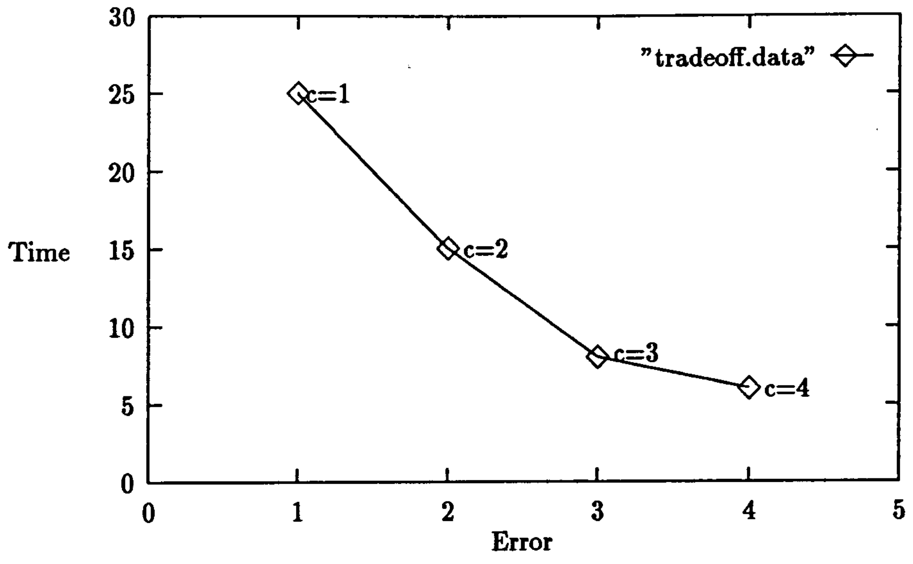


Figure 17. Sample tradeoff curve.

shown when learning curves are produced, the performance of inductive algorithms depends on the number of training examples. Therefore, to produce tradeoff curves, we have chosen to use inductive algorithms at their peak performance; that is, when it is clear from the learning curve that the algorithm will not perform better. For instance, if the learning curve in Figure 16 were to be used, we would choose a training set of size 250. Once the size of the training set has been established, we can start performing the experiments. These consist in running the inductive algorithm with the chosen training set and measuring the resulting time and error for various values of the constant *c*. These values can then be plotted in a graph similar to the one shown in Figure 17. This graph indicates that as the weight *c* of the error increases, it takes more time to produce more reliable results.

Having explained the methodology we used to produce our results, let us turn, in the next three chapters, to illustrate the techniques we have devised and to detail the results.

Chapter 4

Symbolic Methods

This Chapter describes symbolic knowledge compilation (Cerbone and Dietterich 1991) methods to speedup and increase the reliability of numerical optimizers.

Section 4.1 explains the importance of using symbolic methods and Section 4.2 illustrates how to use knowledge about the problem in the optimization process. Section 4.3 outlines the compiler, while the results of the experiments in the skeletal design task domain are reported in Section 4.4. Finally, Section 4.5 summarizes the results in this chapter and the drawbacks of the symbolic approach.

4.1 Introduction

Many important application problems can be formalized as constrained non-linear optimization tasks which, traditionally, are solved using general-purpose iterative numerical methods that "know" nothing about the problem. These methods are typically hillclimbers (see Figure 13) and are CPU intensive because of their pure run-time nature. Therefore, in most cases, they cannot be used in real-time applications. Furthermore, numerical methods for solving such problems are brittle and do not scale well because, for large classes of engineering problems, the objective function cannot be converted into a differentiable closed form. This prevents the application of efficient gradient optimization methods. Only slower, non-gradient methods can be applied. To overcome these limitations, we have augmented numerical optimization by first performing a symbolic compilation stage to produce objective functions that are faster to evaluate and that depend less on the choice of the starting point. These goals are accomplished by successive specializations of the objective function that, in the end, reduce it to a collection of independent functions that are fast to evaluate, that can be differentiated symbolically, and that represent smaller regions of the overall search space. This allows us to replace a single inefficient non-gradient-based optimization by a set of efficient numerical gradient-directed optimizations that can be performed in parallel.

4.2 Knowledge-Based Optimization

In engineering design, the objective function is typically very expensive to evaluate, since it reflects many of the specifications for the design problem. Furthermore, it is often the case that the objective function lacks a differentiable closed-form. For example, in our objective function in Equation 2.2, the fact that the constant c is applied only to compressive members makes it impossible to obtain a differentiable closed-form. The signs of the internal forces must be computed before it is possible to determine which members are compressive. Given that the speed of numerical optimization is determined by the cost and frequency of evaluating the objective function, there are two obvious ways to speed up the process: (a) reduce the cost of each evaluation of the objective function and (b) reduce the number of evaluations by finding a closed-form for the derivative of the objective function, so that gradient descent methods can be applied.

We have developed an approach that pursues these directions. The basic idea is to perform a *compilation* stage prior to run time numerical optimization. As shown in Figure 18, the optimization strategy then becomes a mixture of compile and run time operations. During compilation, a series of fast-to-evaluate special cases of the original function are obtained. These functions are then used at run

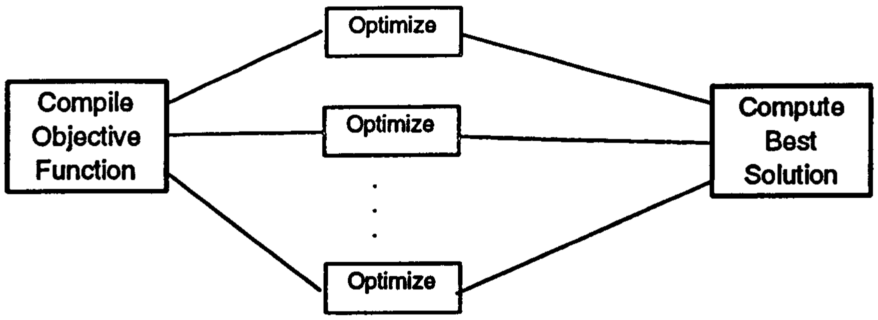


Figure 18. Optimization schema with compilation stage.

time as inputs to traditional gradient-based numerical optimizers to find the local minimum for each special case. The global minimum is then computed by choosing the best solution among all minima. The overall result is a faster and more reliable optimization procedure that can be further sped up by solving each special case in parallel on independent machines. Therefore, as a side effect, compilation also produces a parallelization of the optimization process.

The compilation stage is the core of our approach, and the next section illustrates how it is accomplished.

4.3 Compilation Stages

Our compilation schema is based on the knowledge compilation techniques described in Section 3.4. Unlike most current machine learning techniques, it relies on the specialization of the problem via successive incorporation of constraints into the objective function. Our method is analogous to previous attempts in machine learning at specializing abstractions to obtain specific rules. Braudaway (Braudaway 1988) designed a system along the same principle. However, to our knowledge, very little work has been done in using knowledge compilation techniques to speed up numerical tasks. In contrast, the current trend in the machine learning community focuses on methods, such as Explanation Based Learning (EBL) (Ellman

1989), capable of generating rules for guiding combinatorial search. Section 3.3 gives a brief description of EBL and its drawbacks in numerical tasks.

The inputs to the compilation stage are the objective function, a method to compute terms of the objective function, and values for some of the non-design variables in the function. As an example, for the skeletal design problem, the compiler is given the formula

$$Weight = \sum_{\substack{\text{tensile}\\\text{members}}} \|F_i\|\, l_i + c \sum_{\substack{\text{compressive}\\\text{members}}} \|F_i\|\, l_i, \tag{4.3}$$

a method to solve a system of equations, and values for loads and supports. The goal of the optimization process is to derive the locations of the connection points. Compilation is divided into four phases. The first three incorporate knowledge about the problem into the objective function to obtain a large number of special cases in closed-form. In principle, this number can be exponential and Chapter 6 describes how to circumvent this problem. During the last phase, the compiler derives the gradient for each of these special cases and its Taylor series expansion. To see how the compiler works, let us follow each compilation phase through an example taken from the skeletal design domain.

4.3.1 Topological Simplification

During this phase, the compiler performs a case analysis of topologies at an abstract level. It explores the topological search space to a certain depth and, for each topological configuration, it specializes the method it was given to compute the elements of the objective function. The result is a series of special cases of the objective function in symbolic form. For the skeletal design domain this means that, once the topology is fixed, it is possible to determine explicitly the number of connection points, the number of members, and the connectivity of the members. This, in turn, allows the compiler to determine symbolically the expressions for the length of each member. Moreover, knowledge of the topology implies that the compiler can use the method of joints (see Section 2.1.3) to compute symbolically the elements of the axial matrix and of the load vector. For the problem in Figure 11a, the compiler first searches the space of topologies and for each topology it creates an optimization task. Let us assume that it has reached the topology

Table 5. Symbolic setup of the method of joints for the example in Figure 2.

$$\begin{pmatrix} cos(\alpha_1) & cos(\alpha_2) & 0 & 0 \\ sin(\alpha_1) & sin(\alpha_2) & 0 & 0 \\ 0 & cos(\alpha_2+180) & cos(\alpha_3) & cos(\alpha_4) \\ 0 & sin(\alpha_2+180) & sin(\alpha_3) & sin(\alpha_4) \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix} = \begin{pmatrix} Lcos(\gamma) \\ Lsin(\gamma) \\ 0 \\ 0 \end{pmatrix}$$

where:

$$\begin{aligned} cos(\alpha_1) &= (x_1 - x_l)/l_1, & cos(\alpha_2) &= (x - x_l)/l_2 \\ cos(\alpha_3) &= (x_1 - x)/l_3, & cos(\alpha_4) &= (x_2 - x)/l_4 \\ sin(\alpha_1) &= (y_1 - y_l)/l_1, & sin(\alpha_2) &= (y - y_l)/l_2 \\ sin(\alpha_3) &= (y_1 - y)/l_3, & sin(\alpha_4) &= (y_2 - y)/l_4 \end{aligned}$$

and l_i's are Euclidean distances:

$$\begin{aligned} l_1 &= \sqrt{(x_1 - x_l)^2 + (y_1 - y_l)^2} \\ l_2 &= \sqrt{(x - x_l)^2 + (y - y_l)^2} \\ l_3 &= \sqrt{(x - x_1)^2 + (y - y_1)^2} \\ l_4 &= \sqrt{(x - x_2)^2 + (y - y_2)^2}. \end{aligned}$$

shown in Figure 11c (and 11d). The abstract objective function for the skeletal design problem is:

$$Weight = \sum_{\substack{\text{tensile} \\ \text{members}}} \|F_i\|\, l_i + c \sum_{\substack{\text{compressive} \\ \text{members}}} \|F_i\|\, l_i. \tag{4.4}$$

Using the fact that the topology has 4 members and by unrolling the loop implementing the summations, the compiler produces the objective function:

$$Weight = c_1\|F_1\|\, l_1 + c_2\|F_2\|\, l_2 + c_3\|F_3\|\, l_3 + c_4\|F_4\|\, l_4 \tag{4.5}$$

where the c_i's depend on the unknown stress state of the solution; that is, their value is:

$$c_i = \begin{cases} 1 & \text{if } F_i > 0 \\ 50 & \text{otherwise.} \end{cases}$$

Besides specifying the number of members, the topology also gives the connectivity among members. This allows the compiler to symbolically compute all quantities in the method of joints. For the example in question, these are illustrated in Table 5. The system of equations must still be solved to obtain a closed form solution for the objective function in Equation 4.5.

The novel symbolic techniques initiated by Macsyma and nowadays available in powerful off-the-shelf packages like Maple and Mathematica (Wolfram 1988) make this latter task much easier. In fact, we have used a compiler written in the programming language of Mathematica (Maeder 1989) to solve and simplify such systems of equations symbolically. We have written a procedure that solves the linear system of equations using Cramer's rule and the simplification routines available in Mathematica. Cramer's rule gives a solution as the ratio of two determinants computed from the matrix of coefficients. It turns out that numerator and denominator of this ratio are much easier to simplify individually, because they are only polynomials. Therefore, our compiler first uses Cramer's rule to solve the linear system, then simplifies the two polynomials separately. Finally, the compiler combines all terms by factoring out and deleting common factors between numerator and denominator. The simplified output is then combined and simplified again to produce the symbolic solution to the system of equations. Table 6 shows the expression of the force in the member E1 that joins the load L and the support S1 in Figure 2. This expression has been obtained using our compiler and it is the solution for the force F_1 computed from the system of equations in Table 5. Similar expressions have been derived by our compiler for the remaining forces but, for the sake of brevity, are not shown. It must be noticed that the symbolic expression in Table 6 explicitly contains the unknown coordinates x and y of the connection point C in addition to all givens of the problem. These solutions are then plugged into Equation 4.5. The resulting expression, which is omitted for brevity, is not yet in closed form, because the c_i's depend on the sign of each solution and this cannot be computed symbolically.

It is important to stress that all the operations described above were performed on symbolic expressions, a task that is traditionally difficult to automate. However, symbolic manipulation packages like Mathematica make these operations easier to perform and the corresponding compilers relatively easy to write.

Table 6. Closed-form of the internal force for member E1 in Figure 2.

Internal Force in member E1 =

$$
\begin{aligned}
p\,\Big[&+ \big((x_1 - x_l)^2 + (y_1 - y_l)^2\big) \\
&\qquad \big((x_2 - x)(y_1 - y)(y - y_l) - (x_1 - x)(y_2 - y)(y - y_l)\big)cos(\gamma) + \\
&\qquad \big((x_1 - x)(x - x_l)(y - y_l) - (x_2 - x)(x - x_l)(y_1 - y)\big)sin(\gamma)\Big] \Big/ \\
&\Big[\big((x_2 - x)(y_1 - y) - (x_1 - x)(y_2 - y)\big)\big((x - x_l)(y_1 - y_l) - (x_1 - x_l)(y - y_l)\big)\Big]
\end{aligned}
$$

4.3.2 Instance Simplification

The second specialization step is to plug the givens of the problem into each of the expressions obtained during topological simplification and to partially evaluate the resulting mixed symbolic/numeric expression. For the skeletal design domain, the givens of the problems are the loads and supports; however, one may wish to analyze a structure subject to different inputs such as various loading conditions or support locations. In such cases it is possible to leave those values in symbolic form and substitute their numerical values at run time. For the example in Figure 2, we choose loads and supports as givens. The expression of the internal force in E1 is shown in Table 7. This indicates that the force is now reduced to a closed-form expression of the coordinates x and y of the (unknown) connection point C.

From a formal standpoint, instance simplification is partial evaluation. The compiler is given a symbolic expression representing the objective function and a set of assignments for some of the variables. It then uses Mathematica to plug these givens into the objective function and to simplify the resulting expression. The result is a specialized objective function in symbolic form in which the values have been substituted for the variables. Moreover, the resulting expression might be faster to evaluate than the original objective function. Again, the power of symbolic packages comes into play by making the substitution and simplification steps easier to implement.

Table 7. Internal force for member E1 in Figure 2 with givens.

Internal Force in member E1 = $2236(y - 6000)/(y - 2x - 2000)$

4.3.3 Case Analysis

The third compilation step is to split the objective function obtained from instance simplification into even smaller cases. For the function in Equation 4.5, this corresponds to combinatorially exploring all values of the signs for the forces and substituting the related values of c_i into the equation. From a physical standpoint, each combination corresponds to a stress state. Therefore, at compile time, it is possible to tell which terms should be multiplied by 1 and which ones have a coefficient of 50. Hence, Equation 4.5 can be expressed in a closed form. This also implies that each function becomes differentiable. A differentiable objective function enables the problem solver to employ, at run time, gradient-based optimization techniques which are typically faster than methods based only on evaluations of the objective function alone.

For illustrative purposes, let us refer to Figure 7 in which we have plotted the weight of the structure for the topology in Figure 2 as a function of the coordinates x and y of the connection point C. Each unimodal region in the figure corresponds to one or more stress states; for instance, $(+1, -1, +1, -1)$ corresponds to region R1. This correspondence between stress states and unimodal regions is exploited by the case analysis, which partitions the whole region and produces one objective function per stress state. Each function is then obtained by abductively assuming that the stress state is known and substituting this knowledge into the objective function. In the skeletal design domain, once the stress state is known, it is possible to determine if a member is tensile or compressive. For instance, having assumed the stress state $(+1, -1, +1, -1)$, the specialized objective function for Equation 4.5 is shown in Table 8. This expression is a closed form expression which is in contrast with the original, more general, formulation of the weight function.

Table 8. Simplified objective function for the problem of Figure 2.

$$\begin{aligned}
&Weight = \\
&\big(1.14\ 10^{13}x - 5.66\ 10^{9}x^2 + 8.16\ 10^{5}x^3 + \\
&\ 3.28\ 10^{13}y - 3.26\ 10^{9}xy + 2.44\ 10^{5}x^2y - \\
&\ 6.70\ 10^{9}y^2 + 8.16\ 10^{5}xy^2 + 2.44\ 10^{5}y^3 - 4.08\ 10^{16}\big) \Big/ \\
&\big(1.28\ 10^{1}xy - 2.56\ 10^{4}x + 2.56\ 10^{4}y - 6.40\ y^2 - 2.56\ 10^{7}\big)
\end{aligned}$$

Table 9. Quadratic approximations of the gradient

$$\begin{aligned}
&Quadratic\ Approximation\ of\ the\ gradient = \\
&\quad -1060187.5 + 159.4\ x + 0.05x^2 - 68.7\ y + 0.2\ x\ y - \\
&\qquad 0.00005\ x^2\ y + 0.03\ y^2 - 0.00003\ x\ y^2 + 5.9\ 10^{-9}\ x^2\ y^2, \\
&\quad -2909656.25 + 1860.4375\ x - 0.3\ x^2 + 1207.6\ y - 0.8\ x\ y + \\
&\qquad 0.0001\ x^2\ y - 0.14\ y^2 + 0.00009\ x\ y^2 - 1.5\ 10^{-8}\ x^2\ y^2
\end{aligned}$$

4.3.4 Gradients

The closed form expressions obtained at the previous step are differentiable and allow the use of gradient-based numerical optimizers. An advantage of gradient descent methods is that they need to evaluate the objective function less often because they are able to take larger, and more effective steps. Of course, they incur the additional cost of repeatedly evaluating the partial derivatives of the objective function. Hence, they produce substantial savings only when the reduction in the number of function evaluations offsets the cost of evaluating the derivatives.

For highly non-linear functions like the one in Table 8, the resulting gradient is at least as expensive to compute as the original function. Therefore, as we will show, overall we do not obtain a speed up for the optimization process. This obstacle can be circumvented at compile time by approximating the gradient via Taylor series expansion to produce a further speedup in the optimization process. The quadratic approximations obtained at compile time of the gradient of the weight function in Table 8 is shown in Table 9. The polynomial is fast to evaluate and allows another significant speed up in the optimization process as shown in the next section.

4.4 Experiments

To test the efficacy of this approach, we have solved a series of design problems, and we have measured the impact of the compilation stages on the evaluation of the objective function, on the optimization task, and on the reliability of the optimization method. The measurements presented are averages over five randomly generated design problems and, for each design, over 25 randomly generated starting points.

4.4.1 Objective Function

The objective function of each design problem was evaluated in four different ways and, for each of them, we averaged the CPU[7] time over the different designs and starting points. The weight was first computed using the traditional, naive, numerical procedure with the method of joints. We then compiled the designs incorporating, in three successive stages, topological information, the givens of the problems, and the stress state. Figure 19 shows the time (summed over 100 runs)

[7]The examples were run on a NeXT Cube with a Motorola 68030 chip.

to evaluate the objective function at the various compilation stages. The biggest speedup was obtained with the numerical substitution of values into the symbolic closed form expression and with the specialization to stress states. This suggests that the gain is related to the elimination of arithmetic operations from the original numerical problem.

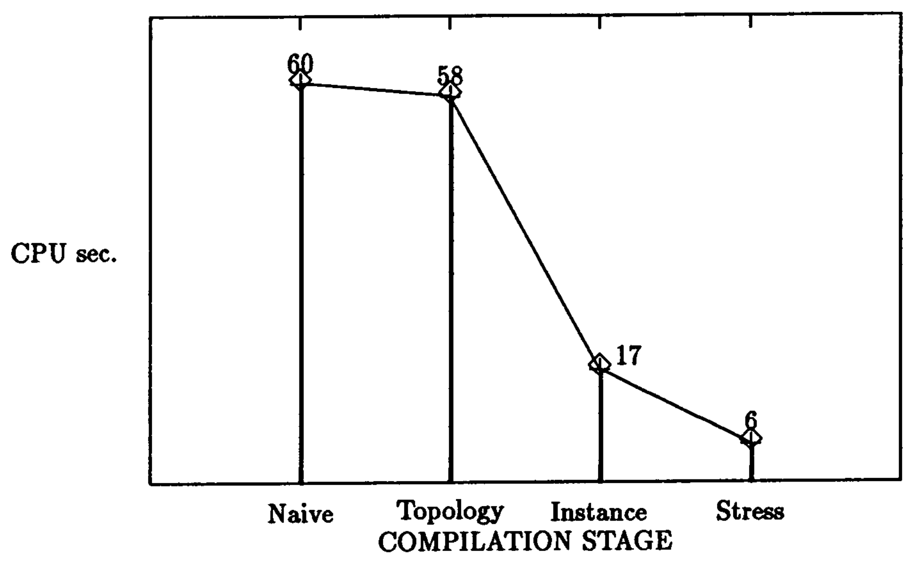


Figure 19. Average CPU time per function evaluation.

4.4.2 Optimization

As indicated in Section 4.2, the running time of the optimizers is influenced by the number of function calls and by the time for each function evaluation. To present the benefits of our approach on the optimization task, we have experimented with two optimization algorithms (a) an optimizer based on Powell's method that does not require gradient information and (b) the version of conjugate gradient descent (Press et al. 1988) provided by Mathematica. The graphs in Figures 20 and 21 report, respectively, the number of objective-function calls and the overall CPU time for each optimizer. The values connected by solid lines correspond to cases where the optimizer had no gradient information, while the values connected by dashed lines indicate averages utilizing the conjugate gradient descent method with alternative approximations for the gradient vector.

As expected, the number of evaluations remains constant throughout the compilation stages when the non-gradient method is used, while it decreases drastically when we switch to the gradient-based optimization method. The overall

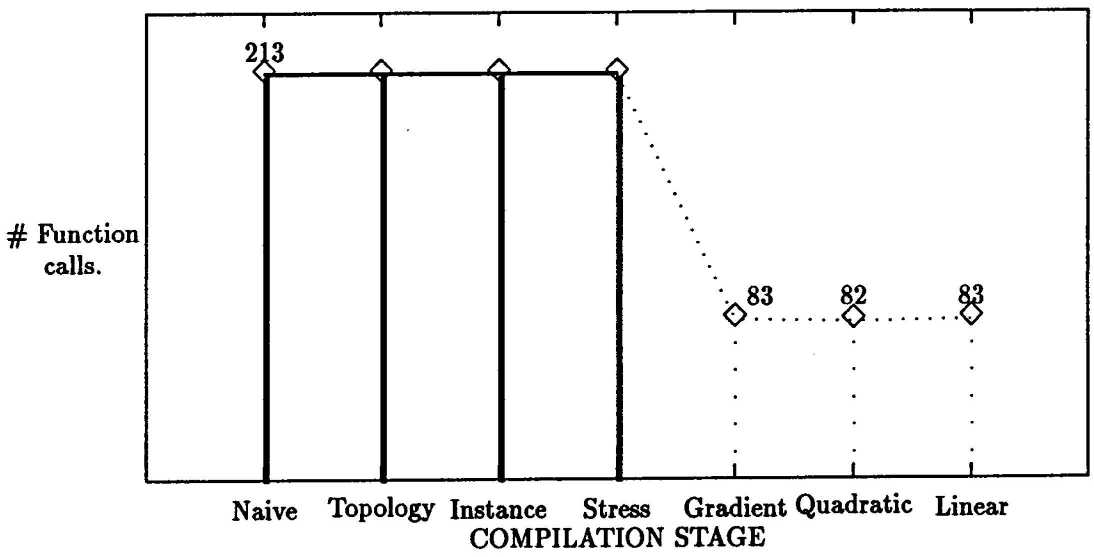


Figure 20. Average number of function calls.

CPU time (Figure 21) steadily decreases as well. For the non-gradient method, the decrease is due to the progressive simplification of the objective function itself, so that it is cheaper to evaluate. When we switch to the gradient method, there is initially no speedup at all, because the cost of evaluating the full gradient offsets the decrease in the number of times the objective function must be evaluated. However, additional speedups are obtained by approximating the objective function as a quadratic and as a linear function (by truncating its Taylor series).

We have found experimentally that there is no appreciable difference between the minima reached using the full gradient vector and the minima computed using quadratic approximations of the partial derivatives. However, the precision of the results obtained with the linear approximation is significantly reduced. Depending

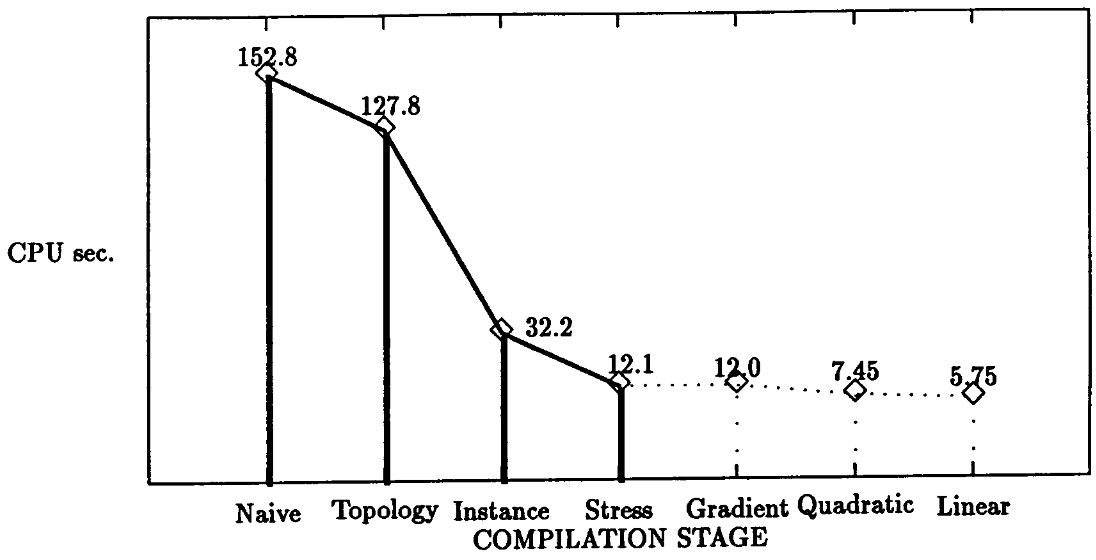


Figure 21. Average CPU time.

on the application, this trade of accuracy for speed may be acceptable. If not, the quadratic approximation should be employed.

Another possibility is to employ the linear approximation for the first half of the optimization search, and then switch to the quadratic approximation once the minimum is approached. In other words, the linear approximation can be applied to find a good starting point for performing a more exact search.

4.4.3 Reliability

An optimization method is reliable if it always finds the global minimum regardless of the starting point of the search. Unfortunately, as shown in Figure 7, the objective function in this task is not unimodal, which means that simple gradient-descent methods will be unreliable unless they are started in the right "basin." It is the user's responsibility to provide such a starting point, and this makes numerical optimization methods difficult to use in practice.

From inspecting graphs like Figure 7, it appears that, over each region corresponding to a single stress state, the objective function is unimodal. We conjecture that this is true for most of 2-D structural design problems. If true, this could imply that optimization can be started from *any* point within a stress state, and it would always find the same minimum. If this is true, then our "divide-and-conquer" approach of searching each stress state in parallel will be guaranteed to produce the global optimum.

We have tested these hypothesis by performing 20 trials of the following procedure. First, a random starting location was chosen from one of the basins of the objective function that did not contain the global minimum. Next, two optimization methods were applied: the non-gradient method and the conjugate gradient method. Finally, our divide-and-conquer method was applied using, for each of the specialized objective functions, a random starting location that exhibited the corresponding stress state. In all cases, our method found the global minimum while the other two methods converged to some other, local minimum.

4.5 Concluding Remarks

Our overall strategy for speeding up numerical optimization methods relies on successive specializations and simplifications of the objective function by: (a) specializing the objective function by incorporating the invariant aspects of the particular problem and (b) splitting the objective function into special cases based on stress states. This produces an optimization problem expressed in terms of the coordinates of the connection points. This approach is in absolute concordance with research in engineering in the context of non-statically-determinate structures. For instance, Reich and Fuchs (1989) show the superiority over other problem formulations of the Explicit Design Method in which the objective function is represented in terms of the coordinates of the connection points. This suggests that our methods have a strong practical basis, because throughout our solution we have used the coordinates of the connection points as unknowns.

From a numerical optimization standpoint, the benefits of the specializations we have introduced are great. First, the cost of evaluating the objective function is reduced. Second, the specializations make it possible to obtain differentiable closed forms for the objective function. This allows us to apply gradient-directed optimization methods. Third, the specializations create opportunities for parallel execution of the optimization calculations.

Our symbolic approach is complementary to the traditionally difficult task of writing compilers (Berliner 1989) for numerical programs. In this chapter we have shown how to use powerful high-level symbolic packages like Mathematica as basic tools to write specialized compilers. Admittedly, however, these compilers are slower than ones written for production. Nevertheless, we can argue that high level symbolic manipulation packages can be used by compiler writers to prototype production compilers and by users, like engineers, who are not required to be experts in compiler technology and, yet, they can still write their own compilers.

While the approach shown in this chapter is extremely useful to speedup a single numerical optimization problem, it does not suffice to solve the entire design problem. This is because the double specialization in the spaces of topologies and stress states introduces a combinatorial explosion. Nevertheless, not all of the 2^m stress states make physical sense or produce an optimal solution. In Chapter 6 we show how our system learns search control rules to prune candidate solutions that are useless or that do not lead to optimal solutions. In the meantime, in the next chapter, we illustrate how to use inductive knowledge compilation techniques to produce another source of constraining knowledge. This can also be incorporated into the objective function producing a further speedup of the individual optimization tasks.

Chapter 5

Eliminating Independent Variables

5.1 Introduction

The number of independent variables (*dimensionality*) in an optimization problem plays an important role in the reliability and speed of numerical optimizers. In fact, Pike (Pike 1986) shows that the larger the dimensionality, the less reliable and the slower the numerical optimization methods are. One way to circumvent this drawback is to decrease the number of independent variables. This can be accomplished by incorporating problem-specific knowledge into the optimization process. As an illustrative example, let us suppose that we wish to minimize the seemingly simple function:

$$f(x,y) = \frac{x^2\, y^3}{(y-1)^2}$$

whose plot is shown in Figure 22. Depending on the starting point, a conjugate gradient optimizer may or may not find a minimum. For instance, when the `FindMinimum[]` package in Mathematica[8] is started at the point of coordinates $(x = 2, y = -5)$ it fails to determine any minimum. On the other hand, when the starting point is $(x = 2, y = 5)$, the numerical package determines a local minimum at $(x = 0, y = 4.8)$ in 9.5 seconds for 20 iterations on a SUN Sparc 2. Now, let

[8] We wish to thank Igor Rivin of Wolfram Research for providing useful details on the `FindMinimum[]` package in Mathematica.

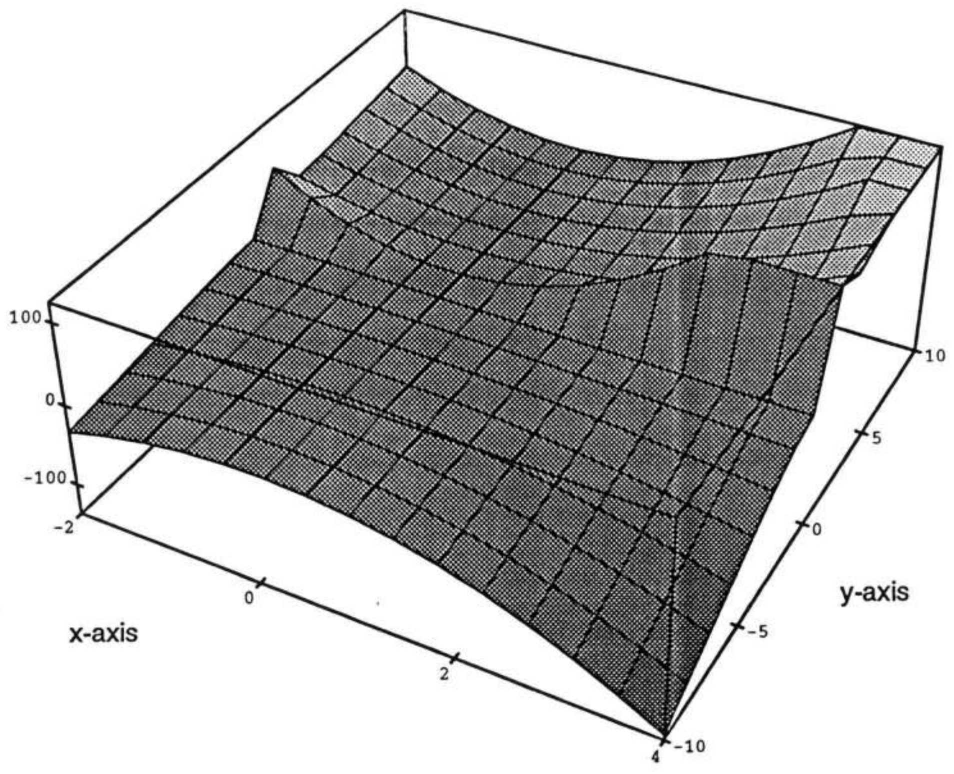


Figure 22. Plot of the function $f(x,y) = \frac{x^2\,y^3}{(y-1)^2}$.

us suppose that, using domain knowledge, we are able to determine that, at the optimum, the relation $y = 2$ holds for all x. Using this relation, the objective function can be simplified to $f(x) = 8x^2$ and the optimization problem becomes trivial. `FindMinimum[]` always finds a solution quickly regardless of the initial point, and it takes 0.9 seconds (10-fold speedup) to solve 20 problems.

There should be little doubt that, under most circumstances, the decreased dimensionality of the problem greatly simplifies the numerical optimization task. The goal then is to identify *regularities*; that is, constraining relations among independent variables. The relation $y = 2$ is an example of a regularity. These relations can then be included in the partial evaluation process illustrated in Chapter 4 to reduce the dimensionality of the optimization task thereby obtaining a much simpler problem for the run time optimization. Our approach mimics strategies that are commonly used in solving (Polya 1973) mathematical problems. In fact, often mathematicians use algebra to transform a problem into an equivalent one for which known relationships can be applied to simplify the search for a solution. In many cases, a simple change of the coordinate system greatly simplifies the formulation of a task. For instance, the equation of a circle whose center is the origin of the axes is simpler than the equation of the same circle in a different coordinate system.

To find regularities, we have employed two inductive techniques that rely on training examples of the optimal solutions. The first, Explanation-Based Regularities, uses domain knowledge to hypothesize regularities and training examples to focus the search for the "appropriate" formulation of the algebraic problem. The second, regression, determines regularities by using statistical (SAS 1989) linear regression techniques.

Ideally, given either sufficient domain knowledge or powerful regression techniques, one should be able to determine regularities among all variables. This would completely circumvent numerical optimization. Realistically, instead, some regularities are difficult to find. The goal is then to determine as many of them as possible, without striving for completeness. For instance, let us consider Figure 23 which shows a polar representation of a point C. One might be able to determine a regularity involving the angle of the vector α but might fail to determine the distance ρ of the point from the origin of the vector. This regularity is *incomplete* because only one of the components needed to locate the point has been determined. Nevertheless, as we demonstrate in the next section, this information is extremely useful, because the regularity can still be used to reduce the dimensionality of the optimization problem.

5.2 A Design Example

To see how the dimensionality can be decreased in the optimization task described in Chapter 2, let us consider again the problem in Figure 24, which we shall refer

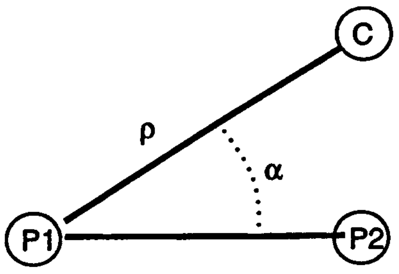


Figure 23. **A point C in polar coordinates α and ρ.**

to as the "bisector" example. In this example, the coordinates x and y of the connection point C are the unknowns. The givens are the load L and the supports S1 and S2. Moreover, let us assume that a set of training examples of the optimal solution has been either provided or derived by the system itself. The goal of this simplification phase is to find constraining relations (*regularities*) between the coordinates of C and the givens of the problem. Regularities are detected by inductively analyzing the training examples.

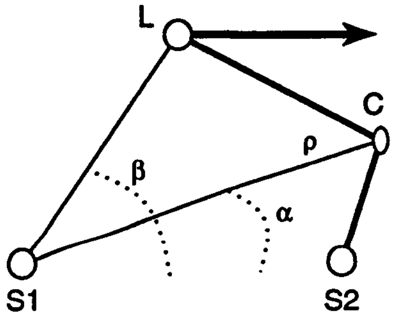


Figure 24. **The bisector example.**

The first step transforms the engineering problem into a geometric one. This is necessary in order to use the geometric knowledge-base that was given to the system. This process can be considered as a translation and it is accomplished by parsing each training example and "recognizing" geometric objects. The translation is possible because the knowledge-base includes rules that bridge the gap between geometry and the domain task. For the bisector example, the system identifies, among others, the following geometric objects:

```
point(S1), point(S2), point(C), point(L),
angle(β, L, S1, S2), angle(α, C, S1, S2),
segment(Sg1, S1, S2), ...
```

where each point is identified by a `point(P)` predicate, an angle (see Figure 23) between three points is recognized by `angle(Ω, P1, P2, P3)`, and a segment with `P1` and `P2` as endpoints is described by the predicate `segment(Sg, P1, P2)`. In the bisector example, `point(C)` and all predicates that involve it in their derivation (e.g. `angle(α, C, S1, S2)`) are unknowns, all others are givens.

With this knowledge, the system then tries to relate the unknown geometric entity `point(C)` to as many other entities as possible with the ultimate goal of expressing it using only given geometric entities. This is accomplished by using a blend of EBL and discovery techniques. To visualize this process, let us refer to the derivation tree in Figure 25. During the first step, a domain rule is used to transform the point from cartesian to polar coordinates. As shown in Figure 23, the domain rule states that a point can be identified by its distance ρ from `S1` and by the angle α between points `C`, `S1`, and `S2`. With this in mind, the system recursively tries to determine `angle(α, C, S1, S2)` and `distance(ρ, C, S1)`. After having explored all proofs, the system concludes that it is not possible to re-express the `angle` and the `distance` in terms of known entities. If we were to follow EBL strictly, we should conclude that the domain theory is incomplete; that is, it is not powerful enough to bridge the gap between unknowns and givens. This, in turn, implies that the search would terminate concluding that `point(C)` cannot be re-expressed in terms of known geometric objects.

To overcome this problem we have used a discovery approach that fills these knowledge gaps. The training examples of the optimal solutions are used to discover relations among variables. For the problem in Figure 24, we analyze the

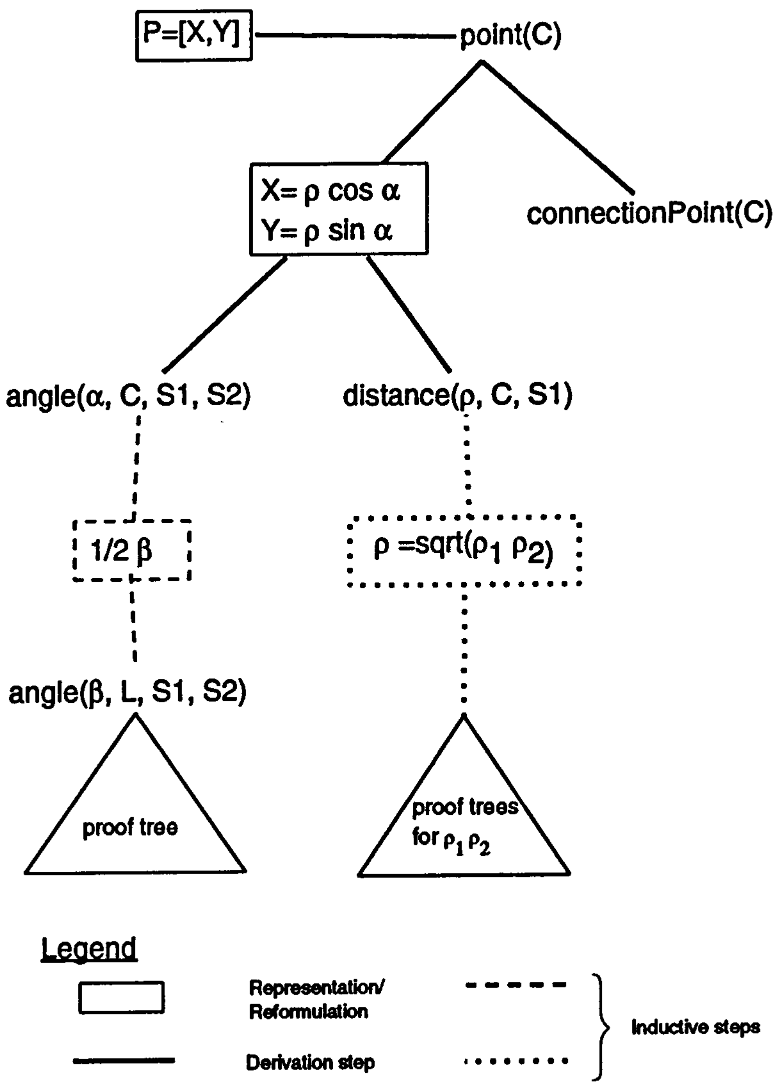


Figure 25. Derivation tree to express the unknown `point(C)` in terms of given geometrical entities.

training examples and determine that the angle α between points C, S1, and S2 is one-half the angle β between points L, S1, and S2. This relation is assumed as a transformation of the unknown angle α. This process is shown by the nodes in Figure 25 connected by the dashed lines. The system then recurs on the points L, S1, and S2. These represent the geometric entities that were used to recognize the angle β. Simple lookup reveals that these entities are givens of the problem, because they were derived from the position of the load and supports; thus, the search terminates. For the time being, let us disregard the branch in the figure indicated by the dotted lines; it will be explained in Section 5.4.

Table 10. Simplified objective function for the problem of Figure 2.

$$\begin{aligned}
&Weight = \\
&\quad \big(1.14\ 10^{13}x - 5.66\ 10^{9}x^2 + 8.16\ 10^{5}x^3 + \\
&\quad\ 3.28\ 10^{13}y - 3.26\ 10^{9}xy + 2.44\ 10^{5}x^2y - \\
&\quad\ 6.70\ 10^{9}y^2 + 8.16\ 10^{5}xy^2 + 2.44\ 10^{5}y^3 - 4.08\ 10^{16}\big) \big/ \\
&\quad \big(1.28\ 10^{1}xy - 2.56\ 10^{4}x + 2.56\ 10^{4}y - 6.40\ y^2 - 2.56\ 10^{7}\big).
\end{aligned}$$

Table 11. Objective function for the structure in Figure 2 with reduced dimensionality.

$$\begin{aligned}
&Weight_{simplified} \approx \\
&\quad \big(1.16\ 10^{13}\rho - 5.19\ 10^{9}\rho^2 + 8.19\ 10^{5}\rho^3 - 4.08\ 10^{13}\big) \big/ \big(3.95\rho^2\big)
\end{aligned}$$

To summarize the derivation process in Figure 25, the x and y coordinates of unknown connection point C can be expressed in terms of the angle α and the distance ρ. In turn, the angle α is substituted by $\frac{\beta}{2}$ which can be computed from the given position of the load and supports. These transformations represent an incomplete regularity, because we have related one of the independent variables needed to identify C to given quantities. This regularity can then be incorporated into the objective function of the optimization problem shown in Table 10. For the bisector example, using the symbolic techniques described in Chapter 4 we obtain the resulting objective function shown in Table 11. This expression shows a reduction of the dimensionality of the problem from two to one. To see this, let us compare the expression in Table 10 with the one in Table 11. The former depends on the two independent variables x and y which represent the cartesian coordinates of the connection point. The latter expression of the objective function, instead, only depends on the distance (ρ) of the point C from support S1. This distance is the only unknown. Therefore, the optimization problem is one-dimensional and can be solved quite efficiently at run time by a numerical optimizer. Having seen how independent variables can be eliminated from the optimization process, let us now turn to illustrate the method in greater detail.

5.3 Knowledge-Based Regularities

To determine regularities in the skeletal design problem we must be able to relate unknown connection points to givens of the problems such as loads and supports. The approach we have used is based on a mix of EBL (Ellman 1989) and discovery techniques. The domain theory for EBL includes domain and geometric knowledge, and it is described in Section 5.3.1. However, this domain theory is incomplete (Tadepalli 1989), (Ellman 1989). In our framework, an incomplete domain theory does not allows the EBL-system to derive unknowns from givens. This implies that independent variables cannot be eliminated. To fill the gaps in the proofs derived by EBL, we have used a slightly modified version of the discovery techniques discussed in Section 3.2.4. During discovery, the training examples of the optimal solutions are analyzed to determine relations among geometric entities inductively. These relations are then substituted into the proof derived by EBL and the system recurs on the geometric entities in the new relation. However, the discovery phase might fail to determine relations among geometric entities. These failures lead to incomplete (see Figure 26) regularities. The remainder of this section introduces first the geometric domain theory and then gives the sequence of steps which lead to the regularities.

5.3.1 Geometric Knowledge Base

There are three parts to the knowledge base: (a) primitive geometric entities, (b) relationships between engineering components and geometrical entities, (c)

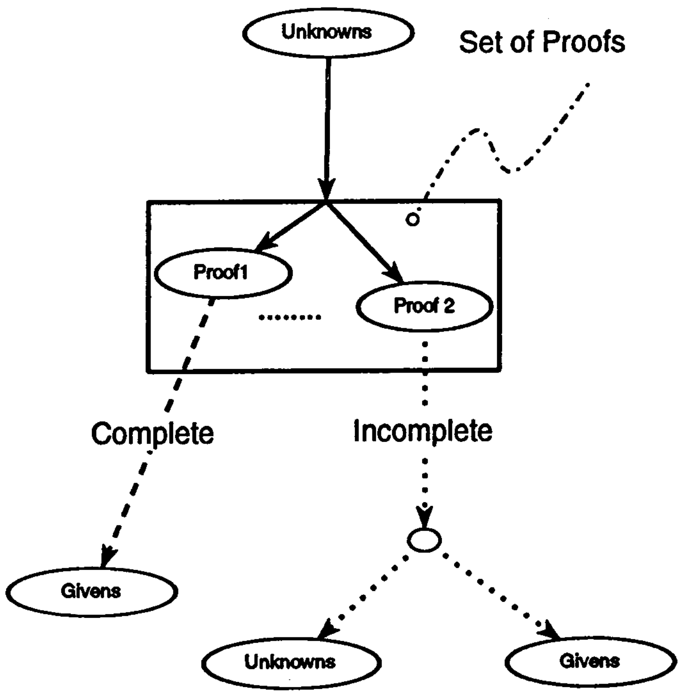


Figure 26. Types of regularities: (a) complete, (b) incomplete.

relationships among geometric entities. These parts are illustrated below in a Prolog-like notation.

Primitive geometric entities. This component of the domain theory establishes the language in which geometric entities are expressed. Table 12 enumerates seven primitive geometric entities: points, angles, distances, lines, line segments, semiplanes, and regions. Each entity has a standard form. For example, points are represented as (x, y) coordinate pairs. Lines are represented by the three coefficients in the equation $ax + by + c = 0$. Semiplanes are represented by a line and a comparison symbol (either $\geq$ or $\leq$). By substituting the comparison symbol for the $=$ sign in the linear equation, one obtains the appropriate semiplane. Line segments are represented by a distinguished endpoint and a direction vector that when added to the distinguished endpoint produces the other endpoint.

Table 12. Relations among geometrical objects.

Relation	*Explanation*
`point(P)`	P is a point.
`angle(`α`,P1,P2, P3)`	α is the angle between points P1, P2, and P3 (See Figure 23 with P3 = C.)
`distance(`ρ`,P1,P2)`	ρ is the cartesian distance between points P1 and P2 (See Figure 23.)
`line(Ln,P1,P2)`	Ln is the line that passes through points P1 and P2.
`segment(Sg,P1,P2)`	Sg is the segment with endpoints P1 and P2. P1 is the designated endpoint.
`semiplane(Sp,FR)`	Sp is a semiplane defining region FR.
`region(FR)`	FR is a collection of semiplanes.

Table 13. Relations between engineering terms and geometric entities.

```
point(P)  :-support(P).
          :-load(P).
          :-connectionPoint(P).
segment(E, P1, P2) :- edge(E, P1, P2).
```

Relations between problem and geometry. The engineering components of the problem (loads, supports, connection points, and edges) are connected to the geometric entities by the rules in Table 13. These rules indicate that supports, loads, and connections are also points and that edges are line segments. In addition, it is known to the system that loads and supports are givens of the problem and that connection points are unknowns.

Relations among geometric entities. There are many relationships that can be defined among geometric primitives. The full domain theory provides rules for defining each relationship and a sample is given in Table 14. It must be noticed that the rules shown in the table are recursive and, hence, so is the whole domain

Table 14. Rules to derive geometric entities.

```
line(L, P1, P2) :- /* line L passes through points P1 and P2 */
    point(P1), point(P2), P1=[X1,Y1], P2=[X2,Y2],
    B=X1 - X2,
    (isZero(B) THEN A=1, C=-Y2 ELSE A=Y1-Y2, C=X1 Y2-Y1 X2)
    L = [A, B, C].
point(P):- /* point P is the intersection of lines L1 and L2 */
    line(L1), L1 = [A1, B1, C1], line(L2), L2 = [A2, B2, C2],
    notParallel(L1, L2), D = (A2 B1 - A1 B2),
    X = (C1 Y2 - C2 Y1) / D, Y = (C2 X1 - C1 X2) / D,
    P = [X, Y].
```

theory. For instance, a line is defined in terms of points, and a point is defined as the intersection of two lines. Recursive rules represent a problem during search because they can lead to infinite loops.

5.3.2 Search Control

Unlike most EBL systems which reason backward from the goal, our search strategy mixes backward and forward chaining. This is necessary because of the recursive nature of the domain theory. For expository purposes, we divide this search into phases which are interleaved in the implementation:

- Identify geometric objects (parsing),
- Use the geometric knowledge to construct proofs for unknowns, and
- Discover relations that relate given and unknown geometric objects.

The three phases are described in the remainder of this section.

Phase 1: Parsing. Reasoning begins with the engineering entities: loads, supports, connection points, and edges. It then progresses by identifying geometric objects listed in Table 12 using the domain theory in Tables 13 and 14. Each geometric object extracted by this process is labelled either as **given** or as **unknown**. An object is a **given** if it is derived using a geometric rule from other **given** objects; it is an **unknown** otherwise. At the outset, connection points and edges with connection points as endpoints are unknown.

Phase 2: Proof of unknown entities. This is the EBL phase. For each unknown, the system uses the rules in the given domain theory to transform unknowns into givens. All rules used in the process are chained and proofs are collected. If there is a proof that contains only given entities then the process stops and outputs the proof. Otherwise, it uses with the next step to discover regularities to fill gaps in the proof.

Phase 3: Discovery. This is the inductive stage of the search. When this phase is reached, it means that the system was unable to complete the proof with given entities. Therefore, it tries to fill the gaps in the proofs inductively. The system analyzes the proof and, starting from the most abstract level, it extracts all unknown geometric entities. For instance, in the bisector example, it determines that in order to compute the point, it must determine an angle and a distance (see Figure 25.) All new unknowns are set as subgoals, and the domain rules are used to determine if the unknowns can be derived from known geometric entities. In the bisector example, the unknown angle α is one half of β, a known angle. Once the relation is discovered in the example at hand, the system tests the hypothesis for consistency against all training examples. For each one of them, the system determines if the relation is true or false. A threshold is then used to determine if the relation is to be considered true. If it is assumed true, the relation is substituted into the proof for the unknown entity so that it can be later incorporated into the objective function. In the end, when all proofs are analyzed, the system chooses the first one with the least number of unknowns in the proof tree. In other words, it chooses the proof which inductively determines the largest number of unknowns.

5.3.3 Incorporation into the Objective Function

Ths chosen proof is the regularity to be incorporated in the objective function to decrease the dimensionality. This is accomplished by analyzing the proof tree and collecting the algebraic relationships among variables. The objective function and these relations are then used as input to the partial evaluator implemented in Mathematica and discussed in Chapter 4. For instance, for the bisector example, the system analyzes the tree in Figure 25 and determines the following relations:

$$\begin{aligned} x &= \rho\cos(\alpha) \\ y &= \rho\sin(\alpha) \\ \alpha &= \beta/2 \\ \beta &= 63^\circ. \end{aligned}$$

These are substituted in the expression in Table 10 to yield the simplified expression in Table 11. After all these steps are taken, the objective function *may* contain fewer independent variables. As demonstrated by the experiments in Section 5.5, the simplification of the optimization task triggers a reduction in the time it takes the numerical optimizer to produce a minimum.

5.4 Numerical Discoveries

In section 5.3.2, we used domain rules to discover relations among geometric entities and to fill gaps in the explanations constructed by EBL. The power of this approach is determined by the rules provided to the system. These rules may involve relations that are not always obvious to experts. For instance, in the bisector example, the rules are unable to determine a relation to compute the distance from the givens of the problem. This is shown in Figure 25 by the dotted lines. Therefore, a different technique is needed to determine relations which are difficult to capture and formalize.

Linear regression is a techniques for which explicit rules are not needed. Nevertheless, this technique can be used to infer constraining relationships among unknown variables. We have experimented with the regression package REG in SAS. As an example, in the bisector problem in Figure 24, the relation between angles α and β can also be discovered by linear regression. We computed the values of the angles α and β for all training examples and fed them to REG. The result was the linear expression $\alpha = 0.5\beta$. In addition, for the bisector example, once we assume the bisector relation between α and β, it is possible to determine analytically[9] that there is a functional relationship among the distances. In fact, it is possible to determine that:

$$\begin{aligned}&\mathtt{distance}(\rho,\ \mathtt{S1},\ \mathtt{c})\\ &= \sqrt{\mathtt{distance}(\rho_1,\ \mathtt{S1},\ \mathtt{L})\ \mathtt{distance}(\rho_2,\ \mathtt{S1},\ \mathtt{S2})}\\ &= \sqrt[4]{((\mathtt{S1}_x - \mathtt{L}_x)^2 + (\mathtt{S1}_y - \mathtt{L}_y)^2)((\mathtt{S1}_x - \mathtt{S2}_x)^2 + (\mathtt{S1}_y - \mathtt{S2}_y)^2)},\end{aligned}$$

where the distance is expressed in terms of the cartesian coordinates of the given load L and supports S1 and S2. Once again, the linear regression package REG was used to derive the expression

$$\widehat{\mathtt{distance}}(\rho, \mathtt{S1},\ \mathtt{c}) = 0.428467\rho_1 + 0.564770\rho_2 \tag{5.6}$$

which is a linear approximation of the analytical relation above.

For the bisector example, knowledge of the relations among angles and distances completely circumvents the need for numerical optimization at run time. In fact, the functional form in Equation 5.6 can be substituted for ρ in the expression in Table 11 to obtain an objective function which is a function of the given positions of the load and supports. The position of the connection point is then obtained at

[9]The proof is carried out by applying the law of sines to the triangles S1 - L1 - C and C - S1 - S2.

compile time by collecting all nodes in the proof. This is illustrated by the dotted line in Figure 25. At run time, it is only necessary to substitute the given quantities for the loads and supports and compute the coodinates of the connection point. No run time search is necessary.

Although regression is a powerful technique that can be used to decrease the dimensionality of an optimization problem, it is not the *panacea.* This is because efficient regression techniques only produce a linear model of the unknown relationships and this may not be sufficient. On the other hand, as illustrated in Section 3.2.3, non-linear methods require that the functional form and the model-selection problems be solved a priori and this is too restrictive. In our experiments, we used only linear regression. Model selection was performed automatically by REG. In the bisector example, the system tried all possible combinations of dependent and independent variables. For each of these, it computed the values of the unknown distance ρ and the values of all other known distances for all training examples. These values were given to REG, which was then asked to determine the linear model using forward and backward model selection techniques (see Section 3.2.3). As illustrated by the functional form in Equation 5.6, both methods indicate that only ρ_1 and ρ_2 are significant in the functional form. This approach, however, is very much prone to errors and, ideally, one should indicate to the regression package the variables to be included in the relation.

5.5 Experiments

To demonstrate the efficacy of the approach illustrated in this chapter we have produced two plots. The first indicates that when a reduction in dimensionality is possible, there is a drastic reduction of the run time numerical optimization process. This is shown in Figure 27 which reports the average time it takes to optimize 300 randomly generated design problems with different numbers of connection points. We recall that each connection point corresponds to 2 independent

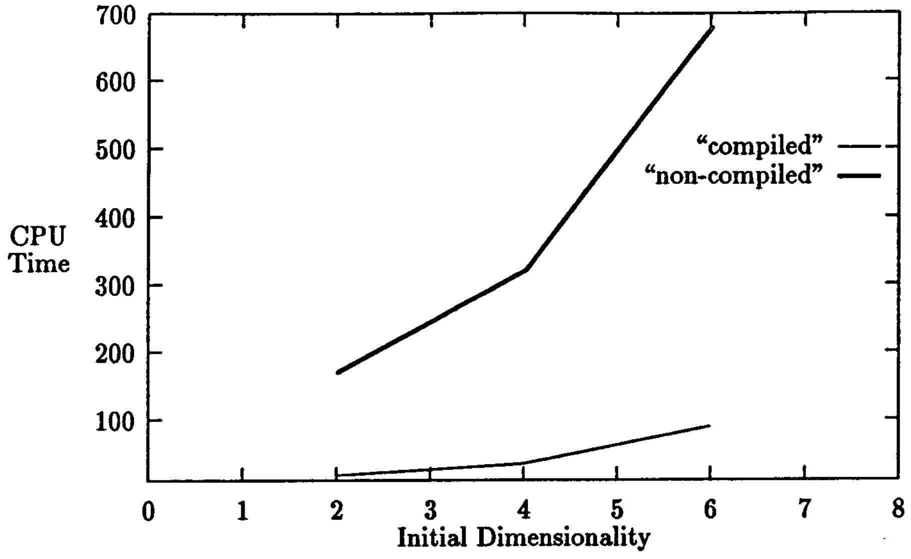


Figure 27. CPU time of the optimization process before and after dimensionality reduction.

variables (coordinates) in the optimization problem. The x axis in the figure indicates the number of independent variables given to the numerical optimizer. The time it takes to optimize is reported on the y axis. The thin line indicates the time before the compilation that yields a reduction in dimensionality. The thick line

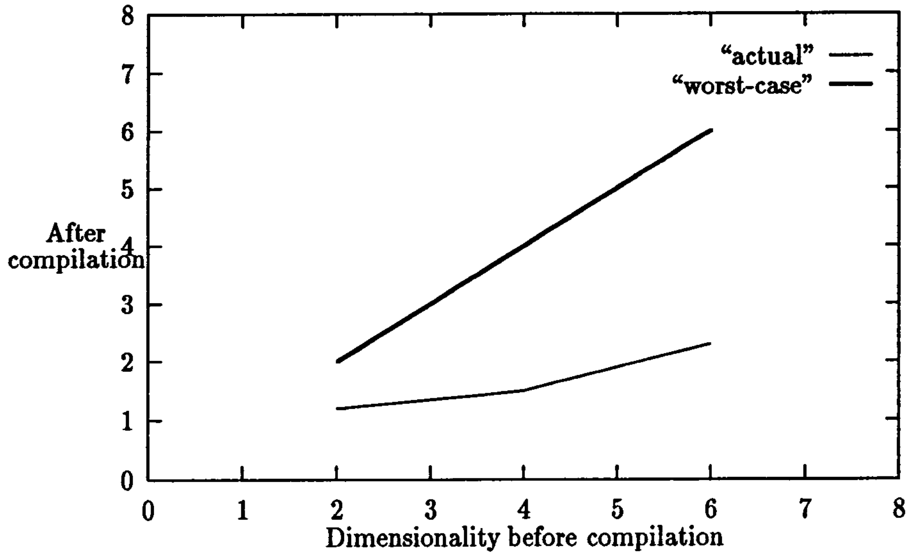


Figure 28. Decrease in the number of independent variables after elimination procedure.

shows the values after the compilation. As expected, there is a drastic reduction in the raw CPU time.

In addition to demonstrating the efficacy of the method when applicable, we have also proved that the method is, indeed, widely applicable. This is necessary, because for the method presented in this chapter to apply, one must consider the following factors:

- Presence of regularities in the domain
- Completeness of the domain theory.

The first suggests that solutions to the optimization problem in the chosen domain must present regularities that can be captured during the search. The second indicates that the domain theory is powerful enough to detect such regularities. For instance, in the bisector example, the solution presents regularities, and the domain theory is complete in the sense that there is a transformation that allows the detection of the regularity. Figure 28 indicates on the x axis the number of independent variables in the initial optimization problem. On the y axis we report the number of independent variables after the compilation stage that reduces the dimensionality. The thin $45°$ line indicates the worst case performance in which the compilation would not have produced any reduction in the number of independent variables. The thick line indicates the average number of variables after the incoporation of the regularities. The results show that there is a reduction in the dimensionality of the problem demonstrating the efficacy of the approach.

5.6 Concluding Remarks

The techniques illustrated in this chapter are used to derive relations among variables in an optimization problem. Combined with the symbolic methods in Chapter 4, these techniques produce a simpler objective function with fewer variables.

The elimination of variables is accomplished by discovering regularites; that is, relating unknown variables to given ones whose values are known. This transformation process applies a mix of EBL and discovery techniques to a set of training examples of the optimal solution. EBL is given a geometrical domain theory and derives proofs of the unknowns in terms of the givens. However, it is almost always the case that these proofs cannot be completed. Under these circumstances, the discovery mechanism bridges the gap between the unknown quantities and the given ones. This is done by analyzing inductively a set of examples of the optimal solutions and deriving simple relations. These are then used to complete the proofs derived by the EBL engine. The trasformations of the variables are derived from the first proof with the fewest number of unknowns. These are incorporated into the objective function which is partially evaluated and simplified using the techniques in Chapter 4. The overall result is the elimination of the transformed variables from the optimization task and, hence, a reduced dimensionality.

We have illustrated two antithetic approaches to discovery which are used in the procedure. The first approach is knowledge-based and uses given relations among geometrical objects. For instance, an angle is one-half of another. The second approach is knowledge-free. It uses linear regression and a set of training examples of the optimal solution to derive relations among variables. The knowledge-based approach provides a higher level of abstraction and the derivations can be used as explanations for a human. However, the rules that they use may not be always obvious. This limits the effectiveness of the approach. On the other hand, the regression technique is certainly more general, because it requires far less knowledge transfer from the expert to the system. However, linear regression only provides linear approximations of the true relations and this might not always be sufficient. Non-linear approximation was discarded, because it requires that the model of the relation be given to the regression package. This is a difficult problem. In our study, we found that regression methods are useful for problems in which it is possible to circumscribe the number of combinations among independent variables.

This is because for these problems all combinations of variables can be tried and the best approximations can then be chosen.

However, certain relations among variables are obvious to humans and cumbersome to express as functions. For these relations, the knowledge-based approach is superior to the knowledge-free one. As an example, in Appendix E we have considered the identification of a point as the intersection of two lines. In conclusion, in large scale mathematical applications we conjecture that to capture relations among variables it is necessary to use a blend of knowledge-free and knowledge-based heuristics.

The experimental results in this chapter are encouraging. The reduced dimensionality of the optimization problem triggers a reduction in the raw CPU time of the run time optimization.

Chapter 6

Learning Search Control

The specialization techniques in Chapter 4 and the simplifications in Chapter 5 have shown how we can quickly find the optimal locations for connection points for a given stress state. Unfortunately, there are exponentially many stress states for a given topology, so even with the speedups obtained from these techniques, we cannot afford to search *every* possible stress state to find the globally optimal solution. In this chapter, we address the task of learning rules that analyze a given problem and topology and select a small number of stress states to be searched. The goal is to learn rules that are correct—that is, the stress state containing the optimal solution is always selected—and efficient—that is, only a few stress states (preferably only one) are selected. The selection rules (Cerbone and Dietterich 1992b) form a kind of control knowledge: we are deciding which parts of the large space (of possible stress states) are worth searching.

Section 6.1 poses the problem of learning optimal search control and stresses the need of a new learning framework. Section 6.2 introduces a formal definition of SETMAXUTIL, the task of learning optimal search control for the problem solver. Section 6.3 contains the proof of $\mathcal{NP}$-completeness of SETMAXUTIL and gives an approximation algorithm. Section 6.4 describes three learning algorithms to learn optimal search control. Section 6.5 demonstrates the effectiveness of the traditional and novel learning algorithms in acquiring optimal control knowledge

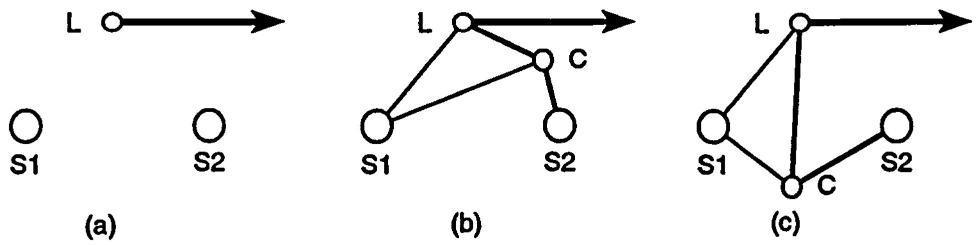


Figure 29. A design problem and two stress states. Each stress state can be an optimal solution.

for the skeletal design domain. Finally, Section 6.6 summarizes the results in this chapter and outlines future research.

6.1 Introduction

In a parallel environment with a sufficient number of independent processing units, we could solve all subproblems derived during the specialization and return the best solution. However, this becomes impractical as the number of stress states grows. To overcome this problem, in this chapter we define a new learning framework to acquire search heuristics to map a description of a truss design problem to a small set of stress states to be explored with traditional numerical optimizers. Each stress state suggested by the heuristic is one of the simplified versions of the original optimization task and is obtained by applying the techniques illustrated in the previous two chapters. Hence, it is fast to evaluate.

Ideally, the selected stress state should be a singleton so that only one stress state needs to be examined. Let us consider the design problem in Figure 29a, and let us assume that the topology in Figure 29b (and 29c) was selected. The task is to determine the optimal location for the connection point C so that the weight is minimized. As shown in the figures, the stress state depends on the location of the connection point. Figure 29b indicates a stress state with two tensile and two compressive members. Moving the connection point below the supports changes

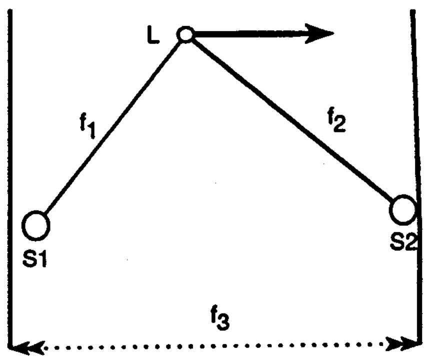


Figure 30. Features used in the skeletal design task. Forces f_1 and f_2 are obtained from the method of joints applied to a frame with no connection points. The position f_3 takes into account the location of the load w.r.t the supports.

the stress state to three tensile members and one compressive. This is shown in Figure 29c. These are not the only two feasible stress states. In fact, although not shown in the figure, there are 7 realizable stress states that can be obtained by placing the connection point in various positions in the 2-D space. In this chapter, we detail the method used to derive rules that, starting with the information from Figure 29a, allow the problem solver to select one or more stress states such as those in Figures 29b and 29c. These stress states reduce the space to be searched by the numerical optimizer to determine the location of the connection point that yields the structure of minimum weight.

Ideally, we would be able to analyze Figure 29a and choose only one stress state to be examined. However, we believe this is not a realistic goal. When we presented Figure 29a to our expert engineers, they were able to indicate that one of the two stress states shown in Figure 29b and 29c would contain the optimal solution. However, they were unable to formulate rules to choose between the two stress states. Similarly, in our experiments with learning algorithms, we have been unable to learn a good rule that can select only *one* stress state for a given problem. Hence, we have adopted the more modest goal of selecting a small set of stress states.

To apply inductive learning to this problem, we must first develop a set of features that describe the design task. In consultation with our experts, we selected features that engineers use to prospect optimal solutions. For each load and pair of supports, we have defined four features. As it is shown in Figure 30, the first two features (f_1 and f_2) take into account the direction of the load w.r.t. the supports. These features are computed by applying the method of joints to the structure composed of only two members that connect the load to each of the two support points. The result of the method of joints is the internal force in each member (tensile or compressive.) The stress state of this 2-member structure is an array composed of two elements. These constitute the first two features. For the sake of comprehensibility, the values of these features are `C` (compressive) and `T` (tensile.) It must be noticed that the stress state of the simplified structure does not depend on the solution, of course. Instead, it is a function of the givens of the problem. Thus, it can be computed a *priori*. Moreover, the stress state of a 2-member structure is very fast to compute since it only requires the solution of a 2×2 system of linear equations. The third feature, indicated by f_3 in Figure 30, gives information on the position of the load w.r.t. the supports. Its values are left, middle, and right; these are encoded as `L`, `M`, and `R`, respectively. The fourth feature (not illustrated in the figure), measures the perpendicular distance from the load point to the line passing through the two supports.

Given these features, and a collection of example design problems and solutions, the goal of learning is to find a rule that balances the tradeoff between the optimality of the final structure and the efficiency of the optimization process. This tradeoff can be expressed as a utility function, and the goal of learning is to find a stress-state selection rule of maximal utility. This kind of learning task has not been studied previously in machine learning, so we formalize it in the next section.

6.2 Formal Definition of SETMAXUTIL

The problem-solving task is to perform gradient descent search on one or more stress states to find a good solution. This can be formalized as a problem space in which X is the set of all problem instances and $A = \{a_1, a_2, \ldots, a_k\}$ is the set of actions available to the problem solver. Each individual action a_i, when applied to a problem instance $x \in X$, either produces a complete solution or fails to produce a solution. Hence, the solution depth is 1; actions are *not* chained together to construct a solution. A problem solver can be described by a function $h : X \mapsto 2^A$ that examines the problem instance x and selects a set S of actions to apply.

Let us first consider how we can evaluate a problem solver h. Then, we will formalize the problem of finding a good h by learning from examples.

Informally, a good problem solver is one that usually finds a near-optimal solution without searching too many stress states. To formalize this, consider the two important outcomes of an action: the cost of performing the action, $c(a_i, x)$, and the quality of the resulting solution, $q(a_i, x)$. (We will assume small values of $q(a_i, x)$ are desired and, if a_i fails to produce *any* solution when applied to x, $q(a_i, x)$ is infinite.) In our domain—and many others—we are willing to trade some quality to reduce the cost of finding the solution. To formalize this tradeoff, we define a utility function. Let S be a set of actions chosen for instance x. Then the cost of executing S is simply the sum of the cost of each action: $\sum_{a \in S} c(a, x)$. The quality of the solution produced by a set of actions is expressed by a loss function $l(S, x)$. Let

$$a'(S, x) = \operatorname*{argmin}_{a \in S} q(a, x) \text{ and}$$

$$a^*(x) = \operatorname*{argmin}_{a \in A} q(a, x).$$

$a'(S, x)$ is the best action among the selected actions S, but $a^*(x)$ is the best of all possible actions A. The loss taken by selecting S rather than performing all possible actions A is defined as

$$l(S,x) = \begin{cases} 0 & \text{If } \frac{q(a'(S,x),x)}{q(a^*(x),x)} < 1.1 \\ \frac{q(a'(S,x),x)}{q(a^*(x),x)} - 1.1 & \text{Otherwise.} \end{cases}$$

This says that there is no loss for solutions within 10% of the optimum, but beyond that point, the loss increases linearly.

Given these definitions, the utility $U(S,x)$ of a set of actions is defined as

$$U(S,x) = -\left[\sum_{a\in S} c(a,x) + k \cdot l(S,x)\right]. \tag{6.7}$$

where k is a parameter expressing how much additional cost we are willing to pay to reduce the loss by one unit (or equivalently, improve the quality $q(a'(S,x),x)$ by an amount $q(a^*(x),x)$).

With the above definitions, we can evaluate the performance of a problem solver h when it selects the set S of actions for a particular problem x. More generally, let $D(x)$ be the probability that the problem solver will need to solve problem instance x. Then we will measure the performance of h by the expected utility over $D(X)$:

$$\sum_{x\in X} U(h(x),x) \cdot D(x).$$

This completes our discussion of how to evaluate a heuristic function h.

Now let us consider how we will learn h from examples. Suppose we draw a sample $W \subset X$ according to distribution D, and, for each instance $x \in W$, we perform all possible actions in A and measure the cost of each action and the quality of the result. Then, for each $x \in W$, we would obtain a "training example" of the following form:

$$\begin{aligned} \langle x, \quad & \langle a_1, c(a_1,x), q(a_1,x)\rangle, \\ & \langle a_2, c(a_2,x), q(a_2,x)\rangle, \\ & \ldots \\ & \langle a_k, c(a_k,x), q(a_k,x)\rangle\rangle. \end{aligned}$$

From this training example, we can determine $a^*(x)$, and hence, for any subset S of the actions, we can compute $a'(S,x)$, $l(S,x)$, and $U(S,x)$. If we have an hypothesis, $\hat{h}$, we could estimate its expected utility by taking the average utility over the sample W:

$$EU(\hat{h}, W) = \frac{1}{|W|} \sum_{x \in W} U(\hat{h}(x), x).$$

As an example, let us consider the training data in Table 15. This is a small subset of the data used in the design domain. The first column in the table is simply a unique identifier of the examples. The second column lists (a subset of) the features. The third and last column indicates the actions taken by the numerical optimizer. For instance, the training example *Ex1* indicates that the numerical optimizer explored stress states 57 and 66. Using stress state 57, it took the optimizer 5.61 seconds to complete, and the resulting solution was about 400% the globally optimal one. The global optimum was instead obtained running the numerical optimizer on stress state 66 for which it took 9.00 seconds to complete the optimization. In addition to showing a sample of the complex format of the training examples, the table also shows an effect of the many-to-one representational mapping r. Training examples *Ex1* and *Ex2* have the same values for the features but different actions. This is due to a loss in information when the the features are computed from the original training examples. Shapiro (Shapiro 1987) considers these occurrences noise and calls them *clashes.* For exisiting ID3-like algorithms, considering clashes as errors in the data is necessary because of the implicit assumption that the features provided to the learning algorithm will allow it to associate a conjunction of feature values to an individual action. To overcome clashes, Shapiro proposes to advise the user that the data is incorrect and that the training examples need to be modified. This solution is unacceptable in the design domain because of the assumption of a weaker representation of the problem. The fact that the representation r is many-to-one, implies that clashes can be an integral part of the data. Thus, our learning framework allows for a weaker representation of the knowledge and permits clashes.

The goal of a learning program, therefore, is to search the space H of hypotheses

Table 15. A sample of training examples to learn optimal search control.

Example	*Features*	*Actions*
Id	f_1 f_2 f_3	
Ex1	N T L	(57 5.61 4.0029) (66 9.0 1.0)
Ex2	N T L	(57 15.61 5.0) (30 5.24 1.0) (68 9.13 1.3518)
Ex3	N C L	(1 2.90 1.0) (60 8.67 2.3)
Ex4	C C L	(2 7.59 1.21) (60 6.9 3.1)
Ex5	N C R	(60 4.59 1.0021) (42 10.6 1.0156) (69 8.07 1.0)
Ex6	N C R	(60 2.59 1.0) (43 11.7 1.0) (69 28.70 1.5)
Ex7	N T L	(57 14.73 52.329) (33 5.6 6.5954) (66 9.04 1.3518)

for an hypothesis $\hat{h}$ that maximizes the average utility over the training set W.

There is one further elaboration that we must make to this definition. In our case, each $x \in X$ is a description of a truss graph along with the location, magnitude and direction of each load, and the location of each stable attachment point. Because this description can vary in size and structure from one instance to another, we have found it convenient to construct a fixed-length vector of features to *represent* each problem instance. The learning algorithm must work with these feature vectors rather than with the original structural descriptions. Formally, we let $Y = \prod_{i=1}^{n} Y_i$ be a space of *representations* of problem instances in terms of n individual features Y_i $(i = 1, \ldots, n)$. We will assume there is a function $r : X \mapsto Y$ that maps from problem instances to their representations, and we will denote the feature vector representing x as $r(x) = (r_1(x), \ldots, r_n(x))$. In our framework, we will assume that r is many-to-one, because it is often difficult to come up with a completely adequate set of features. Hence, the same feature vector $\mathbf{y} \in Y$ may represent more than one problem instance in X.

Consequently, training examples will have $r(x)$ in place of x, and each hypothesis h can be written as $g \circ r$ (i.e., $h(x) = g(r(x))$), where $g : Y \mapsto 2^A$. The goal of finding a good $\hat{h}$ can be restated as the problem of finding a good $\hat{g}$ that maximizes utility on the training set W.

We shall call this learning problem SETMAXUTIL. Two differences set apart SETMAXUTIL from the usual inductive learning frameworks. First, the training examples have complex structure. This is a consequence of the goal of finding *optimal* solutions in the original problem space. This complex structure means that the usual notion of an hypothesis being "consistent" with a sample must be revised. We will say that an hypothesis h is "consistent" with a sample W if, on each example $x \in W$, the hypothesis selects at least one action a_i such that $q(a_i, x)$ is finite (i.e., a_i can be legally applied to x and returns a solution). In Section 6.3, we prove that the problem of finding a consistent and optimal hypothesis in this sense is $\mathcal{NP}$-complete (by reduction from HITTING-SET (Garey and Johnson 1979)).

The second difference is that each hypothesis recommends a *set* of actions. Strictly speaking, this should not be necessary, because it should be possible to learn a function $f : X \mapsto A$ such that $f(x) = a^*(x)$ (the best action to apply to instance x). This is a simple classification learning problem. However, because of the many-to-one representational shift $y = r(x)$, there may not exist a function $g : Y \mapsto A$ such that $g(r(x)) = a^*(x)$. One might argue that, in such cases, a better representation r should be found. However, an important consideration is the cost of run-time calculation of the heuristic function h (the so-called "utility problem," (Minton 1988)). We have therefore chosen to employ a few easy-to-extract features and learn a weaker mapping.

6.3 Complexity Analysis and Approximation

Learning the set of actions that (a) is consistent with the training examples, and (b) has maximum utility is $\mathcal{NP}$-complete. To prove this assertion, we shall first introduce the HITTING-SET, a known (Garey and Johnson 1979) $\mathcal{NP}$-complete problem, and then show the equivalence between HITTING-SET and SETMAXUTIL.

$\mathcal{NP}$-completeness leaves no hope (unless $\mathcal{P} = \mathcal{NP}$) for a polynomial time algorithm for the learning task. To overcome this problem, in Section 6.3.3 we introduce an approximation algorithm due to Chvatal (Chvatal 1979) and that, however, we derived independently.

6.3.1 HITTING-SET

Proofs of $\mathcal{NP}$-completeness are typically carried out by *reducing* a known $\mathcal{NP}$-complete problem to the problem at hand. In principle, one should be able to perform the reduction step from any known problem; however, in practice, the choice of the $\mathcal{NP}$-complete problem is crucial in simplifying the proof. This is the reason behind the choice of HITTING-SET (Garey and Johnson 1979). This problem can be posed as a *yes-no* task as follows:

Given: A finite set `S`, a collection `C` of subsets of `S`, and a positive integer `K`

Is there: A subset `S'` $\subseteq$ `S` of size $\|C'\| \leq$ `K` such that `S'` contains at least one element from each subset in `C`?

The proof of equivalence between this formulation and the classical (Garey and Johnson 1979) HITTING-SET is trivial. The $\mathcal{NP}$-completeness of HITTING-SET can be proved by reduction from VERTEX-COVER, another well-known $\mathcal{NP}$-complete problem. To illustrate the HITTING-SET problem, let us suppose that `S = {1, 2, 3, 4, 5}` and let `C = {X, Y, W, Z}` be a collection of sets. The following matrix M indicates the characteristic function of the sets (columns) w.r.t. elements (rows):

	X	Y	W	Z
1	1	0	0	1
2	1	1	0	0
3	0	0	1	1
4	1	0	1	0
5	0	0	0	1

A value of 1 in $M[i,j]$ indicates that the i-th element belongs to the j-th set. The set `S'` = `{2, 3}` is a solution to the HITTING-SET problem described by the matrix above. In fact, `S'` is smallest subset of `S` such that the intersection of `S'` with any element of the original collection `C` is non-empty. The reader can readly convince her/himself by taking the `OR` of the second and third row and noticing that the result is a vector of 1's. This indicates that each set in the collection `C` is "hit", that is, each set in `S'` contains at least one element of each set in the collection `C`. The minimality of `S'` can be proved by noticing that no row contains all 1's, therefore at least two sets are necessary in the solution.

Finally, it is important to notice that the chosen matrix representation highlights the similarities between HITTING-SET and MINIMUM-SET-COVER, a more common $\mathcal{NP}$-complete problem. In fact, for the example in the matrix above, the MINIMUM-SET-COVER is `X` ∪ `Z`. This assertion can be verified by `OR`-ing the corresponding columns rather the rows as in the HITTING-SET problem.

We shall see now show that SETMAXUTIL is equivalent to HITTING-SET.

6.3.2 $\mathcal{NP}$-completeness

Let us first illustrate a matrix formulation of the learning problem that is equivalent to the one shown for HITTING-SET. For each of the n training examples presented to the learning algorithm, let us build a set of pairs $\texttt{A}_i = \{(\texttt{a}, \texttt{u})\}$ where `a` is one of the actions associated with the example and `u` its utility. With the exception of the utility, the collection of sets $\texttt{A}_i$ is equivalent to the collection of sets `C` in the HITTING-SET problem. As an example, Table 15 indicates the matrix representation of the training examples in Table 16. Each row in the matrix indicates an action. Columns represent training examples. The values in the matrix are ∞ if the action is not applicable to the example. Otherwise, the value in the matrix is the real number $(u_{i,j})$ which represents the utility of an action. This matrix can be trasformed to resemble the characteristic function shown for the HITTING-SET problem in the previous section by setting all real values $(u_{i,j})$

to 1 and by changing the symbol ∞ to 0.

To prove $\mathcal{NP}$-completeness, we observe that the requirement that the output of SETMAXUTIL must be consistent with the training examples is equivalent to the requirement that the solution contains at least one action applicable to an example. This, in turn, can be expressed as the requirement that the solution contains at least one element from each set $\mathtt{A}_i$. Moreover, the optimality requirement of SETMAXUTIL can be formulated as a *yes-no* problem. In fact, to determine that a solution is optimal (w.r.t. the utility) is equivalent to determine that there is no solution of a lesser utility. With these observations in mind and assuming unitary cost for each applicable action, SETMAXUTIL can be re-stated as follows:

Given: A finite set of actions $\mathtt{A}$, a collection $\mathtt{C}$ of subsets of $\mathtt{A}_i \subseteq \mathtt{A}$, and a positive integer $\mathtt{K}$

Is there: A subset $\mathtt{A}' \subseteq \mathtt{A}$ of size $\|A'\| \leq \mathtt{K}$ such that $\mathtt{A}'$ contains at least one element from each subset in $\mathtt{C}$.

A glance at the formulations of SETMAXUTIL and HITTING-SET indicates the obvious equivalence between the two problems. This completes the $\mathcal{NP}$-completeness proof.

6.3.3 Approximation Algorithm

$\mathcal{NP}$-completeness results are usually bad news for the computer scientists that wish to find exact solutions. On the other hand, they are good news for AI researchers and for people interested in approximation. In fact, for many $\mathcal{NP}$-complete problems there are approximation algorithms which give results that can be proven to be within a small factor from the optimal solution. To compute approximate solutions we derived an algorithm (MinCover) for SETMAXUTIL that, however, had already been derived by Chvatal (Chvatal 1979) for MINIMUM-SET-COVER. In addition to giving the algorithms, Chvatal also proved that the algorithm produces results that are within a logarithmic factor from the (unknown) optimal

Table 16. Training examples for UTILITYID3 in matrix format.

	$Ex1$	$Ex2$	$Ex3$	$Ex4$	$Ex5$	$Ex6$	$Ex7$
1	∞	∞	$u_{1,3}$	∞	∞	∞	∞
2	∞	∞	∞	$u_{2,4}$	∞	∞	∞
30	∞	$u_{3,2}$	∞	∞	∞	∞	∞
33	∞	∞	∞	∞	∞	∞	$u_{4,7}$
42	∞	∞	∞	∞	$u_{5,5}$	∞	∞
43	∞	∞	∞	∞	∞	$u_{6,5}$	∞
57	$u_{7,1}$	$u_{7,2}$	∞	∞	∞	∞	$u_{7,7}$
60	∞	∞	$u_{8,3}$	$u_{8,4}$	$u_{8,5}$	$u_{8,6}$	∞
66	$u_{9,1}$	∞	∞	∞	∞	∞	$u_{9,7}$
68	∞	$u_{10,2}$	∞	∞	∞	∞	∞
69	∞	∞	∞	∞	$u_{11,5}$	$u_{11,6}$	∞

minimum. Given the similarities among MINIMUM-SET-COVER, HITTING-SET, and SETMAXUTIL, the same theoretical results apply *sic et simpliciter* to all three problems. The greedy algorithm given by Chvatal (and us) is shown in Table 17 and generalizes to arbitrary costs a previous algorithm of Johnson (Johnson 1974) which only handled unitary costs. The optimal set of actions is built incrementally by choosing the action a that maximizes the metric $(P_{a_j}/-U_{a_j})$. This balances the number of occurrences of each action (P_{a_j}) with its expected utility(U_{a_j}). The expected utility $EU(h_{a_j}, W_{a_j})$ of an action a_j is only computed with respect to those training examples W_{a_j} where the action produced a solution (i.e., where $q(a_j, x_i)$ was not infinite). If an action is good and appears frequently in W, then the denominator will be small and the numerator will be large. Once the action is chosen, all examples that contain it are removed from consideration. The actions are accumulated until no more examples are left in W.

With the assurance that our approximation algorithms finds satisfactory solutions for SETMAXUTIL, we can introduce the learning algorithms to acquire optimal search control knowledge.

Table 17. Greedy algorithm to compute the set of actions of maximum utility.

```
Function MinCover($W$)
  $W$ = set of training examples $\langle r(x_i), v_{a_1}(x_i), \ldots, v_{a_k}(x_i)\rangle$
      where $v_{a_j}(x_i) = \langle c(a_j, x_i), q(a_j, x_i)\rangle$
  1.  Let $OptimalSet$ := $\emptyset$
  2.  While $W \neq \emptyset$ do
  3.    For each action $a_j$
  4.      Let $W_{a_j} = \{\langle r(x_i), \ldots, \langle c(a_j, x_i), q(a_j, x_i)\rangle \ldots\rangle \in W | q(a_j, x_i) < \infty\}$
  5.      Let $P_{a_j}$ be $|W_{a_j}|$
  6.      Let $h_{a_j}(x) = a_j$ be the heuristic of choosing action $a_j$
  7.      Let $U_{a_j} = EU(h_{a_j}, W_{a_j})$
  8.    endFor
  9.    Let $\mathrm{a} = \operatorname{argmax}_{a_j}(P_{a_j} / - U_{a_j})$
  10.   Set $OptimalSet := OptimalSet \cup \{\mathrm{a}\}$
  11.   Set $W := W - W_{\mathrm{a}}$
  12. endWhile
  13. Return $OptimalSet$
```

6.4 Three Learning Algorithms

In this section, we present three algorithms for learning control knowledge for the skeletal design domain.

6.4.1 ID3

The first algorithm is a simple version of ID3 (Quinlan 1986) that attempts to predict $a^*(x)$, the optimal action, given non-unique feature vector $r(x)$. It is

Table 18. The basic ID3 algorithm.

```
Function ID3($W$, $F$)
  $W =$ a set of training examples $\langle r(x), a^*(x)\rangle$.
  $F =$ a set of features $r_i$.
  1.If $\exists$a $\forall\langle r(x), a^*(x)\rangle \in W$, $a^*(x) =$ a Then
      Return leaf node recommending action a.
  2.If $\exists$y $\forall\langle y, a^*(y)\rangle \in Wy =$ y or $F = \emptyset$ Then
      Compute the action a that appears most frequently in $W$
      Return leaf node recommending action a.
  3.Let $r_c$ be the feature in $F$ of highest mutual information
  4.Split $W$ into sets $W_j = \{\langle r(x), a^*(x)\rangle \in W | r_c(x) = j\}$
      where $j = 1, \ldots, nval(r_c)$ are the values of feature $r_c$.
  5.Return DecisionTreeNode(ID3($W_1$, $F - \{r_c\}$), ...,
                            ID3($W_{nval(r_c)}$, $F - \{r_c\}$))
```

straightforward, of course, to convert the complex training examples described in the previous section into examples of the form $\langle r(x), a(x)\rangle$. We then apply the familiar top-down separate-and-conquer algorithm for constructing a decision tree (see Table 18).

The algorithm works by recursively splitting the training set W into subsets determined by the values of the selected features r_c. The features are selected using the mutual information heuristic (also known as information gain). The mutual information between two random variables A (with values $a_1, \ldots, a_n$) and B (with values $b_1, \ldots, b_m$) is

$$MI(A;B) = I(A) - \sum_{i=1}^{n} Pr(A = a_i) I(B|A = a_i),$$

where

$$I(A) = \sum_{i=1}^{n} -Pr(A = a_i) \log_2 Pr(A = a_i).$$

In the algorithm, the probabilities in these formulas are estimated from the frequencies observed in the set W.

The splitting process terminates when either (a) all examples in W agree on the desired action $a^*(x)$, (b) all examples in W have the same representation $\mathbf{y} = r(x)$, or (c) there are no more features left to split on ($F = \emptyset$).

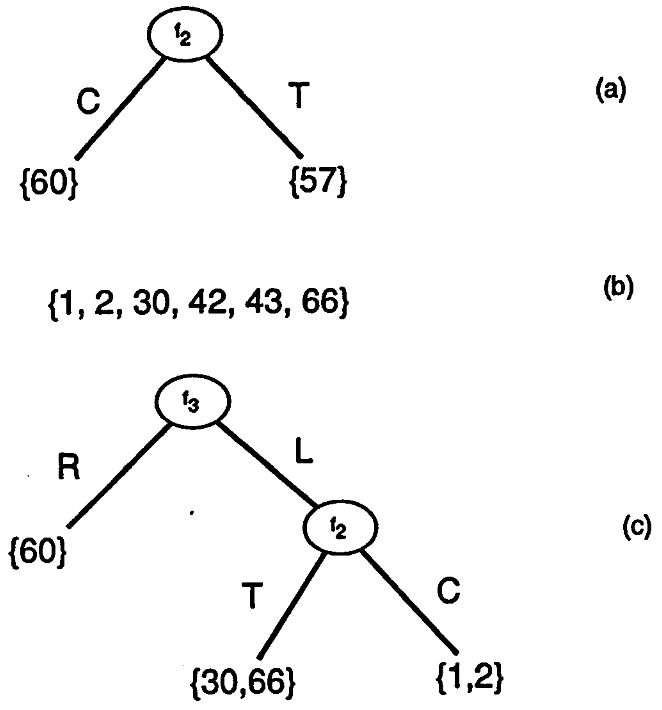


Figure 31. Output of the three learning algorithms on the training data in Table 16. (a) ID3, (b) MinCover, and (c) UTILITYID3.

To compute the action to be associated to a leaf node, ID3 uses a majority rule. At the bottom of the recursion, it outputs the action with the highest frequency in the training data—Step 2 in Table 18. As an example, Figure 31a indicates the decision tree output by `C4.5` (a widely used implementation of multi-class ID3) on the training examples in Table 15. ID3 completely neglects the cost and quality associated with each example. The decision tree reflects this approach. In fact, it suggests action 57 for f_2 =T. However, as we can see from the training data in the table, this action can yield the worst results. Nevertheless, it is the majority class for the feature selected. In the design domain this leads to poor performance of the algorithm when the correctness of the solution is taken into consideration. The reason for this behaviour of the algorithm is that the features do not allow ID3 to partition the examples so that at each leaf the class that yields the minimum is always chosen. Intuitively, this is due to the many-to-one nature of the representational mapping r. That is, training examples with the same features may have different optimal stress states. Moreover, different stress states can produce vastly different local optima. On the positive side, ID3 always suggests a single action. As we shall see in Section 6.5, this implies that ID3 performs better than other learning algorithms in terms of CPU time. This is because only one stress state is explored. However, the necessity of exploiting the time-quality tradeoff leads us to the two algorithms described in the next sections.

6.4.2 Minimum Cover

The second algorithm we have studied takes a set of training examples W and tries to find a set S of actions that has the maximum expected utility over W. In other words, it develops a function $h(x) = S$, which returns S regardless of the value of x. This problem is equivalent to MINIMUM-COVER and we have applied the MinCover algorithm in Table 17. When used as a learning algorithm, MinCover disregards any domain knowledge and produces a set of actions that is consistent with the training data. However, as the experimental results demonstrate, the algorithm produces a set of actions which contains a large number of elements. To see this, let us consider the training data in Table 15. Chvatal's algorithm applied to this data produces the set shown in Figure 31b. This means that at run time the problem solver will always try 6 stress states. The large number of stress states is due to the requirement that the set of actions be consistent with the training examples. It turns out that this solution is very expensive in terms of computational resources. To overcome this problem, we have devised an inductive learning algorithm that uses features to map design problems to sets of actions.

This is described in the next section.

6.4.3 UTILITYID3

The final algorithm (UTILITYID3) that we have explored combines aspects of both ID3 and MinCover. Like ID3, it uses features of the domain to construct a decision tree. However, the training examples have the complex structure that includes the cost and quality of each action. The differences between UTILITYID3 and ID3 are described below.

Feature Selection. The algorithm for selecting the splitting feature r_c (Step 3 of Table 18) is changed as follows:

```
3. Let $W_{i,j} = \{\langle r(x), \ldots \rangle \in W | r_i(x) = j\}$
   Let $S_{i,j} = MinCover(W_{i,j})$
   Let $G_{u,v}$ be the event that $g(a_u, x_v) < \infty$.
   Let $P(G_u|W_{i,j})$ = the probability of $G_{u,v}$ for $x_v$ in $W_{i,j}$.
   Compute $I(G|r_i(x) = j) = \sum_{a_u \in S_{i,j}} -P(G_u|W_{i,j}) \log_2 P(G_u|W_{i,j})$
   Compute $gain(r_i) = I(a^*) - \sum_{j=1}^{nval(r_i)} \frac{|W_{i,j}|}{|W|} I(G|r_i(x) = j)$
   Choose the feature $r_c = \operatorname{argmax}_{r_i} gain(r_i)$.
```

This is a modified calculation for the mutual information between features r_i and actions a_u. We restrict attention only to those actions returned by $MinCover(W_{i,j})$. Note, however, that we calculate the probability of each action by considering all training examples in which that action returns a solution (i.e., $g(a_u, x_v) < \infty$), even if the solution is not optimal. The intuition is that we want to find a feature that will grow the decision tree so that the new nodes have small sets of actions (as computed by $MinCover$).

Construction of Leaf Nodes. In addition to changing the feature-selection procedure, UTILITYID3 also changes how leaf nodes are created in the tree.

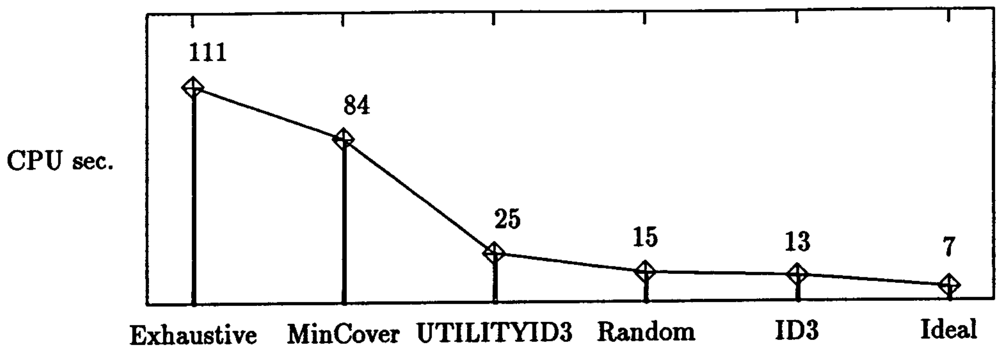


Figure 32. Average CPU time as a function of search control knowledge.

Whenever ID3 would terminate and create a leaf node, UTILITYID3 first invokes MinCover to compute an *OptimalSet* of actions. These are then placed in the leaf node instead of choosing a single best action.

Figure 31c illustrates the output of UTILITYID3 on the training data in Table 15. UTILITYID3 produces small sets of alternatives to be explored by the problem solver. In fact, for f_3 = L and f_2 = T, only stress states 30 and 66 are suggested. Therefore, UTILITYID3 is capable to prune stress states that will not lead to optimal solutions.

6.5 Experiments

To compare these algorithms, we generated training and test data sets as follows. We chose three classes of truss design problems: one having 2 support points, one load, and 4 members, and two each having 3 support points, 2 loads, and 6 members. For each problem class, we generated 700 problem instances by randomly varying the location, magnitude, and direction of the loads and the locations of the supports. We randomly divided these 700 instances into 500 training cases and 200 test cases (for a total of 1500 training and 600 test cases).

For each of the three algorithms, we trained and tested it three times, once on each of the three data sets, and measured the CPU time required to evaluate the

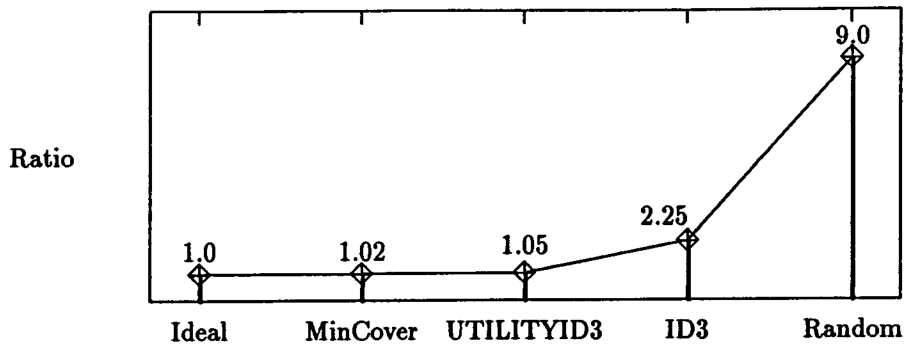


Figure 33. Average ratio $\frac{q(a'(S,x),x)}{q(a^*(x),x)}$ as a function of search control knowledge.

test data, the ratio $\frac{q(a'(S,x),x)}{q(a^*(x),x)}$ between the best solution found and the best possible solution, and the utility of the problem-solving process. A value of $k = 20$ was used in equation 6.7. It is important to note that in all cases, the algorithm is tested on data drawn from the same problem class as the training data. We have not tested the ability of these algorithms to generalize across problem classes.

In addition to running the three algorithms, we also computed the CPU time, solution-quality ratio, and utility of three other strategies: (a) exhaustive search, in which all stress states are evaluated on each test instance, (b) random search, in which a stress state is chosen at random, and (c) perfect search, in which only the stress state containing the optimal solution is evaluated on each test instance. Exhaustive and perfect search find the optimal solution, of course. For ID3 and random search, if an infeasible action was chosen, it was replaced with the worst feasible action.

Figures 32, 33, and 34 show the CPU time, ratio, and utility figures for these 6 strategies. The results reveal the tradeoff between efficiency (where ID3 and random are better than UTILITYID3 and MinCover) and quality of the solution as measured by the ratio (where MinCover and UTILITYID3 are better than ID3). Figure 34 shows that, according to our utility function, UTILITYID3 is clearly superior to all of the other strategies (except Ideal, of course).

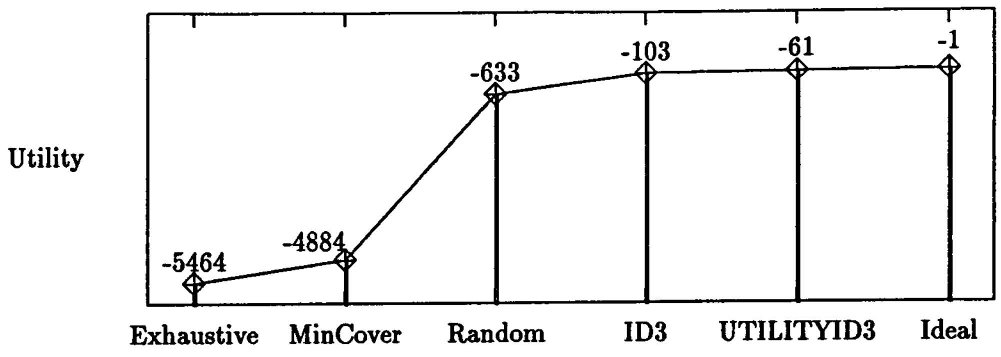


Figure 34. Average utility as a function of search control knowledge.

Compared to exhaustive search, UTILITYID3 gives a speedup of 4.4. From Figure 33 we can see that the average quality of the solutions found by UTILITYID3 is within 5% of optimal. By comparison, ID3's solutions are, on the average, more than double the ideal values (i.e., the trusses weigh 2.25 times more than optimal).

The speed of ID3 and Random results from the fact that they always evaluate exactly one stress state. By comparison, we can infer from Figure 32 that UTILITYID3 is evaluating two or three stress states in each problem. Because the number of possible stress states is exponential in the number of members in the truss, this is very encouraging. UTILITYID3 has managed to prune away all but a fixed number of stress states in each problem instance. Additional results of UTILITYID3 are reported in Appendix D.

6.6 Concluding remarks

Speedup learning in numerical optimization tasks requires new methods and poses new learning tasks. In this chapter we have defined SETMAXUTIL, a novel framework to learn optimal search control knowledge. The problem is to determine a decision tree that maps features of the problem into a small set of alternatives to be exploited by the problem solver. In the truss design problem, partitioning the search space for the non-linear optimization problem might produce an exponential number of regions. The learning task is then to decide which regions are likely to contain the optimum, and perform gradient-descent searches in those regions only.

To generate training data, we performed exhaustive search and determined which region contains the optimum. We developed an algorithm, UTILITYID3, that analyzes these training examples and constructs a function $f : X \mapsto 2^A$ that selects a set of regions (actions) to be searched. The performance of UTILITYID3 was measured in three problem classes and found to give near-optimal solutions while yielding a speedup of more than a factor of 4.

Future work in the truss-design domain must develop techniques that can generalize from examples of one problem class (e.g., 4-member trusses with two loads) to examples in another problem class (e.g., 6-member trusses with 3 loads). We believe the UTILITYID3 algorithm will be able to solve these problems without modification.

Another avenue for future work is to test other ways of deploying standard ID3 to solve this problem. One approach would be to try to learn a function $f : X \times A \mapsto \{good, bad\}$ that, given an instance and an action tries to predict whether the action will lead to a solution within, say, 10% of optimum. Another approach would be to employ a regression algorithm (such as regression trees, (Breiman, Friedman et al. 1984) to learn $q(a, x)$ and $c(a, x)$. These would serve as evaluation functions and would allow the problem solver to compute (a guess concerning) $a^*(x)$. However, the run-time cost of such an approach could be large.

Finally, we believe that UTILITYID3 can also be applied to learn control knowledge in problem spaces where the solution lies at depths greater than 1. To construct training examples, it would be necessary to estimate the expected quality and cost of solving the problem $x' = a_j(x)$ that remains after applying each operator a_j to x. This information is closely related to the values computed by reinforcement learning algorithms such as Q-learning (Barto et al. 1991). Hence, UTILITYID3 provides a way to convert the evaluation-function representation for search-control knowledge (which can be highly inefficient to apply at run time) into the operator-selection form of search-control knowledge (which can be evaluated very quickly if inexpensive features are employed).

Chapter 7

Conclusions and Future Work

Speedup learning in numerical optimization tasks requires new methods and poses new learning tasks. In this thesis we have demonstrated how machine learning (ML) techniques can be applied to optimization tasks in engineering design. This has been accomplished by using existing methods and devising new techniques in the inductive and symbolic machine learning paradigms. These techniques have been used to

- Speed up existing numerical optimization methods, and
- Bridge gaps in the knowledge transfer between engineers and computer systems.

Our approach opens new research directions into the so far unexplored area of applications of machine learning to numerical optimization. It is our hope that, in the medium-to-long-term, our techniques will allow the use of specialized numerical optimizers in real-time applications like intelligent CAD systems.

In the remainder of this chapter we highlight the techniques used, discuss some benefits from an engineering standpoint, and outline future work.

7.1 Solution Highlights

The strategy used in this thesis to speed up numerical optimization tasks relies on (a) successive specializations and simplifications of the objective function and

(b) identification of a few specialized sub-problems that are fast to optimize and that can be solved independently. Our techniques have been applied to the design of lightweight 2-dimensional structures made of rods and columns (members). In this task the problem is twofold. First one must identify the topology. That is, the number of extra connection points, members, and their connectivity. Second, the geometry must be fixed; that is, the length of each member must be derived. This is equivalent to locating the connection points in the 2-dimensional plane. Topology and geometry must be chosen so that the overall weight is a minimum.

To partition the original optimization task, we enumerate a few topologies and use a novel abstraction (*stress state*) that considers the type of member (rod or column) in a structural configuration. These are invariant aspects of a particular problem and are incorporated into the objective function. This specialization produces an optimization problem expressed in terms of the coordinates of the connection points. From a numerical optimization standpoint, the benefits of the specializations are great. First, the cost of each evaluation of an objective function is greatly reduced. Second, the specializations make it possible to obtain differentiable closed forms for the objective function. This allows us to apply gradient-directed optimization methods. Third, the specializations create opportunities for parallel execution of the optimization calculations. Fourth and last, each of the specialized functions can be further simplified by reducing the number of independent variables.

The elimination of variables is accomplished by discovering *regularities*; that is, relating unknown variables to given ones. This transformation process uses a mix of Explanation-Based Learning (EBL), inductive, and regression techniques. EBL is given a geometrical domain theory and derives proofs of the unknowns in terms of the givens. However, most often these proofs cannot be completed. Under these circumstances, the inductive and regression mechanisms bridge the gap between unknown and given quantities. In the case of inductive techniques, the relations among variables are derived using given knowledge of geometry and relations among geometric entities. When regression is used, the relations are derived in functional form using the off-the-shelf SAS statistical package. The regularities are then used to complete the proofs derived by the EBL engine. The first of such proofs with the fewest number of unknowns is then analyzed to derive algebraic relations among givens and unknowns. These relations are incorporated into the objective function which is simplified using symbolic methods. The overall result is the elimination of the transformed (unknown) variables from the optimization task. This means that at run time the numerical optimizer will be given an optimization task with fewer independent variables. This simplification, in turn, yields optimization tasks that are faster and simpler.

While the specialization of the objective function is extremely useful to speedup a single numerical optimization problem, it does not suffice to solve the entire design task. This is because the double specialization in the spaces of topologies and stress states introduces a combinatorial explosion. Nevertheless, not all of the 2^m stress states make physical sense or produce an optimal solution. To overcome the combinatorial explosion, we have defined a new learning framework which is more appropriate to optimization tasks. This learning framework requires (a) that the output of the learning algorithm be a set of alternatives and (b) that the algorithm attempts to maximize the utility of these sets. The utility is a function of the time it takes to obtain a solution and of its quality (ratio w.r.t. the global minimum.) Within this framework, we have developed and tested several learning algorithms which generate search control knowledge for the problem solver. We found experimentally that one of the algorithms we devised, UTILITYID3, outperforms all others in the skeletal design domain. This is a contribution to basic research in machine learning.

7.2 Engineering Perspective

The approach presented in this thesis simplifies the task of the engineer when s/he is faced with an optimization problem. In the 2-D optimal design task, our methods speed up numerical optimization and help bridge the knowledge gap between the engineer and the computer system. The speedup allows the user a quicker interaction with the computer system. This increases the human throughput without affecting the quality of the solutions. The knowledge gap is reduced because methods like UTILITYID3 only require a weak featural representation of the problem. Therefore, UTILITYID3 trades computer time with human time. In fact, UTILITYID3 presents the problem solver with an array of alternatives while requiring only a very *weak* featural representation. Thus, UTILITYID3 requires the computer to solve a few more problems while saving time and effort to the engineer who is required to derive the features only. On the other hand, traditional learning algorithms, like ID3, propose only one possible solution but require that the features are *strong* enough to allow the learning algorithm to select the correct action. As we have seen in this thesis, strong features are not always available. With the ever increasing computing power available in modern computers, our approach helps engineers focus on high-level activities and let computers (a) prune out infeasible alternatives and (b) explore a few meaningful solutions.

7.3 Future Work

This thesis is one of the first attempts at using machine learning techniques in numerical optimization. Thus much work remains to be done.

In the task of designing lightweight 2-D structures, we identify four open topics. First, the test cases should include "real" life size optimization tasks. Second, techniques that can generalize from examples of one problem class (e.g., 4-member trusses with two loads) to examples in another problem class (e.g., 6-member trusses with 3 loads) must be developed. Third, we hope to prove our unimodality conjecture. This would provide a proof of correctness for our divide-and-conquer schema. Fourth and last, the system should be able to autonomously derive abstractions, such as *stress state*, that partition the search space.

The symbolic techniques illustrated in Chapter 4 pave the way for machine learning techniques to speed up the simplification process. In fact, one avenue of research is to analyze the traces of the partial evaluation and to *learn* algebraic simplifications. This is necessary to drastically reduce compilation time. These methods would find large applications in modern symbolic manipulation packages like Mathematica.

Scale-up is one of the foremost issues to be overcome for the procedures shown in Chapter 5 to eliminate independent variables. We foresee that, as the number of variables grows, the search control methods we have adopted will not suffice. Therefore, it will be necessary to add (or learn) search control heuristics. Moreover, the regression techniques we have adopted in our approach should be employed to derive the starting point for the numerical optimization process. This would completely relieve the engineer of the burden of choosing a correct starting point.

The work in Chapter 6 should be extended to test other ways of deploying standard ID3 to learn optimal search control knowledge. One approach would be to try to learn a function $f : X \times A \mapsto \{good, bad\}$ that, given an instance and an action, predicts whether the action will lead to a solution within, say, 10% of optimum. Another approach would be to employ a regression algorithm (such as regression trees, (Breiman et al. 1984)) to learn $q(a, x)$ and $c(a, x)$. These would serve as evaluation functions and would allow the problem solver to compute (a guess concerning) $a^*(x)$. However, the run-time cost of such an approach could be large. In addition, we believe that UTILITYID3 can also be applied to learn control knowledge in problem spaces where the solution lies at depths greater than 1. To construct training examples, it would be necessary to estimate the expected quality and cost of solving the problem $x' = a_j(x)$ that remains after applying each operator a_j to x. This information is closely related to the values computed by reinforcement learning algorithms such as Q-learning (Barto et al. 1991). Hence, UTILITYID3 provides a way to convert the evaluation-function representation for search-control knowledge (which can be highly inefficient to apply at run time) into the operator-selection form of search-control knowledge (which can be evaluated very quickly if inexpensive features are employed).

In addition to the extensions to the work we have presented in this thesis, we are planning to tackle the improvement of performance of numerical optimizers from a different perspective. In this thesis we have treated the numerical optimizers as *black boxes* and tried to predict their outcomes from past input-output pairs. Another approach is to *open-up* the numerical optimizers. We believe that domain knowledge and traces of the solution paths taken by the optimizers can be used to make optimization algorithms adaptive. Moreover, another important drawback of numerical optimizers is that they cannot explain their solutions in a language that is comprehensible to, say, an engineer. Again, traces and domain knowledge can be used to produce such explanations.

Finally, a longer term goal is to incorporate efficient optimization procedures into intelligent CAD systems.

Bibliography

Agogino, A. and Almagren, A.: 1987, Symbolic computation in Computer Aided Design, *in* J. Gero (ed.), *Expert-Systems in CAD.*

Aho, A. V., Sethi, R. and Ullman, J. D.: 1986, *Compilers Principles, Techniques, and Tools*, Addison-Wesley.

Bakiri, G.: 1991, *Converting English Text to Speech: A Machine Learning Approach*, PhD thesis, Oregon State University, Computer Science Dept., Corvallis, OR.

Barto, A. G. et al.: 1991, Real-time learning and control using asynchronous dynamic programming, *Tech.Rep. 91-57*, Department of Computer Science University of Massachusetts, Amherst MA.

Beckman, L. et al.: 1976, A partial evaluator, and its use as a programming tool, *Artificial Intelligence Journal* **7**, 319–357.

Berliner, A. A.: 1989, A compilation strategy for numerical programs based on partial evaluation, *Tech.Rep. AI-TR 1144*, MIT AI Lab.

Braudaway, W.: 1988, Constraint incorporation using constrained reformulation, *Tech.Rep. LCSR-TR-100 Computer Science Dept.*, Rutgers University.

Breiman, L., Friedman, J. et al.: 1984, *Classification and Regression Trees*, Wadsworth.

Burns, S.: 1989, Graphical representation of design optimization processes, *Computer Aided Design* **21**(1), 21–24.

Burstall, R. and Darlington, J.: 1977, A transformation system for developing recursive programs, *Journal of the ACM* 24(1), 44–67.

Cerbone, G. and Dietterich, T. G.: 1991, Knowledge compilation to speed up numerical optimization, *Proceedings of the Machine Learning Workshop*, pp. 600–604.

Cerbone, G. and Dietterich, T. G.: 1992a, Inductive learning in engineering: A case study, *Proceedings of the Adaptive and Learning Systems Conference of the IEEE Society for Optical Engineers*, to appear.

Cerbone, G. and Dietterich, T. G.: 1992b, Learning optimal search control knowledge for engineering optimization, *Tech.Rep. CS-90-30-1*, Oregon State University, Dept. of Computer Science.

Cerbone, G.: 1992, Machine learning techniques in optimal design, *Proceedings of the Second International Conference on Artificial Intelligence and Design, Carnegie-Mellon Univerisity, PA*, to appear.

Chien, S. et al.: 1991, Machine learning group in engineering automation, *Proceedings of the Eighth International Machine Learning Workshop*, pp. 577–580.

Chvatal, V.: 1979, A greedy heuristic for the set covering problem, *Math.Oper.Res* 4(3), 233–235.

Cook, G.: 1991, ALPAL, a program to generate physics simulation codes from natural description, *Journal of Computational Physics* 2, 387–410.

Dietterich, T. G. (ed.): 1986, *Proceedings of the Workshop on Knowledge Compilation.*

Dietterich, T. G.: 1987, Research methodologies in AI, Class Notes.

Draffin, J. and Collins, W.: 1950, *Statics and Strength of Materials*, Ronald Press, New York.

Duffey, M. and Dixon, J.: 1988, Automating extrusion design: a case study in geometric and topological reasoning in mechanical design, *Computer Aided Design* **20**(10), 589–596.

Ellman, T.: 1988, Approximate theory formation: An Explanation-Based approach, *Proceedings AAAI*, pp.570–574.

Ellman, T.: 1989, Explanation-Based Learning: A survey of programs and perspectives, *Computing Surveys* **21**(2), 163–222.

Ershov, A.: 1982, Mixed computation: Potential applications and problems for study, *Theoretical Computer Science* **18**, 41–67.

Feigenbaum, E.: 1977, The art of artificial intelligence 1: themes and case studies of knowledge engineering, *Tech.Rep. STAN-CS-77-621*, Stanford University, Dept. of Computer Science.

Friedland, L.: 1971, *Geometric Structural Behavior*, PhD thesis, Columbia University at New York, N.Y.

Furuta, H., Tu, K.-S. and Yao, J.: 1985, Structural engineering applications of expert systems, *Computer Aided Design* **17**(9), 410– 420.

Futamura, Y.: 1971, Partial evaluation of a computation process – an approach to a compiler-compiler, *Systems, Computers, and Controls* **2**(5), 45–50.

Garey, M. J. and Johnson, D. S.: 1979, *Computers and Intractability, A Guide to $\mathcal{NP}$-completeness*, Freeman.

Genesereth, M. and Nilsson, N.: 1987, *Logical Foundations of Artificial Intelligence*, Morgan Kaufman, Los Altos, CA.

Hagen, P. and Tomiyama, T.: 1987, *Intelligent CAD Systems*, Springer Verlag, Berlin.

Hemp, W.: 1973, *Optimum Structures*, Clarendon Press, Oxford.

IEEE Expert: 1991, intelligent systems and their applications, **6** (2).

Johnson, D. S.: 1974, Approximation algorithms for combinatorial problems, *Journal of Computer Systems* **9**, 256–278.

Keller, R. M.: 1991, Building the scientific modeling assistant: An interactive environment for specialized software design, *Tech.Rep. FIA-91-13*, NASA Ames Research Center.

Kibler, D. and Langley, P.: 1988, Machine learning as an experimental science, *Proceedings of the Third European Working Session on Learning.*

Langley, P., Simon, H. et al.: 1987, *Scientific Discoveries: Computational Explorations of the Creative*, MIT Press.

Lenat, D. B.: 1978, The ubiquity of discovery, *Artificial Intelligence Journal* 9, 257–285.

Maeder, R.: 1989, *Programming in Mathematica*, Redwood City, Calif. : Addison-Wesley, Advanced Book Program.

Maxwell, C.: 1869, Reciprocal figures, frames, *Scientific Papers* ii, 175–177.

McClelland, J. and Rumelhart, D.: 1986, *Parallel Distributed Processing*, MIT Press.

Michell, A.: 1904, The limits of economy of material in frame structures, *Philosophical Magazine* 8(47), 589–597.

Minton, S.: 1988, Empirical results concerning the utility of Explanation-Based Learning, *Proceedings AAAI*, pp. 564–569.

Minton, S.: 1990, Quantitative results concerning the utility of Explanation-Based Learning, *Artificial Intelligence Journal* **42**, 363–392.

Mitchell, T., Keller, R. and Kedar-Cabelli, S.: 1986, Explanation-Based Generalization: A unifying view, *Machine Learning Journal* **1**, 47–80.

Mittal, S. and Araya, A.: 1986, A knowledge-based framework for design, *Proceedings AAAI*, pp.856–865.

Mostow, J.: 1989, Research directions for AI in design, *Proceedings of the 1989 Workshop on Research Directions for AI in Design*, pp. unavailable.

Nevill, G.E. Jr., Garcelon, J.H. et al.: 1989, Automating preliminary mechanical configuration design: The MOSAIC perspective, *NSF Engineering Design Research Conference.*

Newell, A.: 1966, Heuristic programming: Ill-structured problems, *Operation Research* pp. 362–413.

Palmer, A. and Sheppard, D.: 1970, Optimizing the shape of pin-jointed structures, *Proc. of the Institution of Civil Engineers*, pp. 363–376.

Palmer, A.: 1968, Optimal structure design by dynamic programming, *American Society of Civil Engineers, Journal of Structural Engineering* **94**(ST8), 1887–1906.

Papalambros, P. Y. and Chirehdast, M.: 1990, An integrated environment for structural configuration design, *Journal of Engineering Design* **1**(1), 73–96.

Papalambros, P. Y. and Wilde, D. J.: 1988, *Principles of optimal design: modeling and computation.*, Cambridge University Press, Cambridge.

Paul, G. and Nevill, G.E. Jr.: 1987, Knowledge based spatial reasoning for designing structural configurations, *Computers in Engineering*, pp. 155–160.

Pike, R. W.: 1986, *Optimization for Engineering Systems*, Van Nostrand.

Polya, G.: 1973, *How To Solve It*, Princeton University Press, Princeton.

Press, W. H. et al.: 1988, *Numerical Recipes in C: the art of scientific computing*, Cambridge University Press, Cambridge.

Quinlan, R. J.: 1986, Induction of decision trees, *Machine Learning* **1**(1), 81–106.

Quinlan, R. J.: 1987, Simplifying decision trees, *International Journal of Man-Machine Studies* **27**, 221–234.

Reich, Y. and Fuchs, M. B.: 1989, A comparision of explicit optimal design methods, *Computers and Structures* **32**(1), 175–184.

Ritchie, G. and Hanna, F.: 1984, AM: A case study in AI methodology, *Artificial Intelligence* **23**, 249–268.

SAS, Institute Inc.: 1989, *SAS/STAT User's Guide, Version 6, Fourth Edition, Vols.1 and 2.*

Shah, J.: 1988, Synthesis of initial form for structural shape optimization, *ASME Transactions. Journal of Vibration, Acoustics, Stress, and Reliability in Design* **110**, 564–570.

Shapiro, A. D.: 1987, *Structured Induction in Expert System*, Turing Institute Press and Addison-Wesley.

Shavlik, J. W. and Dietterich, T. G.: 1990, *Readings in Machine Learning*, Morgan Kaufmann, Los Altos, CA.

Sheppard, D. and Palmer, A.: 1972, Optimal design of transmission towers by dynamic programming, *Computers and Structures* **2**, 455–468.

Simon, H. A.: 1983, Why should machines learn?, *in* Michalski et al.(ed.), *Machine Learning: An Artificial Intelligence Approach.*

Spillers, W. and Friedland, L.: 1972, On adaptive structural design, *American Society of Civil Engineers, Journal of Structural Design* **98**(ST10), 2155–2163.

Spillers, W.: 1963, Applications of topology in structural analysis, *American Society of Civil Engineers, Journal of Structural Design* **89**(ST4), 301–313.

Tadepalli, P.: 1989, Lazy Explanation-Based Learning: A solution to the intractable theory problem, *Proceedings of the International Internation Joint Conference on Artificial Intelligence*, pp. 694–700.

Topping, B.: 1983, Shape optimization of skeletal structures, *American Society of Civil Engineers, Journal of Structural Engineering* **109**(8), 1933–1951.

Van Harmelen, F. and Bundy, A.: 1988, Explanation-Based Generalisation = partial evaluation, *Artificial Intelligence Journal* **36**(3), 401–412.

Vanderplaats, G. N.: 1984, *Numerical Optimization Techniques for engineering design with applications*, McGraw Hill, New York.

Wang, C.-K. and Salmon, C. G.: 1984, *Introductory Structural Analysis*, Prentice Hall, New Jersey.

Wolfram, S.: 1988, *Mathematica*, Addison-Wesley.

Appendices

Appendix A

Simplifying Assumptions

We make a number of simplifying assumptions from an engineering perspective that greatly reduce the complexity (from an engineering standpoint) of the analysis task in the skeletal design domain:

1. Strucural members are joined by frictionless pins
2. Only a single loading case is allowed
3. Only statically determinate structures are considered
4. Only two-force members are used
5. Columns are *short and stocky*
6. The cross section of a column is square
7. Manufacturing problems are not taken into account
8. Columns and rods of any length and cross sectional area are available
9. Supports have no freedom of movement along either axis

A frame is subjected to a single loading condition when the forces applied do not change over time and the structure must only support one given set of loads.

A roof is an example of a structure that must support multiple sets of loading conditions because it must withstand its own weight, rain, wind, snow, and so on.

A structure is said to be *statically determinate* if the external reactions and the internal member forces can be determined using only the equations of static equilibrium. A necessary condition for a structure to be statically determinate is (Wang and Salmon 1984):

$$\text{Number of Possible Forces} = \text{Number of Internal Forces.}$$

The number of possible forces is computed by counting for each joint the degrees of freedom along the cartesian axis. The number of internal forces is equivalent to the number of members.

We point out (see Section 2.2) that the requirement of statical determinacy, the strongest of the above assumptions, stems from the ample experimental evidence that optimal structures subject to a single loading condition are almost always statically determinate.

Appendix B

Objective Function

In this appendix we illustrate the derivation of the weight function. Let n and m be the number of compression and tension members in a given structure, respectively. We define the weight of a structure as the sum of the weights of the compressive and tensile elements:

$$Weight = Weight^{(c)} + Weight^{(t)} \tag{B.8}$$

and

$$\begin{aligned} Weight^{(c)} &= \sum_{i=1}^{m} Weight_i^{(c)} \\ Weight^{(t)} &= \sum_{i=1}^{n} Weight_i^{(t)} \end{aligned} \tag{B.9}$$

where $Weight_i^{(c)}$ and $Weight_i^{(t)}$ are the weights of the i-th compressive and tension member, respectively.

In practical applications the weight of each member is a function of force, length, shape, and material; however, given our assumptions, we can simplify the expression for the weight.

Since columns are short and stocky there is a *direct stress* relationship between the axial (internal) force and the cross sectional area. As an example, for yellow pine the relation (Draffin and Collins 1950) is:

$$\frac{F^{(c)}}{A} = 4600 \tag{B.10}$$

where $F^{(c)}$ is the magnitude of the force and A is the cross sectional area, both measured in SI units. Furthermore, having assumed the cross section of a column is square and only one material is used, the weight of each column can be expressed as

$$Weight_i^{(c)} = k^{(c)} A_i l_i \tag{B.11}$$

where $k^{(c)}$ is a constant dependent on the material, A_i and l_i are cross sectional area and length of each compressive member, respectively.

From (B.10) we can derive that $A_i = F_i^{(c)}/4600$ which substituted into (B.11) gives:

$$Weight_i^{(c)} = \frac{k^{(c)}}{4600} F_i^{(c)} l_i. \tag{B.12}$$

Substituting (B.12) into (B.9) we have:

$$Weight^{(c)} = \frac{k^{(c)}}{4600} \sum_{i=1}^{n} F_i^{(c)} l_i \tag{B.13}$$

Similarly, assuming that for rods there exists a linear relationship between force and cross sectional area, we obtain:

$$Weight^{(t)} = k^{(t)} \sum F_i^{(t)} l_i \tag{B.14}$$

where $k^{(t)}$ is a constant that depends on the material, $F_i^{(t)}$ and l_i are the tensile force and length of each member, respectively.

Hence, substituting (B.13) and (B.14) into (B.8) we obtain:

$$Weight = \frac{k^{(c)}}{4600} \sum_{i=1}^{n} F_i^{(c)} l_i + k^{(t)} \sum_{i=1}^{m} F_i^{(t)} l_i \tag{B.15}$$

To simplify notation, the superscripts c and t will be omitted, whenever no confusion results. By using appropriate scaling, we can always transform (B.15) into:

$$Weight \approx Weight_{approx} = \sum_{i=1}^{n} F_i l_i + c \sum_{i=1}^{m} F_i l_i \tag{B.16}$$

which will be considered as the **objective function to be optimized.** With a slight abuse of notation, we shall use $Weight$ instead of $Weight_{approx}$.

Appendix C

Mathematica in the Design Domain

Mathematica (Wolfram 1988) is an-off-the-shelf "... *System for Doing Mathematics with Computers* ..." It includes numerical routines that range from the solution of simple equations to systems of partial differential equations. Mathematica also provides powerful 2-D and 3-D color plotting functions, symbolic manipulation of expressions, and a Turing-equivalent[10] programming language. In this thesis, we have used Mathematica extensively to perform most of the symbolic manipulation of algebraic expressions.

As an example, here is a program written in the Mathematica language to compute a symbolic solution for a system of two equations in the unknowns x and y:

```
Simplify[Solve[a x + b y == c, x - y == d, {x, y}]].
```

`Solve[]` and `Simplify[]` are two built-in Mathematica functions. The former, among other things, produces a symbolic solution of the system of equations. The latter simplifies any given expression by trying to minimize the number of terms. In theory, using these built-in symbolic packages, a symbolic solution is very easy to obtain. In practise, however, the built-in routines soon become inadequate as the number of equations grows. In fact, in the skeletal design domain, `Solve[]`

[10]The programming language provides constructs for composition, iteration, and conditionals.

produces expressions that are so lengthy and complex that `Simplify[]` does not produce any simplification. As an example, the system of equations in Table 5 could not be solved by the combination of `Solve[]` and `Simplify[]`. We speculate that the heuristics used in `Simplify[]` are not powerful enough to handle a ratio of polynomials. The simplification step is necessary because `Solve[]` can produce very lengthy expressions. The length of the expression affects the evaluation time. Thus, it might happen that the time it takes to evaluate the compiled expressions is greater than running a numerical method thereby offsetting the advantages of the compilation stage.

To circumvent these problems we have written a procedure using the programming language in Mathematica that solves the linear system of equations using Cramer's rule instead of the built-in `Solve[]` routine. Cramer's rule gives a solution of a system of equations as the ratio of two determinants computed from the matrix of coefficients. It turns out that numerator and denominator of this ratio are polynomials which are easy to simplify. Therefore, our compiler first uses Cramer's rule to solve the linear system, then simplifies the two polynomials separately, and, finally, it combines them by factoring out and deleting common factors between numerator and denominator.

Appendix D

Additional Experimental Results for UTILITYID3

This appendix complements the experimental results reported in Section 6.5 providing learning and tradeoff curves. These have been obtained using the experimental approach outlined in Section 3.5.2.

The curve in Figure 35 shows the average quality of a solution (ratio of the solution found to the optimal one) as a function of the number of examples. As the figure shows, the quality of the solution improves (slightly, after an initial drop) as more examples are given. This improvement in quality corresponds to longer

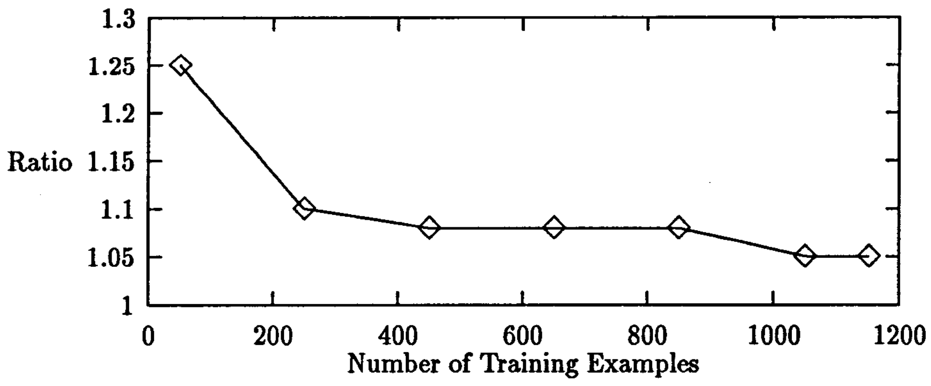


Figure 35. Average quality ($Ratio = \frac{\text{solution found}}{\text{optimal solution}}$).

processing time to obtain a solution. This can be seen from the curve in Figure 36 which shows the average serial CPU time (time, for short) as a function of the number of examples. The time increases until a sufficient number of examples is presented to the learning algorithm. As the program learns more from the examples, the problem solver must run the numerical optimizer on more stress states. This justifies the increase in time. Time required to obtain a solution and

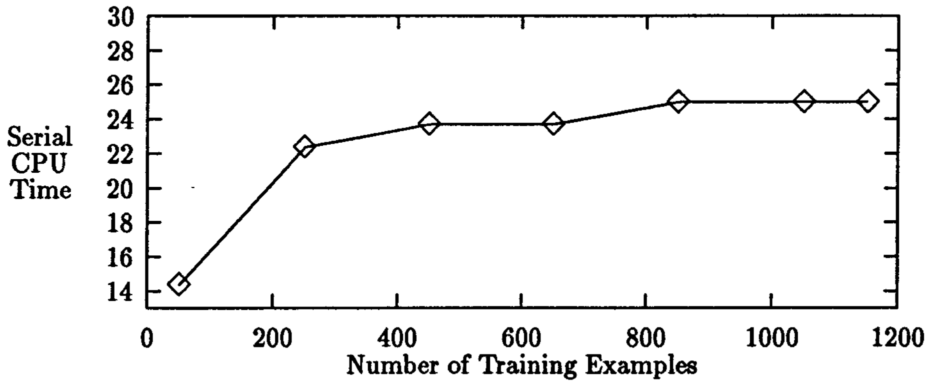


Figure 36. Average problem solving time.

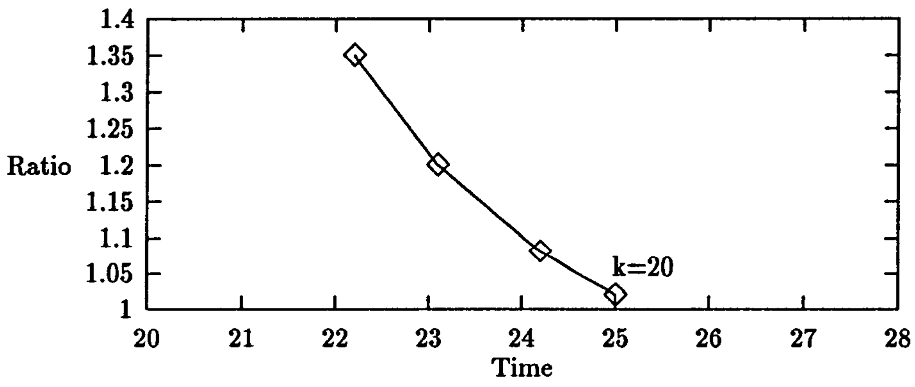


Figure 37. Ratio / Time tradeoff.

its quality induce a tradeoff that is illustrated in Figure 37. This plot indicates that a better ratio is obtained by spending more time to derive the solution. The plot also suggests that UTILITYID3 can also provide search control knowledge to derive solutions quickly. However, in this case, the solutions will be of poor quality. The time/quality tradeoff is typical in engineering endeavors and it is usually resolved by considerations like the engineers' intuition and experience. An analysis of these factors goes beyond the scope and nature of this thesis.

Appendix E

Design with Forbidden Regions

In this appendix we present a variation on the design task in Chapter 2 to demonstrate how the techniques introduced in this thesis can be applied to solve this problem as well.

As shown in Figure 38, we include forbidden regions (e.g., R) in the 2-D plane and assume that structural members cannot enter these regions. For instance, in the figure a solution with a member connecting directly load L and support S1 must be discarded because the member would cross the forbidden region R. The design task with forbidden regions is similar to the one used by Paul and Nevill, G.E. Jr. (1987) in their MOSAIC system. The remainder of this appendix shows the results of the application of the techniques we have developed to the design problem in Figure 38.

Symbolic Methods. The symbolic methods in Chapter 4 are used to unfold the system of equations derived applying the method of joints to the truss in Figure 38. Once the closed form expression of the objective function has been determined, the givens of the problem are incorporated into the function and the resulting expression is partially evaluated. Table 19 shows the final expression derived by our system for the stress state in the figure. The coordinates of the connection points C1 and C2 in Figure 38 are (x_1, y_1) and (x_2, y_2), respectively.

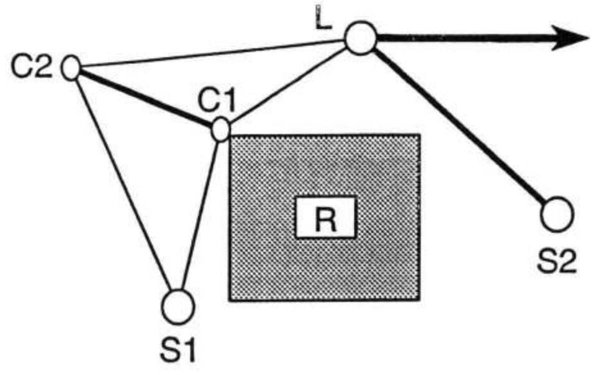


Figure 38. Skeletal design task with a forbidden region R.

The compiled expression in the table is much faster to evaluate than the original one. In fact, it takes 9.083 seconds to perform 100 evaluations of the compiled function in Table 19. In contrast, for the same values of the unknown variables, it takes 19.6 seconds to evaluate 100 times the non-compiled function.

Regularities. The optimization problem associated to the design task in the figure is 4-dimensional. This is because both coordinates of the extra connection points C1 and C2 must be determined. However, the regularity detection mechanisms in Chapter 5 allow us to decrease to one the dimensionality of the optimization problem. To see how this is accomplished, let us first consider the connection point C1. The EBL system is asked to prove that the unknown connection point C1 can be expressed in terms of known quantities. The system is able to derive a complete proof for the point. This is because C1 can be computed as the intersection of the boundary lines of the forbidden region and these are known entities. Therefore, at run-time, to compute the coordinates of C1, the problem solver tries all coordinates of the forbidden region. Once C1 has been identified, the system tries to prove that connection point C2 can be expressed in terms of given entities. This proof is similar to the one in Figure 25. In fact, C2 is first transformed in polar coordinates. Then, the system tries to determine the angle and the distance from C1 which, at this stage, is a known quantity. Angle and dis-

Table 19. Simplified objective function for the problem of Figure 38.

$$
\begin{aligned}
&Weight = \\
&\big(2.55\ 10^{25}x_1 - 5.1\ 10^{21}x_1{}^2 - 2.55\ 10^{25}x_2 - 5.1\ 10^{21}x_1x_2 + 2.04\ 10^{18}x_1{}^2x_2 + \\
&2.53\ 10^{18}x_2{}^2y_1 - 4.49\ 10^{14}x_2{}^2y_1{}^2 + 5.3\ 10^{17}x_1{}^2y_2 + 3.46\ 10^{18}x_2y_1y_2 + \\
&1.02\ 10^{22}x_2{}^2 - 1.02\ 10^{18}x_1x_2{}^2 - 2.04\ 10^{14}x_1{}^2x_2{}^2 - 1.02\ 10^{18}x_2{}^3 + \\
&2.04\ 10^{14}x_1x_2{}^3 + 2.55\ 10^{25}y_1 + 2.45\ 10^{21}x_1y_1 + 192.x_1{}^2y_1 - \\
&1.52\ 10^{22}x_2y_1 - 4.9\ 10^{17}x_1x_2y_1 - 0.03x_1{}^2x_2y_1 + 0.06x_1x_2{}^2y_1 - \\
&1.02\ 10^{14}x_2{}^3y_1 - 1.24\ 10^{22}y_1{}^2 + 4.73\ 10^{18}x_2y_1{}^2 - 192.y_1{}^3 - \\
&0.03x_2y_1{}^3 - 2.55\ 10^{25}y_2 + 7.75\ 10^{21}x_1y_2 - 0.03x_1{}^3y_2 + \\
&5.\ 10^{21}x_2y_2 - 2.02\ 10^{18}x_1x_2y_2 + 2.04\ 10^{14}x_1{}^2x_2y_2 + 1.02\ 10^{14}x_1x_2{}^2y_2 + \\
&1.72\ 10^{22}y_1y_2 - 2.69\ 10^{18}x_1y_1y_2 - 0.015x_1{}^2y_1y_2 - 4.9\ 10^{14}x_1x_2y_1y_2 + \\
&1.02\ 10^{14}x_1y_2{}^3 + 1.02\ 10^{18}x_2y_2{}^2 + 2.04\ 10^{14}x_2y_1{}^2y_2 - 5.1\ 10^{17}y_2{}^3 - \\
&5.1\ 10^{13}x_2{}^2y_1y_2 - 1.02\ 10^{18}y_1{}^2y_2 + 0.03x_1y_1{}^2y_2 + 0.01y_1{}^3y_2 - 4.8\ 10^{21}y_2{}^2 + \\
&2.44\ 10^{18}x_1y_2{}^2 - 2.96\ 10^{14}x_1{}^2y_2{}^2 - 2.04\ 10^{14}x_1x_2y_2{}^2 + 5.1\ 10^{17}y_1y_2{}^2 - \\
&1.02\ 10^{14}x_2y_1y_2{}^2 - 5.1\ 10^{13}y_1{}^2y_2{}^2 + 5.1\ 10^{17}x_2{}^2y_2 - 5.1\ 10^{13}y_1y_2{}^3\big) \big/ \\
&\big(4.\ 10^{13}x_1y_1 - 4.\ 10^{13}x_2y_1 - 8.\ 10^{9}x_1x_2y_1 + 8.\ 10^{9}x_2{}^2y_1 - 1.2\ 10^{14}y_1{}^2 + \\
&4.4\ 10^{10}x_2y_1{}^2 - 4.\ 10^{6}x_2{}^2y_1{}^2 - 4.\ 10^{13}x_1y_2 + 8.\ 10^{9}x_1{}^2y_2 + 4.\ 10^{13}x_2y_2 - \\
&8.\ 10^{9}x_1x_2y_2 + 2.4\ 10^{14}y_1y_2 - 4.4\ 10^{10}x_1y_1y_2 - 4.4\ 10^{10}x_2y_1y_2 + 8.\ 10^{6}x_1x_2y_1y_2 - \\
&1.2\ 10^{14}y_2{}^2 + 4.4\ 10^{10}x_1y_2{}^2 - 4.\ 10^{6}x_1{}^2y_2{}^2\big)
\end{aligned}
$$

Table 20. Simplified objective function for the problem of Figure 38 with only one remaining design variable.

$$Weight_{simplified} = 10^4\ \frac{4.78\ 10^9 + 3.06\ 10^7\rho + 4.53\ 10^4\rho^2 + 4.56\rho^3}{1.49\ 10^5 + 1.75\ 10^3\rho + 3.28\rho^2}$$

tance cannot be directly related to givens. Therefore, as in Section 5.2, the EBL system uses geometric knowledge and discovers that the angle $\mathtt{L1} - \widehat{\mathtt{C1}} - \mathtt{C2}$ is one half of the angle $\mathtt{L1} - \widehat{\mathtt{C1}} - \mathtt{S1}$. Since C1 is marked as a given, this latter angle is also a given. Thus, the system is inductively able to determine the angle of C2 in polar coordinates. On the contrary, EBL cannot relate the distance from C1 to givens; hence, it is left as an independent variable (ρ) to be determined at run time. The regularities decrease the number of independent variables (dimensionality) in the optimization problem from the original four cartesian coordinates to the one polar coordinate (ρ) needed to locate C2. The simplified expression for this one-dimensional optimization problem is reported in Table 20. The `FindMinimum[]` package in Mathematica determines the correct minimum ($\rho = 1143$) in 0.15 seconds. The same package took 24.82 seconds to determine the minimum of the 4-dimensional objective function before compilation.