THE USE OF ARTIFICIAL INTELLIGENCE TO IMPROVE THE NUMERICAL OPTIMIZATION OF COMPLEX ENGINEERING DESIGNS

BY MARK A. SCHWABACHER

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Thomas Ellman and Andrew Gelsey

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 1996

ABSTRACT OF THE DISSERTATION

The Use of Artificial Intelligence to Improve the Numerical Optimization of Complex Engineering Designs

by Mark A. Schwabacher

Dissertation Directors: Thomas Ellman and Andrew Gelsey

Gradient-based numerical optimization of complex engineering designs promises to produce better designs rapidly. However, such methods generally assume that the objective function and constraint functions are continuous, smooth, and defined everywhere. Unfortunately, realistic simulators tend to violate these assumptions. We present several artificial intelligence-based techniques for improving the numerical optimization of complex engineering designs in the presence of such pathologies in the simulators. We have tested the resulting system in several realistic engineering domains, and have found that using our techniques can greatly decrease the cost of design space search, and can also increase the quality of the resulting designs.

Acknowledgements

I thank all members of the Rutgers HPCD (Hypercomputing and Design) project. In particular, I thank my advisors, Tom Ellman and Andrew Gelsey, for their advice, support, and collaboration over the years. I thank Haym Hirsh for being on my thesis committee, for his advice, and for his help with the inductive learning work. I thank Doyle Knight for being on my thesis committee, and for his help with the inlet domain. I thank Don Smith and Keith Miyake for their help with the aircraft simulator. I thank John Keane and Tim Weinrich for their help with the yacht simulator. I thank Gerard Richter for his help with numerical analysis and his collaboration on search-space reformulation. I thank Saul Amarel and Lou Steinberg for setting the overall direction of the HPCD project's research.

I would also like to thank some of the HPCD project's industry collaborators without whom this research would not have been possible: Gene Bouchard of Lockheed Martin, Marty Haas of United Technologies Research Center, John Letcher of AeroHydro, and Nils Salvesen and Marty Fritts of SAIC. And finally, I want to thank my parents.

This research was partially supported by a Rutgers University Excellence Fellowship, and partially supported by the Advanced Research Projects Agency of the Department of Defense (ARPA) and the National Aeronautics and Space Administration (NASA) under the following grants: NASA grant NAG2-817, DARPA-funded NASA grant NAG 2-645, and ARPA contract DABT 63-93-C-0064.

Publication notes. Much of the research presented in this thesis has appeared elsewhere with various co-authors, all of whom I thank. The work in Chapter 2 was described in the *Proceedings of the Tenth Conference on Artificial Intelligence for Applications*, with Haym Hirsh and Tom Ellman as co-authors [Schwabacher *et al.* 1994]. The work in Chapter 3 was described in *Artificial Intelligence in Design '96*, with Tom Ellman, Haym Hirsh, and Gerard Richter as co-authors [Schwabacher *et al.* 1996b]. The work in Chapter 4 was described in the same volume, but with Andrew Gelsey and Don Smith as co-authors [Gelsey *et al.* 1996a]. Finally, the work in Chapter 6 will appear in the *6th AIAA/NASA/USAF Multidisciplinary Analysis & Optimization Symposium*, with Andrew Gelsey as the co-author [Schwabacher and Gelsey 1996b].

Table of Contents

Chapter 1

Introduction

1.1 The Design Problem

Informally, the engineering design problem is to find quickly a good design in a complex engineering domain. Here "good" can, for example, mean low cost or high quality.

Examples of design problems include designing the fastest possible racing yacht, subject to the constraints that it satisfies the race committee's rules regarding what yachts can be entered in the race, and that its cost not exceed the available budget, or designing a supersonic aircraft with the lowest possible life-cycle cost, subject to the constraint that it be capable of carrying 70 passengers from New York to Tokyo at Mach 2.

Once a particular design problem has been stated formally, it can be attacked using nonlinear optimization (see section Section A.2). Nonlinear optimization works reasonably well when the objective and constraint functions defined by the simulator satisfy certain assumptions of continuity and smoothness. Unfortunately, these assumptions often do not hold for realistic simulators, because of the *pathologies* described in section Section 1.4.

1.2 Importance

We have defined the problem to be that of quickly producing better designs in complex engineering domains. Solving this problem has tremendous commercial, military, and scientific importance.

Automated engineering design is likely to be extremely beneficial to commercial enterprises, as it would allow them to reduce the cost of producing new designs, to produce

better designs than their competitors, and to bring new concepts to the production line faster. All of these improvements would allow the companies that take advantage of them to increase market share and profitability.

Automated engineering design should also be very beneficial to the military, as it would reduce the cost of the design process, reduce the cost of building defense systems, and increase the performance of these systems. These advances would allow a strong defense to be maintained in the presence of shrinking defense budgets. They would also allow the military to rapidly adapt to emerging threats.

Automated engineering design might be highly beneficial to science. When applied to the design of scientific instruments such as telescopes and electron microscopes, it could conceivably allow the creation of improved instruments that might allow startling new discoveries to be made.

1.3 Testbed domains

To demonstrate that the techniques described in this thesis are valuable when applied to the important class of realistic engineering problems, we tested them in the following four realistic engineering domains:

- The design of racing yachts of the type used in the America's Cup race.
- The design of airframes for supersonic transport aircraft.
- The design of exhaust nozzles for the jet engines of supersonic transport aircraft.
- The design of inlets for supersonic cruise missiles.

These domains are described in Appendix A.

1.4 Pathologies

In realistic engineering domains such as those described in the previous section, engineers often rely on *legacy codes* to simulate proposed designs. These codes are software programs that have generally been developed over many years, by many people. They

often are based both on empirical data and theoretical equations. Engineers are reluctant to change or replace the legacy codes, because they have faith in the codes due to years of comparisons between the codes and physical experiments. Unfortunately, legacy codes tend to have have many *pathologies* (described below) which make it difficult to use them within an automated design system.

Even if it were practical to re-write the legacy codes using modern software engineering techniques, some pathologies would always exist. Many simulators require the solution of a set of equations that has no closed-form solution. In these cases, numerical root finding must be used. Numerical root finding is undecidable [Gill *et al.* 1981]. Therefore, imperfect root-finding algorithms must be used.

There are several types of pathologies:

- **Multiple local optima.**

  Physical. There can be multiple physically meaningful local optima.

  Noise. Noise can be caused, for example, by numerical solvers that do not find the exact root, or by gridding[1] systems that change the number of grid points based on the values of design parameters. It shows up as spurious nonphysical oscillations in the objective and constraint functions. These oscillations can cause apparent local optima that are not in fact locally optimal in the true physics.

- **Nonsmoothness**

  Discontinuity in function. For example, when the afterburner in a supersonic jet turns on, there is a discontinuous increase in thrust.

  Discontinuity in first derivative, also known as a "ridge". For example, when the depth of the keel of a 12-Meter yacht exceeds the maximum depth permitted under the 12-Meter Rule[2] without a sail-area penalty, the sail-area

[1] Gridding is the process of dividing a surface or volume into discrete panels for use with a computational method.

[2] The 12-Meter Rule is a set of constraints set forth by the America's Cup race committee which defines what yachts are permitted to participate in the race. It is further described in Chapter 3 and Appendix A.

penalty begins to be applied, which causes a discontinuity in the first derivative of velocity with respect to keel depth.

- **Unevaluable points**

  Undefined points. At these points, the function has no value. An example of such a point is the course time (amount of time it takes to complete the race course) for a yacht that cannot complete the course.

  Unevaluably bad points. At these points, it is known that the design is bad, but it is not possible to compute a number indicating just how bad it is. For example, if a yacht floats with its rail below the water line, it may be known known that the drag from the rail will cause the yacht to be very slow, but the simulator may not be able to model this drag, and therefore might not be able to compute just how slow the yacht is.

  Unevaluable points due to limitations of simulator. These points may be perfectly good designs, but because of limitations in the simulator, it is not possible to evaluate them. For example, a simulator that was designed to simulate monohull yachts may not be able to simulate a catamaran.

The pathologies encountered in each of our testbed domains are further described in Appendix A.

1.5 Current Methods

1.5.1 Manual design

Most complex engineering design is done today manually. Engineers often use computer-aided design (CAD) tools to edit their designs, and computational simulators to simulate the designs (or, more often, portions of the designs). The process often works as follows: The engineer uses the CAD tool to make a modification to the design. He or she then evaluates the design using a computational simulator. Next, he or she examines the output of the simulator, and uses the CAD tool to make further changes to the design on the basis of this output.

This design-evaluate-redesign loop seems like it would benefit from gradient-based optimization. Some engineers have tried to use automated optimization, and have been unhappy with the results; others have heard about these failures, and have therefore not even tried to use automated optimization [Knight 1994]. We believe that the pathologies described in Section 1.4 are a major reason for these failures.

The manual design process described above is slow. It requires large amounts of human time and calendar time. It also usually fails to produce an optimal design. Human designers tend to use satisficing rather than optimization — that is, they tend to stop the design process when they find a design that is "good enough," instead of seeking to find a design that is even locally optimal. They also tend to decompose problems in ways that sacrifice optimality. This decomposition is necessary to divide the design problem into more manageable pieces that can be attacked by separate groups of people, but because of interactions among the subparts into which the problem is decomposed, the decomposition tends to result in a suboptimal design.

1.5.2 Existing optimization techniques

Conventional, gradient-based numerical optimization has been successfully applied to only a very limited fraction of relatively small engineering problems. In more complex problems, which tend to include pathologies, conventional numerical optimization techniques tend to produce poor designs, or even to crash.[3]

Stochastic optimization methods such as Simulated Annealing (SA) and Genetic Algorithms (GA) are able to deal with some of the pathologies described in section Section 1.4. SA's are based on an analogy with thermodynamics, specifically the way in which metals cool and anneal[Press *et al.* 1986]. GA's are based on an analogy with Darwin's theory of evolution[Goldberg 1989]. The problem with applying SA's and GA's to complex engineering design is that they are slow. They tend to require thousands (if not tens of thousands) of simulations before they converge. It is not feasible to apply such algorithms to complex design problems in which a single simulation can

[3]To say that a program *crashes* means that it exits abnormally. On UNIX systems, this usually means that it produces a floating point exception or a segmentation fault.

Table 1.1: Techniques to handle pathologies during optimization.

technique	multiple optima	nonsmoothness	unevaluable points
prototype selection	√	√	
reformulation selection		√	
model constraints			√
rule-based gradients			√
multilevel design	√		

take hours.

Powell [Powell 1990, Powell and Skolnick 1993] has built a system called Inter-GEN that seeks to combine the ability of genetic algorithms to handle certain pathologies with the speed of numerical optimization algorithms. It contains a genetic algorithm, and a numerical optimizer, and uses a rule-based expert system to decide when to switch between the two. Inter-GEN is discussed further in the related work chapter (Chapter 7).

1.6 Our Approach

Our approach is to use a state-of-the-art gradient-based optimization method, such as sequential quadratic programming [Gill *et al.* 1981], augmented with artificial-intelligence-based techniques. (See Appendix A for an overview of numerical optimization.) We use machine learning-based techniques to set up the optimization with *prototype selection* (see Chapter 2) and *reformulation selection* (see Chapter 3). (See Appendix A for an overview of machine learning.) Prototype selection helps the optimizer deal with the problems of multiple local optima and nonsmoothness. Reformulation selection also helps the optimizer deal with nonsmoothness. We use knowledge-based techniques to overcome pathologies during the optimization in *model constraints* (see Chapter 4) and *rule-based gradients* (see Chapter 5). Both of these techniques allow the optimizer to deal with unevaluable points. Finally, we use a multilevel design paradigm (see Chapter 6) to deal with multiple local optima. Table 1.1 summarizes the techniques and the pathologies that they alleviate.

1.6.1 Prototype selection

The first step for most case-based design systems is to select an initial prototype from a database of previous designs. The retrieved prototype is then modified to tailor it to the given goals. For any particular design goal the selection of a starting point for the design process can have a dramatic effect both on the quality of the eventual design and on the overall design time, particularly when the search space is pathological. We present a technique for automatically constructing effective prototype-selection rules. Our technique applies a standard inductive-learning algorithm, C4.5, to a set of training data describing which particular prototype would have been the best choice for each goal encountered in a previous design session. We have tested our technique in the domains of racing-yacht-hull design and airframe design, comparing our inductively learned selection rules to several competing prototype-selection methods. Our results show that the inductive prototype-selection method leads to better final designs when the design process is guided by a noisy evaluation function, and that the inductively learned rules will often be more efficient than competing methods.

1.6.2 Reformulation selection

It is well known that search-space reformulation can improve the speed and reliability of numerical optimization in engineering design. We argue that the best choice of reformulation depends on the design goal, and present a technique for automatically constructing rules that map the design goal into a reformulation chosen from a space of possible reformulations. We tested our technique in the domains of racing-yacht-hull design and airframe design, where each reformulation corresponds to incorporating constraints into the search space. We used a standard inductive-learning algorithm, C4.5, to learn rules from a set of training data describing which constraints are active in the optimal design for each goal encountered in a previous design session. We then used these rules to choose an appropriate reformulation for each of a set of test cases. Our experimental results show that using these reformulations improves both the speed and the reliability of design optimization, outperforming competing methods

and approaching the best performance possible.

1.6.3 Model constraints

Automated search of a space of candidate designs seems an attractive way to improve the traditional engineering design process. To make this approach work, however, the automated design system must include both knowledge of the modeling limitations of the method used to evaluate candidate designs and also an effective way to use this knowledge to influence the search process. We suggest that a productive approach is to include this knowledge by implementing a set of *model constraint* functions which measure how much each modeling assumption is violated, and to influence the search by using the values of these model constraint functions as constraint inputs to a standard constrained nonlinear optimization numerical method. We test this idea in the domains of conceptual design of supersonic transport aircraft and detailed design of nozzles for supersonic jets. Our experiments indicate that our model constraint communication strategy can decrease the cost of design space search by one or more orders of magnitude.

1.6.4 Rule-Based Gradients

Gradient-based numerical optimization of complex engineering designs offers the promise of rapidly producing better designs. However, such methods generally assume that the objective function and constraint functions are continuous, smooth, and defined everywhere. Unfortunately, realistic simulators tend to violate these assumptions. We present a knowledge-based technique for intelligently computing gradients in the presence of such pathologies in the simulators, and show how this gradient computation method can be used as part of a gradient-based numerical optimization system. We tested the resulting system in the domains of conceptual design of supersonic transport aircraft and detailed design of a missile inlet, and found that using rule-based gradients can decrease the cost of design space search by one or more orders of magnitude.

1.6.5 Multilevel Design

Multi-level representations have been studied extensively by artificial intelligence researchers. We utilize the multi-level paradigm to attack the problem of performing engineering design optimization in the presence of many local optima, using multiple levels of simulation, paired with a multi-level search space. We tested the resulting system in the domain of conceptual design of supersonic transport aircraft, and found that using multi-level simulation and optimization can decrease the cost of design space search by one or more orders of magnitude.

1.7 Claims

We present here several claims about our techniques. We will review these claims in the conclusion chapter (Chapter 8).

1.7.1 Effectiveness

The techniques described in this thesis make design optimization faster, and they enable the production of better designs. In this thesis, we present experimental results to show that each technique is effective. We demonstrate effectiveness by performing optimizations with and without each technique, and showing that using the technique improves the design quality or reduces the amount of CPU time needed to achieve a particular level of design quality. The experimental results presented in this thesis show increases in optimization speed ranging from 35% to three orders of magnitude, with no loss of quality. They also show that these techniques can be used in a way that has a high probability of producing high-quality designs. Each technique was shown to be useful in two different realistic engineering domains (see Table 1.2). In the conclusions chapter we will summarize the effectiveness of the different techniques in the different domains, and we we will attempt to describe the properties of a domain that enable our techniques to be effective.

1.7.2 Degree of Automation

Knowledge needed

All of the techniques described in this thesis require one or more simulators, a set of constraint functions, and a set of upper and lower bounds on the design parameters. These simulators, constraint functions, and bounds embody knowledge of the domain. In addition, some of the techniques exploit *background knowledge* about the domain that must be obtained from domain experts. For example, model constraints require knowledge of the modeling assumptions in the simulator. Some use general knowledge of the types of pathologies that tend to occur when numerical optimization is applied to complex simulators. This general knowledge is, for the most part, knowledge that we discovered ourselves, and that we believe is applicable across many engineering domains. For example, the rules used in rule-based gradients embody such knowledge. We will summarize this knowledge in the conclusion chapter.

Human intervention needed

All of the optimizations described in this thesis were fully automated, in the sense that there never was a human inside the optimization loop.[4] In addition, there was some automation of the process of setting up the optimizations. For example, choosing a starting prototype or a reformulation of the search space has traditionally been done manually, but is automated in our prototype selection and reformulation selection experiments. There of course is still considerable human effort needed in constructing the simulators and constraint functions, creating an initial formulation of the search space, and selecting the stopping criteria for the optimizer. In the conclusion chapter, I will summarize the intervention that is needed for setting up each technique in a new domain.

1.7.3 Generality

[4]That is, a sequence of simulations leading to an apparent optimum was carried out without human intervention.

Table 1.2: The domains in which we tested our techniques.

technique	racing yachts	supersonic airframes	supersonic nozzles	supersonic inlets
prototype selection	√	√		
reformulation selection	√	√		
model constraints		√	√	
rule-based gradients		√		√
multilevel design		√	√	

Each of the techniques described in this thesis was tested in two realistic engineering domains. The domains in which each technique were tested are listed in Table 1.2. We believe that the techniques are applicable across a wide variety of engineering domains. Specifically, each technique is applicable to domains that have the pathologies listed in the relevant row of Table 1.1. If there is little or no pathology, then the techniques will not be helpful, but we argue that they will not hurt optimization performance very much. If there is a moderate amount of pathology — as there is in the domains we have encountered — then our techniques will be helpful. If, however, there is an extremely large amount of pathology, then optimization will fail with or without our techniques, so again our techniques will neither help nor hurt. We will return to this issue in the conclusion chapter.

1.7.4 Novelty

We believe that all of the techniques presented in this thesis are new. In addition, we believe that we are attacking a problem that has received little attention in the past: solving design optimization problems that require numerical simulators that are both expensive and pathological. There has been much past work in gradient-based optimization, but most of it has not focused on the pathologies described above, and has not incorporated artificial intelligence-based techniques. There has been considerable work on applying artificial intelligence to design, but most of it has focused on symbolic rather than numerical techniques. Genetic algorithms and simulated annealing are able to deal with numerical design spaces that include certain pathologies, but they are much slower than gradient-based methods and are therefore not very useful when each

iteration requires an expensive simulation. The Related Work chapter (Chapter 7) further discusses this issue.

Chapter 2

Prototype Selection and Synthesis

2.1 Introduction

This chapter describes two ways of using inductive learning to choose a starting prototype for a design optimization. *Prototype selection* maps the design goal into a selection of a prototype from a database of existing prototypes. *Prototype synthesis* synthesizes a new prototype by mapping the design goal into the design parameters that define a prototype. In some domains, some design goals are not achievable. We therefore also used inductive learning to predict whether a given design goal is achievable, before we attempt to synthesize a starting prototype for the goal. We performed experiments to measure not only the error rate of the machine learning algorithms when applied to each of these problems, but also the impact on optimization performance of using these techniques.

2.2 The Problem

Many automated design systems begin by retrieving an initial prototype from a library of previous designs, using the given design goal as an index to guide the retrieval process [Sycara and Navinchandra 1992]. The retrieved prototype is then modified by a set of design modification operators to tailor the selected design to the given goals. In many cases the quality of competing designs can be assessed using domain-specific evaluation functions, and in such cases the design-modification process is often accomplished by an optimization method such as hillclimbing search [Ramachandran *et al.* 1992, Ellman *et al.* 1992]. Such a design system can be seen as a *case-based reasoning* system [Kolodner 1993], in which the prototype-selection method is the *indexing* process, and

the optimization method is the *adaptation* process.

In the context of such case-based design systems, the choice of an initial prototype can affect both the quality of the final design and the computational cost of obtaining that design, for three reasons. First, prototype selection may impact quality when the design process is guided by a nonlinear evaluation function with unknown global properties that may include pathologies. Since there is no known method that is guaranteed to find the global optimum of an arbitrary nonlinear function [Press *et al.* 1986], most design systems rely on iterative local search methods whose results are sensitive to the initial starting point. Second, prototype selection may impact quality when the prototypes lie in disjoint search spaces. In particular, if the system's design modification operators cannot convert any prototype into any other prototype, the choice of initial prototype will restrict the set of possible designs that can be obtained by *any* search process. A poor choice of initial prototype may therefore lead to a suboptimal final design. Finally, the choice of prototype may have an impact on the time needed to carry out the design modification process — two different starting points may yield the same final design but take very different amounts of time to get there. In design problems where evaluating even just a single design can take tremendous amounts of time, we believe that selecting an appropriate initial prototype can be the determining factor in the success or failure of the design process.

2.3 The Technique

To use inductive learning to form prototype-selection rules, we take as training data a collection of design goals, each labeled with which prototype in the library is best for that goal. "Best" can be defined to mean the prototype that best satisfies the design objectives, the prototype that results in the shortest design time, or the prototype that optimizes some combination of design quality and design time.

Inductive learning is particularly suitable in the context of an automated design system because training data can be generated in an automated fashion. For example, one can choose a set of training goals and perform an optimization for all combinations

of training goals and library prototypes. One can then construct a table that records which prototype was best for each training goal. (The cost of generating this table is discussed in section Section 2.5.1.) This table can be used by the inductive-learning algorithm to generate rules mapping the space of all possible goals into the set of prototypes in the library. If learning is successful this mapping extrapolates from the training data and can be used successfully in future design sessions to map a new goal into an appropriate initial prototype in the design library.

The specific inductive-learning systems used in this work are C4.5 [Quinlan 1993] (release 3.0, with windowing turned off) and CART[1] [Breiman 1984]. (See Appendix A for a summary of inductive learning and C4.5.)

2.4 Results

Our hypotheses were the following:

1. The best choice of prototype depends on the goal, so the inductive learning methods (C4.5 and CART) would have lower error rates than the simpler learning methods [most frequent class (MFC) and constant regression].

2. Using the prototypes chosen by the inductive learning methods would produce better optimization performance (better designs and/or lower CPU cost) than using the prototypes chosen by the simpler learning methods, or by random guessing, or by the heuristic methods (closest goal and best initial evaluation).

We were interested in determining not only whether these hypotheses were true, but also the magnitude of the differences in performance among methods. We tested prototype selection using C4.5 in the yacht domain, and tested prototype synthesis using CART in the airframe domain.

In the yacht domain, our initial results led us to make additional hypotheses, and we then performed additional experiments to test the additional hypotheses.

[1] CART stands for Classification And Regression Trees

2.4.1 Yacht domain

We tested our prototype-selection method in the yacht domain. In this domain, a design is represented by eight design parameters which specify the magnitude with which a set of geometric operators are applied to the B-spline surfaces of the starting prototype. The goal is to design the yacht which has the smallest course time for a particular wind speed and race course. (See Appendix A for a more complete description of this domain.)

We conducted several sets of experiments. In each case we compare our approach with each of four other methods:

1. **Closest goal**. This method requires a measure of the distance between two goals, and knowledge of the goal for which each prototype in the design library was originally optimized. It chooses the prototype whose original goal has minimum distance from the new goal. Intuitively, in our yacht-design problem this method chooses a yacht designed for a course and windspeed most similar to the new course and windspeed.

2. **Best initial evaluation**. This method requires running the evaluation function on each prototype in the database. It chooses the prototype that, according to the evaluation function, is best for the new goal (before any operators have been applied to the prototype). In the case of our yacht-design problem this corresponds to starting the design process from whichever yacht in the library is fastest for the new course and windspeed.

3. **Most frequent class**. This is actually a very simple inductive method that always chooses a fixed prototype, namely the one that is most frequently the best prototype for the training data.

4. **Random**. This method involves simply selecting a random element from the design library, using a uniform distribution over the designs.

We compare these methods using two different evaluation criteria:

1. **Error rate**. How often is the wrong prototype selected?

2. **Course-time increase**. How much worse is the resulting average course time than it would be using the optimal choice that an omniscient selection would make?

In our experiments we focus primarily on the question of how well our inductive-learning prototype-selection method handles problems where the prototypes lie in disjoint search spaces. Our experiments therefore explore how prototype selection affects the quality of the final design.

For the prototype selection experiments in the yacht domain, we used the Rutgers Hillclimber as our optimizer. It is an implementation of steepest-descent hillclimbing (see Section A.2), that has been augmented so as to allow it to "climb over" bumps in the surface defined by the objective function that have less than a certain width or a certain height.

For our first set of experiments we created a database of four designs that would serve as our sample prototype library (and thus also serve as the class labels for the training data given to our inductive learner). To simulate the effect of having each prototype define a different space, the design library was created by starting from a single prototype (the Stars and Stripes '87) and optimizing for four different goals using all eight of our design-modification operators. All subsequent design episodes used only four of the eight operators, so that each yacht would define a separate space.[2]

We defined a space of goals to use in testing the learned prototype-selection rules. Each goal consists of a windspeed and a racecourse, where the windspeed is constrained to be 8, 10, 12, 14, or 16 knots, the racecourse is constrained to be 80% in one direction, and 20% in a second direction, and each direction is constrained to be an integer between 0 and 180 degrees. This space contains 162,900 goals.

To generate training data we defined a set of "training goals" that spans the goal space. This smaller set of goals was defined in the same fashion as for the testing set of

[2]The four operators we chose were *Scale-X*, *Scale-Y*, *Prism-Y*, and *Scale-Keel*. We chose these operators because the results of our earlier work on operator-importance analysis suggested that these are the four most important operators [Ellman and Schwabacher 1993].

Table 2.1: A portion of the input to C4.5 for prototype selection in the yacht domain.

Long-Leg	Short-Leg	Windspeed	Initial-Design
180	0	8	Design 1
180	0	10	Design 2
180	0	12	Design 2
180	0	14	Design 2
180	0	16	Design 2
180	90	8	Design 1
180	90	10	Design 4
180	90	12	Design 4
180	90	14	Design 4
180	90	16	Design 1

```
long-leg <= 90 :
|   windspeed > 10 : Design-1
|   windspeed <= 10 :
|   |   short-leg <= 90 : Design-1
|   |   short-leg > 90 : Design-2
long-leg > 90 :
|   windspeed > 14 : Design-2
|   windspeed <= 14 :
|   |   windspeed <= 10 : Design-4
|   |   windspeed > 10 : Design-4
```

Figure 2.1: Example of a prototype-selection decision tree generated by C4.5.

goals except that the directions in the racecourse are restricted to be only 0, 90, or 180 degrees, yielding a smaller space of 30 goals. To label the training data we attempted to find designs for each of the 30 goals starting from each of the four prototypes using the restricted set of operators, and determined which starting point was best.

To generate test data we randomly selected ten "testing goals" from the goal space. We then generated designs starting from each of the four prototypes in the database for each of these testing goals to determine which prototype was best, as well as to determine how much of a loss in course time each incorrect selection would impose. Table 2.1 shows a portion of the input to C4.5, and Figure 2.1 gives an example of a decision tree output by C4.5. Table 2.2 compares the results using C4.5 with the other prototype-selection methods. (Since there are four prototypes, one would expect

Table 2.2: Comparison of prototype-selection methods when trained on a set of goals that spans the goal space, using AHVPP.

Method	Error Rate	Course-Time Increase (sec)
Inductive Learning	30%	24
Most Frequent Class	70%	47
Random Guessing	75%	62
Best Init Eval	70%	64
Closest Goal	70%	78

random guessing to get 75% of the test examples wrong.)

In this experiment, the inductive method (C4.5) performed better than the other methods on both measures of performance, confirming our two hypotheses. Moreover, we were particularly surprised by how poorly the non-inductive prototype-selection methods (closest goal and smallest initial evaluation) performed — our expectation was that the prototypes chosen by these methods would be close in "design space" to the optimal final design, thus yielding better final designs than starting from the other prototypes.

After studying these results we generated two new hypotheses for why these two prototype-selection methods did not work well. The first is that the shape of the design space may be such that there is little relationship between the distance between two designs and the ability of the hillclimber to climb from one design to the other. If the space contains "bumps" or "ridges" over which the hillclimber cannot climb, then it might be more important for the initial prototype to be on the "right side" of a bump or a ridge than for it to be close to the optimal point. Our second new hypothesis was that some of the prototypes in the database may be "bad" prototypes. This could be the case if the hillclimber got stuck at a local (non-global) optimum during the run that produced the prototype. This latter hypothesis was supported by the fact that one of the four prototypes was never found to be a good starting point for any of the 30 goals in the training data (not even the goal for which it was supposedly optimal, since it wound up being a local optimum and starting from another prototype yielded a superior result). In a realistic design scenario, when there is no control over the source

of a design library, there could easily be "bad" prototypes included. Unlike the non-inductive prototype-selection methods, the inductive methods learn to avoid the bad prototypes.

We performed some experiments to test our first new hypothesis that the closest-goal and smallest-initial-evaluation methods performed poorly because of the "bumps" in the evaluation function. We repeated the earlier experiments using a simplified, "smooth" velocity prediction program, called "RUVPP," that we developed at Rutgers. RUVPP differs from the more complex AHVPP in several respects. To begin with, RUVPP represents a yacht as a list of major geometric dimensions such as length, depth, and beam, rather than B-spline surfaces. Furthermore, RUVPP embodies a number of simplifying assumptions about the physics of sailing that are not made in AHVPP. Nevertheless, the simple version, RUVPP, is useful for two reasons: RUVPP is much faster to execute than AHVPP, and RUVPP has fewer of the bumps and ridges that appear in AHVPP. We therefore expect that a hillclimbing search algorithm is less likely to get stuck on the wrong side of a bump or ridge when the simple version, RUVPP, is used as an evaluation function. Table 2.3 presents the results of experiments comparing the performance of inductively learned prototype-selection rules to the other prototype-selection methods, repeating our earlier experiments, but using RUVPP as the evaluation function, and using forty random test cases instead of just ten.

Because RUVPP is much faster than AHVPP, we conducted additional supporting experiments to test our first new hypothesis, to see if using a spanning set of goals as training data was significant for our results. In particular, rather than using inductive learning on a set of goals that span the space of possible goals, we also performed experiments where C4.5 was trained on a random sample of goals selected from the same space as the testing data. This was done using ten trials of four-fold cross-validation (see Appendix A) on a set of forty random goals. Each such trial involves randomly dividing the data into four sets of size ten, using three of the sets for training data and the remaining one as testing. This is repeated four times, using each ten-element set once for testing, and this process was repeated ten times with different random partitionings of the data. Table 2.4 reports the results of these experiments.

Table 2.3: Comparison of prototype-selection methods when trained on a set of training examples that spans the goal space, using the simplified VPP.

Method	Error Rate	Course-Time Increase (sec)
Best Init Eval	12%	26
Inductive Learning	37%	57
Closest Goal	40%	76
Most Frequent Class	45%	175
Random Guessing	75%	257

Table 2.4: Comparison of prototype-selection methods when trained and tested on random goals, using cross-validation and the simplified VPP.

Method	Error Rate	Course-Time Increase (sec)
Best Init Eval	12%	26
Inductive Learning	30%	35
Closest Goal	40%	76
Most Frequent Class	45%	175
Random Guessing	75%	257

Table 2.5: Comparison of prototype-selection methods when trained on a set of goals that span the space, using the simplified VPP, and a "bad" prototype in the database.

Method	Error Rate	Course-Time Increase (sec)
Best Init Eval	10%	80
Inductive Learning	30%	82
Closest Goal	32%	89
Most Frequent Class	45%	171
Random Guessing	75%	348

Table 2.6: Comparison of prototype-selection methods when trained and tested on a set of random goals, using cross-validation, the simplified VPP, and a "bad" prototype in the database.

Method	Error Rate	Course-Time Increase (sec)
Inductive Learning	19%	38
Best Init Eval	10%	80
Closest Goal	32%	89
Most Frequent Class	45%	171
Random Guessing	75%	348

Consistent with our first new hypothesis, the closest-goal and best-initial-evaluation methods both did much better in both cases with the simplified VPP than they did with AHVPP, while C4.5 did about the same as it did before. We believe that because the simplified VPP is much smoother than AHVPP, the hillclimber is much less likely to get stuck, so that the distance in goal space or the difference in initial evaluation becomes much more relevant when choosing a prototype. In fact, the improvement in the best-initial-evaluation method was so great that it significantly outperformed the inductive method.

We performed another set of experiments to test our second new hypothesis of why the closest-goal and smallest-initial-evaluation method performed so poorly using AHVPP, namely that they were unable to avoid the "bad" prototype in the database. We repeated our preceding experiments using the simplified VPP, except that we intentionally put a "bad" prototype into the database. To generate a bad prototype, we started with the Stars and Stripes '87, and added a random number between -0.2 and +0.2 to each of the operator parameters. We then randomly chose one of the four prototypes in the database to replace with the bad prototype (but we left the class label the same). The results of repeating the experiments with the bad prototype in the database are presented in Table 2.5 for training on goals that span the space, and Table 2.6 for training on random goals.

Consistent with our second new hypothesis, C4.5's ability to avoid the "bad" prototype improved its performance relative to the other methods. When trained on the spanning goals, C4.5 performed only slightly worse than the smallest-initial-evaluation method. When trained on the random goals, C4.5 performed markedly better than any other method as measured by average course-time increase, although the smallest-initial-evaluation method had a lower error rate. This apparent anomaly can be explained as follows: The "bad" prototype was very bad, so that choosing it even a few times resulted in large increases in average course time. C4.5 never chose the bad prototype. The best-initial-evaluation method occasionally chose the bad prototype, so that even though it chose the best prototype more frequently than C4.5, the few times when it chose the bad prototype worsened its average course-time increase.

```
distance > 8982.46 : infeasible
distance <= 8982.46 :
|   distance <= 6384.94 : feasible
|   distance > 6384.94 :
|   |   overland <= 23.6023 : feasible
|   |   overland > 23.6023 : infeasible
```

Figure 2.2: Learned decision tree for deciding if a mission is feasible.

2.4.2 Airframe domain

A variation on prototype selection is *prototype synthesis*. Instead of selecting a prototype from a database of prototypes, prototype synthesis synthesizes a new prototype by mapping the design goal directly into the design parameters that define a prototype. What is learned is not a set of rules for selecting a prototype, but rather a set of functions that map the design goal into the design parameters. We performed some experiments to test prototype synthesis in the domain of supersonic transport aircraft design. In this domain, the design goal is to minimize take-off mass (a rough estimate of life-cycle cost) for a specified mission. We defined the following space of missions:

```
distance between 1000 and 10,000 miles
percentage over land between 0 and 100%
mach number over land of 0.85, altitude 40,000 ft
mach number over water between 1.5 and 2.2, altitude 60,000 ft
optional takeoff phase, no climb phase
```

A mission within this space can be represented using three real numbers (distance, percentage over land, and mach number) and one Boolean value (whether the takeoff phase is included). We generated 100 random missions as follows: The distance and mach number were uniformly distributed over their possible ranges. There was a 1/3 probability of having the mission entirely over land, a 1/3 probability of having it entirely over water, and a 1/3 probability of having the percentage over land uniformly distributed between 0 and 100%. There was a 1/2 probability of including the takeoff phase.

Table 2.7: Accuracy of CART in predicting each design parameter in the airframe domain.

Design Parameter	Relative RMSE
engine size	0.65
wing area	0.59
wing aspect ratio	0.06
fuselage taper length	0.07
effective structural thickness over chord	0.08
wing sweep over design mach angle	0.08
wing taper ratio	0.21
fuel annulus width	1.02

In order to generate training data to test our hypotheses in the airframe domain, we performed a 10-point random multistart CFSQP optimization for each of the 100 random missions. We found that for many of these missions, CFSQP was unable to find a feasible design in any of the ten runs — that is, it was unable to design a plane that could fly the mission. It occurred to us that it would be valuable if we could predict in advance whether a given mission was achievable, so that we could avoid attempting to synthesize prototypes for infeasible missions. We hypothesized that C4.5 would be able to make this prediction, and that it would be able to do so with greater accuracy than MFC.

We trained C4.5 on a set of training examples showing whether each of our 100 missions was feasible. It produced the decision tree in Figure 2.2. This decision tree shows that missions are infeasible if they are very long, or if they are moderately long and have a significant portion over land. Further analysis revealed that building a plane to fly such a mission would require an engine larger than the largest engine that we allowed. Our upper bound on engine size can be considered to be representative of the largest commercially available engine.

Tenfold cross validation showed that C4.5 has a 4% error rate on this learning task, compared with 50% for random guessing and 24% for most frequent class, confirming our hypothesis. The decision tree of figure 2.2 can be used to predict, without doing any simulation or optimization, whether a new proposed mission is feasible.

Table 2.8: Comparison of prototype-synthesis methods.

Method	Success	Cost (number of simulations)
CART	13/16	7394
mean	14/16	11963
1 random	8/16	16893
2 random	13/16	33883
3 random	14/16	47395

In order to map the new mission into the numerical design parameters that define a prototype, we need to use *continuous-class induction* (which is also known as regression). We used CART (Classification And Regression Trees), which builds a "regression tree" that has a numerical constant at each leaf [Breiman 1984]. We trained CART on the 100 randomly generated training goals as follows: For each design parameter, we gave CART a set of training data, where each item in the training data included the goal and the optimal value of the design parameter. CART thus generated a set of trees to map the design goal into a set of design parameters that we hope will be near the optimal values for that goal. Table 2.7 shows the root mean squared error in CART's prediction of each design parameter, relative to the error of "constant regression," which always uses the mean of the training data. A value less than one in this table indicates that CART's prediction was more accurate than that of constant regression. Our hypothesis that these relative errors would be low was confirmed for all of the parameters except fuel annulus width.

We performed a set of experiments to test our second hypothesis — that using these trees to do prototype synthesis would produce better optimization performance than using the mean prototype or a random prototype. We used 25 randomly generated testing goals. Table 2.8 compares using the prototypes synthesized by CART with using a 1-, 2-, or 3-point random multistart, or always using the prototype which is the mean of all the optimized prototypes in the training data. Of the 25 randomly generated test goals, 16 were feasible. The "success" column shows the number of optimizations

Table 2.9: Performance of one random probe, averaged over ten trials.

Measure	Success	Cost (number of simulations)
Mean	8.8/16	15062
Standard Deviation	1.9/16	3102

that came within 1% of the point that we believe to be the global optimum.[3] Some of the failures occurred because the learning method produced an unevaluable prototype that could not be simulated, and therefore could not be optimized. Other failures occurred because the optimizer, when started from the synthesized point, failed to get within 1% of the apparent global optimum. The "cost" column shows the total number of simulations used in the 16 optimizations. Using the mean prototype instead of a single random prototype resulted in much greater success, at 33% lower cost. Using CART produced a success rate about the same as using the mean prototype, with an additional 38% cost reduction, supporting our hypothesis. Using a 2-point random multistart produced the same success rate as using CART, but it required more than four times as many simulations, further supporting our hypothesis.

To test the significance of the result that CART performed better than one random probe, we repeated the one-random-probe test ten times, with ten different seeds to the random number generator. The mean and standard deviation of the success rate and cost are shown in Table 2.9. CART's success rate was more than two standard deviations higher than that of one random probe, and its cost was more than two standard deviations lower than that of one random probe — further supporting our hypothesis.

[3]Because CFSQP failed to find a feasible point in some of these optimizations, it was not possible to compute the average design quality.

2.5 Analysis

2.5.1 The cost of learning

One important question to answer is whether the inductive prototype-selection method is worth the considerable "off-line" expense of collecting training data — every training example requires one design run for each design in the prototype library. An alternative, possibly cheaper method would be to take an "on-line" approach: for each new design problem optimize starting from every prototype in the database, and then use whichever of the resulting designs is the best.

If the quality of the final design is extremely important and there is ample CPU time available, this "exhaustive" method is the one to use (out of the methods listed in Table 2.2). On the other hand, if limiting CPU time is important, our inductive learning method becomes cost effective when the computational expense of learning can be amortized over a sufficiently large number of new design goals. More specifically, the inductive prototype-selection method is less expensive than the exhaustive method whenever the number of hillclimbing runs taken by the inductive approach is less than the number of runs taken by the exhaustive approach, i.e., $TP + G < PG$ or

$$G > \frac{T}{1 - \frac{1}{P}}$$

where T is the number of training examples, P is the number of prototypes in the database, and G is the number of new goals for which prototypes need to be selected. (When using the inductive prototype-selection method, TP is the cost of generating the training data, and G is the cost of performing optimizations for the new goals. When using the exhaustive method, each prototype in the database must be optimized for each new goal, at a cost of PG.) In all of the experiments that we performed, there were four prototypes and 30 training examples, so our inductive approach will be less expensive than the exhaustive approach as long as at least 40 out of the more than 150,000 remaining design goals must be attempted.

When doing prototype synthesis rather than prototype selection, it is not necessary to collect training data in which each prototype in a database is used as a starting

point of an optimization for each of a collection of goals. (Prototype synthesis takes as training data the optimal design parameters for each goal, rather than the selection of the best prototype from a database for each goal.) Instead, any optimizations that have been previously done (within the same goal space) can be used as training data. Hopefully, such data will already exist in a design library, so additional optimizations will not be needed to generate training data.

2.5.2 Future work

This chapter presents an initial exploration of our inductive approach to the prototype-selection problem, and there are a number of directions for future work. First, the experiments reported here explore the sensitivity of our prototype-selection method to the nature of the design library, specifically with respect to the quality of the stored designs. It would be helpful to more fully explore the sensitivity of our approach to the design library, for example by studying how our approach scales up as the library size increases.

Further, C4.5 is simply one out of many available inductive-learning methods, and it would be useful to compare our results to those obtained with other learning algorithms (such as neural networks) to see how dependent our results are on the particular inductive method. We also plan to explore how our results can be improved even further through the development of inductive-learning methods that can use background knowledge of the domain (e.g., racing-yacht design) while learning prototype-selection rules. Finally, learning methods operating on more expressive representations, such as inductive logic programming systems like FOIL [Quinlan 1990], may enable going beyond the simple representation of goals used here and handling more complicated goals, such as those involving multiple disciplines.

2.5.3 Conclusion

We have described and demonstrated the utility of two methods of choosing an initial prototype for optimization. Both methods allow better designs to be produced rapidly. Prototype selection is especially appropriate in domains such as the yacht domain in

which there is a database of previous designs available, and the available simulators are noisy. Prototype synthesis is especially appropriate in domains such as the airframe domain in which finding a feasible design is difficult.

Chapter 3

Reformulation Selection

3.1 The Problem

In a simulation-based automated engineering design system that uses numerical optimization, the decision on how to formulate the search space can substantially affect the performance of the optimizer in two ways. First, using a lower-dimensional formulation of the search space makes optimization faster, since each gradient computation requires fewer runs of the simulator, and the distance in design space from the starting point to the optimum is smaller. Second, different formulations of the search space can result in different degrees of "smoothness" or "pathology" of the search space, which can impact not only the speed of the optimizer, but also the ability of the optimizer to get to the optimum, and therefore the quality of the resulting designs.

3.2 The Technique

We present a method of reformulation called "constraint incorporation," which reduces the dimensionality of the search space and increases its smoothness by incorporating constraints into the search space.

Traditionally, numerical optimization has dealt with explicit, "hard" constraints. The optimizer assumes that these constraints can never be violated. A hard constraint can be expressed as

$$f(x_1, x_2, \ldots, x_n) \leq k$$

(Here $x_1, x_2, \ldots, x_n$ are the *design parameters* that represent the design.) The constraint is said to be *inactive* if $f(x_1, x_2, \ldots, x_n) < k$, *active* if $f(x_1, x_2, \ldots, x_n) = k$, and *violated* if $f(x_1, x_2, \ldots, x_n) > k$. Hard constraints can result from the laws of physics, for

example.

Another type of constraint is the "soft" constraint, for which there is some sort of known penalty for violating the constraint. A soft constraint can be expressed as

$$\text{if } f(x_1, x_2, \ldots, x_n) > k \text{ then apply penalty } P(x_1, x_2, \ldots, x_n)$$

These usually arise from human-written laws, such as regulations specifying a monetary penalty for exceeding a certain noise level. In either case, if it is known that the constraint will be active at the optimal design point, and the constraint function f is invertible, then the constraint can be *incorporated* into the search space by using the inverse of f to eliminate one of the design parameters. This incorporation is done by making the inequality constraint into an equality constraint, and then solving for one of the design parameters in terms of the other design parameters. [Papalambros and Wilde 1988] describe how monotonicity knowledge can be used to determine that certain constraints will be active at the optimum. Incorporating these constraints produces a new search space with lower dimensionality, since the incorporation eliminates a design parameter, and greater smoothness, since the incorporation eliminates the "ridge" (or nonsmoothness) in the search space caused by the "if" statement in the constraint. If there are n constraints that can be incorporated in this way, then there are 2^n possible reformulations that can be produced by incorporating different subsets of constraints.

Optimization can be done for a variety of *design goals*. A design goal consists of *environment parameters*, which are inputs to the simulator other than the design parameters (and which typically describe the environment in which the designed artifact will operate), and the thresholds on the various constraints.

Constraint activity depends on the goal (some constraints are active at the optimum for only some design goals), for two reasons: First, the constraint thresholds are part of the design goal. Second, different design goals will result in different optimal values of the design parameters on which the constraint functions depend.

Because constraint activity depends on the goal, different reformulated search spaces are appropriate for different design goals. We describe a way in which inductive learning can be used to map the design goal into the appropriate reformulation.

3.2.1 Learning Reformulation Rules

To use inductive learning to form reformulation-selection rules, we take as training data a collection of design goals, each labeled with the set of constraints that are active at the optimal design point. We run the inductive learner once for each constraint, producing for each constraint a set of rules that can be used to predict whether the constraint will be active for new design goals.

The training data can be generated in an automated fashion. For example, one can choose a set of training goals and perform an optimization for each goal. One can then evaluate each constraint function for each optimal design, and then construct a table that records which constraints were active (within a threshold) for each training goal. This table can be used by the inductive-learning algorithm to generate a set of rules for each constraint, mapping the space of all possible goals into a prediction of whether or not that constraint will be active at the optimal design point for that goal. If learning is successful, these mappings extrapolate from the training data and can be used successfully in future design sessions to map a new goal into an appropriate reformulation.

The specific inductive-learning system used in this work is C4.5 [Quinlan 1993] (release 6.0).

3.3 Results

Our hypotheses were the following:

1. The best choice of reformulation depends on the goal, so the C4.5 learning method would have a lower error rate than the most frequent class (MFC) learning method.

2. Using the reformulations chosen by C4.5 would produce better optimization performance (better designs and/or lower CPU cost) than using the reformulations chosen by MFC, or by random guessing, or not using any reformulation.

3. C4.5's performance would improve as the size of the training set increases.

We were interested in determining not only whether these hypotheses were true, but also the magnitude of the differences in performance among methods, and the rate at which C4.5's performance changes as the size of the training set changes.

We ran experiments in two domains — the yacht domain and the airframe domain — with the expectation that our hypotheses would hold in both domains.

3.3.1 Yacht domain

We performed some experiments to test the performance of reformulation selection in the yacht domain. In the experiments described in this subsection, we ran CFSQP with *course-time* as the objective function, and with one explicit, nonlinear, "hard" constraint. This constraint specifies that the mass of the yacht, before adding any ballast, must be less than or equal to the mass of the water that it displaces. (In other words, the boat must not sink.)

Yachts entered in the 1987 America's Cup race had to satisfy a hard constraint known as the 12-Meter Rule [IYRU 1985]. Instead of using this rule as an explicit constraint, we incorporated it into the search space. (How we incorporated it is described below.) The basic formula in the rule is:

$$\frac{length - freeboard + \sqrt{sailarea}}{2.37} \leq 12m$$

In addition to the basic formula, the rule contains several soft constraints, along with associated penalties for violating these constraints. These soft constraints are:

- draft constraint
- beam constraint
- displacement constraint
- winglet span constraint

For example, the *beam constraint* states

> if $beam < 3.6m$, then add four times the difference to $length$

While constructing the simulator, we used a reasoning process similar to that described in [Papalambros and Wilde 1988] to determine that the constraint described by the basic formula of the 12-Meter Rule, above, will always be active, since the objective function being minimized, *course-time*, is monotonically non-increasing in *sail-area*,[1] and the left-hand-side of the constraint is monotonically increasing in *sail-area*. We therefore *incorporated* this constraint into the simulator by solving for *sail-area* in terms of the other design parameters. So, for example, when the optimizer makes *length* bigger, *sail-area* is automatically made smaller. In addition, because we also implemented the soft constraints as penalty functions, reducing *beam* beyond $3.6m$ causes the quantity *length* in the formula to increase, which causes *sail-area* to decrease.[2]

Because the beam constraint contains an *if* statement, this incorporation causes a nonsmoothness in *course-time* as a function of *beam*. That is, there is a discontinuity in the first derivative of *course-time* with respect to *beam*. Figure 3.1 illustrates this nonsmoothness by showing the cross-section of the search space corresponding to the *beam* design parameter.[3] This nonsmoothness can cause a gradient-based optimizer such as CFSQP to get stuck, and to fail to get to the optimum.

For many design goals, the optimal design is right on the constraint boundary. The optimal beam is often $3.6m$. If we expect the optimal beam to be $3.6m$, then we can incorporate the beam constraint into the operators. In the case of the beam constraint, this incorporation is trivial — we simply set *beam* to $3.6m$ and leave it there. For other constraints, the incorporation is more complicated. For example, there is a constraint that specifies a penalty if *displacement* does not vary with a certain cubic polynomial in *length*. *Displacement* is not a design parameter; rather, it is a quantity computed from all of the design parameters. In order to incorporate the displacement constraint,

---

[1] The simulator assumes that there is perfect reefing, so additional sail area can never hurt the yacht's performance.

[2] Because we incorporated the 12-Meter rule into the simulator, we did not need to use it as an explicit constraint.

[3] Although this figure shows only a "snapshot" of the search space for specific values of the other design parameters, we believe that the trend shown in the figure is generally applicable.

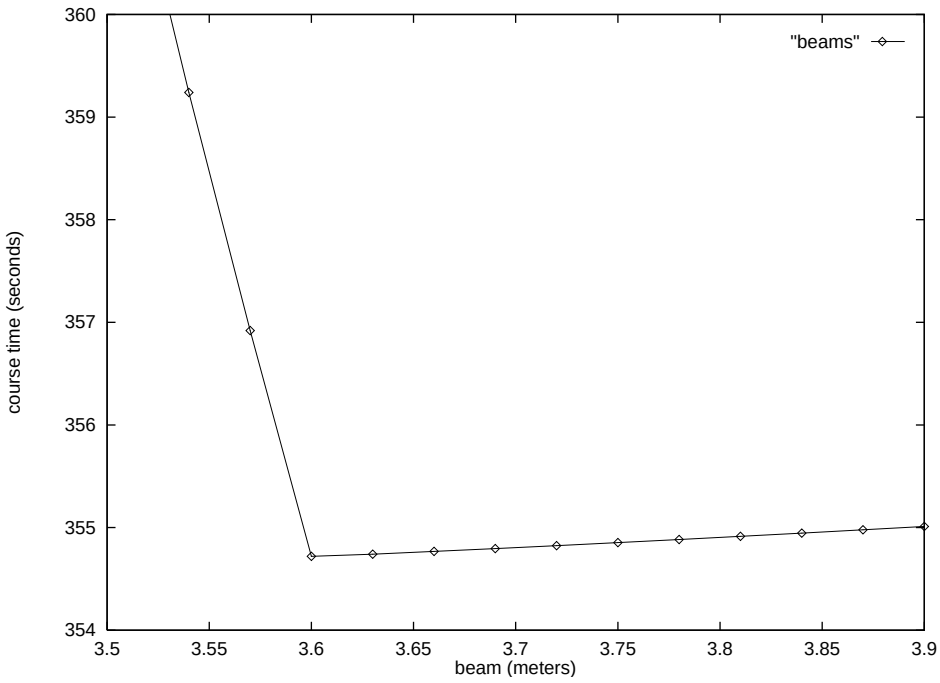

Figure 3.1: The nonsmoothness in the search space caused by the beam constraint.

we used Maple [Char *et al.* 1992], a symbolic algebra package, to invert the displacement formula, and created a new set of operators that vary certain parameters while maintaining *displacement* at the minimum displacement allowed by the constraint. For still-more-complicated constraints, it might not be possible to invert the constraint function using Maple; it might therefore be necessary for the operators to contain numerical solvers that find the right values of the incorporated design parameters so as to put the design on the constraint boundary.[4]

We created operators to incorporate all four of the above-listed 12-Meter Rule constraints: the draft constraint, the beam constraint, the displacement constraint, and the winglet constraint. Using these operators, we are able to either incorporate or not incorporate each of these four constraints independently. We thus defined a set of sixteen (2^4) possible reformulations of the search space. From our initial experiments with these operators, we determined empirically that incorporating the draft constraint substantially improved the reliability and speed of optimization for any design goal. We

[4]Operators containing numerical solvers would probably be more computationally expensive than operators containing the algebraic solutions of the constraint functions, so the CPU time savings from reformulation would probably be smaller.

therefore decided to always incorporate the draft constraint, leaving us with a space of eight possible reformulations that we used in the experiments described below.

Having defined eight reformulations of the search space, we used inductive learning to decide, based on the design goal, which reformulation to use. As training data, we used 100 previous optimizations.[5] For each previous optimization, we evaluated each 12-Meter Rule constraint function at the optimum, and determined if the constraint was active (within a tolerance). Each of these previous optimizations had as its design goal minimizing course time for a single-leg racecourse, which can be represented using two numbers: the wind speed, and the heading (the angle between the yacht's direction and the wind direction). The design goal can therefore be represented using these two numbers. We ran the inductive learner once for each of the three constraints. Each time, the inductive learner was provided with a set of triples: the wind speed, the heading, and a ternary value indicating whether the constraint was inactive, active, or violated. One of the constraints was violated at the optimum in 10 of these optimizations. Figure 3.2 gives an example of a decision tree output by C4.5. This decision tree predicts whether the displacement constraint will be active at the optimum, based on the design goal. By running a new design goal down three decision trees, one for each of the three constraints that can be incorporated, the system can make predictions of whether each constraint will be active at the optimum. These three yes/no predictions directly map into one of the eight (2^3) reformulations of the search space.

We used C4.5 to perform tenfold cross-validation (see Appendix A), and obtained the error rates shown in Table 3.1. Here we compare the error rates of C4.5 with and without pruning, and of C4.5rules, a variant of C4.5 that extracts rules from the trees, with the expected error rate of random guessing (which is two-thirds since there are three classes from which to guess), and the error rate of the Most Frequent Class (MFC) learning method. MFC always chooses the class that occurs most frequently in the training data. In this case, that means that it always chooses the same reformulation, namely the one that is most often the best reformulation in the training data.

[5]The optimizer failed for one of these goals, so we used the remaining 99 goals as training data in the results that follow.

```
heading <= 109 :
|   windspeed <= 6.3 : active
|   windspeed > 6.3 :
|   |   windspeed > 8.2 : violated
|   |   windspeed <= 8.2 :
|   |   |   heading <= 65 : violated
|   |   |   heading > 65 : active
heading > 109 :
|   windspeed > 11.5 : active
|   windspeed <= 11.5 :
|   |   heading <= 135 : active
|   |   heading > 135 : inactive
```

Figure 3.2: Learned decision tree for the displacement constraint.

Table 3.1: Cross-validated error rates for selecting whether to incorporate each constraint.

method	Beam	Displacement	Winglet
C4.5 w/ pruning	11.1%	15.1%	7.0%
C4.5 w/o pruning	11.1%	15.1%	10.0%
C4.5rules	11.1%	15.1%	10.0%
MFC	33.3%	53.5%	13.1%
Random	66.7%	66.7%	66.7%

As Table 3.1 shows, C4.5 with pruning performed slightly better than C4.5 without pruning or C4.5rules (and so in our further experiments reported below we use only C4.5 with pruning), and all three substantially outperformed MFC, which in turn substantially outperformed random guessing. These results supported our first hypothesis.

These results are for error rates, the proportion of cases where learning makes an incorrect guess. A more important question in this domain is how learning affects the overall problem-solving task, namely how it improves the speed and reliability of the design optimization process. Does learning make the design process faster or slower? Are the resulting designs better or worse? To measure these effects, we performed optimizations for 25 new randomly generated goals using the reformulations suggested by each learning method. Table 3.2 shows the effect that C4.5 (with pruning) and MFC had on the average course time (the quality of the design), and average number of evaluations (the speed of the optimization), as compared with the "old way" of doing

Table 3.2: Effect of using reformulations chosen by learner on optimization performance. A positive quality change indicates an improvement in quality (which is a reduction in course time).

method	quality change	CPU time change
omniscient	+0.085%	-36%
exhaustive	+0.085%	+384%
C4.5	+0.080%	-35%
MFC	+0.029%	-32%
none	0	0
random	-0.276%	-40%
all	-0.599%	-74%

optimization without incorporating any of the three constraints into the operators. The first column in the table shows the percentage difference between the optimized course-time produced without reformulation, and the optimized course time produced with the specified reformulation. The second column shows the percentage difference between the cost of performing the optimization without reformulation, and the cost of performing it with the specified reformulation.

We also include in this table the performance of several other methods. A hypothetical "omniscient" problem solver always magically guesses the best possible choice (the one with that results in the best course time). [6] No learning method will enable results superior to this. The "exhaustive" optimization method performs eight optimizations for each goal, using all eight possible reformulations, and then chooses the best resulting design. Incorporating "all" constraints all the time results in the fastest possible optimization within this set of reformulations (at the cost of quality loss).

C4.5 produced a significant speedup in optimization, with no quality loss. In fact, it produced a small quality increase. (This quality increase suggests that without any reformulation, the optimizer gets "stuck" on the "ridges" that the constraints cause the search space to have, and therefore sometimes fails to get the optimum.) MFC produced a slightly smaller speedup and a slightly smaller quality improvement.

[6] We simulated the omniscient learner by performing optimizations using all eight reformulations for each goal (as in the "exhaustive" method), and then ignoring the cost of the seven optimizations that turned out not to be best.

The difference between C4.5 and MFC in quality change was, however, statistically significant at the 99% confidence level, according to the paired t-test. Both learning methods performed substantially better than random guessing. C4.5 performed almost as well as the hypothetical omniscient learner, which means it performed almost as well as any learner could possibly do. [7] These results supported our second hypothesis.

Incorporating all of the constraints all of the time resulted in a very large speedup, with a modest quality loss. This method may be appropriate if one wants a quick and approximate optimization. It might, for example, be used in the early stages of design when the engineer wants to get a feel for the search space by asking "what-if" questions.

One question that these results raise is how training-data quantity affects performance. If one does not have results from a large number of previous optimizations available, then one can either run some extra optimizations to generate training data (which is expensive), or do the learning with less training data (which is likely to produce higher error rates and lower optimization performance). We ran some experiments to determine how C4.5's performance varies with training-set size, and how its performance compares with that of MFC for various training-set sizes. We applied our learning approach to datasets of varying sizes, with the error rates shown in Figure 3.3. For each training-set size in the figure, we randomly chose 10 different subsets of our training data of that size, and performed 10-fold cross-validation on each subset. The figure shows the averages. The three symbols at the right side of the figure show MFC's performance on the full training set. C4.5 outperformed MFC for every training-set size, but C4.5's error rate on smaller training sets was significantly larger than C4.5's error rate for larger training sets (with performance reaching an asymptote for training sets of about 60 cases or more). These results supported our third hypothesis.

---

[7] Interestingly, according to the t-test, the difference between C4.5 and the omniscient method was not statistically significant, but this just illustrates a limitation of the t-test, since we know that the omniscient method really is better, on average, than C4.5.

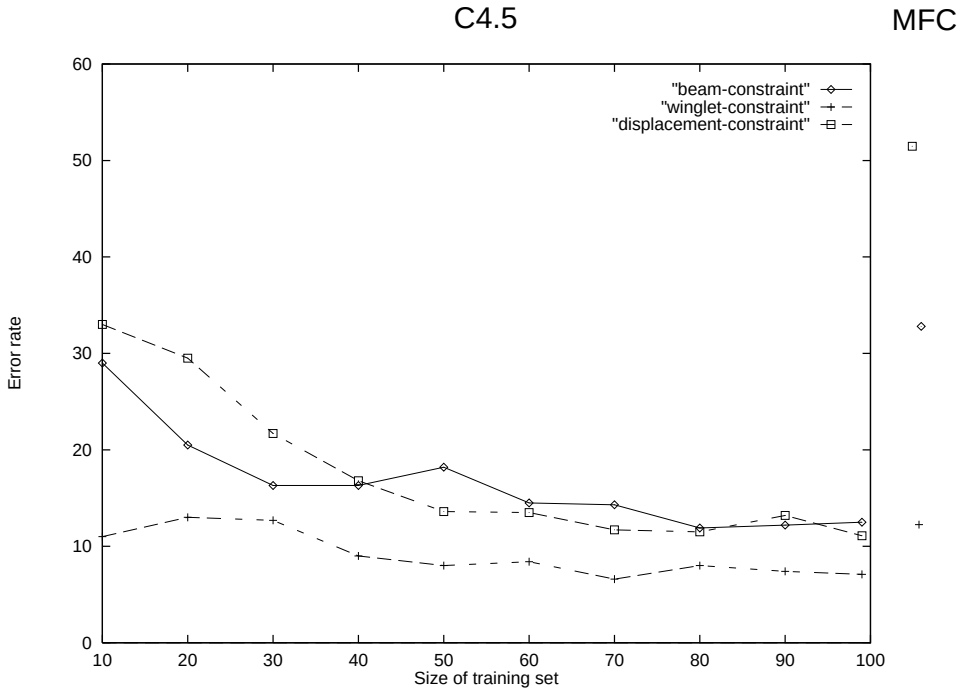

Figure 3.3: Effect of training set size on learner performance.

3.3.2 Airframe domain

We believe that our reformulation selection technique is applicable to a broad range of design optimization problems. To test the domain-independence of the reformulation selection technique, we performed additional experiments in the airframe domain, and compared the impact on optimization performance of C4.5 with that of MFC.

In the airframe domain, there are eight design parameters, each of which can have an upper and lower bound. The optimal design sometimes lies at the bounds of some of these parameters, depending on the mission.

We used the space of missions described in section Section 2.4.2. This mission space describes a mission using three real-valued attributes and one Boolean attribute. We used the C4.5 decision tree, described in the same section, to predict which missions are feasible. As training data, we used the same 100 10-point random multistart CFSQP optimizations, 76 of which are feasible (that are also described in section Section 2.4.2).

We used the 76 feasible missions to train C4.5 for reformulation selection. Of the eight design parameters, four were never at their upper or lower bounds at the apparent

```
overland <= 95.0872 : zero (54.0)
overland > 95.0872 :
|   takeoff = no: zero (12.0/1.0)
|   takeoff = yes: nonzero (10.0)
```

Figure 3.4: Learned decision tree for predicting if the taper ratio will be at its lower bound of zero.

Table 3.3: Cross-validated error rates for selecting whether to incorporate each lower bound, in the airframe domain.

design parameter	C4.5	MFC	Random
wing taper ratio	2.7%	14.5%	50.0%
wing sweep	2.5%	27.6%	50.0%
fuselage taper length	3.9%	22.4%	50.0%
fuel annulus width	13.6%	5.3%	50.0%

optima for any of the 76 missions. The other four had optima at their lower bounds for some missions. We trained C4.5 to predict whether these four design parameters would be at their lower bounds, depending on the mission. C4.5 produced a separate decision tree for each of these four design variables. For example, Figure 3.4 shows the decision tree for wing taper ratio. This decision tree says that wing taper ratio will be at its lower bound of zero, unless the mission includes a takeoff phase and is almost entirely over land. The four decision trees can used to select among 16 (2^4) possible reformulations.

Table 3.3 compares the cross-validated error rates of C4.5 with those of most frequent class and random guessing for each of the four design parameters. For the first three parameters, C4.5 did much better than most frequent class, supporting our first hypothesis. For the fourth parameter, fuel annulus width, C4.5 did much worse than most frequent class, violating our expectations. In this case, only 4 of the 76 training examples were positive examples. We suspect C4.5 would need more training examples to be more accurate. Interestingly, in our prototype synthesis experiments, CART had difficulty predicting the optimal value of fuel annulus width (see Chapter 2).

To determine the impact of the reformulations selected by the various methods on

optimization performance (and to test our second hypothesis), we randomly generated 25 new missions. Table 3.4 compares the performance of the various methods of reformulation selection when doing optimizations for these new missions. For the methods that used C4.5, we used the decision tree of Figure 2.2 to predict whether each new mission was feasible, and only performed optimizations for those missions that were predicted to be feasible. For the other methods, we performed optimizations for all 25 missions. Each optimization was a 10-point multistart. The "success" column indicates for how many of the missions the specified method came within 1% in takeoff mass of the best design found.[8] The "time change" column shows the change in total number of simulations used in all of the optimizations performed, compared with not incorporating any constraints.

Because cross-validation showed that C4.5 under-performs MFC for predicting whether to incorporate fuel annulus width, we did not use C4.5 to decide whether to incorporate this parameter. We used C4.5 to decide whether to incorporate the other three parameters, and used two different methods to decide whether to incorporate fuel annulus width. The first method used MFC to decide whether to incorporate the fuel annulus width, which resulted in always incorporating it. The results of this method are labeled "C4.5/MFC" in Table 3.4. For the second method, we decided to play it safe and never incorporate fuel annulus width, since cross validation suggests that we are not able to accurately predict when this parameter will be at its bound. The results of this method are labeled "C4.5/none" in Table 3.4. We compare these methods with most frequent class, and with the exhaustive method that does optimizations for all 16 (2^4) reformulations, and the omniscient method which magically guesses the best reformulation.

The first interesting thing to note about Table 3.4 is that there is one mission for which CFSQP failed to reach the optimum without reformulation. The only way to reach the optimum for this mission is the use the "omniscient" method (which does not exist), or the "exhaustive" method (which is extremely expensive). The next thing

[8]Because CFSQP failed to find a feasible point in some of these optimizations, it was not possible to compute the average design quality.

Table 3.4: Effect of using reformulations chosen by learner on optimization performance, in airframe domain.

method	success	time change
omniscient	16	-51%
exhaustive	16	+1206%
C4.5/none	15	-36%
none	15	0
C4.5/MFC	13	-57%
MFC	13	-21%
all	3	-55%

to note is that using the reformulations selected by C4.5 for the first three parameters, while not incorporating fuel annulus width ("C4.5/none"), reduces cost by 36% compared with not incorporating any constraints ("none"), without any loss of quality. Using C4.5 for the first three parameters, and MFC for fuel annulus width (C4.5/MFC), causes CFSQP to fail to find the optimum in two additional cases. Using MFC for all parameters causes the same number of missed optima, at a higher cost. And incorporating all of the parameter bounds all of the time results in CFSQP almost always failing to get to the optimum.

The airframe domain results are surprisingly similar to the yacht domain results. In the yacht domain, using the reformulations selected by C4.5 reduced the cost of optimization by 35% (Table 3.2), while in the airframe domain the speedup was 36%. In the yacht domain, using C4.5 also resulted in a small quality increase, while in the airframe domain, quality remained the same. This may be because the yacht domain reformulations increase the smoothness of the search space (by eliminating the 12m-rule penalties), while the airframe domain reformulations do not. Another interesting thing to note is that while the difference between MFC and C4.5 was small (but statistically significant) in the yacht domain, it was much larger in the airframe domain. The fact that C4.5 outperformed MFC in both domains supports our second hypothesis.

3.4 Analysis

3.4.1 Future Work

This chapter has described on-going work, and there are thus a number of directions for future work. These fall into two groups: extending this work to more difficult design tasks, and improving results by using other learning methods.

Other Design Tasks

The yacht domain results presented here apply to a constrained class of yacht-design goals, those comprised of a single leg. One question is how this approach can be applied to courses comprised of varying numbers of legs. We believe that we could get reasonable optimization performance by using the trees learned from single-leg courses to perform multi-leg optimization in the following way: If a constraint should be incorporated for every leg of the racecourse, then incorporate it for the full, multi-leg course. We need to test how well optimization performs when handling racecourses in this manner. We could also attempt to learn directly for multi-leg racecourses. Doing so would raise an interesting machine-learning question, since describing a multi-leg racecourse requires a variable number of attributes, and thus traditional learners such as C4.5 do not directly apply.

In the results presented here, we assume that the only change between the previous design sessions and the current design session is the design goal (expressed as a (*wind speed*, *heading*) pair in the yacht domain). An interesting question is what would happen if in addition to changing the goal, we also changed the constraints, or the simulator, or the form of the goal. We would need to find a way to encode as a set of attributes for the learner whatever had changed.

We believe that the results presented here will easily generalize to situations in which there are more than sixteen reformulations. We used the results from the same set of 100 optimizations to perform three separate learning tasks (for three constraints), and then combined the rules generated by these three learning sessions to select one of the eight reformulations. As the number of reformulations grows, the number of

constraints, and therefore the amount of CPU time needed for the learning, will grow logarithmically with the number of reformulations. The CPU time needed for learning is currently insignificant compared with the CPU time needed for the subsequent optimizations. We expect that as the number of reformulations grows, the number of training examples needed will remain constant (since the same training examples are used for each constraint), and the amount of CPU time needed for learning will remain insignificant. We plan to test this hypothesis by using other constraints within the yacht design domain, such as the "boat doesn't sink" constraint.

The learning approach could also be used to decide when to reformulate soft constraints as hard constraints. If it were known with a high degree of confidence that a certain soft constraint will not be violated at the optimum for certain goals, then this soft constraint could be converted into a hard constraint for those goals, which would eliminate a ridge from the search space and thereby make optimization more robust (although it would not reduce the dimensionality of the search space). For example, in the training data that we collected, the *beam constraint* was never violated, so it might be replaced safely with a hard constraint.

Other more-difficult problems might involve a less-smooth search space, a higher-dimensional goal space, or a less reliable optimizer. Such problems may arise when we test this method in still other domains.

Other Learning Methods

We found that C4.5 performed nearly as well as a hypothetical "omniscient" learner, for the fairly simple design problems that we used in our experiments. Other learning methods, however, might prove useful when attacking some of the harder problems described in the previous subsection. For example, it would be interesting to see how well neural networks, nearest-neighbor methods, or statistical regression would perform. In particular, C4.5, like most decision-tree learners, uses linear, axis-parallel cuts in its decision trees. However, Figure 3.5 shows how the activity of the beam constraint varies over the goal space in the training data we used — the space is clearly divided into two regions (except for one point which we believe is noise). The border between

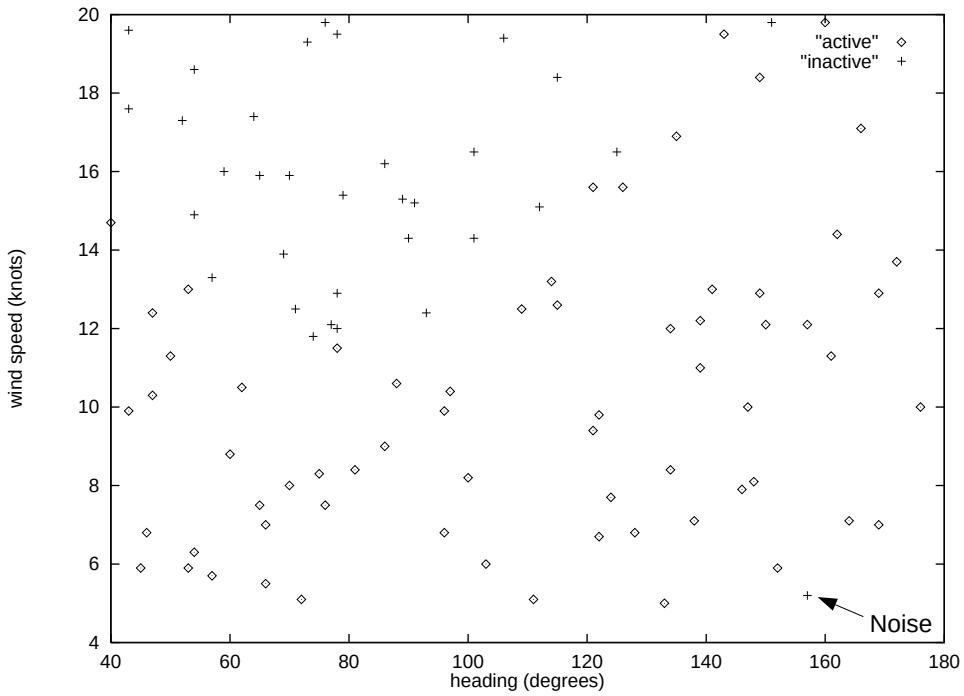

Figure 3.5: Activity of the beam constraint over the goal space.

these regions does not appear to be axis parallel, and appears to be nonlinear. This suggests that better performance might be achieved using an "oblique" decision tree learner, such as OC1 [Murthy *et al.* 1994], or by attempting to learn nonlinear region boundaries.

As would be expected, even though our yacht-domain results with C4.5 were nearly optimal for 100 examples, results degrade when given less training data. Although it would be interesting to see if other learning methods would have better small-dataset performance, for any learner we would expect performance to be inferior for small enough datasets. One approach for improving results in such small-dataset cases — as well as in other cases where off-the-shelf learners such as C4.5 may not perform well even if given larger datasets — is to integrate background knowledge into the learning process. One form of background knowledge that is often available, such as in the yacht-design domain, is *modality constraints*. This is knowledge that expresses the modality of the learned class with respect to the attributes. For example, we believe that optimal *beam* is monotonically increasing in wind speed, and monotonically decreasing in heading. We also know that the activity of any constraint of the form

$f(x_1, x_2, \ldots, x_n) \leq k$ must be monotonic in k, so, for example, the activity of a cost constraint must be monotonic in the cost threshold. One open question is how such knowledge could be integrated into learning. One approach would be to use such modality constraints to remove from the training data points that violate the constraints (on the assumption that these points are noise). A second approach is to modify the tree induction algorithm so that it will never construct a tree that violates the constraints. A similar approach was used to constrain decision lists in [Clark and Matwin 1993].

Finally, even after our learning approach is applied, every additional future optimization can serve as an additional training point for the learning. Thus learning methods that can work in an incremental fashion might also prove useful for this task. In addition, it may prove useful to develop methods that select suitable data prior to learning. For example, when there are not enough existing optimizations to achieve adequate learning results, additional optimizations can be performed to generate further training data. Rather than performing these new optimizations for random goals or for a set of goals that span the goal space, one could allow the learner to choose the goals to be used in the new training data. Background knowledge — such as modality constraints — could prove particularly useful in selecting such goals.

3.4.2 Conclusion

We have shown that using the reformulations selected by inductive learning can make design optimization faster, because the reformulation reduces the dimensionality of the search space, and more reliable, because the reformulation can make the search space smoother.

Chapter 4

Model Constraints

4.1 The Problem

During an automated design optimization, each step of the optimization requires evaluating the quality of candidate designs. For complex artifacts (e.g., aircraft, our main example), this evaluation must be done by computational simulation. However, computational simulation is based on a model of the physics of the artifact, and this model will generally make simplifying assumptions in order to be computationally tractable. A simulator based on such a model will often exhibit pathological behavior when applied to designs that violate these assumptions. Most existing computational simulators are intended to be used by human experts, and thus they typically include no explicit representation of their modeling assumptions. Instead, it is assumed that the experts know enough to stay away from portions of the design space that will violate the simulator's assumptions.

For example, a typical assumption for an aircraft simulator might be that the wings won't stall. Stall is a physical phenomenon that occurs when a wing is operated at too high an angle of attack and therefore ceases to generate lift. The physics of stall is understood, and there is in principle no reason not to model it in a simulator. However, a human expert aircraft designer doesn't want to design a plane that stalls during normal operation, so he doesn't need a detailed prediction of stall behavior. The designer is satisfied with an incomplete model as long as he can recognize "impossibly high" lift coefficients and realize that the design he is considering would actually stall and thus should be discarded.

However, if the simulator is invoked by another program such as an automated search procedure rather than by a human expert, it is quite likely that in exploring the

design space, the automated search procedure will examine designs which violate the simulator's assumptions, and for those candidate designs, the evaluation of the design quality computed by the simulator may be meaningless. Furthermore, this meaningless value may appear better than the value for any physically realizable design, thus leading the search procedure to a worthless but apparently very good design.

[Gelsey 1995b] investigates the types of modeling knowledge that are needed so that a simulator can be reliably invoked by another program, and describes algorithms for detecting assumption violations and other problems that might lead to low-quality or unreliable simulation results. In this chapter, we address the question of how information about model assumption violations can be effectively communicated to an automated search procedure so that the search procedure can find candidate designs that don't violate model assumptions.

4.2 The Technique

Strategies for communicating information about model violations to the search procedure include:

The Null Strategy: ignore the model violation — the search procedure uses whatever value happens to be computed by the inapplicable model for the quality of the candidate design.

The Boolean Strategy: when any model violation occurs, always give the search procedure a standard "very bad value" as the quality of the candidate design.

Model Constraints: when a candidate design is evaluated, give the search procedure not only a value for the quality of the candidate design, but also a vector of values for a set of "model constraint" functions which measure how much the various modeling assumptions are satisfied or violated. The search procedure then treats these model constraint functions as set of additional nonlinear inequality constraints.

Model Penalties: same as the model constraints strategy, except that only the value for the quality of the candidate design is returned to the search procedure, and that value is penalized in proportion to the amount by which the various modeling assumptions are violated. The search procedure does not receive the model constraints as explicit constraints.

We will focus primarily on the Boolean strategy and model constraints. The null strategy is unlikely to be useful unless it coincidentally happens to be the same as either the Boolean strategy (if the constraint is never violated) or the model penalties strategy (if the objective function already gets worse when the constraint is violated). The Boolean strategy can be useful — its advantages include:

- easy to implement: as soon as a violation is detected, just return immediately with a standard "very bad" value for the objective function
- it can be used even with unconstrained search methods

The model constraints strategy is more complicated to implement than the Boolean strategy, but our experimental results later in this chapter show that when used with a search method that allows constraints, the performance of the model constraints strategy is considerably better than that of the Boolean strategy.

Most of the experiments in this chapter were performed in the airframe domain. The system computes the following model constraint functions, which are less than or equal to zero if a constraint is satisfied and positive otherwise:

ETUB = <maximum throttle required during mission simulation> − <maximum throttle setting allowed in engine table>. If an impossibly high throttle is required to fly the mission, the simulation will continue using extrapolation, but the value of ETUB will indicate the extent to which the engine model assumptions are violated. Note that this constraint function is negative when the throttle is within the table, and positive when the simulator has to extrapolate outside the table.

ETLB = <minimum throttle setting allowed for engine> − <minimum throttle required during mission simulation>.

A1LB, A1UB, A2LB, A2UB: Table constraints similar to the above engine table constraints — violation of bounds for a two-dimensional table of experimental data on supersonic drag.

WLUB = <maximum wing loading during mission simulation> – <maximum wing loading simulator can validly model>.

FM = <fuel mass that current candidate design requires to complete mission> – <fuel mass that can be stored in available volume for current candidate design>.

STALL = <maximum lift coefficient during mission simulation> – <maximum lift coefficient simulator can validly model>. The simulator assumes wings won't stall, and this constraint function computes how well that assumption is satisfied.

These model constraint functions are continuous and usually smooth with respect to the design parameters as their values change sign, which is very important so that when the system is using the model constraint communication strategy, CFSQP (the numerical optimizer) can follow constraint boundaries if necessary as it searches for an aircraft design which can fly the given mission with minimal takeoff mass. If the system is following the "Boolean" communication strategy, it does not give the values of the model constraint functions to CFSQP: instead, any candidate design for which some model constraint function is positive will be evaluated to have a standard "very large" takeoff mass.

In addition to these model constraints, the system computes the following design constraint:

PASS = <passenger capacity required for the mission> – <passenger capacity available with current design parameters>.

Differences between model constraints and design constraints include:

- Design constraints can be extracted directly from design goals, while formulating model constraints requires carefully examining the underlying assumptions of the model on which the simulator is based.

- Design constraints can be violated without reducing the accuracy of the objective function computed by the simulator, but when a model constraint is violated, the value of the objective function computed by the simulator cannot be trusted. For example, even if the PASS constraint is violated, the simulator can still correctly compute the takeoff mass needed to fly though the mission carrying whatever number of passengers the aircraft is actually able to hold. However, if a model constraint is violated, then the takeoff mass computed by the simulator may be wildly wrong. For example, if the simulator is allowed to violate the STALL constraint, the optimizer may design an aircraft with very small wings operated at a very high angle of attack which may appear to be a very efficient aircraft, much better than the best physically plausible design, but which in fact is not capable of flying at all.

- If a design constraint happens to be inactive at the optimal design (i.e., the constraint is satisfied for all designs near the optimal design, so the optimum does not lie on a constraint boundary), then the "null" communication strategy will be effective when applied to this constraint — i.e., the constraint may safely be ignored without a detrimental effect on the optimization. However, the null communication strategy will not in general be effective when applied to model constraints, even if they are inactive at the optimal design. In the region where a model constraint is violated, the value of the objective function computed by the simulator may include random meaningless values, so if the model constraint violations are ignored by the null strategy, the region where the model constraint is violated may include local optima of the objective function or, even worse, points having (spurious) values of the objective function better than the best value for any design satisfying all the model assumptions. Either of these conditions can "trap" the optimizer and keep if from getting to the true optimum, even though the model constraint in question is inactive at the true optimum.

Design Parameter	Small Box low	Small Box high	Big Box low	Big Box high
engine size	0.5	3	0.1	5
wing area (sq. ft.)	1500	13500	500	20000
wing aspect ratio	1	2	0.5	3
fuselage taper length (ft.)	100	200	50	300
effective structural thickness over chord	1	5	0.5	10
wing sweep over design mach angle	1	1.45	0	1.45
wing taper ratio	0	0.1	0	0.1
fuel annulus width (ft.)	0	4	0	8

Figure 4.1: Subsets of design space explored

4.3 Results

We formed the following hypotheses:

1. Using the model constraints communications strategy will produce better optimization performance (lower CPU cost to get the same probability of getting a certain design quality) than the Boolean strategy.

2. The individual model constraints that are active at the global optimum will be more important, in that replacing them with the Boolean strategy will hurt optimization performance more.

3. Using model constraints to guide CFSQP to the feasible region will be less expensive than using random probes to find the feasible region. The difference in CPU time between these two methods will be greater when the "box" in which we are searching is larger.

4. The model constraints will perform similarly for different design goals.

We performed experiments in two domains — the airframe domain and the nozzle domain — to test these hypotheses. We expected all three hypotheses to hold in both domains.

4.3.1 Airframe domain

To test various model communication strategies experimentally, we used a design space in which the optimizer varied the following aircraft conceptual design parameters over a continuous range of values:

1. engine size
2. wing area
3. wing aspect ratio
4. fuselage taper length (how "pointed" the fuselage is)
5. effective structural thickness over chord (a nondimensionalized measure of wing thickness)
6. wing sweep over design mach angle (a nondimensionalized measure of wing sweep)
7. wing taper ratio (wing tip chord divided by wing root chord)
8. fuel annulus width (space available in fuselage for fuel storage)

Figure 4.1 shows the two boxes we explored in the design space defined by these eight design parameters. We used a small box and a large box in order to test our third hypothesis — that model constraints will be more helpful in finding the feasible region when the box is larger.

To test the effect of the communication strategy on the design process, we considered the following strategy combinations:

1. Return values of all model constraint functions to the optimizer as nonlinear inequality constraints.
2. Return values of all model constraint functions except ETLB and ETUB to the optimizer, but for candidate designs where the engine table constraints were violated (ETLB or ETUB positive), use the "Boolean" strategy and return a standard "very large" value for takeoff mass.

3. Return values of all model constraint functions except A1LB, A1UB, A2LB, and A2UB to the optimizer; use "Boolean" strategy for points which required extrapolation outside the aerodynamics table bounds.

4. Return values of all model constraint functions except FM, which is "Boolean".

5. Return values of all model constraint functions except STALL, which is "Boolean".

6. Return values of all model constraint functions except WLUB, which is "Boolean".

7. Use the "Boolean" communication strategy for all model constraint functions.

8. A two-level approach in which the "Boolean" communication strategy is used to find a feasible point, and then all model constraints are used to find an optimum.

Comparing strategy combination 1 with strategy combination 7 allows us to test our first hypothesis by comparing the performance of the model constraints strategy with that of the Boolean strategy. Strategy combinations 2 through 6 allow use to test our second hypothesis by seeing whether replacing a given set of related model constraints with the Boolean strategy hurts optimization performance. Strategy 8 allows us to test our third hypothesis — that the model constraints strategy is more helpful for finding a feasible point when the box is bigger.

For each strategy combination, our system randomly choose points in the "small box" of Figure 4.1 until it found 74 "evaluable" points (i.e., points whose objective function was not assigned the Boolean strategy standard "very bad" value).[1] Each of these 74 points was then used as a starting point for a design optimization using CFSQP to try to find an optimal aircraft design for the mission shown in Figure A.5. (We required the starting points to be evaluable because if CFSQP happened to be started in an unevaluable region, then all components of the gradient would be zero and the optimization would terminate immediately.) The best design found for this mission in all the experiments is shown in Figure 4.2, and a diagram of this aircraft appears in Figure A.4.

[1] 74 is not a "magic" number; it was just a convenient choice given available disk space.

Design Parameters:	
engine size	1.532
wing area	4652 sq. ft.
wing aspect ratio	1.570
fuselage taper length	121.3 ft.
effective structural thickness over chord	3.002
wing sweep over design mach angle	1.158
wing taper ratio	0
fuel annulus width	0
Objective Function:	
Takeoff Mass	167.4 tonnes
Model Constraints:	
ETUB	−41.57
ETLB	−0.76
A1LB	−2.2
A1UB	−1.8
A2LB	−1.5
A2UB	−8.5
WLUB	−149.8
FM	−0.0011 tonnes
STALL	0
Design Constraint:	
PASS	−2

Figure 4.2: Best design found for mission of Figure A.5

Strategy Combination	Success	Start Cost	Opt. Cost	Est. 99% Cost
All model constraints returned	65/74	16	42375	1252
ETLB and ETUB "Boolean"	52/74	3203	67158	3609
FM "Boolean"	0/74	603	99215	$\gg$ 456565
STALL "Boolean"	18/74	5441	81566	19427
A1LB/A1UB/A2LB/A2UB "Boolean"	67/74	57	47042	1242
WLUB "Boolean"	62/74	721	39404	1372
All model constraints "Boolean"	0/74	21946	75804	$\gg$ 447106
Two level	72/74	18098	39389	990

Figure 4.3: Performance of the various strategy combinations

The performance of the strategy combinations is shown in a table in Figure 4.3. The "Success" column for each strategy combination shows what fraction of the 74 optimizations found aircraft designs having takeoff masses within 1% of the takeoff mass of the apparent "global optimum" — the best design we found for this mission (Figure 4.2). The "Start Cost" column shows how many simulations had to be run on unevaluable points while finding the 74 optimization starting points, and "Opt. Cost" shows the total number of simulations that were run during each set of 74 optimizations.[2] The "Est. 99% Cost" column in Figure 4.3 gives the estimated cost with each strategy combination to have a 99% chance of finding the global optimum, which is computed by multiplying the average cost per optimization times $\log(1 - P_{\text{desired}})/\log(1 - P_{\text{success}})$, where P_{desired} is the desired probability of finding the global optimum (99% in this case) and P_{success} is the probability of any single optimization finding the global optimum (which we estimate with the value in the "Success" column).[3] As the table indicates, for the cases whose success was 0/74, the only information we can compute about the "Est. 99% Cost" is that it will be greater than the cost we would have computed if the success rate had been 1/74. Figure 4.4 shows graphically the "Est. 99% Cost" to achieve a range of different design qualities.

The data in Figure 4.3 indicates that the model constraints communication strategy can find the global optimum with a 99% confidence at a cost which is **one or more orders of magnitude smaller** than the cost to achieve comparable results with the Boolean communication strategy, consistent with our first hypothesis. Examination

---

[2] As mentioned earlier, a complete mission simulation requires about 1/4 second of CPU time on a DEC Alpha 250 4/266 workstation.

[3] Derivation of formula: $(1 - P_{\text{success}})$ is the probability that a single optimization will **not** find the global optimum, so $(1 - P_{\text{success}})^n$ is the probability that **none** of n optimizations will find the global optimum, and thus $(1 - (1 - P_{\text{success}})^n)$ is the probability that at least one of n optimizations **will** find the global optimum. To find the cost of P_{desired}, a given desired probability of finding the global optimum, solve

$$P_{\text{desired}} = 1 - (1 - P_{\text{success}})^n$$

for n, which gives

$$n = \log(1 - P_{\text{desired}})/\log(1 - P_{\text{success}})$$

and finally multiply n by the the average cost per optimization. Note: the computed value of n is not necessarily an integer, so a more precise calculation would round n up to the nearest integer.

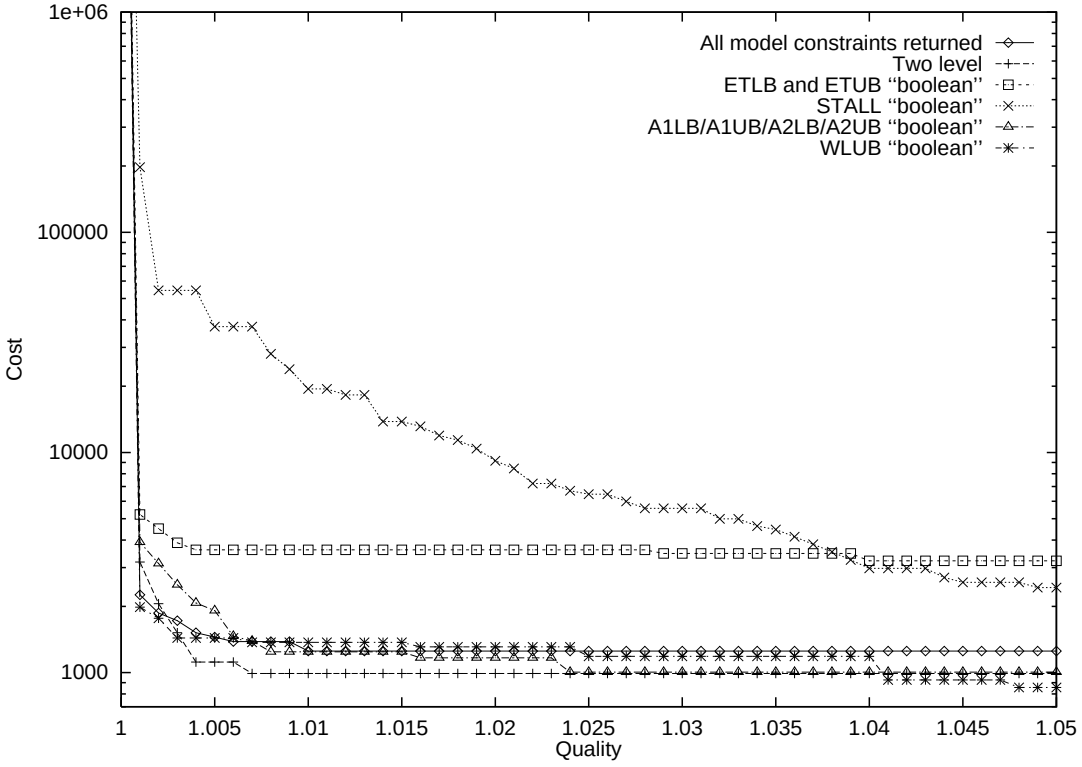

Figure 4.4: Cost to achieve a range of design qualities with 99% confidence. Quality is takeoff mass, normalized by the best takeoff mass found (Figure 4.2), so quality = 1.01 corresponds to Figure 4.3. The 'FM "Boolean"' and 'All model constraints "Boolean"' strategy combinations do not appear as they did not find designs within 5% of the best takeoff mass found.

of the different strategy combinations indicates that the model constraints which contribute most to this performance difference are the constraints active at the global optimum (constraint values ≈ 0 in Figure 4.2) — consistent with our second hypothesis — but that even the constraints which are inactive at the global optimum may give a factor of two to three speedup when handled using model constraints rather than the Boolean strategy. If a constraint is active at the global optimum, then CFSQP must "navigate" along the constraint boundary when searching for the optimum. This navigation is easy when the boundary is defined by a smooth model constraint, but much more difficult when the boundary is marked only by a sudden jump in the objective function from a reasonable value to the Boolean "very bad" value. Model constraints which are inactive at the optimum may still be active during some parts of the search and thus can help guide the search and prevent the optimizer from getting stuck.

Model constraints which are active at the global optimum are more critical, but it is important to note that there will typically be no reliable *a priori* way to determine which model constraints will be active at the global optimum. This fact suggests that the model constraints communication strategy should be used to handle all model assumptions, even though implementing smooth model constraint functions may require more work than implementing the simpler Boolean communication strategy.

An issue that should be considered is the question of why any model constraints are active for the globally optimal design. Does this situation indicate that there are actually better designs on the other side of the constraint boundary which the optimizer would be able to find if only we had a more sophisticated model that didn't need as many constraints? Not necessarily. For example, lift initially rises as a function of angle of attack and later begins falling rapidly as stall occurs for higher angles of attack. The STALL constraint, which is active at our global optimum (see Figure 4.2) cuts off this function at its peak so that the lift function is monotonic where the constraint is satisfied. A more sophisticated simulator which modeled stall would not find better designs on the other side of the STALL constraint boundary — it would just find that the lift function ceased to be monotonic when the boundary was crossed. The ETLB constraint is also active at our global optimum (see Figure 4.2). In this case, the engine

Strategy Combination	Success	Start Cost	Opt. Cost	Est. 99% Cost
All model constraints returned	55/74	48	58376	2674
ETLB and ETUB "Boolean"	14/74	6979	124796	39102
FM "Boolean"	0/74	879	128618	$\gg$ 592317
STALL "Boolean"	15/74	21669	78508	27520
A1LB/A1UB/A2LB/A2UB "Boolean"	40/74	280	176113	14114
WLUB "Boolean"	60/74	2181	58976	2285
All model constraints "Boolean"	0/74	354761	80835	$\gg$ 1992408
Two level	54/74	301917	56198	17034

Figure 4.5: Performance of the various strategy combinations in a bigger box

stops running when the throttle is too low. Modifying the engine model to predict correctly the sudden low temperatures and pressure produced by the engine when it stops running would not uncover better designs.

To test our third hypothesis — that model constraints would be a bigger help in finding the feasible region when the box size is bigger — we repeated our experiments in a larger box. Figure 4.1 shows the two "boxes" in the design space used in our experiments. The bigger box contains the smaller box, and the volume of the larger box is about 300 times greater than the volume of the smaller box. Figure 4.5 shows the performance of the various communication strategies in the larger box, and Figure 4.6 shows graphically the "Est. 99% Cost" to achieve a range of different design qualities. Search cost increases in the larger box, as expected, but model constraints still cost orders of magnitude less than the Boolean strategy.

It is important to compare the performance of the "two-level" strategy combination for the two boxes. In the "small" box, the two-level approach was actually superior to the pure model constraints approach: it was slightly better to use a Boolean strategy to find a feasible point before starting to use model constraints to find the optimum. The reverse was true in the big box, however: the pure model constraints approach was a factor of six less expensive then the two-level approach, confirming our third hypothesis. These results are quite plausible, because the "start cost" data for "all Boolean" combination indicates that the density of feasible points in the small box is 74/21946 ($\approx$ 1/300) while in the big box it is only 74/354761 ($\approx$ 1/4800). The big box has such a small feasible region that the benefit of using model constraints to search for

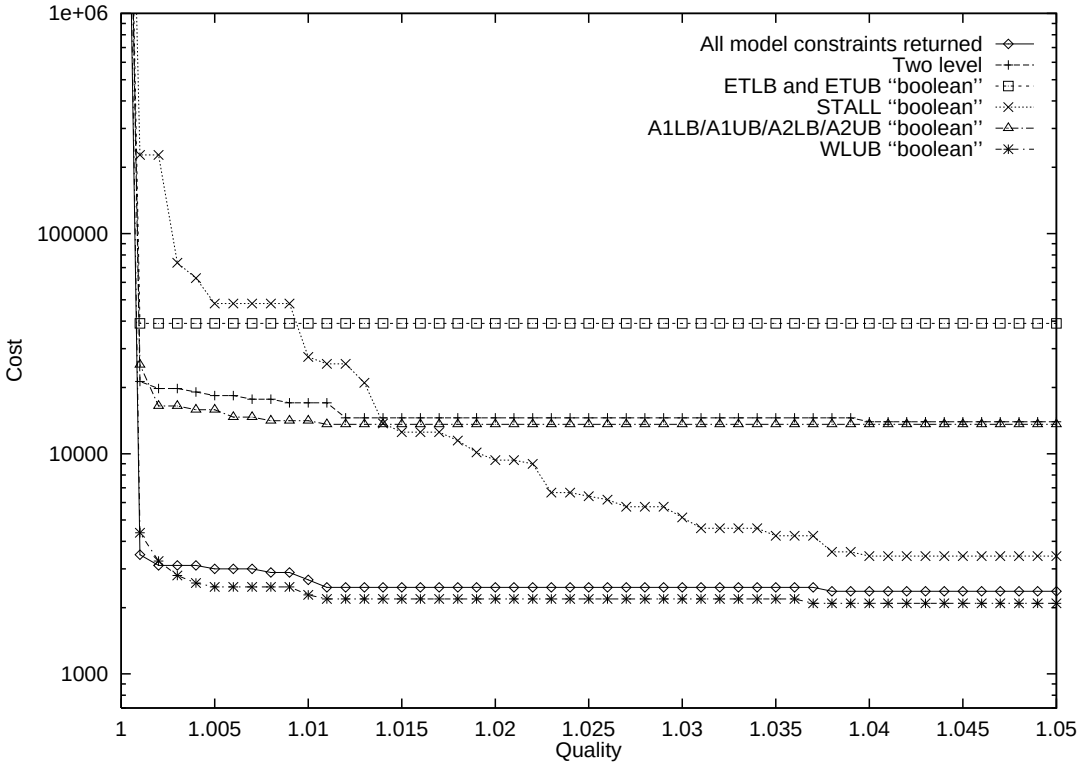

Figure 4.6: Cost to achieve a range of design qualities with 99% confidence in a bigger box. Quality is takeoff mass, normalized by the best takeoff mass found (Figure 4.2), so quality = 1.01 corresponds to Figure 4.5.

Phase	Mach	Altitude (ft.)	Duration (min.s)	comment
1	0.227	0	5	"takeoff"
2	0.85	40,000	50	subsonic cruise (over land)
3	2.0	60,000	225	supersonic cruise (over ocean)

capacity: 70 passengers.

Figure 4.7: Another mission specification

the feasible region outweighs the model constraints overhead, while in the smaller box random probes can find the feasible region cheaply enough that the overhead of using model constraints to find the feasible region is not justified. However, even in the small box model constraints are still extremely useful for searching within the feasible region in order to find an optimum.

To test our fourth hypothesis — that the model constraints will perform similarly for different design goals — we repeated our experiments with a different goal. We used the same boxes as for the previous experiments, but instead the goal was to design the best aircraft for the mission shown in Figure 4.7. Figure 4.8 and Figure 4.9 show the best design found, which differs considerably from the optimal design for the previous mission. We performed the same set of experiments for this case, and the experimental data which appears in Figures 4.10, 4.11, 4.12, and 4.13 supports our previous conclusion that the model constraint communication strategy can cut search cost by an order of magnitude or more.

4.3.2 Nozzle domain

We have also implemented model constraints for a different design problem having significantly different sorts of model assumptions. Here we look at the problem of designing a particular subcomponent of an aircraft: the engine's exhaust nozzle. In a supersonic aircraft, exhaust nozzles are complex, adjustable mechanical systems, and a computational simulation requires a number of model assumptions which have a geometrical flavor not found in the work we described above.

We used a multilevel design framework to design a nozzle and an airframe that can

Design Parameters:	
engine size	1.146
wing area	3690 sq. ft.
wing aspect ratio	1.089
fuselage taper length	130.1 ft.
effective structural thickness over chord	2.728
wing sweep over design mach angle	1.235
wing taper ratio	0
fuel annulus width	0
Objective Function:	
Takeoff Mass	134.8 tonnes
Model Constraints:	
ETUB	−2.89
ETLB	−18.19
A1LB	−1.83
A1UB	−2.17
A2LB	−2.03
A2UB	−7.97
WLUB	−143.8
FM	−0.00038 tonnes
STALL	0
Design Constraint:	
PASS	−2

Figure 4.8: Best design found for the 2nd mission (Figure 4.7)

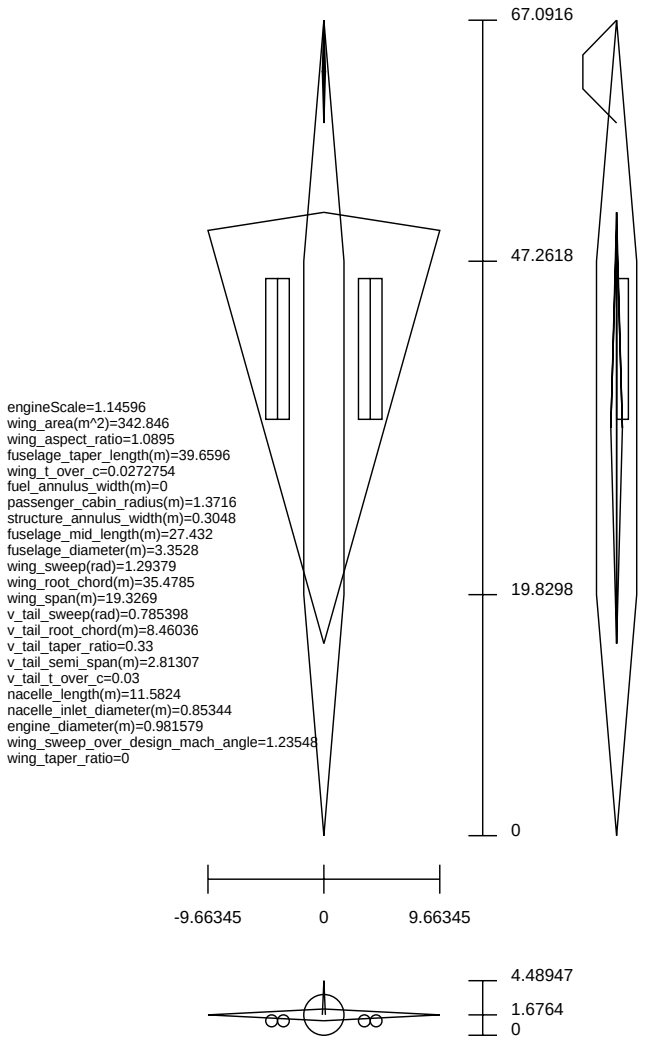

Figure 4.9: Supersonic transport aircraft designed by our system for the second mission (Figure 4.7)

Strategy Combination	Success	Start Cost	Opt. Cost	Est. 99% Cost
All model constraints returned	62/74	13	36204	1238
ETLB and ETUB "Boolean"	57/74	1275	65556	2827
FM "Boolean"	1/74	31	65105	297930
STALL "Boolean"	36/74	1681	50847	4904
A1LB/A1UB/A2LB/A2UB "Boolean"	65/74	55	40377	1194
WLUB "Boolean"	64/74	227	34899	1092
All model constraints "Boolean"	0/74	6576	67046	$\gg$ 336745
Two level	64/74	4307	34477	1205

Figure 4.10: Performance of the various strategy combinations for the 2nd mission

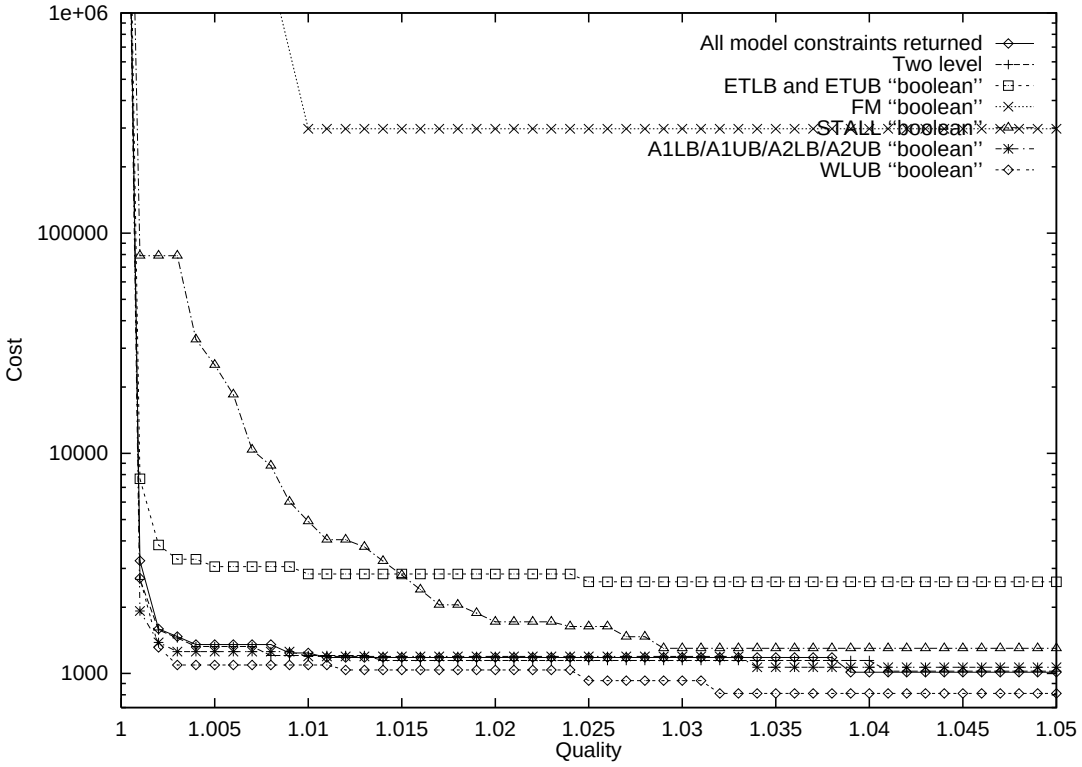

Figure 4.11: Cost to achieve a range of design qualities with 99% confidence. Quality is takeoff mass, normalized by the best takeoff mass found (Figure 4.8), so quality = 1.01 corresponds to Figure 4.10.

Strategy Combination	Success	Start Cost	Opt. Cost	Est. 99% Cost
All model constraints returned	48/74	41	57607	3429
ETLB and ETUB "Boolean"	13/74	5616	145770	48765
FM "Boolean"	0/74	162	93146	$\gg$ 426789
STALL "Boolean"	39/74	7602	64000	5951
A1LB/A1UB/A2LB/A2UB "Boolean"	34/74	342	131652	13352
WLUB "Boolean"	51/74	1420	56171	3066
All model constraints "Boolean"	0/74	133265	117224	$\gg$ 1145732
Two level	49/74	231396	48049	16025

Figure 4.12: Performance of the various strategy combinations for the 2nd mission in the bigger box

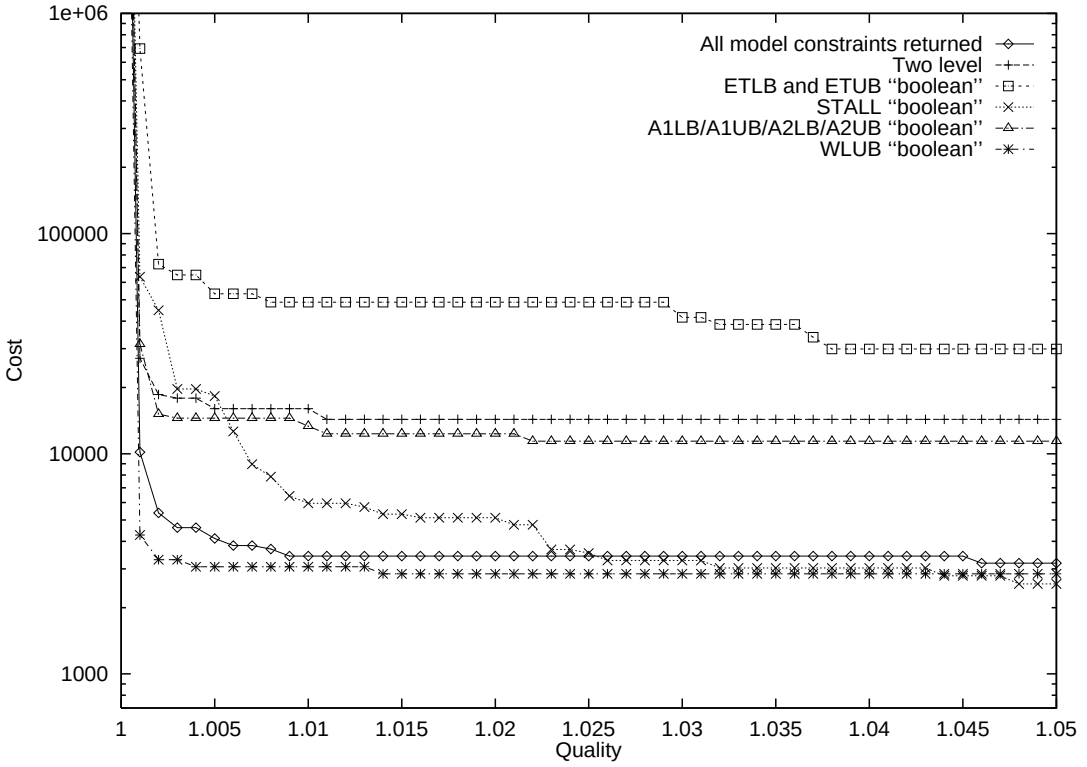

Figure 4.13: Cost to achieve a range of design qualities with 99% confidence. Quality is takeoff mass, normalized by the best takeoff mass found (Figure 4.8), so quality = 1.01 corresponds to Figure 4.12.

be used together. (See Chapter 6 for a full description of our multilevel nozzle and airframe design algorithm.) We approach the exhaust nozzle design problem as follows:

1. Design an aircraft for a particular mission as described in earlier in this chapter. A high level of abstraction is used, and the exhaust nozzle is not modeled explicitly.

2. Fix (almost) all of the major aircraft parameters at the optimal values found in the abstract optimization, add an explicit model of the exhaust nozzle to the simulation, and vary the parameters describing the nozzle geometry (Figure A.6), using the same optimization criterion that was used for airframes (i.e., minimize takeoff mass of the aircraft required to complete a given mission).

The explicit model of the exhaust nozzle used in the second stage is based on one-dimensional gas dynamics heavily supplemented by experimental data tables. The following additional model constraint functions are used (which, as before, are less than or equal to zero if a constraint is satisfied and positive otherwise):

ELMAX = <length l_e of external nozzle flap> − <maximum length external nozzle flap could have, with the given values for the rest of the nozzle geometry, while still allowing the nozzle to be connected as a convergent-divergent nozzle>.

ELMIN = <minimum length external nozzle flap could have, with the given values for the rest of the nozzle geometry, while still allowing the nozzle to be connected as a convergent-divergent nozzle> − <length l_e of external nozzle flap>.

CA = <minimum angle to which convergent flap can move, while still maintaining a convergent-divergent configuration> − <maximum angle to which convergent flap can move, while still maintaining a convergent-divergent configuration>.

R8LOW = <smallest value r_8 can achieve with current geometry> − <smallest value for r_8 required during mission simulation>.

R8HIGH = <largest value for r_8 required during mission simulation> − <largest value r_8 can achieve with current geometry while maintaining a convergent-divergent configuration>.

Strategy Combination	Success	Start Cost	Opt. Cost	Est. 99% Cost
All model constraints returned	5/100	21504	54297	68055
All model constraints "Boolean"	3/100	2674928	39106	4103386

Figure 4.14: Performance of the various strategy combinations

NG1 $= 0 - z_7$. Nozzle geometry bound.

NG2 $= r_6 - r_{10}$. Nozzle geometry bound.

NG3 $= r_7 - r_{10}$. Nozzle geometry bound.

NG4 $= z_{10} - (z_7 + l_c + l_d)$. Nozzle geometry bound.

NG5 $= (r_7 - l_c) - r_6$. Nozzle geometry bound.

CA1LB, CA1UB, CA2LB, CA2UB: violation of bounds for a two-dimensional table of experimental data on nozzle angularity thrust loss.

CV1LB, CV1UB, CV2LB, CV2UB: violation of bounds for a two-dimensional table of experimental data on nozzle friction velocity/thrust loss.

CB1LB, CB1UB, CB2LB, CB2UB: violation of bounds for a two-dimensional table of experimental data on nozzle boattail (external) drag.

The model constraints used with airframes also continue to be used.

We experimentally compared the model constraints and Boolean communication strategies for this new design problem, to see if the model constraints strategy again proved superior, as it did in the airframe domain. For our experiments, we focused on stage two of the two-stage design process described above, since stage one does not involve the explicit nozzle model. (Note that stage one is just the design process described extensively earlier in this chapter.) In stage two we used a five-dimensional design space consisting of the four nozzle geometry parameters l_c, l_d, l_e, and r_7, and also the aircraft wing area. Wing area was added to give some "looseness" to the main aircraft design, to avoid difficulties in finding a nozzle that will fit the exact optimal design found in stage one.

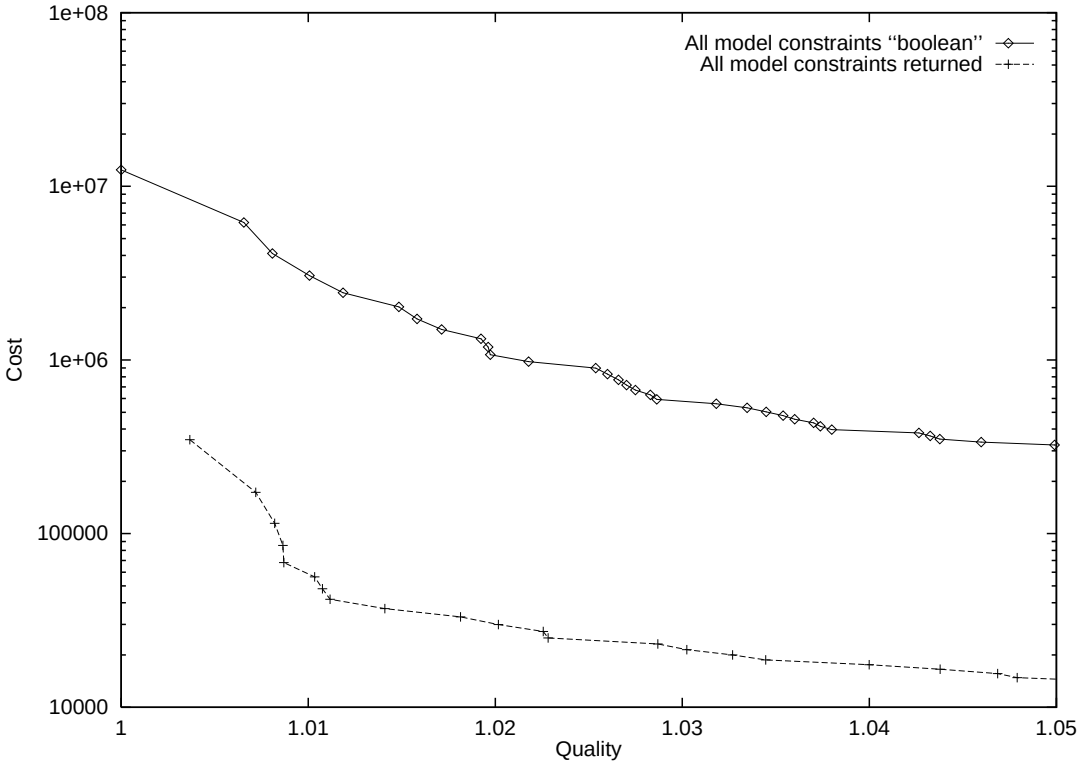

Figure 4.15: Cost to achieve a range of design qualities with 99% confidence. Quality is takeoff mass, normalized by the best takeoff mass found.

The mission of Figure A.5 was used again for these experiments. For the nozzle problem, we did not do extensive experiments testing each individual model constraint for its effect, but instead simply compared a multistart optimization in which all model constraints are available to CFSQP to an optimization in which all model constraints are handled with the "Boolean" strategy. Figures 4.14 and 4.15 show the experimental results, indicating that using model constraints reduces search cost by orders of magnitude, as we found in the airframe domain, providing additional support for our first hypothesis. Note that the density of feasible points in this space is only 100/2674928 ($\approx 1/27000$), and that model constraints locate this small feasible region much more efficiently than do random probes.

Note: after the second stage of the two-stage design process, we actually added a third stage in which a single optimization with all twelve design parameters was started from the best point found in stage two. This third stage improved the design somewhat, but the improvement was rather small since the stage one and stage two design problems

are fairly decomposable. Chapter 6 further describes the three stages.

4.4 Analysis

4.4.1 Limitations and Future Work

The model constraints communication strategy requires (of course) that model assumptions be representable using model constraint functions, which is a possible limitation since some problem domains may include assumptions which are not amenable to such representation. However, we did not encounter such a difficulty in the conceptual design of aircraft domain, or the exhaust nozzle domain.

CFSQP, the numerical optimizer used in most of the results in this thesis, assumes design variables take a continuous range of values, so our approach would not work as it stands for problems whose design variables are fundamentally limited to a discrete set of values. However, the idea of representing model assumptions as constraints makes sense for discrete problems as well as continuous, assuming appropriate search methods are used, for example methods from the extensive body of research on constraint satisfaction problems.

Our experiments have been performed in a domain in which the global optimum has a fairly large "basin of attraction", so that a local optimization method like Sequential Quadratic Programming will give a high confidence of finding the global optimum if started from a small number of random starting points. For domains in which this property fails to hold, global optimization methods such as Simulated Annealing will often be preferable. Such methods would not typically be able to make direct use of model constraint functions, so for such a domain investigating the "model penalties" communication strategy described in Section 4.2 might be a worthwhile area for future work.

An interesting but very challenging area for future research would be the automated generation of model constraint functions from a declarative representations of a model and its assumptions.

4.4.2 Conclusion

Automated search of a space of candidate designs seems an attractive way to improve the traditional engineering design process. To make this approach work, however, the automated design system must include both knowledge of the modeling limitations of the method used to evaluate candidate designs and also an effective way to use this knowledge to influence the search process. We suggest that a productive approach is to include this knowledge by implementing a set of *model constraint* functions which measure how much each modeling assumptions is violated, and to influence the search by using the values of these model constraint functions as constraint inputs to a standard constrained nonlinear optimization numerical method. Our experiments indicate that our model constraint communication strategy can decrease the cost of design space search by one or more orders of magnitude.

Chapter 5

Rule-Based Gradients

5.1 The Problem

The existence of unevaluable points in a search space (see Section 1.4) is a pathology that presents a major barrier to automated design optimization. The previous chapter describes how model constraints can be used to replace unevaluable points with infeasible points, by constructing nonlinear constraints that embody knowledge of the limitations of the simulator. This chapter presents *rule-based gradients*, which are another method for handling unevaluable points. Rule-based gradients have two advantages over model constraints. First, they are easier to implement. Second, when using model constraints, it is usually not possible to capture all of the limitations of the simulator in the constraints, so some unevaluable points remain. In these cases, rule-based gradients can be used to handle the remaining unevaluable points. Our experimental results show that using rule-based gradients together with model constraints produces better optimization performance than using either technique alone.

Gradient-based optimization methods, such as sequential quadratic programming (see Appendix A), are reasonably fast and reliable when applied to search spaces that satisfy their assumptions — namely that the objective function and constraint functions are continuous, smooth, and defined everywhere. Unfortunately, realistic simulators tend to violate these assumptions. In particular, for some designs they tend to exit without returning values of the objective or constraint functions. We call these designs *unevaluable points* in the search space.

One approach to dealing with unevaluable points is to simply assign such points a "very bad value," and to return this value to the optimizer for all such points. If the optimizer computes gradients numerically (such as by forward differences), this can

result in gradients that are wildly inaccurate. When the optimizer attempts to make use of these inaccurate gradients, it often fails to reach the optimum.

We present a rule-based technique for computing gradients in the presence of unevaluable points. Our technique features a set of rules which specify how to compute the gradient with a reasonable degree of accuracy in each of several cases that arise in the presence of unevaluable points. These rules embody knowledge of the way optimization algorithms use gradients, and of the types of pathologies that tend to exist in search spaces defined by complex simulators.

Our experimental results show that using these rule-based gradients, together with a line search that terminates when it encounters an unevaluable point, as part of a gradient-based optimization system, produces optimization performance that is one or more orders of magnitude better than that obtained without the rule-based gradients.

5.1.1 Unevaluable Points

Unevaluable points arise for several reasons:

- **Undefined points.** At these points, the function has no value. For example, in the aircraft domain, if the fuselage is shorter than the wing chord, then the design is meaningless, and its takeoff mass is undefined.

- **Unevaluable points due to limitations of the simulator.** These points may be perfectly good designs, but because of limitations in the simulator, it is not possible to evaluate them. They fall into two categories:

  Unevaluably bad points. At these points, it is known that the design is bad, but it is not possible to compute a number indicating just how bad it is. For example, if an airplane must be flown at a large angle of attack in order to get the necessary lift, then the plane will be very inefficient, and will have a very large takeoff mass. If the simulator does not model the increased drag resulting from the large angle of attack, then it will not be able to accurately compute takeoff mass, but it will know that takeoff mass is very large.

Points of unknown quality. For some unevaluable points, the simulator has no idea whether the design is good or bad. For example, many simulators require the solution of a set of equations that has no closed-form solution. In these cases, numerical root finding must be used. Numerical root finders are not always able to find a solution. The designs for which the root solvers are unable to find a solution are classified as unevaluable points.

If the simulator is a *legacy code* (a simulator that has been in use for a long time), the engineers who use it may be reluctant to change or replace it, because they have faith in the code due to years of comparisons between the codes and physical experiments. Such legacy codes may contain bugs that cause them to crash or go into infinite loops for some designs. In such cases, a *software wrapper* around the legacy code can be used to detect when the code crashes or runs for longer than a certain amount of time, and to classify these designs as unevaluable points.

5.2 The Technique

For each component of the gradient, the rule-based gradient computation method evaluates three points: the current point, and two points produced by changing the specified design parameter by plus or minus δ, where δ is a small step size. The method has stored in its knowledge base an appropriate value of δ for each component of the gradient. The method then does one of three things: it computes the partial derivative using all three points according to the central-difference formula, or it computes the partial derivative using just two of the points according to the forward-difference formula, or it returns zero for the partial derivative. It chooses one of these three options based on the following rules:

- **Basic Rule 1:** If all three points are evaluable, one outer point is worse than the middle point, and the other outer point is better than the middle point, use the central difference formula. This rule should produce an accurate partial derivative.

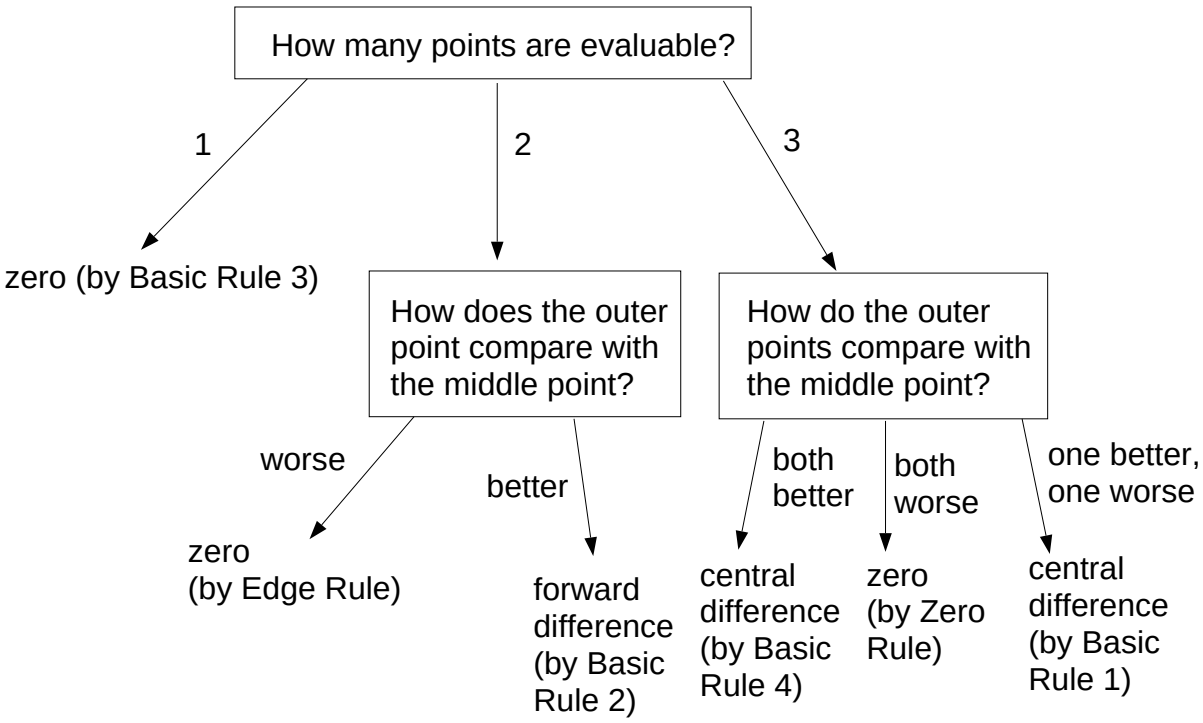

Figure 5.1: Flowchart of the rules used to determine how to compute a partial derivative from three points.

- **Zero rule:** If all three points are evaluable, and the two outer points are worse than the middle point, return zero for the partial derivative. This rule prevents the optimizer from "bouncing back and forth" in the vicinity of the optimum.

- **Basic Rule 2:** If one of the outer points is unevaluable, and the other is better than the middle point, compute the partial derivative by applying the forward difference formula to the two evaluable points. This rule produces a reasonably accurate derivative in the presence of an unevaluable point.

- **Edge rule:** If one of the outer points in unevaluable, and the other one is worse than the middle point, return zero for the partial derivative. This rule allows the optimizer to trace the edge of the evaluable region by moving along the other axes.

- **Basic Rule 3:** If both of the outer points are unevaluable, return zero as the partial derivative. This rule allows the optimizer to move through a narrow evaluable region by moving along the other axes.

- **Basic Rule 4:** If all three points are evaluable, and the two outer points are both better than the middle point, use the central difference formula. This point is the opposite of a local optimum — it is a point that is maximally bad (at least locally). This type of point usually does not occur during optimization.

These rules are summarized in a flowchart in Figure 5.1.

5.3 Results

We made the following hypotheses:

1. Rule-based gradients would result in improved optimization performance (lower CPU cost to get the same probability of a given design quality).
2. Some rules would be more helpful in some domains than in other domains, but would not hurt performance very much when used in the domains where they are not helpful.
3. Using rule-based gradients together with model constraints would produce better optimization performance than using either rule-based gradients or model constraints alone.
4. The same set of rules would perform well across different design goals within a given domain, and across different domains.

5.3.1 Airframe domain

To test rule-based gradients experimentally, we used an eight-dimensional design space in which the optimizer varied the following aircraft conceptual design parameters over a continuous range of values:

1. engine size
2. wing area
3. wing aspect ratio

4. fuselage taper length (how "pointed" the fuselage is)

5. effective structural thickness over chord (a nondimensionalized measure of wing thickness)

6. wing sweep over design mach angle (a nondimensionalized measure of wing sweep)

7. wing taper ratio (wing tip chord divided by wing root chord)

8. fuel annulus width (the amount of space left in the fuselage for fuel)

We explored the subset of this eight-dimensional design space defined by the larger box in Figure 4.1.

We compared four optimization strategies:[1]

- **Plain CFSQP:** In this strategy, unevaluable points are replaced with a standard "very bad value" which is passed to the optimizer, CFSQP, and to the gradient routine, which computes gradients using the standard central-difference formula.
- **Rule-based gradients:** This strategy handles the unevaluable points using the rule-based gradient computation method described in this chapter.
- **Model Constraints:** This strategy replaces some of the unevaluable points with infeasible points, using the model constraints strategy described in Chapter 4.
- **Rule-based gradients and model constraints:** This strategy replaces some of the unevaluable points with infeasible points, and uses rule-based gradients to handle the remaining unevaluable points.

For each strategy, our system randomly chose points until it found 74 evaluable points.[2] Each of these 74 points was then used as a starting point for a design optimization using CFSQP to try to find an optimal aircraft design for the mission shown

[1] The experiments described in Chapter 4 all included rule-based gradients. Therefore, the second and fourth strategies listed here are the same as two of the strategies described in Chapter 4. Two of the four curves in each of the current chpater's plots are based on the same data used in Chapter 4.

[2] 74 is not a "magic number"; we ran optimizations until we ran out of disk space.

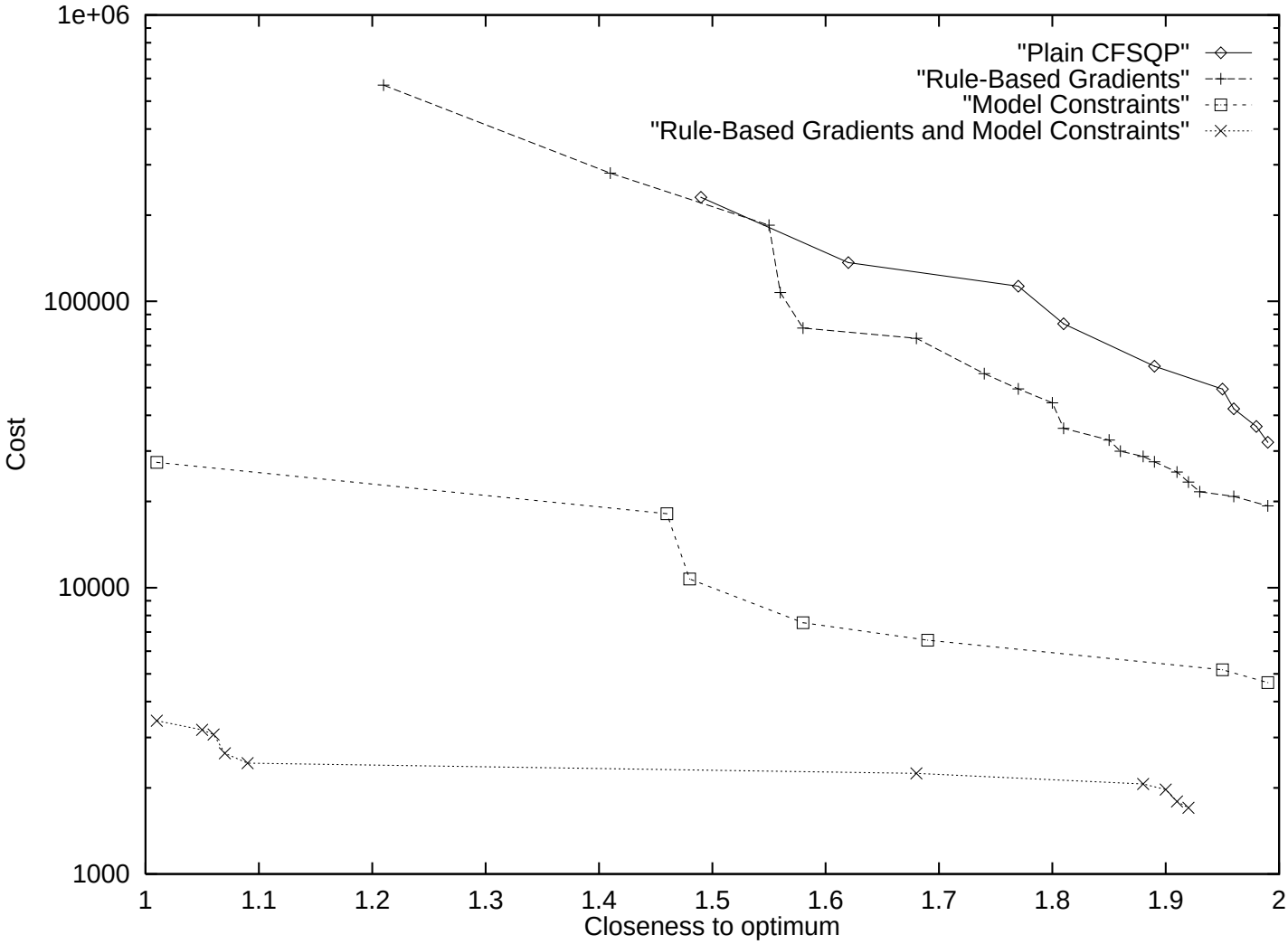

Figure 5.2: Performance of the various strategies. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

in Figure 4.7. (We required the starting points to be evaluable because if CFSQP happened to be started in an unevaluable region, then all components of the gradient would be zero and the optimization would terminate immediately.) Without model constraints, 76% of the points encountered were unevaluable. With model constraints, 4% of the points encountered were unevaluable. The best design found for this mission in all the experiments is shown in Figure 4.8, and a diagram of this aircraft appears in Figure 4.9.

The performance of the strategy combinations is shown in a graph in Figure 5.2. This graph shows the estimated cost with each strategy to have a 99% chance of getting

within a certain fraction of the optimum. The horizontal axis shows the best takeoff mass found divided by the takeoff mass of the design that is believed to be the global optimum. A value of one indicates that the method has found the global optimum. The vertical axis shows the estimated number of simulations needed to have a 99% chance of obtaining a particular closeness to the optimum.[3]

The data in Figure 5.2 indicates that the strategy of combining rule-based gradients with model constraints is one or more orders of magnitude better than any of the other strategies, supporting our hypotheses. Without model constraints, the optimizer fails to even get within 30% of the optimum. With model constraints, but without the rule-based gradients, the optimizer does get to the global optimum, but takes *eight times as long* to have the same probability (99%) of getting there.

To test our hypothesis that the rules would perform well for different design goals, we repeated our experiments with a different goal. The goal was to design the best aircraft for the mission shown in Figure A.5. Figure 4.2 and Figure A.4 show the parameters and diagram of the best design found, which differs considerably from the optimal design for the previous mission. We performed the same set of experiments for this case, and the experimental data appears in Figure 5.3. For this mission, the strategy that combines rule-based gradients with model constraints is the only strategy that got to the global optimum. The other strategies were orders of magnitude worse, consistent with our hypothesis.

Of the rules listed in Section 5.2, all except two make sense for any search space. The edge rule and zero rule may or may not be beneficial, depending on the characteristics of the search space. To test our hypothesis that different rules within the set would be more helpful than others, depending on the domain, we first tested the utility of these rules in the aircraft domain. We did 130 optimizations, with model constraints,

---

[3] As mentioned earlier, a complete mission simulation requires about 1/4 second of CPU time on a DEC Alpha 250 4/266 workstation. The estimated number of simulations needed is computed by multiplying the average cost per optimization times $\log(1 - P_{\text{desired}})/\log(1 - P_{\text{success}})$, where P_{desired} is the desired probability of getting within the specified fraction of the global optimum (99% in this case) and P_{success} is the probability of any single optimization getting within that fraction of the global optimum (which we estimate using the fraction of our 74 optimizations that got within that fraction of the global optimum). For a derivation of this formula, see chapter 4.

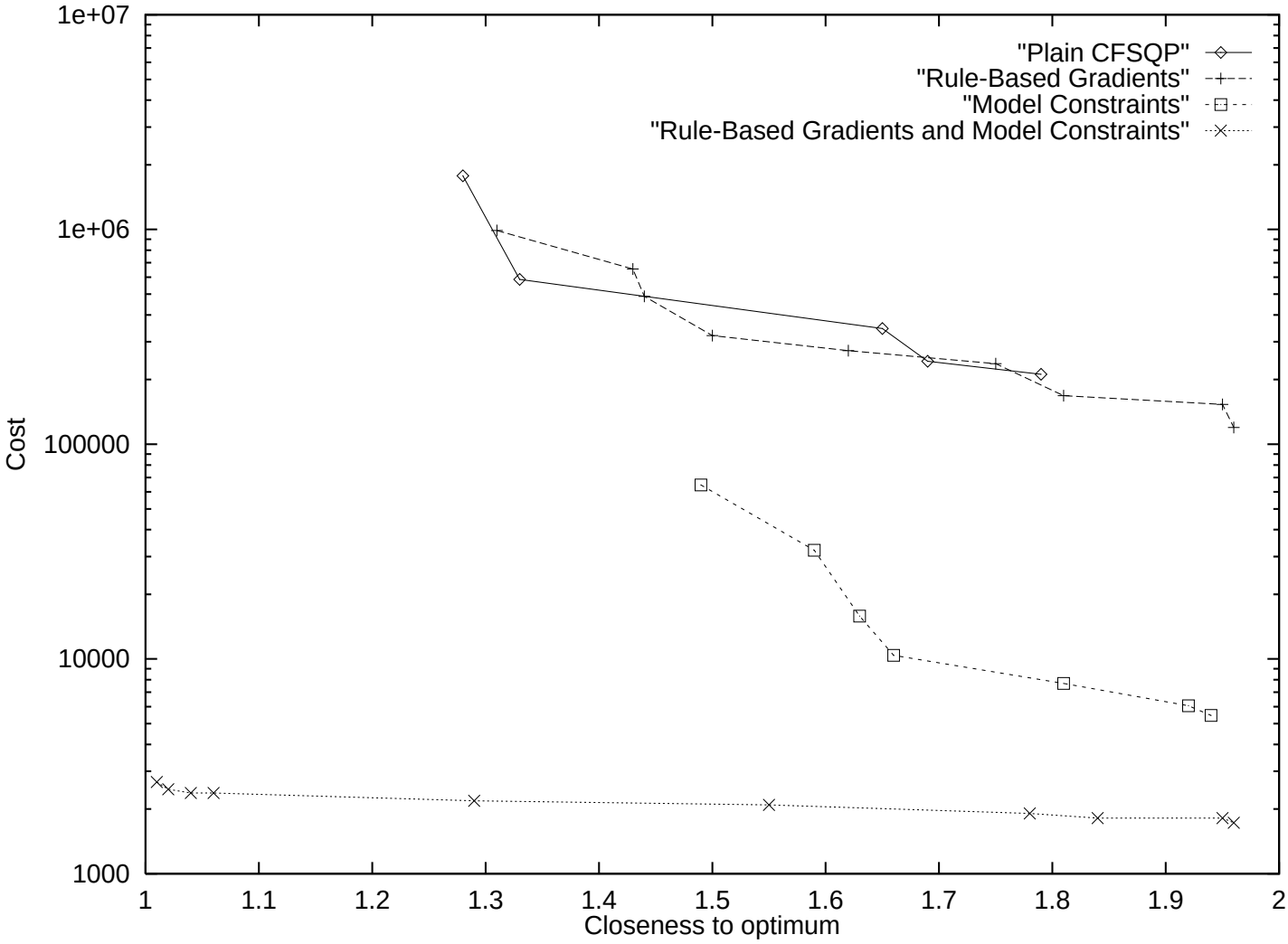

Figure 5.3: Performance of the various strategies for a different mission. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

Rules used	Success	Start Cost	Opt. Cost	Est. 99% Cost
basic rules	73/130	108	88979	3827
basic rules and edge rule	99/130	91	85551	2116
basic rules and zero rule	67/130	89	90768	4443

Figure 5.4: Optimization performance using subsets of the rule-based gradient rules, for the first mission.

Rules used	Success	Start Cost	Opt. Cost	Est. 99% Cost
basic rules	81/130	100	102058	3708
basic rules and edge rule	97/130	126	98035	2536
basic rules and zero rule	83/130	119	111345	3881

Figure 5.5: Optimization performance using subsets of the rule-based gradient rules, for the second mission.

for each of the two missions, with just the "basic rules" (the rules listed in Section 5.2 other than these two), and with each of these two rules added individually. Figure 5.4 and Figure 5.5 show the results for the two missions. For each subset of the rules, it shows the number of optimizations that came within 1% of the point that we believe is the global optimum, the number of simulations used to find an evaluable starting point using random probes, the number of simulations used in the optimizations, and the estimated number of simulations needed to obtain a 99% chance of getting within 1% of the global optimum. Adding the edge rule lowers the cost of finding the optimum by about 40% for either mission. Adding the zero rule actually hurts optimization performance slightly in this domain.

We believe that the zero rule is useful when the search space is "V-shaped" at the optimum — that is, when the first derivative is discontinuous at the optimum. In the yacht domain (see next section), we found that the optimum often had this property because of penalties specified in the 12 Meter Rule, and that the zero rule was therefore helpful. It apparently is not helpful in the aircraft domain, but it does not hurt performance very much — consistent with our second hypothesis. It therefore might be a good idea to include this rule if it is not known whether or not the search space has this property.

5.3.2 Other domains

We have found that using rule-based gradients results in good optimization performance in several other design domains. These domains include the design of racing yachts (see Chapter 2 and Chapter 3), the design of exhaust nozzles for supersonic jets [Gelsey *et al.* 1996b], the design of inlets for hypersonic jets (see Chapter B), and the design of inlets for supersonic missiles (Project Cheapshot) [Zha *et al.* 1996]. All of these domains use simulators that are more expensive than our airframe simulator. For example, the simulators that we use for hypersonic inlets are computational fluid dynamics (CFD) codes that solve the two-dimensional Navier-Stokes equations, and take between 2 and 5 hours of CPU time for a single simulation. It would be prohibitively costly to perform extensive comparisons of different optimization strategies using such a simulator.

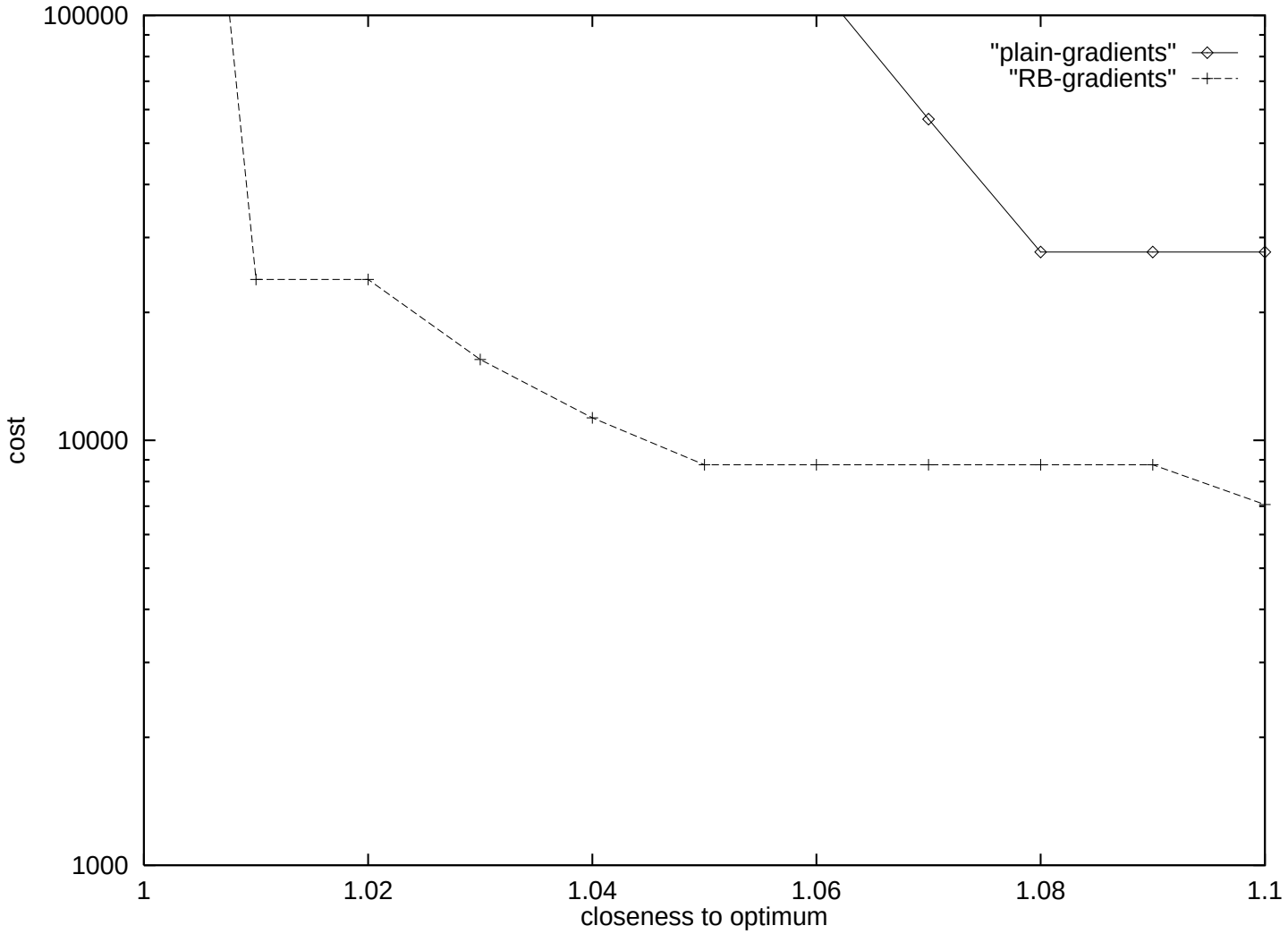

Figure 5.6: Performance of rule-based gradients for the missile inlet domain. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

We have preliminary results demonstrating the value of rule-based gradients in the domain of supersonic missile inlet design. This domain has 9 design parameters. It takes about 7 seconds of CPU time to evaluate a design on a DEC Alpha 250 4/266 workstation, using an analysis code called NIDA, which uses a one-dimensional aerodynamic model supplemented with empirical data. We use model constraints with NIDA, but still 15% of the points encountered during optimization are unevaluable. To test the utility of rule-based gradients, we ran 20 CFSQP optimizations with and without rule-based gradients. The optimization performance is shown in Figure 5.6. Using rule-based gradients allows the optimizer to get much closer to the optimum, at a significantly lower cost, supporting our hypothesis.

5.4 Analysis

5.4.1 Limitations and Future Work

We have shown that using rule-based gradients produces much better optimization performance than not using rule-based gradients, when optimizing real-world simulators that have unevaluable points. We believe that performance can probably be enhanced further by adding additional rules. Preliminary results show that adding an additional rule that avoids, whenever possible, using points that violate model constraints (but are evaluable) in gradient computations greatly improves optimization performance in the domain of supersonic engine exhaust nozzle design. In this domain, points that violate model constraints tend to have less-accurate values of the objective function and the other constraints, so gradients can be more accurately computed by avoiding these points. This new rule requires further testing, and other new rules could be developed.

5.4.2 Conclusion

Gradient-based numerical optimization of complex engineering designs offers the promise of rapidly producing better designs. However, such methods generally assume that the objective function and constraint functions are continuous, smooth, and defined everywhere. Unfortunately, realistic simulators tend to violate these assumptions. We

present a rule-based technique for intelligently computing gradients in the presence of such pathologies in the simulators, and show how this gradient computation method can be used as part of a gradient-based numerical optimization system. We have implemented and tested rule-based gradients in several realistic design domains. Our experimental results show that using rule-based gradients can decrease the cost of design space search by one or more orders of magnitude.

Chapter 6

Multilevel Design

6.1 The Problem

A major barrier to the use of gradient-based search methods for engineering design is that complex, multidisciplinary design spaces tend to have many apparent local optima — both real and pathological. We define an *apparent local optimum* to be a point that our optimizer, CFSQP, declares to be a local optimum. Such a point may be a true local optimum in the true physics of the problem, or it may be a *pathological local optimum.* Pathological local optima occur for several reasons. Numerical truncation error in the solvers within the simulator can result in local optima in the search space defined by the simulator that are not in fact local optima in the true physics. Discontinuities in the objective or constraint function or their derivatives — also known as "ridges" — violate CFSQP's assumptions and can therefore fool CFSQP into thinking that it is at a local optimum when in fact it is not. "Near ridges" — portions of the search space that are smooth, but that have very large second derivatives — can similarly fool CFSQP. Apparent local optima are a barrier to the use of optimization, whether they are real or pathological.

One approach to this problem is to use global search methods such as genetic algorithms and simulated annealing. We would, however, like to exploit the power of gradient-based optimization methods to quickly converge on the optimum. Our approach is therefore to use gradient-based optimization at multiple levels of search space abstraction (where each level has a much smaller number of apparent local optima), coupled with multiple levels of abstraction in the simulator.

Multi-level representations have been studied extensively by artificial intelligence researchers. Multi-level techniques for planning and theorem proving go back as far

as [Sacerdoti 1974]. The importance of decomposing a problem into multiple levels was discussed at length in Simon's classic work on AI and Design, reprinted as [Simon 1981]. Some researchers have applied the multi-level paradigm to engineering design [Sobieszczanski-Sobiesky 1982, Sobieszczanski-Sobieski *et al.* 1985, Rogers 1989, Ellman and Schwabacher 1993], but have not focussed on the use of multi-level optimization to deal with multiple apparent optima in the search space.

It might be asked how these problems are approached today. Human designers often decompose multidisciplinary design problems into smaller components that are then designed by different groups of people, but this entire process is generally carried out without the use of automated design. Our multi-level optimization method can therefore be seen as an automation of the approach typically taken by groups of human designers.

We tested our technique in the domain of conceptual design of supersonic aircraft, focusing on the airframe and the jet engine exhaust nozzle, and found that using multiple levels of simulation and optimization improves optimization performance by one or more orders of magnitude.

6.2 The Technique

Our approach is to use gradient-based optimization at multiple levels of search space abstraction, coupled with multiple levels of abstraction in the simulator. The search space abstractions are formed by decomposing the set of design parameters into two or more subsets. These subsets will typically correspond to different components of the artifact, such as the airframe and the nozzle of an aircraft. These subsets may be disjoint, or it may be necessary for a small number of design parameters to occur in more than one subset. The most important parameters of one component may be included in another component's abstract space in order to serve as a proxy for the first component during optimization. The search space abstraction should be defined in such a way that the different levels are as independent as possible — that is, the optimal values in one subset should not depend strongly on the values in the other subset. If the

overall space is approximately a product space of the abstract spaces, and the abstract spaces each have a moderate number of local optima, then the number of local optima in the overall space will be approximately equal to the product of the number of optima in each of the abstract spaces. Thus by decomposing the problem into multiple levels, it should be possible to optimize in search spaces with a much smaller number of local optima. For example, it may be the case that the two decomposed spaces have n and m apparent local optima, respectively, and the overall space has mn apparent local optima. The number of multistarts needed to find the global optimum with a certain probability will vary linearly with the number of apparent local optima, so the cost of having a certain probability of finding the global optimum will be $O(mn)$ in the overall space, and only $O(m+n)$ in the decomposed space.

It is also helpful to have a simulator which can simulate at different levels of abstraction corresponding to the different levels of search space abstraction. When optimizing one component of the overall design, it helps to have a simulator that simulates the other components at a lower level of detail. Such a simplified simulator can be faster and less pathological than the full simulator.

6.3 Results

We made the following hypotheses:

1. Using an appropriately selected two-level abstraction for optimization will produce better optimization performance (lower CPU cost for the same probability of getting a given design quality) than using one-level optimization.

2. Using an appropriately selected three-level abstraction for optimization will produce better optimization performance than using the two-level abstraction for optimization.

3. The same abstractions will produce good optimization performance for different missions.

Design Parameter	low	high
engine size	0.5	3
wing area (sq. ft.)	1500	13500
wing aspect ratio	1	2
fuselage taper length (ft.)	100	200
effective structural thickness over chord	1	5
wing sweep over design mach angle	1	1.45
wing taper ratio	0	0.1
fuel annulus width (ft.)	0	4
nozzle convergent flap length (in.)	3	48
nozzle divergent flap length (in.)	9	120
nozzle external flap length (in.)	24	120
nozzle radius 7 length (in.)	1	100

Figure 6.1: Subset of design space explored

To test these hypotheses experimentally, we used a twelve-dimensional design space in which the optimizer varied 12 aircraft design parameters over a continuous range of values. Our optimizations focussed on two aspects of the aircraft: the airframe, which is described by the first eight parameters (see Figure A.4), and the exhaust nozzle, which is described by the last four parameters. Figure 6.1 shows the subset we explored in the design space defined by these twelve design parameters.

The optimizations described in this chapter were performed subject to a set of constraints. Figure 6.3 lists the constraints, along with the values of the constraint functions at the point which be believe is the global optimum. The nozzle geometry bound constraints require, for example, that the specified nozzle flaps be connectable. The table bound constraints require that the simulator not have to extrapolate outside the tables of experimental data which it uses. The aerodynamic bounds require, for example, that the lift coefficient required to fly the specified design over the specified mission not exceed one. There is a sanity check to make sure that the work performed by the actuators in the nozzles is positive. Finally, the passenger constraint requires that there be enough room in the plane for the specified number of passengers.

To test our hypotheses, we performed optimizations using one, two, or three levels of abstraction, and then compared the results.

6.3.1 One-level optimization

First we tried doing optimization without the use of any multi-level techniques. We used the four-bar nozzle simulator, and used CFSQP to optimize in the search space defined by all twelve design parameters. Because this search space has many apparent local optima, we used a "random multistart" to attempt to find the global optimum. The system randomly generated starting points within the box of Figure 6.1 until it found 100 evaluable points[1], and then performed optimizations from each of these points. The first curve in Figure 6.2 shows the estimated cost of having a 99% chance of getting within various fractions of the point we believe to be the optimum using this method. This estimate is computed by multiplying the average cost per optimization times $\log(1-P_{\text{desired}})/\log(1-P_{\text{success}})$, where P_{desired} is the desired probability of getting within within the specified fraction of the apparent global optimum (99% in this case) and P_{success} is the probability of any single optimization getting within the specified fraction of the apparent global optimum (which we estimate using the fraction of our 100 optimizations that got within this fraction of the apparent global optimum)[2].

6.3.2 Two-level optimization

Since the one-level optimization was unacceptably expensive, we decided to attempt to reduce the optimization cost by decomposing the search space into two levels. As our first level, we used the eight airframe parameters, and the ideal nozzle simulator. As our second level, we used the four nozzle parameters, and the four-bar nozzle simulator. CFSQP quickly found the point that we believe to be the optimum in the eight-dimensional airframe space, which was encouraging. We then fixed the values of the eight parameters at their optimized values from the first level, and attempted to find the optimum in the nozzle space, using random multistart. After performing 1000 optimizations from random starting points, CFSQP failed to find even a single feasible point, so we declared this particular multi-level strategy to be a failure. We determined

[1] Some randomly generated designs, which we call "unevaluable points," cannot be simulated, either because the designs are meaningless or because of limitations of the simulator.

[2] For a derivation of this formula, see chapter 4

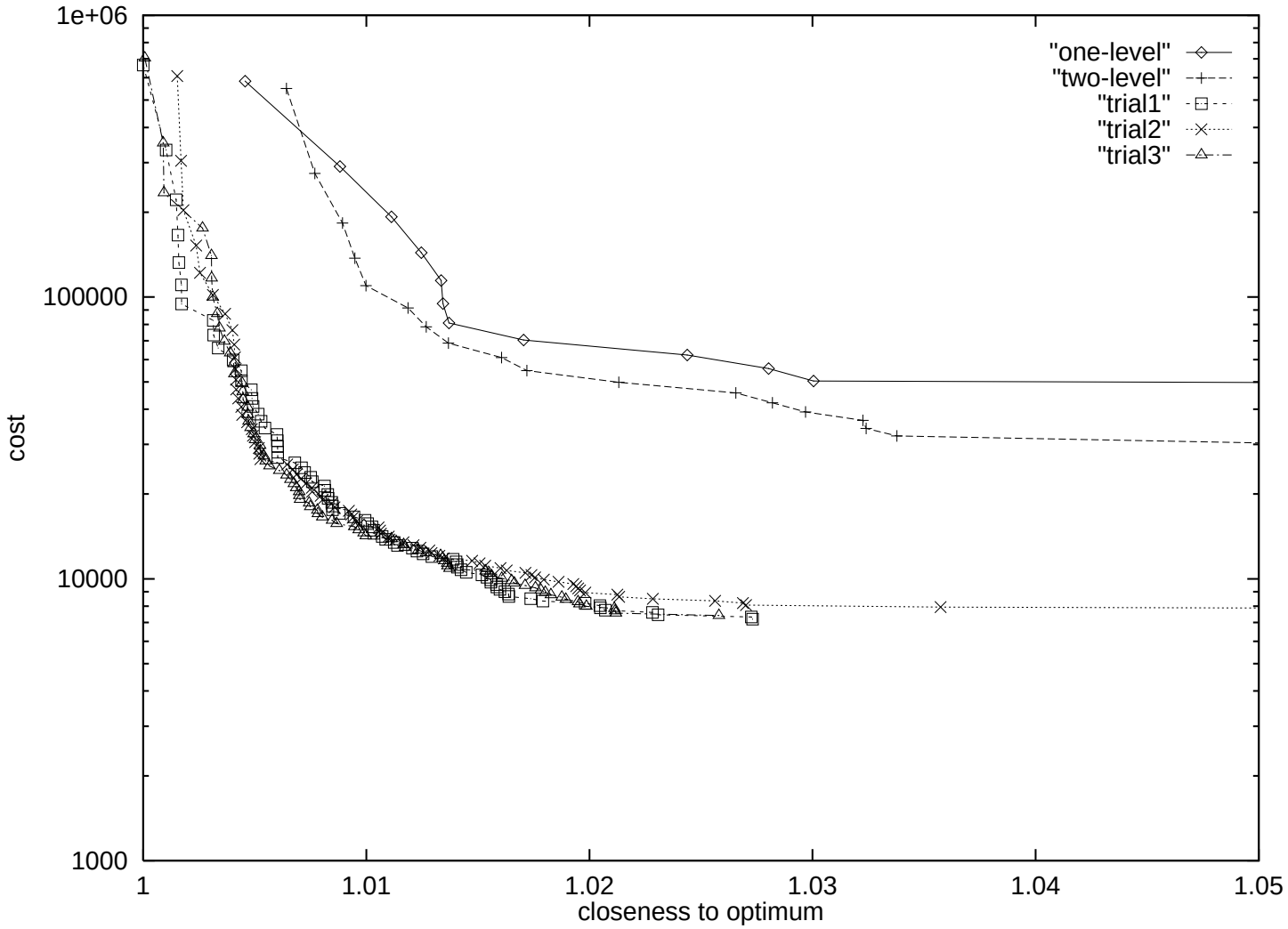

Figure 6.2: Performance of multilevel strategies for the first mission. Optimization performance increases as one moves down (lower cost) and to the left (closer to apparent optimum).

that the airframe designed in the first level, which had been designed using an ideal nozzle in the simulator, was only suitable for use with an ideal nozzle, so it was not possible to design a four-bar nozzle that would work with this airframe.

To circumvent this problem, we allowed CFSQP to vary all twelve design parameters in the second level. We performed a 5-point random multistart at the first (8-dimensional) level, followed by a 100-point random multistart optimization at the second (12-dimensional) level, where each optimization starting point had the eight airframe parameters set at their optimized values from the first level, and had randomly generated values of the four nozzle parameters. The second curve in Figure 6.2 shows the estimated cost of having a 99% chance of getting within various fractions of the apparent optimum using this method. Each point in this curve is based on the cost of doing the 5-point 8-dimensional multistart, plus an n-point 12-dimensional multistart, for a value of n corresponding to the cost.

Two-level optimization resulted in roughly a twofold reduction in the cost of getting within a certain distance of the apparent optimum, supporting our first hypothesis.

6.3.3 Three-level optimization

Two-level optimization significantly reduced the cost of finding the apparent optimum in the 12-dimensional airframe/nozzle space, confirming our first hypothesis. However, we believed that further improvements in optimization performance would be possible if we could allow CFSQP to optimize the nozzle without at the same time optimizing all of the airframe parameters. We decided to try a new strategy for the second level: letting CFSQP optimize the nozzle parameters, and just one airframe parameter. We chose wing area as the one airframe parameter to optimize in the second level, because we believe that it is the most important airframe parameter. One can think of wing area alone as an abstraction of the entire airframe, to be used while optimizing the nozzle, much as the ideal nozzle is used as an abstraction of the four-bar nozzle while optimizing the airframe. Each run at the second level started with the eight airframe parameters set to their optimized values from the first level, and with

Design Parameters:	
engine size	1.462
wing area	4508 sq. ft.
wing aspect ratio	1.557
fuselage taper length	121.0 ft.
effective structural thickness over chord	2.978%
wing sweep over design mach angle	1.159
wing taper ratio	0.0
fuel annulus width	0.0
nozzle convergent flap length	15.31 in.
nozzle divergent flap length	70.54 in.
nozzle external flap length	101.48 in.
nozzle radius 7 length	14.63 in.
Objective Function:	
Takeoff Mass	162.2 tonnes
Nozzle geometry bounds:	
0.0-z7	-0.3971
r6-r10	-0.09784
r7-r10	-0.2806
z10-(z7+cl+dl)	-2.578
(r7-cl)-r6	-0.572
camin-camax	-1.272
el-elmax	-0.01806
elmin-el	-1.328
minRadius8-idealThroatRadius	-0.1240
idealThroatRadius-maxRadius8	-0.0001676
Table bounds:	
ECD lbte	-5.699
ECD ubte	-42.12
rae x min	-2.175
rae x max	-1.825
rae y min	-1.549
rae y max	-8.451
CA x min	-0.8136
CA x max	-98.03
CA y min	-4.121
CA y max	-35.71
CV x min	-0.8136
CV x max	-98.03
CV y min	-3.871
CV y max	-35.70
CB x min	-3.381
CB x max	-11.62
CB y max	-1.000
CB z min	-0.4876
CB z max	-0.5124
Aerodynamic bounds:	
wing-loading bound	-0.1497 tonnes
fuel mass constraint	0.0 tonnes
Lift coef-1	0.0
0.0-wing sweep	-1.214
wing sweep-pi/2	-0.3569
Sanity check:	
0.0-4barWork	-191015
Design Constraint:	
passenger constraint	-2

Figure 6.3: Best design found for mission of Figure A.5. Negative values of constraint functions indicate that the constraints are satisfied.

the the four nozzle parameters set randomly, and then did an optimization in the five-dimensional space defined by the four nozzle parameters and wing area.[3] We knew that optimizing in this space would not allow the optimizer to get exactly to the apparent global optimum, so we added a third level in which the optimizer is allowed to vary all twelve design parameters. The third level was run each time that the level two optimization ended at a feasible point. The three curves in Figure 6.2 labeled "trial1," "trial2," and "trial3" show the estimated cost of having a 99% chance of getting within various fractions of the apparent optimum using the three-level method. Each of these curves is based on a different 5-point multistart in the 8-dimensional space of level one, followed by an n-point 5-dimensional multistart (for various values of n corresponding to the costs) at level two, followed by a third level in which there is a 12-dimensional optimization from each of the feasible apparent optima of level two. We did three trials of the three-level method to see if the results would vary significantly based on what happens in the random multistart of level one; the graph in Figure 6.2 shows that there is not much variation.

Using the three-level method provided roughly an order of magnitude reduction in cost compared with the two-level method, confirming our second hypothesis. We believe that there are three reasons for this speedup. The first is that computing the gradient is less expensive in the five-dimensional space. The second is that in the two-level method, when CFSQP is started from a point in which eight of the design parameters are nearly optimal and the other four are set to random values, it does not know that the eight airframe parameters are near their globally optimal values, so it initially changes the airframe parameters to make the airframe more appropriate for the suboptimal nozzle, and later has to change them back as the nozzle becomes closer to optimal, resulting in the need to perform more line searches. The third reason is that when doing 5-dimensional optimizations, CFSQP has a higher success rate at finding a feasible point than when doing 12-dimensional optimizations. We believe that the reason for this higher success rate is that the constraint functions have fewer apparent

[3]For each starting point, we kept wing area at its optimized value, rather than setting it to a random value, because we believe that the optimized value, although not optimal, should be better than a random value.

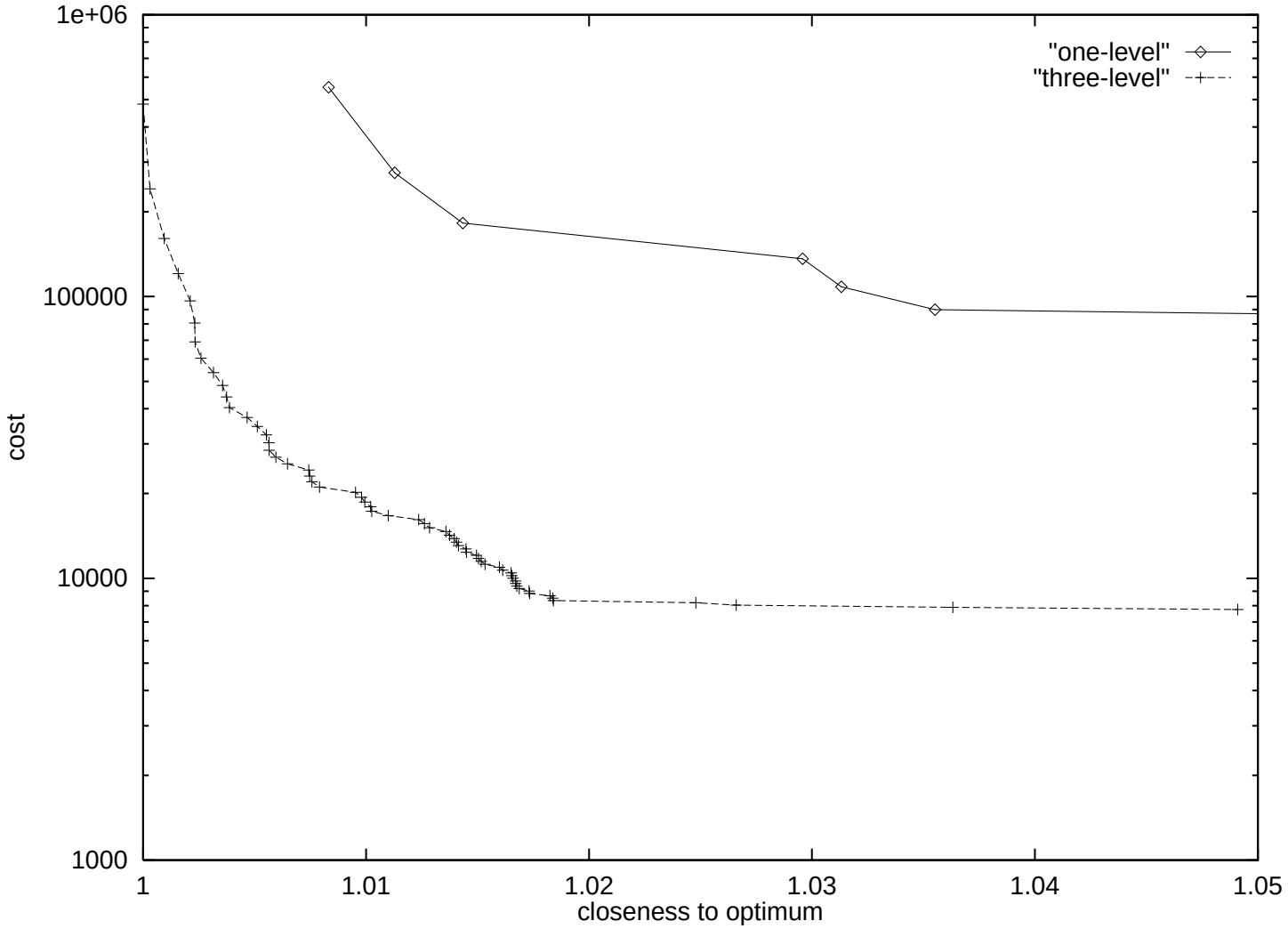

Figure 6.4: Performance of multilevel strategies for the second mission.

local optima in the 5-dimensional space than they do in the 12-dimensional space.

6.3.4 Another mission

To test the effect of the mission on our results, we repeated the experiments for another mission — the mission of Figure 4.7. We compared the single-level method with the three-level method. The results are shown in Figure 6.4. The best design found for this mission is shown if Figure 6.5. We again obtained an order of magnitude reduction in cost using the multilevel method, confirming our third hypothesis.

Design Parameters:	
engine size	1.155
wing area	3617 sq. ft.
wing aspect ratio	1.091
fuselage taper length	129.3 ft.
effective structural thickness over chord	2.673%
wing sweep over design mach angle	1.232
wing taper ratio	0.0
fuel annulus width	0.0
nozzle convergent flap length	28.25 in.
nozzle divergent flap length	50.34 in.
nozzle external flap length	94.40 in.
nozzle radius 7 length	14.89 in.
Objective Function:	
Takeoff Mass	132.1 tonnes
Nozzle geometry bounds:	
0.0-z7	-0.4095
r6-r10	-0.1053
r7-r10	-0.2196
z10-(z7+cl+dl)	-2.406
(r7-cl)-r6	-0.8320
camin-camax	-0.5552
el-elmax	-0.02223
elmin-el	-1.172
minRadius8-idealThroatRadius	-0.2032
idealThroatRadius-maxRadius8	0.0
Table bounds:	
ECD lbte	-14.46
ECD ubte	-9.046
rae x min	-1.789
rae x max	-2.211
rae y min	-2.005
rae y max	-7.995
CA x min	-0.9595
CA x max	-97.75
CA y min	-6.426
CA y max	-33.10
CV x min	-0.9595
CV x max	-97.75
CV y min	-6.176
CV y max	-33.10
CB x min	-1.634
CB x max	-11.98
CB y max	-1.000
CB z min	-0.5166
CB z max	-0.4834
Aerodynamic bounds:	
wing-loading bound	-0.1439 tonnes
fuel mass constraint	0.0 tonnes
Lift coef-1	0.0
0.0-wing sweep	-1.290
wing sweep-pi/2	-0.2803
Sanity check:	
0.0-4barWork	-233402
Design Constraint:	
passenger constraint	-2

Figure 6.5: Best design found for mission of Figure 4.7. Negative values of constraint functions indicate that the constraints are satisfied.

6.4 Analysis

We believe that the full twelve-dimensional space has a large number of apparent local optima, so that finding the apparent global optimum requires a large number of random multistarts. The two-level strategy reduces the cost by getting eight of the twelve parameters close to their optimal values in Level 1, so that fewer random multistarts are needed in the twelve dimensional space. The three-level strategy provides a further improvement by getting all twelve parameters near their optimal values in Levels 1 and 2, so that fewer twelve-dimensional optimizations are needed in Level 3.

6.4.1 Limitations and Future Work

One might ask whether the multi-level technique is applicable to design problems outside the aircraft domain. We have formulated (but not yet tested) multi-level techniques for two other domains.

In the yacht domain, we could use two levels of representation and analysis for the keel. At the first level, the keel would be analyzed using the following simple algebraic formula for effective draft (where D is maximum draft, and A_{ms} is the cross-sectional area of the hull at mid-ship):

$$T_{eff} = 0.92\sqrt{D^2 - \frac{2A_{ms}}{\pi}}$$

At this level, the keel would be represented using a small set of parameters that have an effect on the formula, or on other quantities computed by VPP, such as surface area or displacement. This small set of parameters would include the keel's height and taper ratio.

At the second level, the keel would be analyzed using PMARC, a panel method. Since PMARC is sensitive to the shape of the keel, the keel would be represented using a B-Spline surface. The cost of analyzing the keel with PMARC is orders of magnitude greater than the cost of evaluating the algebraic formula, so it would be potentially very beneficial to perform most of the optimization at the first level.

In the inlet domain, we have used NIDA to analyze a missile inlet rapidly, and GASP

to analyze it with greater accuracy. Analyzing a single missile inlet with GASP takes about one CPU week, which makes it infeasible to perform optimizations with GASP using our current computational resources. We have instead performed optimizations with NIDA, and used GASP to verify the optimized designs. If we had greater computational resources available, we could perform inlet optimization at two levels, which would be likely to produce better designs than our current one-level NIDA optimization. The first level would be the same as our current optimizations — it would use NIDA for the analysis, and a nine-parameter design space that allows the optimizer to vary only those aspects of the inlet that are properly modeled by NIDA. The second level would use GASP for the analysis, and would have a higher-dimensional design space (possibly using splines) that allows the optimizer to make a wider range of changes to the shape of the inlet, since GASP is much more sensitive to the inlet's shape.

It may be difficult to identify the appropriate simulator and search-space abstractions in still other domains. Automatically identifying such abstractions is an area for future research. Finally, the performance of our approach of performing optimization in the presence of many apparent local optima by using a gradient-based optimizer at multiple levels of abstraction needs to be compared with that of global methods such as genetic algorithms and simulated annealing. We may even find that it is possible to use these global methods at multiple levels of abstraction, for even better optimization performance.

6.4.2 Conclusion

Multi-level representations have been studied extensively by artificial intelligence researchers. We utilize the multi-level paradigm to attack the problem of performing multidiscipline engineering design optimization in the presence of many apparent local optima. We use a multidisciplinary simulator at multiple levels of abstraction, paired with a multi-level search space. We tested the resulting system in the domain of conceptual design of supersonic transport aircraft, and found that using multi-level simulation and optimization can decrease the cost of design space search by an order of magnitude.

Chapter 7

Related Work

A great deal of work has been done in the area of numerical optimization algorithms. [Gill *et al.* 1981] provides an applications-oriented overview of numerical optimization algorithms. [Peressini *et al.* 1988] describes the mathematics of nonlinear programming algorithms. [Vanderplaats 1984] describes the application of numerical optimization to engineering design. [Moré and Wright 1993] provides a guide to commercially available numerical optimization software. None of this literature addresses the particular difficulties of attempting to optimize functions defined by large "real-world" numerical simulators.

A number of research efforts have combined AI techniques with numerical optimization. [Ellman *et al.* 1993] describes a method for switching between a less expensive, less accurate simulator, and a more expensive, more accurate simulator during optimization, based on the magnitude of the gradient. [Bouchard *et al.* 1988] describes ways in which expert systems could be applied to the parametric design of aeronautical systems. [Hoeltzel and Chieng 1987] describe a system for digital chip design in which design is done at an abstract level, using machine learning to estimate the performance that would be obtained if the design were carried out at a more detailed level. [Orelup *et al.* 1988] describes a system called Dominic II that uses an expert system to switch among various strategies during numerical optimization. None of these efforts is focused directly on the problems of search-space pathology addressed in this thesis.

Simulated annealing (SA) and genetic algorithms (GA) are able to deal with certain pathologies, such as nonsmoothness, but they tend to be much slower than gradient-based optimization. They tend to require thousands, or even tens of thousands, of simulations, and are thus not practical when each simulation is expensive.

Powell [Powell 1990, Tong *et al.* 1992, Powell and Skolnick 1993] has built a module called Inter-GEN, part of the ENGINEOUS system [Tong 1988], that seeks to combine the ability of genetic algorithms to handle multiple local optima with the speed of numerical optimization algorithms. It contains a genetic algorithm, and a numerical optimizer, and uses a rule-based expert system to decide when to switch between the two. Powell has tested his system on a realistic jet engine design problem. He does not, however, address the issues of unevaluable points, or search space selection.

Gage [Gage 1994, Gage *et al.* 1995] has also combined genetic algorithms with gradient-based optimization. He combined GA's with SQP in two ways. The first method, which he tested in the domain of aircraft wing design, first uses a GA to search a space of wing configurations that is described using a grammar, and then uses SQP to optimize the size of the wings. The second method, which he tested in the domain of truss design, uses the GA to search a space of truss configurations that is described using a grammar, while using SQP at each iteration of the GA to optimize the size of the members. Using the GA to search a configuration space before using SQP to optimize the sizes in a particular configuration can be seen as a method of search space selection which addresses the problem of multiple local optima. Further, the GA has the potential to find a smooth subspace of the overall search space before starting SQP. Gage does not, however, directly address the issue of unevaluable points.

Work on the use of numerical optimization in aircraft design includes [Sobieszczanski-Sobieski *et al.* 1985, Kroo *et al.* 1994]. [Bramlette *et al.* 1990] surveys the application of genetic algorithms to the design and manufacture of aeronautical systems. A survey of multidisciplinary aerospace design optimization can be found in [Sobieszczanski-Sobieski and Haftka 1996].

7.1 Prototype Selection

Cerbone [Cerbone 1992] has reported work which applied machine-learning techniques to a problem similar to our prototype-selection problem. His design space, in the domain of truss design, has an exponential number of disconnected search spaces. He uses

inductive learning techniques to learn rules for selecting a subset of these search spaces for further exploration. In contrast, our system has a smaller number of prototypes (each of which defines a search space) from which to choose, and it just chooses one of them. Cerbone uses an ad-hoc utility function to combine solution quality and search time when evaluating his learning methods, while we only consider solution quality in our prototype-selection work. Cerbone also presents two learners that incorporate background knowledge by incorporating the objective function into the learner.

Research on prototype-retrieval strategies for hillclimbing design optimization is reported by Ramachandran et al. [Ramachandran *et al.* 1992], who investigated a number of library-based methods for finding starting points for the DPMED iterative parameter-design system. These included a nearest-neighbor method, a curve-fitting method, and a hybrid method. The curve-fitting method is similar to our prototype synthesis method. It uses regression to find a function mapping goal parameters to initial design parameters. Ramachandran compared their retrieval strategies in terms of the numbers of iterations needed to carry out the hillclimbing design-optimization process. They showed that starting points obtained by curve fitting led to fewer iterations than were required when the nearest-neighbor method was used. In contrast to this, our work has also evaluated retrieval strategies in terms of the quality of the resulting designs.

Researchers in case-based reasoning have investigated the use of library-retrieval techniques for case-based design [Sycara and Navinchandra 1992, Kolodner 1993], but have not used them to initialize an iterative design process. [Bhatta and Goel 1995] describe a system that learns to retrieve a starting point for the design of a high-acidity sulfuric acid cooler. They evaluate the performance of this indexing system based on its effect on retrieval time, and not based on its impact on optimization performance.

7.2 Reformulation Selection

Gelsey et al. [Gelsey and Smith 1995, Gelsey *et al.* 1996b] describe a Search Space Toolkit which assists in determining properties of the search space that can be used for

reformulation. [Choy and Agogino 1986] describe a system that automates [Papalambros and Wilde 1988]'s method of using monotonicity analysis to detect constraint activity.

In [Williams and Cagan 1994], Williams and Cagan present *activity analysis*, a technique inspired by monotonicity analysis. Their technique is similar to the technique described in this thesis, except that they use qualitative reasoning instead of machine learning to determine which constraints will be active at the optimum. Their technique has the advantages that it does not require training data, and that the reformulation is guaranteed not to lose the global optimum. It has the disadvantage that it requires that the objective function and constraint functions be symbolically differentiable and composed of simple arithmetic operations; it would therefore not be applicable to the complex simulators used in the experiments described in this thesis.

As far as we know, nobody has applied machine learning to predicting constraint activity, or to selection of a search space reformulation for design optimization.

7.3 Model Constraints

[Gelsey 1995b] examines the types of modeling knowledge that are needed so that a simulator can be reliably invoked by another program and describes algorithms for detecting assumption violations and other problems that might lead to low-quality or unreliable simulation results, but strategies for communicating information about modeling failures to an automated design systems are not discussed. [Forbus and Falkenhainer 1990, Forbus and Falkenhainer 1992, Forbus and Falkenhainer 1995] discuss the use of qualitative simulation to check the quality of numerical simulation results, but here strategies for communicating information about modeling failures to an automated design systems are also not discussed.

Other automated intelligent controllers for numerical simulators are described in [Gelsey 1991, Gelsey 1995a, Sacks 1991, Yip 1991, Zhao 1994], but these do not address the issue of model and simulation quality assurance.

Intelligent monitoring for complex systems has received considerable attention (e.g.,

[Dvorak and Kuipers 1991]), but this work has focused on diagnosis of problems in dynamically changing physical systems as opposed to problems in the execution of computational algorithms which are attempting to simulate the behavior of physical systems.

7.4 Rule-Based Gradients

Much work has been done on computing derivatives (or "sensitivities") in multidisciplinary simulators. Most of this work falls into two categories. The first approach uses the residuals of the governing equations to compute the derivatives. The second approach takes the derivatives of each subcomponent of the simulator, and then solves a system of linear equations to obtain the derivatives of the entire system with respect to the design parameters. These methods are surveyed in [Sobieszczanski-Sobieski and Haftka 1996]. As far as we know, nobody has addressed the issue of computing gradients in the presence of pathologies such as unevaluable points.

7.5 Multi-level design

Other work which uses the multi-level paradigm, such as [Sobieszczanski-Sobiesky 1982], is described in Section 6.1, but none of this work focuses on the problem of performing optimization in the presence of many apparent local optima. Much work has been done on the use of simulated annealing and genetic algorithms to deal with search spaces with many local optima [Press *et al.* 1986, Powell and Skolnick 1993, Goldberg 1989], but none of this work has proposed the use of multiple levels of abstraction to reduce the number of apparent local optima that must be handled by the optimizer at any given time.

Chapter 8

Conclusion

8.1 Summary

We have described three types of search-space pathologies that can cause gradient-based optimization to fail: multiple optima, nonsmoothness, and unevaluable points. We have presented five new remedies for these pathologies:

- Inductive learning-based prototype selection and synthesis can help avoid the problems of multiple local optima and nonsmoothness.
- Inductive learning-based reformulation selection can reduce nonsmoothness.
- Model constraints can eliminate unevaluable points.
- Rule-based gradients can help optimization in the presence of unevaluable points.
- Multilevel design and analysis can help optimization in the presence of multiple local optima.

The pathologies and their remedies are summarized in Table 8.1, reproduced here from Chapter 1.

Table 8.1: Techniques to handle pathologies during optimization.

technique	multiple optima	nonsmoothness	unevaluable points
prototype selection	√	√	
reformulation selection		√	
model constraints			√
rule-based gradients			√
multilevel design	√		

We found that these pathologies occur in the four realistic engineering domains that we explored: racing yachts, airframes for supersonic aircraft, nozzles for supersonic jet engines, and inlets for supersonic jets. We tested each of our techniques in two of these domains, and found that using the techniques could improve the quality of the resulting designs, and could also reduce the amount of time needed to produce the designs by as much as three orders of magnitude.

8.2 Review of claims

8.2.1 Effectiveness

Positive results

We claimed that our techniques are effective, in that they improve the quality of the resulting designs, or allow a given quality level to be achieved at a lower CPU-time cost. The effectiveness of the various techniques is summarized here:

- Prototype selection in the yacht domain increased design quality significantly.
- Prototype synthesis in the airframe domain produced the same quality at 22% of the cost.
- Reformulation selection in the yacht domain increased quality slightly while reducing cost by 36%.
- Reformulation selection in the airframe domain reduced cost by 36% without affecting quality.
- Model constraints in the airframe domain reduced the estimated cost of getting a particular design quality by as much as three orders of magnitude.
- Model constraints in the nozzle domain reduced the estimated cost of getting a particular design quality by a factor of 60.
- Rule-based gradients in the airframe domain reduced the estimated cost of getting a particular design quality by an order of magnitude.

- Rule-based gradients in the missile inlet domain reduced the estimated cost of getting a particular design quality by an order of magnitude.
- Multilevel design in the combined airframe/nozzle domain reduced the estimated cost of getting a particular design quality by an order of magnitude.

Negative results

It is instructive to examine some of the negative results that we encountered during the course of our research. These results are not included in the main body of this thesis.

We tried to do prototype synthesis in the yacht domain. CART produced a low root mean squared relative error, indicating that the synthesized prototypes were much closer to optimal than was the mean prototype. But optimizing from the synthesized prototypes produced only a 10% speedup, which we deemed to be too small a speedup to be worth the trouble. We believe that the reason prototype synthesis provided a bigger speedup in the airframe domain than it did in the yacht domain is that in the airframe domain a large portion of the cost of the optimization is the cost of finding a feasible point. Using prototype synthesis often results in a synthesized prototype that is feasible, eliminating the need to search for a feasible point. In the yacht domain, almost any randomly generated point is feasible, so the only benefit of prototype synthesis is that the starting prototype is closer to the optimal prototype. Since CFSQP is a quadratically convergent method, starting closer to the optimum does not have a large impact on the amount of time needed to perform the optimization. The lesson here is that prototype synthesis is more helpful in domains where finding a feasible point is difficult.

We tried to do prototype synthesis in the nozzle domain, before we had implemented model constraints for that domain. In the nozzle domain without model constraints, the evaluable region is a "thin slab." Most of the synthesized prototypes were close (in Euclidean distance) to the optimal prototypes, but unfortunately they were in the unevaluable region — they were near the slab without being in the slab. CFSQP was not able go anywhere from the synthesized prototypes, since it can not compute a gradient

at an unevaluable point. We therefore declared the experiment to be a failure. We believe that had we implemented the model constraints first, the prototype synthesis would have succeeded. The lesson here is that prototype synthesis is not useful when the evaluable region is very small, so it may be important to use model constraints to turn some of the unevaluable points into infeasible points before attempting to use prototype synthesis.

We also attempted an additional learning problem besides the ones that are described in the main body of this thesis. We attempted to learn, based on the difference between the design goal for the starting prototype and the new design goal, which design parameters would need to be changed. The hope was that we could then optimize just a subset of the design parameters. We ran many experiments using various machine learning methods, and various formulations of the goal space, and were unable to obtain learning performance significantly better than that of the most frequent class method. We concluded that the set of design parameters that needs to be changed does not depend in a simple way on the change in goal (within our space of goals), so that the best strategy is to always include all of our design parameters in the optimization. In other words, we had chosen to try to use machine learning for a portion of the optimization setup problem to which machine learning was not applicable. The lesson here is that not all parts of the optimization setup problem are appropriate for machine learning. The decision being learned must depend on the goal, and the relationship between the decision and the goal must not be excessively complicated.

8.2.2 Degree of Automation

As we mentioned in Chapter 1, all of the optimizations described in this thesis were fully automated, in the sense that there was never a human within the optimization loop. The process of setting up the optimization was partially automated. In this subsection, we will describe the steps needed to set up an optimization when given a new simulator, and the degree to which each step has been or could be automated. We will also describe what human intervention, if any, is needed to apply each of our techniques.

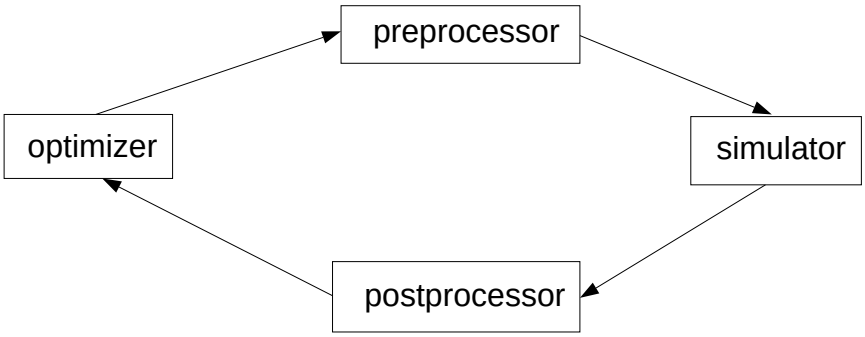

Figure 8.1: The optimizer-simulator interface.

Intervention needed in setting up an optimization

Step 1: Formulate as nonlinear programming problem (choose goal) In this step, it must be decided what the initial design space is, what the objective function to be optimized is, and what constraints must be satisfied. This process amounts to choosing the goal of the optimization. We have always done this by hand, and we believe this step can not be fully automated — the computer can not know what goal the human users wish to achieve. The computer could offer some assistance by automatically generating "trade-study" graphs that illustrate the tradeoffs among multiple conflicting goals, such as cost and quality.

Step 2: Write preprocessor and postprocessor The preprocessor and postprocessor are the software "glue" that binds the simulator to the optimizer. Figure 8.1 illustrates the data flow among the optimizer, the simulator, the preprocessor, and the postprocessor. The preprocessor is run before each simulation. It takes as input the design parameters, and it runs the simulator. It may include gridding and shape modification operators. The postprocessor is run after each simulation, and returns the objective and constraint functions to the optimizer. It might, for example, include numerical integration to extract the numbers needed in the objective and constraint functions from a flow field that is output by a computational fluid dynamics code. We have built the preprocessors and postprocessors by hand. It might be possible to use automatic programming techniques [Lowry and McCartney 1991] to automate this process.

Step 3: Run parameter studies Once the preprocessor and postprocessor have been written, it is possible to run parameter studies to produce graphs of cross-sections of the search space. These parameter studies can be helpful in identifying bugs in the simulator or the preprocessor or postprocessor, and can also be useful in Step 4. We have written programs that automatically run a set of parameter studies and produce a set of graphs. Automatically interpreting these graphs, however, is a challenging area for future research.

Step 4: Choose an optimizer and the parameters to the optimizer By examining the graphs produced in Step 3, one can decide which optimizer is appropriate for the new search space. For example, if the space contains a very large number of apparent local optima, then one might decide to use simulated annealing or a genetic algorithm. If there appears to be only a single global optimum, then one might choose to use a single run of a gradient-based method. For most of the problems described in this thesis, there appeared to be a moderate number of local optima, so we chose to use a multistart CFSQP optimization.

Having chosen to do a multistart CFSQP optimization, one must choose the values of several parameters:

1. The number of multistarts. This number should be higher if there are more apparent local optima.

2. The gradient step sizes. These can be chosen by performing a convergence study in which a wide range of step sizes are chosen, and one looks for a smaller range in which the gradient appears to have converged.

3. The stopping tolerances. They should be tighter if one wants to get closer to the optimum (at the expense of more CPU time).

4. The set of starting points. Options include, for example, randomly generated points, or points provided by human designers.

5. A reformulation of the search space. One might, for example, constrain a parameter to be at its upper or lower bound, or force a constraint to be active.

We automated the last two of these decisions using our prototype selection and reformulation selection techniques. The first three decisions were made manually.

It might be possible to build a system that makes the decision regarding the number of multistarts dynamically by running optimizations from different random starting points until it detects that a large number of them are converging to the same optimum. The choice of gradient step sizes could be chosen by a system that runs convergence studies, and "looks" for the region of convergence. These decisions are both directions for future research.

The choice of stopping tolerances depends of the preferences of the human user (between speed and accuracy), and so can not be fully automated. One might be able to build a system that provides the user with estimates of how long the optimization will take and how close to the optimum it will get for different stopping tolerances, in order to aid the user in making this decision.

Intervention needed to set up our techniques

Another way to look at the amount of human intervention needed is to ask how much human effort is needed in setting up each of the techniques described in this thesis for a new domain.

Inductive-learning based techniques For the inductive-learning based techniques — prototype selection, prototype synthesis, and reformulation selection — the human user must perform the *feature selection*. That is, he or she must find a way to map the space of all possible goals into a finite set of continuous or discrete features that can then be used by the inductive learner — C4.5 or CART — for learning. In the yacht domain, we chose to restrict the set of racecourses to those with one or two legs, since it was not clear how to represent racecourses with an arbitrary number of legs using a finite set of features. A direction for future research is to use inductive logic programming to perform inductive learning on a more complex goal space that could, for example, include racecourses with an arbitrary number of legs. But in that case, the user would still have to formulate an inductive logic representation of the goal

space.

Model constraints The human user must determine what modeling assumptions are made by the simulator, must figure out a way to represent these constraints as non-linear inequality constraints, and must encode these constraints as machine-executable functions, such as C functions. Automating this process is a challenging area for future research.

Rule-based gradients We used the same set of rules across several domains — yachts, airframes, nozzles, supersonic inlets, and hypersonic inlets. Once the rules were integrated into the optimizer, there was no additional human effort needed to use them in additional domains. We believe that our existing set of rules will be applicable across a wide range of problems having unevaluable points. However, there may exist some domains in which additional rules are needed. Formulating and encoding such additional rules would require human effort.

Multilevel design The choice of decomposition of the design parameters into abstraction levels, and the choice of a simulator for each abstraction level, must be made by the human user. Automating these decisions is another challenging area for future research.

8.2.3 Generality

We have shown that our techniques are general by applying each one of them in two realistic engineering domains. Table 8.2 is a reproduction of Table 1.2 from the introduction chapter, with some additional information. In this table, a check ($\surd$) indicates that we have tested the given technique in the given domain, and have included experimental results in this thesis demonstrating that the technique enhances optimization performance. Notice that there are two checks in each row of the table, indicating that we have demonstrated that each technique is effective in two realistic domains.

A circle ($\circ$) in the table indicates that we have used the given technique in the given domain, and that we believe that it is helpful in that domain, although we have not

Table 8.2: The domains in which our techniques are applicable.

technique	racing yachts	supersonic airframes	supersonic nozzles	supersonic inlets	hypersonic inlets	automobiles
prototype selection	√	√	△	NA	NA	△
reformulation selection	√	√	△	NA	NA	△
model constraints	∘	√	√	∘	∘	△
rule-based gradients	∘	√	∘	√	∘	△
multilevel design	△	√	√	△	NA	△

√ Technique has been applied successfully; numerical results included in this thesis.
∘ Technique has been applied; no numerical results available.
△ We conjecture that the technique could be successfully applied.
NA The technique is not applicable.

included in this thesis empirical evidence to show that the technique is helpful in the domain. In these cases, we have performed optimizations using the given technique, and we have obtained good optimization performance, but we have not had a chance to compare empirically optimization performance with and without the technique. In the yacht domain, we used a model constraint to encode the constraint that the boat not sink, and we used rule-based gradients to handle the unevaluable points that result when the solver inside VPP fails to find a solution to the balance-of-forces equations. In the nozzle domain, we used rule-based gradients to handle the unevaluable points that result when the simulator crashes, although we are not sure why it crashes. In the supersonic inlet domain, we used model constraints to encode a set of constraints regarding which sets of design parameters map into physically realizable inlet geometries. In the hypersonic inlet domain, we used a model constraint to to ensure that the given design parameters map into a physically realizable geometry, and we used rule-based gradients to handle cases where the Navier-Stokes code crashes. (The hypersonic inlet domain is further described in Appendix B.)

A triangle (△) in the table indicates that we believe the technique could be helpful in the given domain, although we have not tested it. Chapter 6 describes how our multilevel optimization technique could be applied to the yacht and inlet domains. We believe that prototype selection and reformulation selection could be applied to the nozzle domain in a way similar to the way in which we applied them to the airframe

domain, using a similar space of missions, and forming reformulations based on the activity of the upper and lower bounds on the design parameters.

The letters "NA" in the table indicate that the technique is not applicable in the given domain. Prototype selection and reformulation selection require that there be a space of possible "goals" or "missions" for which the artifact is to be designed. Such is generally the case when designing an entire vehicle, but often is not the case when designing a single component. If the available simulators only simulate one component at one point in the mission, then it is not possible to simulate an entire mission. Instead, the designers use some heuristics to choose a goal for the component that does not directly depend on the mission. For example, when designing hypersonic inlets, our goal was to minimize static pressure distortion. Because we did not have a space of goals or missions, prototype selection and reformulation selection were not applicable to our inlet design work.

We only had one, very expensive simulator for hypersonic inlets available to us. We are not sure if it would be possible to develop a less expensive, but still reasonably accurate simulator for this domain that could use a more abstract design space, so we are do not know if the multilevel design technique would be applicable to hypersonic inlets.

One might ask whether our techniques are applicable to domains beyond the ones that are listed in the first five columns of table Table 8.2 — the domains in which we have conducted our experiments. We believe that our techniques are applicable to a large number of engineering design domains. In particular, each technique is applicable to domains that have the pathologies listed in the relevant row of Table 8.1. The techniques will only be helpful if the amount of these pathologies is not at either extreme, but even in the extreme cases, the techniques will not hurt optimization performance very much.

If there is little or no pathology, then optimization with our techniques will work, but it will work almost as well without our techniques. Using inductive learning to select prototypes or reformulations will result in good optimization performance, but

optimization performance will be almost as good using the default prototype or search-space formulation. In the absence of unevaluable points, model constraints and rule-based gradients will never be activated, so optimization performance will be exactly the same as it would be without them. If there is only a single apparent optimum, then using a multilevel design space may produce a speedup, but the speedup will not be dramatic.

If there is an extremely large amount of pathology, then optimization will fail with or without our techniques. In the most extreme case, the numbers returned by the simulator will provide no information about the search space, and any optimization method, with or without our enhancements, will be reduced to random guessing. Since optimization will fail with or without our techniques, our techniques are not helpful, but again they are not harmful.

We believe that a large number of realistic engineering problems have the types of pathologies to which our techniques are applicable, and have them in the amounts for which our techniques are helpful. To illustrate how one might go about determining whether our techniques are applicable to a new domain, let us consider how the techniques could be applied to the design of automobiles — a domain in which we have no experience beyond having driven automobiles.

As we mentioned above, prototype selection and reformulation selection require a goal space. For automobile design there could be many possible goals, including different combinations of range, speed, passenger space, noise, safety, fuel economy, and manufacturing cost. It should be relatively straightforward to formulate some subset of this goal space as a set of attributes for inductive learning. Prototype selection further requires that the best choice of starting prototype depend on the goal. We believe that this dependency will hold for any design optimization problem, including automobiles. The optimal design depends of the goal, and it is better for the starting point to be closer to the optimal design, so the best choice of starting prototype should depend on the goal. Reformulation selection further requires that there be a set of available reformulations, and that the best choice of reformulation depend on the goal. In the yacht domain, we formed reformulations based on the activity of portions of the 12

Meter Rule. In the automobile domain, government regulations impose constraints on the design that could be considered analogous to those of the 12 Meter Rule. It may be possible to form reformulations of the automobile design search space based on these regulations.

Optimizing a automobile design would require the use of a complex multidisciplinary simulator. Such a simulator would undoubtedly be based upon numerous modeling assumptions. Some of these assumptions could probably be encoded as model constraints, but it is unlikely that they could all be encoded at a reasonable cost (in terms of human effort), so it is likely that some unevaluable points would remain. Rule-based gradients could be used to handle the remaining unevaluable points.

A full analysis of a automobile would require the use of computational fluid dynamics codes to analyze the aerodynamic properties of the automobile. Such a simulation would be very computationally expensive. It would probably be helpful to also use a simplified, less expensive simulator. The less expensive simulator would not model the full shape of the automobile, so it would have fewer design parameters than the full model. We believe that if the design of the automobile were decomposed into two components — the sheet metal and the rest of automobile — the two decomposed search spaces would be fairly independent, and would therefore have fewer optimizer stopping points than the full search space, so that our multilevel optimization technique could be useful.

8.2.4 Novelty

We believe that our techniques are novel for two reasons. The first is that they are new techniques. The second is that they attack a problem, design optimization using simulators that are both expensive and pathological, that has received little attention. See the related work chapter (Chapter 7) for a further discussion of this issue.

8.3 Analysis of contribution

8.3.1 Contribution to the technology of design

We have presented a number of ways of extending conventional numerical optimization that allow it to be used to optimize complex engineering designs that otherwise could not be automatically optimized — those that require a simulator that is both pathological and expensive. We have demonstrated that complex, multidisciplinary designs can be automatically optimized, when using the appropriate augmentations to existing optimization methods such as CFSQP. Using these augmentations, we have produced designs that have been published in the engineering literature, along with descriptions of our techniques [Gelsey *et al.* 1995, Shukla *et al.* 1996a].

8.3.2 Contribution to machine learning

We applied inductive learning to engineering design optimization. It was not immediately obvious that inductive learning would be applicable to this problem. The problem with which we were faced – to produce a good design for a given goal – did not easily fit into the set of classification-type problems usually attacked using inductive learning. We believed that inductive learning might be appropriate, however, because of the existence of a library of past designs that could be used as training data, and because of the existence of a simulator and optimizer that could be used to generate additional training data as needed.

We discovered that at least two pieces of the numerical optimization problem – prototype selection and reformulation selection – could be formulated as inductive learning problems, and that using inductive learning for these pieces of the problem resulted in improved performance for the overall optimization problem. Chapter 2 and Chapter 3 describe these results. It may be possible to use our methodology for analyzing the steps in setting up an optimization to discover other parts of the numerical optimization problem that could be formulated as inductive learning problems.

During this work we have learned a number of lessons about the practical application of inductive learning to real-world problems. The next three subsections describe what

we have learned about the importance of choosing the right evaluation criteria, the importance of getting good data, and the usefulness of visualization.

Evaluation criteria

We feel it is important to note the importance of choosing the correct evaluation criteria when applying inductive learning in a real-world domain such as complex engineering design, rather than on standard test problems such as those found in the UCI repository. Researchers in inductive learning have traditionally evaluated learning performance on the basis of *error rate*, the percentage of test examples incorrectly classified. We argue that it is more important to evaluate the learners on the basis of the impact that they have in the domain, using whatever evaluation criteria are normally used in the particular domain. For example, a learning system used in a business could be evaluated on the basis of its impact on profits. Since our inductive learning systems were used within an engineering design optimization system, we evaluated performance using the two measures of merit normally applied to design optimization systems: the quality of the resulting design, and the amount of CPU time needed to produce the design.

Further, we believe that using error rate instead of the real-world measure of merit can be misleading. For example, at first we used error rate to evaluate the inductive learners for the reformulation-selection problem. We obtained the results shown in Table 3.1. On the basis of this table, it looked like C4.5 performed substantially better than Most Frequent Class (MFC). When we later compared the learners on actual optimization performance, we obtained the results shown in Table 3.2, showing that C4.5 had only a small advantage over MFC.

The reason error rate is not always a good predictor of optimization performance is that it ignores the differences in how "bad" different wrong answers are. For example, when choosing a prototype, the error-rate measure simply counts how many times the learner chose the best prototype. Choosing the second-best prototype and choosing the worst prototype are counted as being equally bad. Optimization performance may be almost as good when using the second best prototype as it is when using the best prototype, but much worse when using the worst prototype.

The importance of getting good data

When the people setting up the learning system also have control over the generation of training data, as we did in our experiments, it is sometimes possible to improve learning performance by improving the training data. In both sets of yacht-domain learning experiments described in this thesis, we at first obtained poor learning performance. We determined that the training data was "noisy" – that is, many of the "optimal" prototypes were not really optimal for the goals for which they were labeled as being optimal. We determined that the reason for this noise was that the optimizer often got "stuck" before reaching the true optimum. We were able to improve the quality of the training data by making several improvements to the optimizer and to the simulator that was used within the optimizer. At first we were using a yacht simulator called AHVPP; we replaced this with a simulator called RUVPP for which the multidimensional surface defined by the search space and the simulator is smoother. At first we used a simple hill climber as our optimizer; we replaced the hill climber with CFSQP, and then later we further improved CFSQP's performance by adjusting its parameters, and by making the constraint that the yacht not sink into an explicit model constraint.

The lesson regarding AI-augmented design optimization is that for the inductive learning techniques — prototype selection and synthesis and reformulation selection — to be effective, it is necessary to first achieve a certain minimum level of optimization performance. If optimization performance is extremely poor, then there will not be enough useful information in the optimization results for the learner to learn anything. If one is faced with a very pathological design space, then one should first use the non-learning techniques described in this thesis — model constraints, rule-based gradients, and multilevel design — as well as conventional numerical optimization techniques, such as properly setting the gradient step sizes, to reduce the pathology as much as possible, and then apply the learning techniques to further improve optimization performance.

By the time we did our machine learning experiments in the airframe domain, we had learned this lesson. We augmented CFSQP with model constraints and rule-based gradients before we collected training data for learning, and we obtained good learning

results on the first attempt.

The value of visualization

We have found that visualization [Tufte 1983] can be very helpful in determining the noisiness of the training data. For example, Figure 3.5 shows the training data from which we learned the activity of the beam constraint for reformulation selection. In this figure, there is one point labeled as "inactive" that is completely surrounded by points that are labeled as "active." We believe that the activity of this constraint should be monotonic in the two goal parameters; we therefore believe that this point is noise. C4.5 has the ability to deal with limited amounts of noise, so this one noisy point did not cause a problem. However, in our earlier experiments we discovered through visualization that our training data was much noisier.

Visualization of the training data can also be used to choose an appropriate inductive learning algorithm. For example, as we mentioned in Chapter 3, C4.5, like most decision-tree learners, uses linear, axis-parallel cuts in its decision trees. However, Figure 3.5 shows that the space is clearly divided into two regions. The border between these regions does not appear to be axis parallel, and appears to be nonlinear. This suggests that better performance might be achieved using an "oblique" decision tree learner, or by attempting to learn nonlinear region boundaries.

8.3.3 Contribution to numerical optimization

We have tested the use of rule-based gradients for engineering design optimization, but we believe that they would be useful for any type of gradient-based optimization in cases where the objective function or constraint functions have unevaluable points. For example, within a simulator for a vehicle there may be an optimizer that simulates the behavior of a human operator. It assumes that the human operator will always control the vehicle in the way that minimizes fuel consumption subject to the constraint that a certain mission be completed. The objective function in this optimization — the cost function — may have unevaluable points, for the same reason that the simulators we used in this thesis have unevaluable points. We believe that rule-based gradients could

be used to augment the optimizer within such a simulator, and that doing so would increase the reliability of the simulator.

8.4 Limitations and Future work

We have shown that our techniques can provide large improvements in optimization performance, but one might ask whether it would be possible to do even better. We know that it is not possible to construct an optimization technique that will always find the optimum in any domain, because the nonlinear optimization problem is undecidable (see Appendix C for a proof). However, there are still numerous directions for future work.

One direction for future work would be to try to improve each of the five techniques described in this thesis. Specific ways of improving these methods are described in the "Future Work" sections of each chapter in the main body of this thesis. These potential improvements include using background knowledge to enhance learning performance in prototype selection and synthesis, and designing and testing more rules for rule-based gradients.

Another direction for future work is to test the techniques in additional domains. The experiments described in this thesis were all performed using relatively inexpensive simulators. These simulators have CPU time requirements ranging from 1/4 second (on a DEC Alpha 250 4/266 desktop workstation) to analyze an airframe with "simulate," to 10 seconds (on the same workstation) to analyze an inlet with NIDA. With simulators as inexpensive as these, it is possible, given a few CPU weeks, to perform optimization without using the techniques described in this thesis. We chose these domains to use for these experiments precisely because it is possible to perform optimization without our techniques; this choice has allowed use to compare experimentally optimization performance with and without our techniques. We see these relatively inexpensive simulators as models for far more expensive simulators with which it may not be possible to do optimization without using the techniques that we have described.

In fact, we have used two of our techniques — model constraints and rule-based

gradients — to perform optimization using a much more expensive simulator, GASP. These optimizations were performed in the domain of hypersonic inlet design. Each simulation took between two and six hours of CPU time (on a DEC Alpha 250 4/266 desktop workstation), and each optimization took weeks of CPU time. The search spaces that we used contained some unevaluable points, which were handled using the techniques described in Chapter 4 and Chapter 5. These optimizations produced inlet designs that are better than the previously best-known designs, and have been published in the aeronautical engineering literature [Gelsey *et al.* 1995, Shukla *et al.* 1996a]. Empirically testing optimization in these domains without our techniques would have been prohibitively computationally expensive, but we believe that the optimizations would not have succeeded without using our techniques. Some of the details of these optimizations can be found in Appendix B.

Another possible way to test the techniques further would be to construct an inexpensive, artificial domain. This domain could be constructed in such a way that the experimenter can easily vary the level of various pathologies. It might have "knobs" to control the number of local optima, the amount of "noise," the number of "ridges," the number of unevaluable points, and the number of discontinuities in the objective function. Such an artificial domain could be used to test the impact of each technique in remedying each of the pathologies in Table 8.1. This table could be further refined by describing subclasses of each of the types of pathologies listed. For example, nonsmoothness could be broken down into discontinuities in the objective function, and discontinuities in the first derivative of the objective function. The experimenter could then use the artificial domain to determine the impact of each technique on each of these more specific types of pathologies. These experiments might inspire the development of additional techniques to remedy the pathologies.

This thesis presents techniques for handling various pathologies that tend to occur in realistic simulators. One question to ask is when given a simulator, how should one determine what pathologies it has? One can try to answer this question by using visualization techniques to examine the search space defined by the simulator, and looking for evidence of multiple optima, nonsmoothness, or unevaluable points. Automatically

determining which pathologies exist is a direction for future research, but fortunately it is not necessary to answer this question in order to use the techniques described in this thesis. These techniques help optimization performance in the presence of pathologies, but they do not significantly hurt performance when there are no pathologies. For example, an optimization method augmented with rule-based gradients will perform exactly the same as the basic optimization method when there are no unevaluable points in the search space. Similarly, if inductive learning is used to select a starting prototype in a search space that is not pathological, then the speedup will be smaller than it would be in a pathological space, since the learning method will not provide the advantage of choosing a portion of the search space that avoids the pathologies, but there still will be a speedup, since the learning method will still provide the advantage of choosing a starting point that is closer to the optimum. Thus, if the user is not certain whether or not the search space contains pathologies, the safe approach is to always use the techniques described in this thesis.

Finally, the experiments described in this thesis tested each of the five techniques individually, and tested some combinations of techniques (such as rule-based gradients combined with model constraints). It would be interesting to test additional combinations of techniques, to see how the various techniques interact with each other, and to determine how close the speedup obtained using multiple techniques is to the product of the speedups obtained using the individual techniques.

Appendix A

Background

A.1 Introduction

This appendix provides background on several subjects that are needed to understand this thesis. Section A.2 covers nonlinear optimization, Section A.3 covers inductive learning, and Section A.4 covers the engineering domains in which the techniques described in this thesis were tested.

A.2 Nonlinear Optimization

Nonlinear optimization [Vanderplaats 1984, Moré and Wright 1993, Gill *et al.* 1981] involves the minimization[1] of a nonlinear[2] *objective function*. One approach to nonlinear optimization is to use *steepest descent hillclimbing*. In this approach, the optimizer computes the gradient[3] of the objective function, moves in the gradient direction (the direction of steepest descent) until the objective function can not be reduced any further, and then repeats. More sophisticated approaches to nonlinear optimization, known as *quasi-Newton* methods, compute an approximation of the inverse of the Hessian[4], which is updated using the gradient, and use this quasi-inverse Hessian to perform Newton's method in multiple dimensions.

A *nonlinear programming* problem consists of a nonlinear *objective function* to be

[1] To instead *maximize* the objective function, just multiply it by -1 and minimize.

[2] A nonlinear function is one that cannot necessarily be represented using a first-order polynomial.

[3] The *gradient* of a function is the vector of first partial derivatives of the function with respect to its inputs.

[4] The *Hessian* of a function is the two-dimensional matrix of second partial derivatives (mixed derivatives) of the function with respect to its inputs.

minimized, and a set of nonlinear *constraints*. It can be expressed as

$$\min f(\mathbf{x}) \tag{A.1}$$

$$\text{subject to } \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \tag{A.2}$$

where $\mathbf{x}$ is a vector, representing the variables, f is a scalar function, representing the objective function, and $\mathbf{g}$ is a vector function, representing the constraints. A particular set of values for the vector $\mathbf{x}$ is known as a *point*. A point that satisfies the constraints is known as a *feasible point.*

A *quadratic programming* problem is a special case of a nonlinear programming problem in which the objective function f is quadratic, and the constraint function $\mathbf{g}$ is linear. The *sequential quadratic programming* (SQP) method solves a nonlinear programming problem by solving a sequence of quadratic programming problems as follows:

1. fit a quadratic program to the nonlinear program
2. solve the quadratic programming problem
3. perform a minimization along the line defined by the current point and the minimum of the quadratic programming problem
4. repeat

The experiments described in this thesis used a state-of-the-art implementation of SQP called CFSQP [Lawrence *et al.* 1995]. [5]

Fitting a quadratic program to the objective function can be done by computing the Hessian of the objective function f with respect to $\mathbf{x}$, and the gradient of each constraint function g_i with respect to $\mathbf{x}$. Since computing the Hessian is expensive, an approximation of the inverse of the Hessian, known as the *quasi inverse Hessian*, is used. The quasi inverse Hessian is updated on each iteration using the gradient of f with respect to $\mathbf{x}$. In CFSQP, this update is done using the Broyden-Fletcher-Goldfarb-Shanno update formula [Gill *et al.* 1981]. SQP is thus a *quasi-Newton* method.

[5] CFSQP stands for "C code for Feasible Sequential Quadratic Programming."

Solving a quadratic programming problem is a much easier task than solving an arbitrary nonlinear programming problem. CFSQP uses the package QLD, an implementation of Powell's method of solving quadratic programming problems, to solve the quadratic programming problem at each iteration.

CFSQP terminates when one of two conditions is met. The first condition is that the solution of the quadratic programming problem is within a certain tolerance of the current point. This means that, according to the quadratic approximation, the current point is approximately the minimum. The second condition is that the improvement in the objective function during the line minimization is less than a certain tolerance.

Gradient-based nonlinear optimization methods such as CFSQP work reasonably well when the objective and constraint functions satisfy the following assumptions [Gill *et al.* 1981]:

- they are smooth (the objective and constraint functions and their first derivatives are continuous)
- they are defined everywhere
- they have only one local optimum (which is the global optimum)

Unfortunately, these assumptions do not always hold for functions defined by realistic simulators (see Section 1.4).

In a space with multiple local optima, the best that (plain) nonlinear optimization can do is to find a local optimum. We therefore often enhance nonlinear optimization by doing a *random multistart*. This technique involves generating random starting points in the space defined by the upper and lower bounds on each design parameter, and performing optimizations from each starting point, for a fixed number of iterations, or until some termination criterion is met. The best point found in any of these optimizations is then taken to be the "optimum" produced by the multistart. This method tends to find the global optimum rapidly if the global optimum has a large "basin of attraction."

```
outlook=sunny:
| humidity <= 75: Play
| humidity > 75: Don't Play
outlook = overcast: Play
outlook = rain:
| windy = true: Don't Play
| windy = false: Play
```

Figure A.1: Example of a decision tree for the training set of Table A.1.

A.3 Inductive Learning

The problem addressed by an inductive-learning [Weiss and Kulikowski 1991] system is to take a collection of labeled *training* data and produce rules that make accurate predictions on future data. Each item in the training data is labeled with a set of *attributes* and a *class*. The inductive learner attempts to learn a mapping from the attributes onto the class. After training, when presented with an unseen attribute set, the learner predicts the class.

A.3.1 C4.5

C4.5 is an inductive learning system developed by Quinlan [Quinlan 1993]. It learns decision trees from the training data. It selects an attribute on which to split the data that produces the greatest *information gain.*[6] It then splits the training data into two or more subsets based on this attribute, and then recursively runs this algorithm on each of the subsets. This recursive procedure terminates when it runs out of training data, at which point it has a tree that correctly classifies all of the training data (unless some case in the training data appears twice with two different classes). C4.5 then does *pruning*, in order to remove branches of the tree that apply to only a small portion of the training data. Pruning can avoid "overfitting" by allowing C4.5 to ignore noise in the training data.

The following example of how C4.5 works is taken from [Quinlan 1993]. Suppose

[6]Information gain is also known in information theory [Shannon and Weaver 1949] as *mutual information.*

Table A.1: A sample training set for C4.5.

Outlook	Temp	Humidity	Windy?	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

we have the training set of Table A.1. This training set has four attributes and two classes. Suppose the first test chosen by C4.5 is **outlook** with three outcomes, **sunny**, **overcast**, and **rain**. The training data is divided into three corresponding sets. The **overcast** set contains only cases of class **Play**, so it does not need to be further divided. If the **sunny** set is divided based on the test **humidity** $\leq$ **75**, and the **rain** set is divided based on the test **windy=true**, then the result is the decision tree of Figure A.1, which divides the training data into five sets, each containing a single class.

A.3.2 Cross validation

Machine learning researchers generally avoid training and testing their learners on the same data. If trained and tested on the same data, a learner that simply "memorized" all the training data would have a zero error rate,[7] although it might have very poor performance when tested on unseen data. Cross validation [Weiss and Kulikowski 1991] is a way of evaluating an inductive learner when only a limited amount of training and testing data is available. Cross validation works as follows:

1. randomly divide the available data into n subsets

[7] Error rate is the percentage of examples that are classified incorrectly during testing

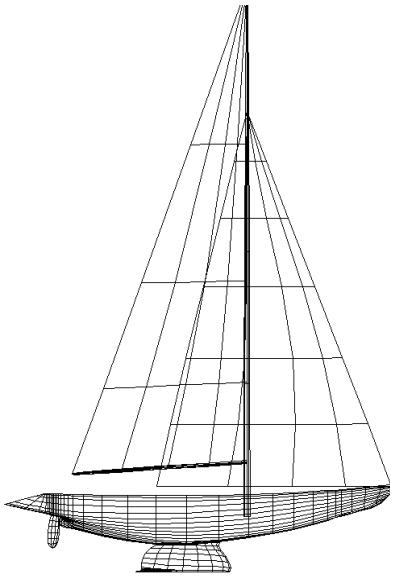

Figure A.2: *Stars & Stripes '87*, winner of the 1987 America's Cup competition

2. for each subset S_i, train the learner on all data except S_i, and then test it on S_i

3. compute the error rate by averaging over the n trials.

This procedure is known as "n-fold cross validation." Sometimes this process is repeated m times, with the data divided into different random subsets each time. It is then called "m trials of n-fold cross validation."

A.4 The Domains

The techniques described in this thesis have been experimentally tested in several realistic, complex engineering domains. These domains include racing yachts, and three parts of a supersonic aircraft: the airframe (wings, fuselage, etc.), and the inlet and exhaust nozzle of the engine.

A.4.1 Yachts

Our experiments in the yacht domain were performed within the class of 12-meter racing yachts. This class was, until recently, the class of sailboats raced in America's

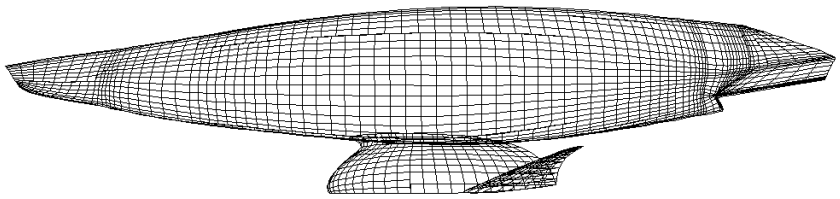

Figure A.3: The hull and keel of *Stars & Stripes '87*.

Cup competitions.[8] An example of a 12-meter yacht is the *Stars & Stripes '87*, which is shown in Figure A.2; a close-up of its hull and keel is shown in Figure A.3. [9]

Racing yachts can be designed to meet a variety of objectives, such as course time or cost. In our work we have chosen to focus on a course-time goal, namely minimizing the time it takes for a yacht to traverse a given race course under given expected wind conditions. A particular course-time goal thus requires the specification of two environment parameters: (1) the race course, represented as a set of (*distance*, *heading*) pairs; and (2) the wind speed, represented as a scalar number, in knots. We used two simulators to analyze yachts. The first is a "Velocity-Prediction Program" called "AHVPP" from AeroHydro, Inc., which is a marketed product used in yacht design [Letcher 1991]. It takes as input a set of surfaces, which are represented within our system using B-splines [Rogers and Adams 1990]. It requires about 10 seconds of CPU time on a Sun SPARCstation 2. The second simulator is a simplified version of AHVPP, called RUVPP, which was developed at Rutgers and represents a yacht geometry by a set of design parameters. It requires about three seconds of CPU time on the same SPARC 2.

In our software, yacht designs are modified by operators that manipulate design parameters and/or the B-spline surfaces. A search space is thus specified by providing the parameters and/or surfaces that define an initial prototype, and a set of operators for modifying that prototype. In the experiments described in this thesis, the following

[8] In 1992, the 12-meter class was replaced with the new America's Cup Class.

[9] This is the boat that won the 1987 America's Cup competition, returning the trophy to the United States after an Australian win in 1983 [Letcher *et al.* 1987].

design parameters were varied:

1. **Length.** The length of the yacht, as measured along the water line.
2. **Beam.** The maximum width of the yacht at the water line.
3. **Hull Depth.** The maximum vertical distance from the water line to the bottom of the "canoe body"[10] of the hull.
4. **Keel Height.** The height of the keel — the vertical distance from the deepest point on the keel to the point where the keel meets the canoe body of the hull.
5. **Keel Taper Ratio.** The tip chord[11] of the keel divided by the root chord of the keel.
6. **Winglet Span.** The width of the winglets that are attached to the keel.

Although the program we use to compute course time (RUVPP) is a state-of-the-art simulator, it nevertheless suffers from a number of pathologies that make optimization difficult. For example, it will sometimes return a spurious root of the balance-of-force equations that it solves. It may also exhibit discontinuities in the objective function or its derivative ("noise"), due to numerical round-off error, or due to truncation error in the numerical solver used to solve the balance-of-force equations. These deficiencies can produce pathologies in the objective function surface over which the optimization algorithm is moving, including discontinuities in the objective function and its derivative. The optimizer can therefore easily get stuck at a point that appears to be a local optimum, but is nevertheless not locally optimal in terms of the true physics of the yacht design space. There are also pathologies in the search space caused by the constraints of the 12-Meter Rule. The rule specifies sail-area penalties for yachts whose dimensions exceed certain limits. When these penalties become active, they cause a discontinuity in the first derivative of the objective function. These "ridges" cased by

[10] The *canoe body* of the hull is the hull without any appendages (keel, winglets, rudder, etc.).

[11] The *chord* of the keel is its width along the x-axis (the axis going from the front of the yacht to the rear of the yacht). The *root chord* is the chord measured at the point where the keel meets the canoe body, and the *tip chord* is the chord at the deepest edge of the keel.

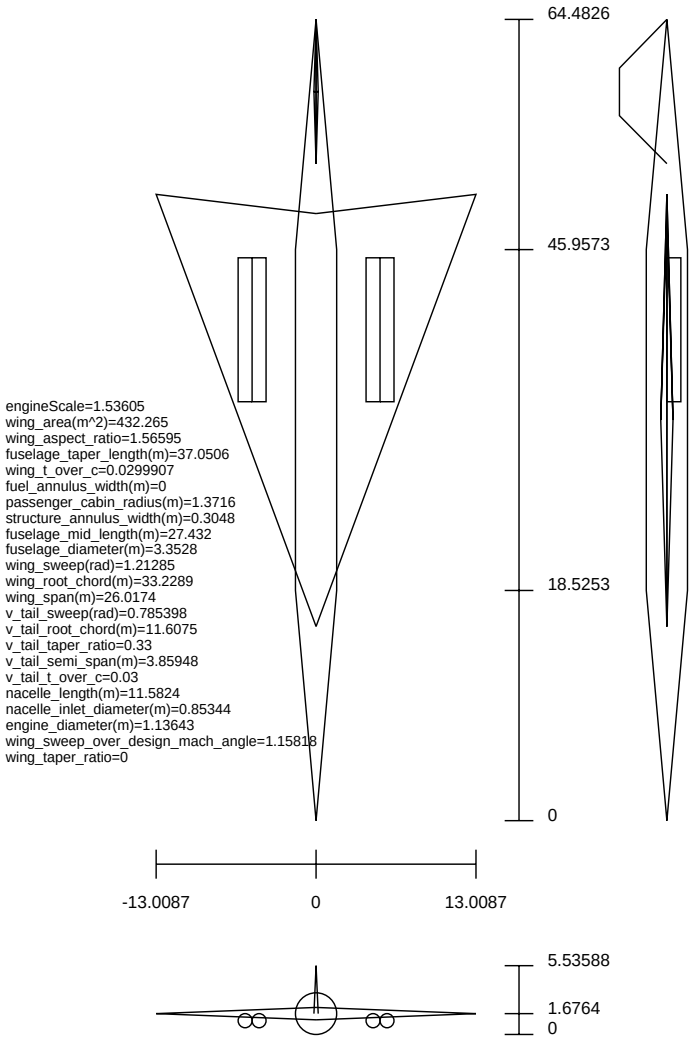

Figure A.4: Supersonic transport aircraft designed by our system

the 12-Meter Rule are discussed further in Chapter 3. Another pathology arises because the numerical root finder will sometimes fail to find a root of the balance-of-force equations, resulting in an unevaluable point.

A.4.2 Airframes and Nozzles

In collaboration with researchers at Boeing and GE aircraft engine, researchers at Rutgers have developed a flight simulator for supersonic transport aircraft. It simulates the entire aircraft, but it focuses on two aspects of the aircraft: the airframe (fuselage, wings, etc.) and the exhaust nozzle on the jet engine. This thesis describes experiments which optimize the airframe separately, the nozzle separately, and both together.

Phase	Mach	Altitude (ft.)	Duration (min.s)	comment
1	0.227	0	5	"takeoff"
2	0.85	40,000	85	subsonic cruise (over land)
3	2.0	60,000	180	supersonic cruise (over ocean)

capacity: 70 passengers.

Figure A.5: Mission specification for aircraft in Figure A.4

Figure A.4 shows a diagram of a typical airplane automatically designed by our software system to fly the mission shown in Figure A.5. The search controller attempts to find a good aircraft conceptual design for a particular mission by varying major aircraft parameters such as wing area, aspect ratio, engine size, etc. using a numerical optimization algorithm. The search controller evaluates candidate designs using a multidisciplinary simulator. In our current implementation, the search controller's goal is to minimize the takeoff mass of the aircraft, a measure of merit commonly used in the aircraft industry at the conceptual design stage. Takeoff mass is the sum of fuel mass, which provides a rough approximation of the operating cost of the aircraft, and "dry" mass, which provides a rough approximation of the cost of building the aircraft. The simulator computes the takeoff mass of a particular aircraft design for a particular mission as follows:

1. Compute "dry" mass using historical data to estimate the weight of the aircraft as a function of the design parameters and passenger capacity required for the mission.

2. Compute the landing mass $m(t_{\text{final}})$ which is the sum of the fuel reserve plus the "dry" mass.

3. Compute the takeoff mass by numerically solving the ordinary differential equation

$$\frac{dm}{dt} = f(m, t)$$

which indicates that the rate at which the mass of the aircraft changes is equal to the rate of fuel consumption, which in turn is a function of the current mass of the

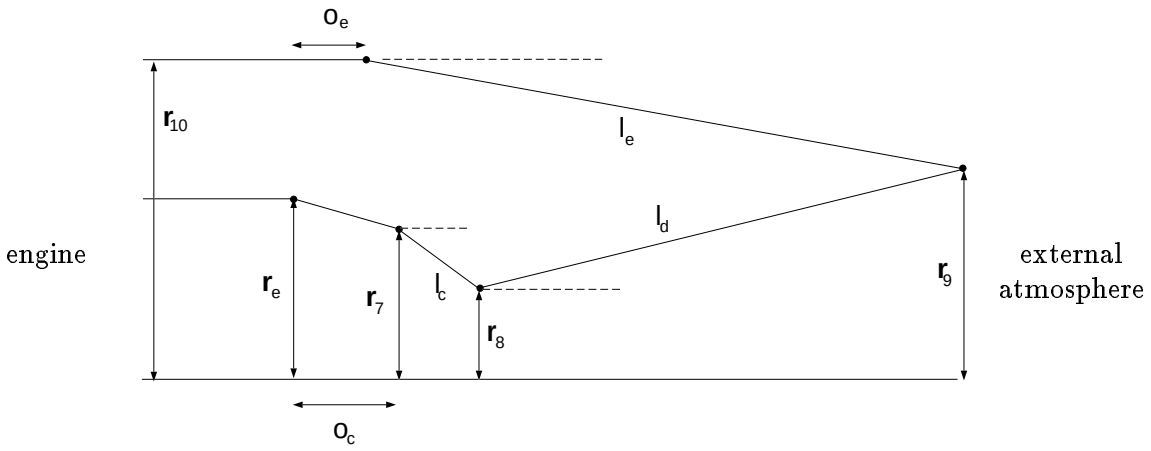

Figure A.6: Axisymmetric convergent-divergent exhaust nozzle (flow from left to right)

aircraft and the current time in the mission. At each time step, the simulator's aerodynamic model is used to compute the current drag, and the simulator's propulsion model is used to compute the fuel consumption required to generate the thrust which will compensate for the current drag.

To test experimentally the techniques described in this thesis, we used a twelve-dimensional design space in which the optimizer varied the following aircraft design parameters over a continuous range of values:

1. engine size
2. wing area
3. wing aspect ratio
4. fuselage taper length (how "pointed" the fuselage is)
5. effective structural thickness over chord (a nondimensionalized measure of wing thickness)
6. wing sweep over design mach angle (a nondimensionalized measure of wing sweep)
7. wing taper ratio (wing tip chord divided by wing root chord)
8. fuel annulus width (the amount of space left in the fuselage for fuel)
9. nozzle convergent flap length (l_c)
10. nozzle divergent flap length (l_d)
11. nozzle external flap length (l_e)
12. nozzle radius 7 length (r_7)

Our optimizations focussed on two aspects of the aircraft: the airframe, which is described by the first eight parameters (see Figure A.4), and the exhaust nozzle, which is described by the last four parameters.

Figure A.6 shows the class of nozzles supported by the current system, the axisymmetric scheduled convergent-divergent exhaust nozzles often found in supersonic aircraft [Mattingly *et al.* 1987]. In Figure A.6, r_{10}, r_e, and r_7 are fixed radii, and r_8 and r_9 are radii which are mechanically varied during aircraft operation. r_{10} is the outer radius of the engine to which the nozzle is attached, r_e is the radius of the duct leaving the engine, r_7 is the radius of the duct at the beginning of the movable convergent section of the nozzle, r_8 is the (variable) radius of the nozzle throat, and r_9 is the (variable) nozzle exit radius. Mechanically, this nozzle is a four-bar linkage, with three movable links labeled in Figure A.6 by their lengths l_c, l_d, and l_e. During aircraft operation, the linkage is moved to change r_8 so that the cross-sectional area at the nozzle throat will produce desired engine performance. Since a four-bar linkage has one degree of freedom, setting r_8 also sets r_9. In the experiments described in this thesis, we allowed the optimizer to vary l_c, l_d, l_e, and r_7.

Our aircraft simulator supports two different ways of simulating the nozzle. The first method takes as input the parameters describing the flap lengths within the nozzle, and simulates the actual operation of the nozzle throughout the mission. The second method uses what is known as an "ideal nozzle." This method does not actually simulate the movement of the flaps within the nozzle, but instead assumes that the nozzle will always produce a certain efficiency. This abstraction of the model allows faster simulation, and does not require the nozzle flap lengths to be input to the simulator. A complete mission simulation requires about 1/2 second of CPU time on a DEC Alpha 250 4/266 desktop workstation when using the four-bar nozzle, and about 1/4 second when using the ideal nozzle.

The aircraft simulator has some pathologies involving small discontinuities, or "noise," in the objective function and its derivative that are caused by truncation error in the numerical solvers within the simulator. Also, most of the designs within our design space violate one or more of the simulator's underlying assumptions, resulting in a

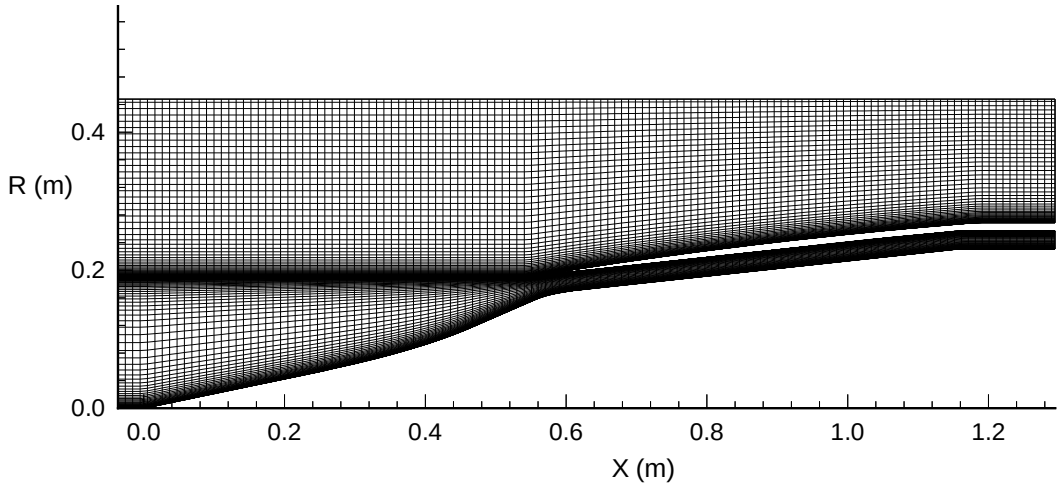

Figure A.7: Mesh for the flow field computation of the missile inlet

large number of unevaluable points. For example, when specifying the flap lengths of a nozzle, it is possible to specify a set of flaps that can not be connected to form a valid convergent-divergent nozzle.

A.4.3 Inlets

The final domain in which we tested the techniques described in this thesis is the design of inlets for supersonic and hypersonic jets. We used three simulators for these inlets. NPARC and GASP are very robust, but very computationally expensive simulators. They require between one hour and one week per simulation. We also used a much faster but much more pathological simulator called NIDA. It takes about 10 seconds per simulation.

The missile inlet in this design work is an axisymmetric mixed compression inlet which cruises at Mach 4 at 60000 feet altitude. Minimum manufacture cost for this inlet is required. Therefore, techniques such as boundary layer bleed and variable geometry are not used. The performance of the inlet will solely rely on the aerodynamic design of the rigid geometry.

The performance of a supersonic missile inlet depends on many factors such as the extent of external and internal compression, contraction ratio, inlet start throat area, throat location, shock train length, divergence of subsonic diffuser, etc.

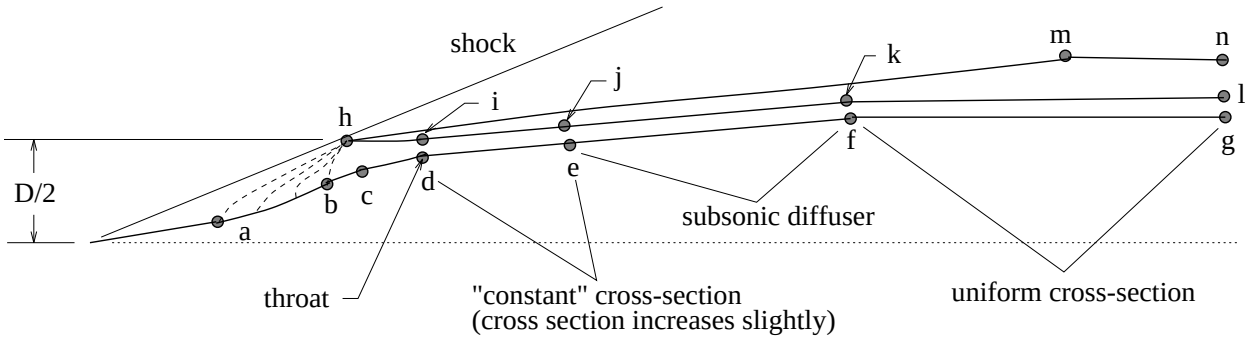

Figure A.8: Supersonic missile inlet geometry model

Table A.2: Fixed Parameters

No.	*Parameter*	*Definition*
1	D	cowl diameter
2	r_f	centerbody radius of constant cross section region
3	x_g	length of inlet for computation
4	x_l	length of inlet for computation ($= x_g$)
5	x_n	length of inlet for computation ($= x_g$)
6	r_m	external diameter

We employed a simple inlet analysis code, NIDA, in the optimization loop. NIDA was developed at United Technology Research Center (UTRC) as an inlet analysis/design tool [Haas *et al.* 1992]. It uses a 1D aerodynamic model with the method of characteristics for the supersonic part upstream of the throat, and empirical correlations based on experimental data downstream of the throat for the region of the terminal shock wave/turbulent boundary layer interaction and subsonic diffuser. After each optimization, we performed a single analysis of the optimized design using GASP. Figure A.7 shows the mesh used to analyze an optimized inlet with GASP.

We defined a set of parameters for the missile inlet geometry to form a design space can be for optimization. Figure A.8 shows the model of the missile geometry which is composed of six fixed parameters and eight design parameters given in Table A.2 and Table A.3, respectively. The missile inlet is axisymmetric and the coordinates are given in terms of axial (x) and radial (r) positions.

The objective function defined by NIDA is very pathological. There are numerous small discontinuities in the function and its first derivative ("noise"), and there are

Table A.3: Parameters to Optimize

No.	*Parameter*	*Definition*
1	θ_1	initial cone angle
2	θ_2	final cone angle
3	x_d	axial location of throat
4	r_d	radial location of throat
5	x_e	axial location of end of "constant" cross section
6	θ_3	internal cowl lip angle
7	H_{ej}	height at end of constant cross section
8	H_{fk}	height at beginning of constant internal cross section

Note: All angles measured positive in counter-clockwise direction

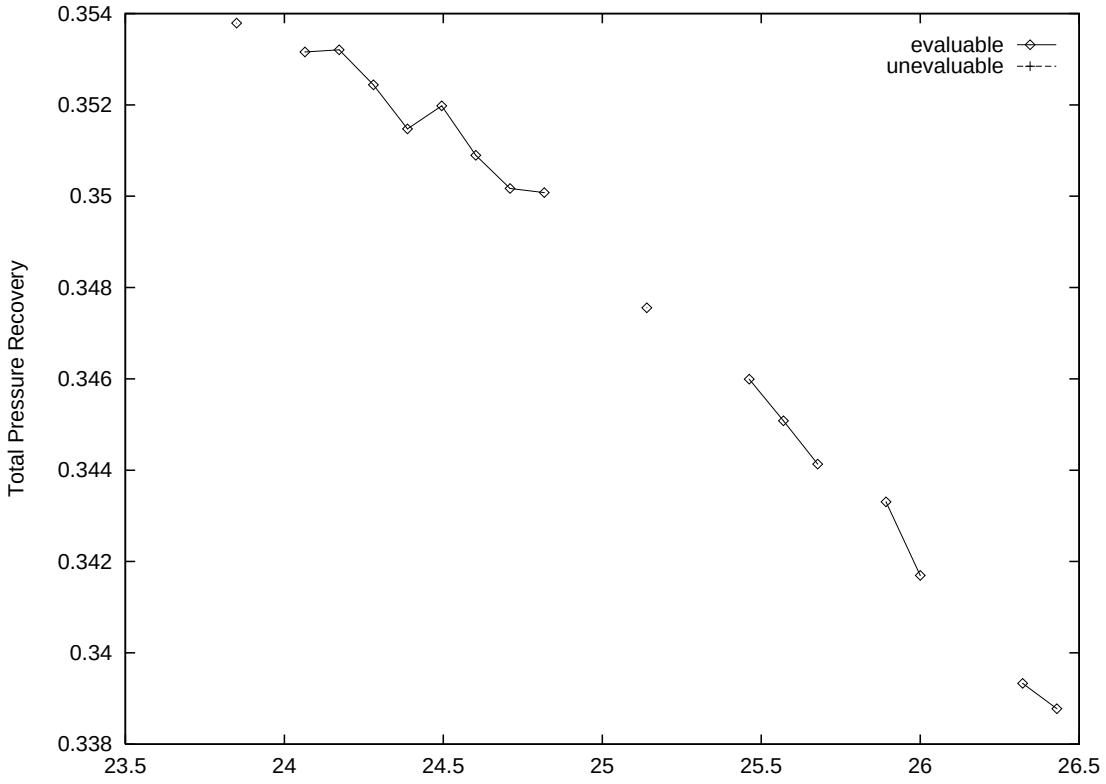

Figure A.9: A cross-section of the search space defined by NIDA.

numerous unevaluable points that result from NIDA crashing or printing an error messages. Figure A.9 shows a cross-section of the search space defined by NIDA, which illustrates these pathologies. The blank regions of the curve are unevaluable points. We don't fully understand what causes the noise and the crashes.

Appendix B

Application to an Expensive Domain

We have used CFSQP augmented with model constraints (see Chapter 4) and rule-based gradients (see Chapter 5) to optimize two hypersonic inlets. These inlets, known as the P2 and the P8, were designed by NASA in the early 1970's. The original inlets were intended to achieve the objective of of canceling the cowl shock, but experimental measurements indicated that a substantial reflected shock was still present. We attempted to minimize this shock using several different measures of merit that were intended to approximate the shock strength. To evaluate the P2 inlet, we used NPARC, which took about two hours per simulation. To evaluate the P8 inlet, we used GASP, which took about five hours per simulation.

To modify the inlets, we used a geometry operator which is illustrated in Figure B.1. The original centerbody of the inlet was replaced with two line segments that are connected by a parameterized curve. We allowed the optimizer to vary x_l, Δx, Δy, and $\Delta\theta$. We used two model constraints to specify that the intersection of the two lines defined by the two line segments must lie between x_l and $x_l + \Delta x$, which ensures that the curve can be fit between the two line segments.

For the P2 inlet, we used a single formulation of the measure of merit — static

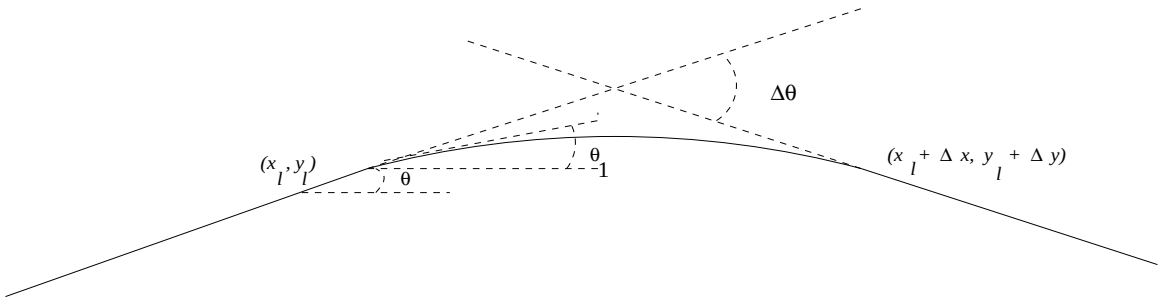

Figure B.1: Parameterized contour for redesign of the inlet centerbody.

Measure	Original	Optimal	Change
σ_p	0.0699	0.0134	**−80.1**%
r_p	2.0053	1.9238	−4.1%
σ_{p_t}	0.3942	0.4014	1.8%
r_{p_t}	0.4957	0.4926	0.6%

Table B.1: Measures of merit for original P2 inlet and best redesigned P2 inlet.

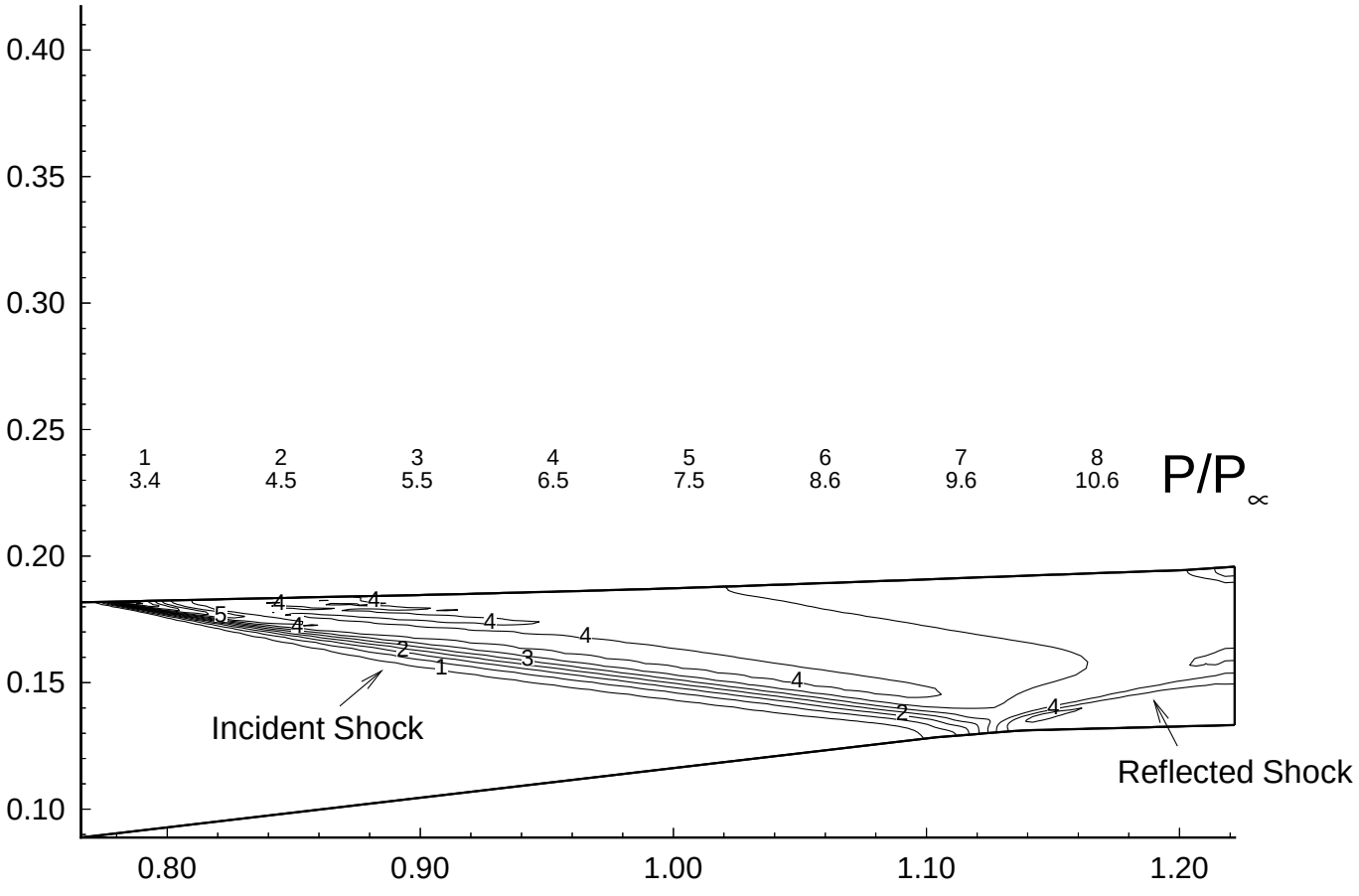

Figure B.2: Pressure P/P_∞ computed by NPARC for original P2 inlet, where P_∞ = 701.4 Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m.

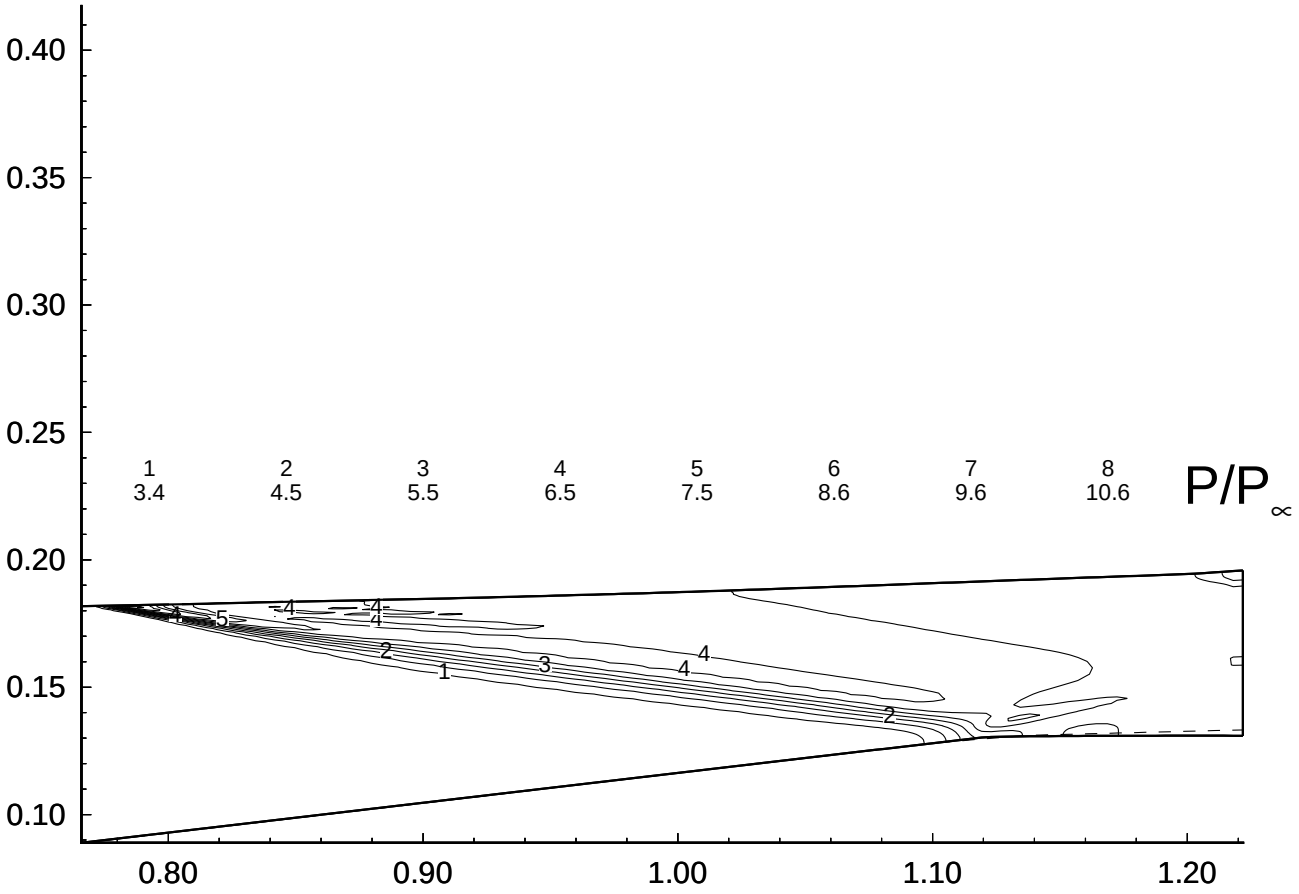

Figure B.3: Pressure P/P_∞ computed by NPARC for optimal redesigned P2 inlet using the σ_p objective function, where $P_\infty = 701.4$ Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m. The original centerbody contour is shown by a dotted line.

Measure name	Original inlet	objective σ_p		objective $\sigma_{\overline{p}}$		objective σ_{sh}	
		optimal	change	optimal	change	optimal	change
σ_p	0.1434	**0.0304**	**-78.8%**	0.0159	-88.9%	0.148	3.2%
$\sigma_{\overline{p}}$	0.1763	0.1226	-30.5%	**0.0603**	**-65.7%**	0.1496	-15.1%
σ_{sh}	0.2688	0.198	-26.3%	0.1102	-59.0%	**0.027**	**-89.9%**
r_p	8.481	7.85	-7.4%	7.503	-11.5%	7.541	-11.1%
σ_{p_t}	0.4605	0.4786	3.9%	0.4803	4.3%	0.465	1.0%
r_{p_t}	0.4505	0.4471	-0.8%	0.4528	0.5%	0.461	2.3%

Table B.2: Measures of merit for starting point P8 inlet and best redesigned P8 inlets using σ_p, $\sigma_{\overline{p}}$ and σ_{sh}. The objective function used during optimization is in bold; other measures are shown for comparison purposes.

pressure distortion, σ_p. We also monitored three other measures of merit to ensure that they did not change too much. Table B.1 shows the measures of merit of the original P2 inlet, and our optimized P2 inlet. Our optimization produced an 80% reduction in static pressure distortion. Figure B.2 illustrates the shock systems in the original P2 inlet, while Figure B.3 illustrates that the shock has been nearly canceled in our optimized P2 inlet.

For the P8 inlet, we tried three different formulations of the objective function as approximate measures of shock strength. Table B.2 shows the effect on the various measures of merit of optimizing each of these three measures of merit. We were able to substantially reduce any of these measures by optimizing it, but minimizing one did not necessarily correspond to minimizing the others. The shock systems in the original P8 inlet are illustrated in Figure B.4. The optimized P8 inlets using three objective functions are shown in Figure B.5, Figure B.6, and Figure B.7. All of these figures show that the shock systems are significantly reduced. Figure B.8 shows a trace of the CFSQP optimization of the σ_{sh} objective function for the P8 inlet, and illustrates how CFSQP performs a sequence of gradient computations, each followed by a line search. Further details of these optimizations can be found in [Gelsey *et al.* 1995, Shukla *et al.* 1996a, Shukla *et al.* 1996b].

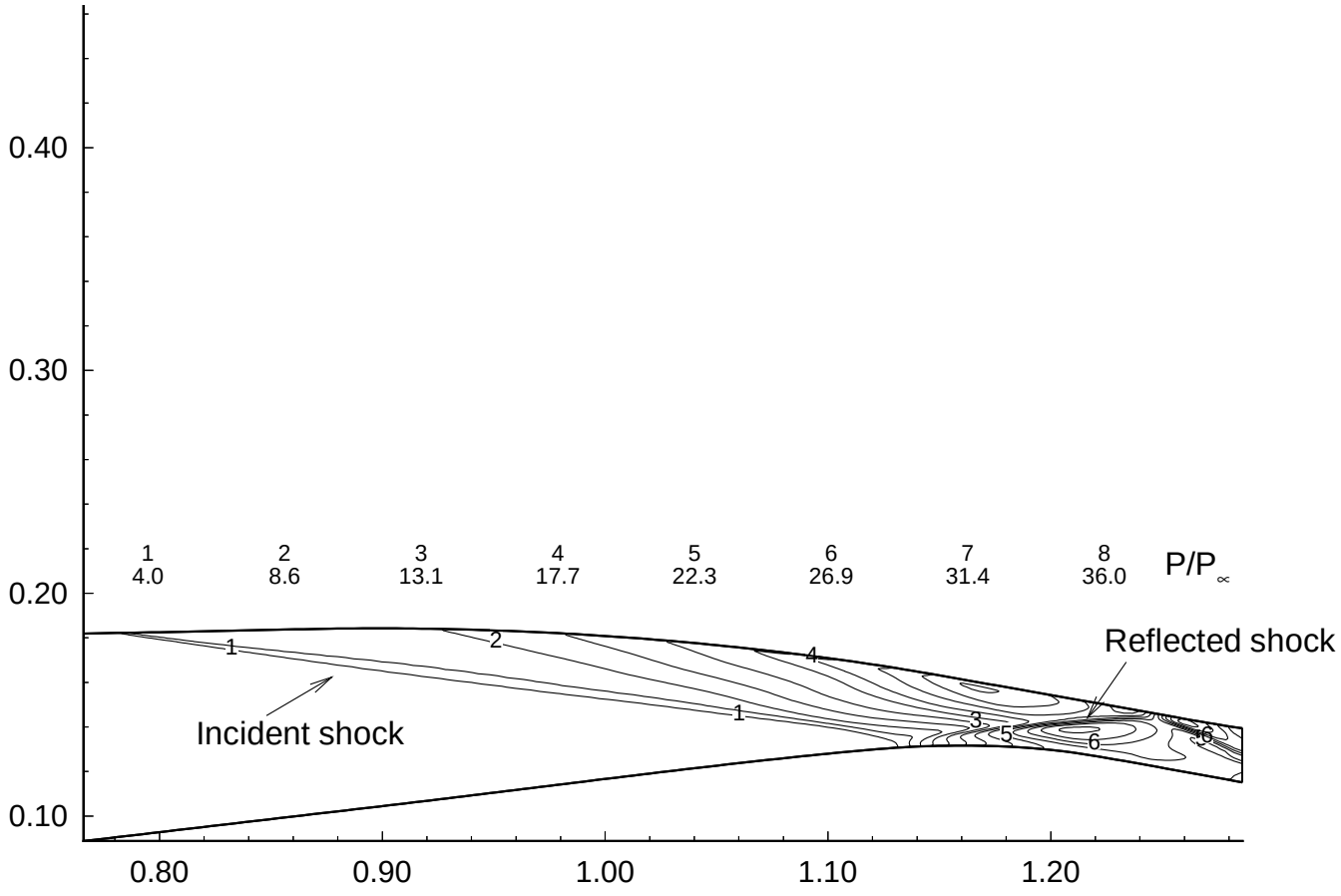

Figure B.4: Pressure P/P_∞ computed by GASP for original P8 inlet, where P_∞ = 701.4 Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m.

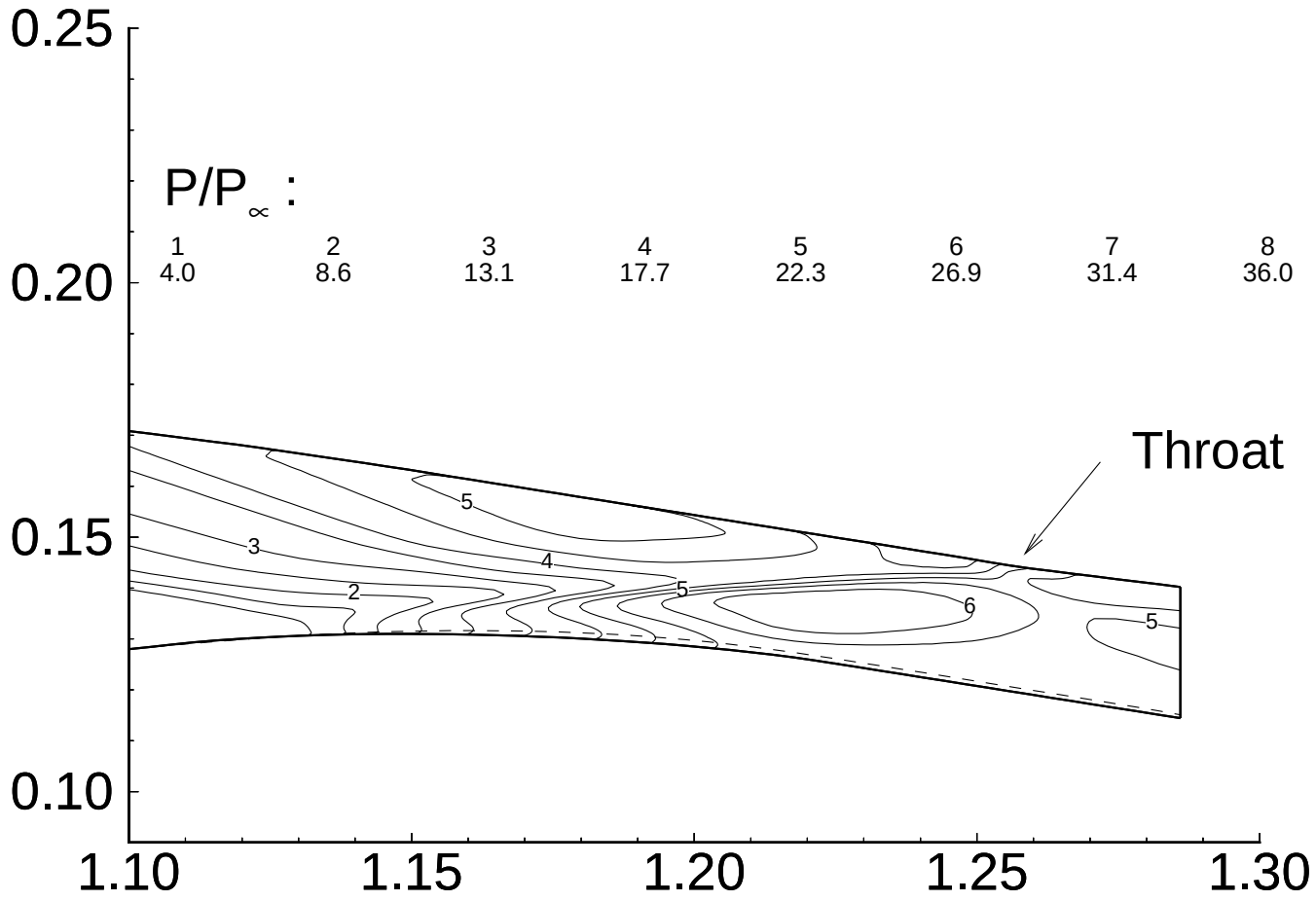

Figure B.5: Pressure P/P_∞ computed by GASP for optimal redesigned P8 inlet using the σ_p objective function, where $P_\infty = 701.4$ Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m. The original centerbody contour is shown by a dotted line.

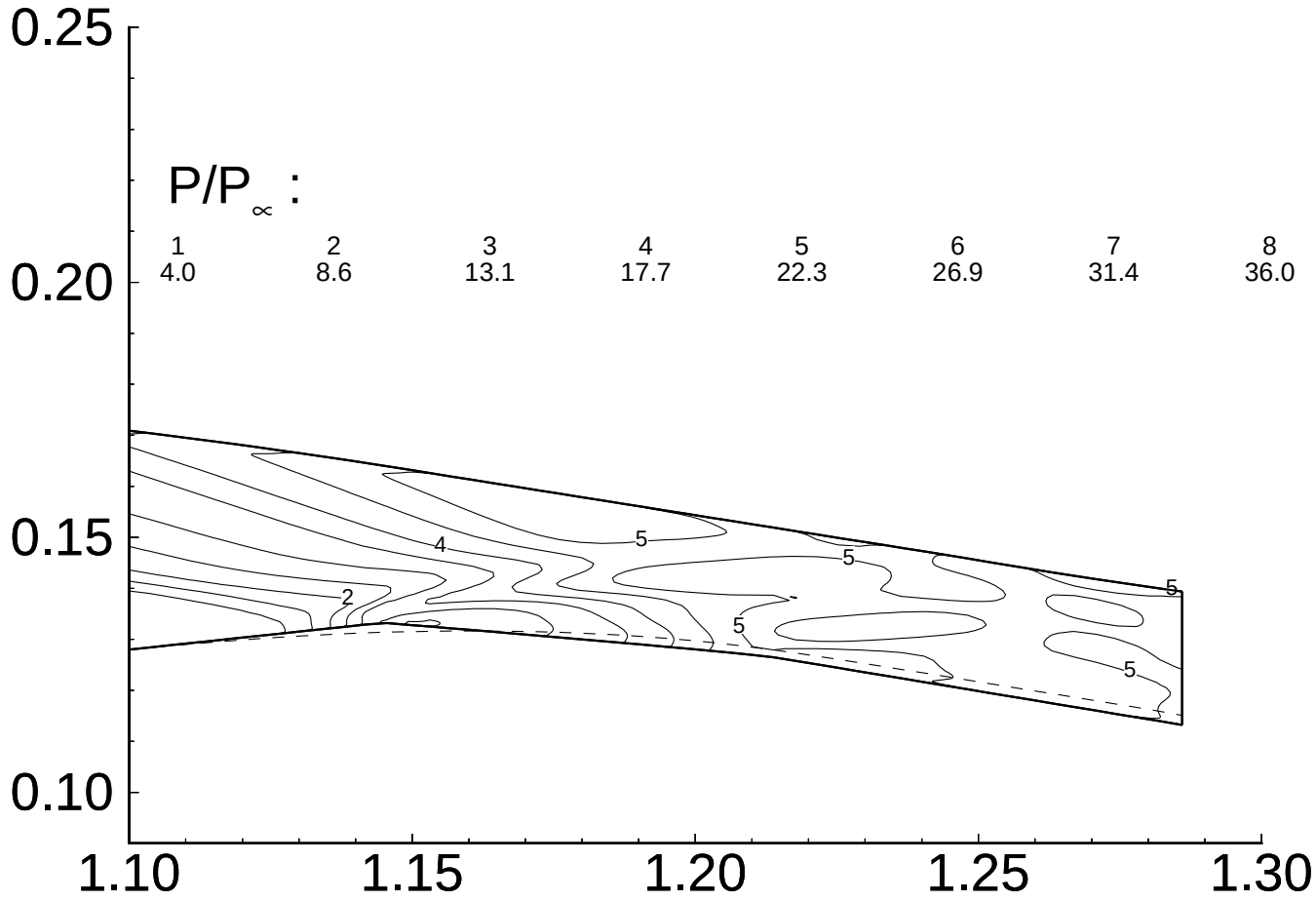

Figure B.6: Pressure P/P_∞ computed by GASP for optimal redesigned P8 inlet using the $\sigma_{\overline{p}}$ objective function, where $P_\infty = 701.4$ Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m. The original centerbody contour is shown by a dotted line.

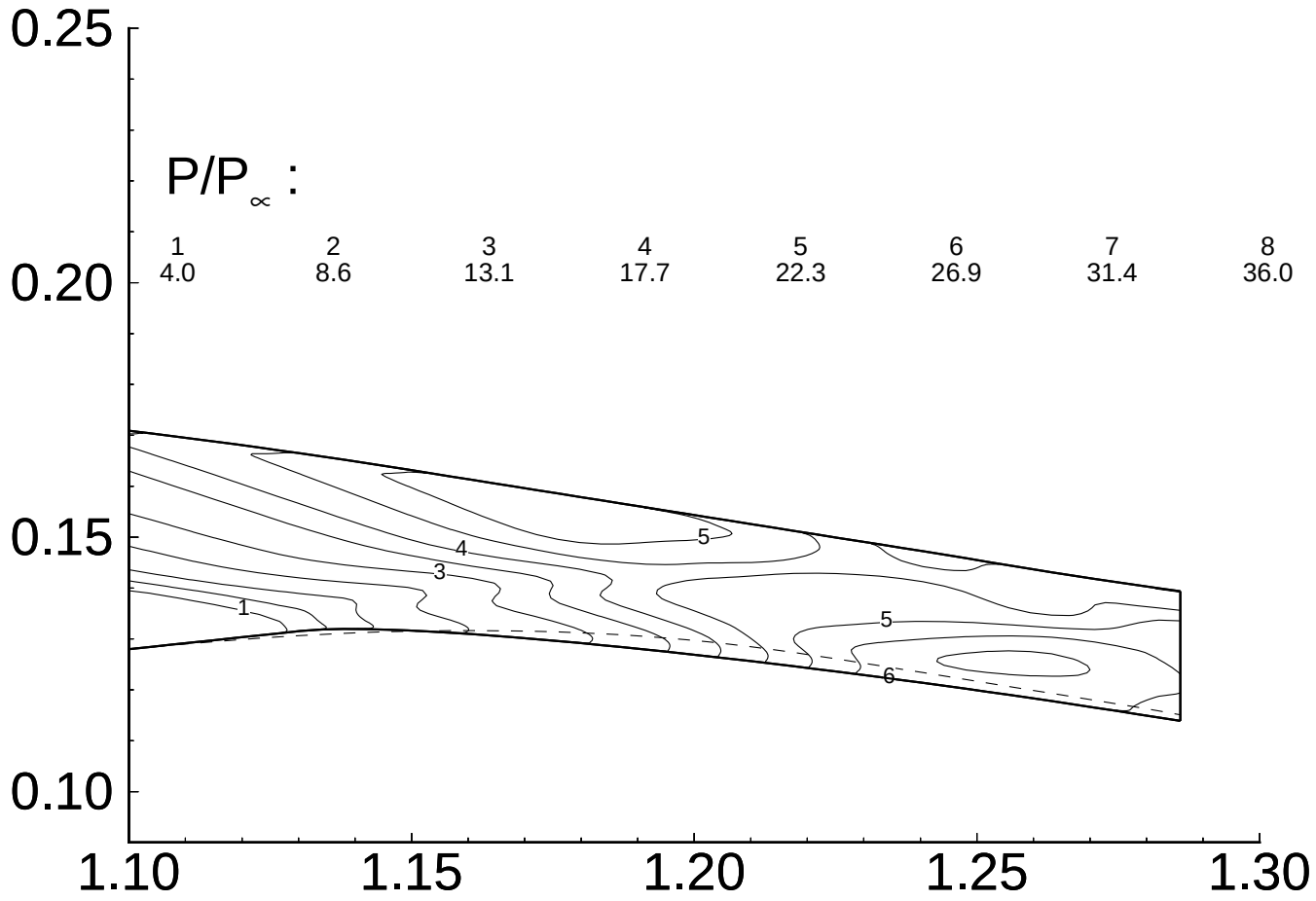

Figure B.7: Pressure P/P_∞ computed by GASP for optimal redesigned P8 inlet using σ_{sh} objective function, where P_∞ = 701.4 Pa (freestream pressure upstream of wedge forebody) and geometric dimensions are m. The original centerbody contour is shown by a dotted line.

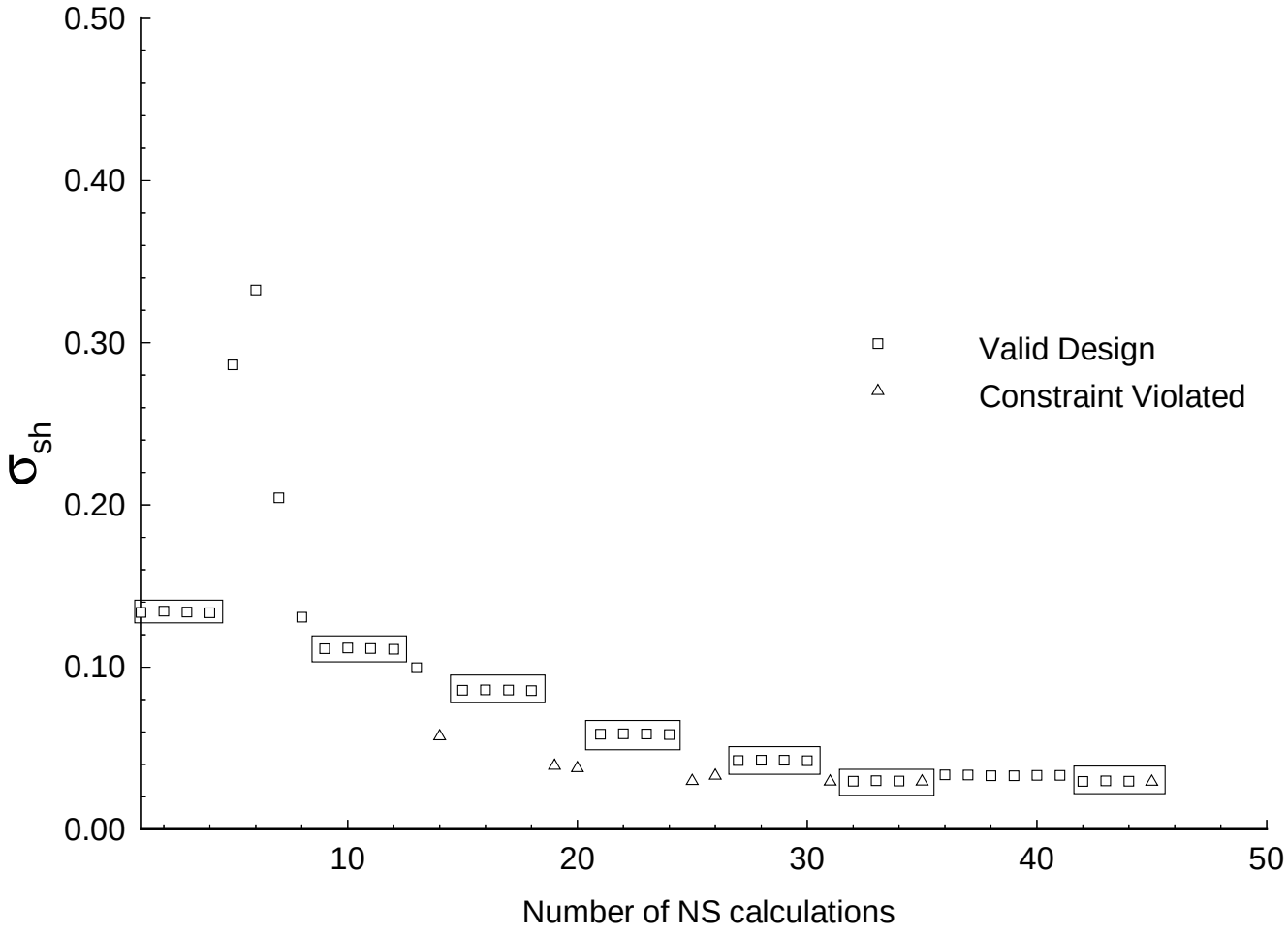

Figure B.8: The change in objective σ_{sh} is shown in terms of the Navier-Stokes calculations. The parameters x_l, Δx and Δy are varied in this optimization. The data points are square for a valid design, triangles for design with constraint violated and boxes show the groups of four Navier-Stokes calculations used in the gradient computation.

Appendix C

The undecidability of nonlinear optimization

C.1 Definition

Given $f(x)$, a function of n variables, and a point $x*$, $x*$ is a *local minimum* of $f(x)$ if there exists a δ such that for all points y within a Euclidean distance δ of $x*$, $f(y) > f(x*)$.

C.2 Claim

The question "Is $x*$ a local minimum of $f(x)$" is undecidable.

C.3 Proof

Given an arbitrary Turing machine T and an arbitrary input to that Turing machine w, we will construct a nonlinear function of two variables $g_{T,w}(x, y)$ such that the point (0,0) is a local minimum of $g_{T,w}(x, y)$ if and only if T does not halt on input w.

The definition of $g_{T,w}(x, y)$ is as follows:

let (r, θ) be (x, y) in polar coordinates.

$$\text{let } i = \begin{cases} 1/\theta & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$$

$$g_{T,w}(x, y) = \begin{cases} -r & \text{if } 0 < \theta \leq 1 \text{ and } i \text{ is an integer and T halts on w in at most } i \text{ steps} \\ r & \text{otherwise} \end{cases}$$

Now, if T does not halt on input w, then $g_{T,w}(x, y) = r$ for all values of x and y, and (0,0) is a local minimum of $g_{T,w}(x, y)$.

If T halts on input w, then there exists some direction θ such that when one moves in direction θ from (0,0), $g_{T,w}(x, y) = -r$, which is a strictly decreasing function. Therefore, (0,0) is not a local minimum of $g_{T,w}(x, y)$.

Hence, if we had a procedure to determine whether a given point is a local minimum of an arbitrary nonlinear function, and we had an arbitrary Turing machine and an arbitrary input, and we wanted to determine whether the Turing machine halted on the given input, then we could construct the above function. We could then use our procedure to determine whether (0,0) is a local minimum of our constructed function, and we would know that our Turing machine halts on the given input if and only if (0,0) is not a local optimum of the constructed function.

But the halting problem is undecidable, so such a procedure must not exist.

QED.

C.4 Corollary

Since it is undecidable whether a given point is a local minimum of an arbitrary nonlinear function, it is clearly not possible to build an optimizer that is guaranteed to find a local minimum (much less the global minimum) of an arbitrary nonlinear function.

References

[Bhatta and Goel 1995] S. Bhatta and A. Goel. Model-based design idexing and index learning in engineering design. In *Working Notes of the IJCAI Workshop on Machine Learning in Engineering*, August 1995.

[Bouchard *et al.* 1988] E. E. Bouchard, G. H. Kidwell, and J. E. Rogan. The application of artificial intelligence technology to aeronautical system design. In *AIAA/AHS/ASEE Aircraft Design Systems and Operations Meeting*, Atlanta, Georgia, September 1988. AIAA-88-4426.

[Bramlette *et al.* 1990] M. Bramlette, E. Bouchard, E. Buckman, and L. Takacs. Current applications of genetic algorithms to aeronautical systems. In *Proceedings of the Sixth Annual Aerospace Applications of Artifical Intelligence Conference*, October 1990.

[Breiman 1984] L. Breiman. *Classification And Regression Trees.* Wadsworth International Group, Belmont, Calif., 1984.

[Cerbone 1992] G. Cerbone. Machine learning in engineering: Techniques to speed up numerical optimization. Technical Report 92-30-09, Oregon State University Department of Computer Science, 1992. Ph.D. Thesis.

[Char *et al.* 1992] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, and S.M. Watt. *First Leaves: A Tutorial Introduction to Maple V.* Springer-Verlag and Waterloo Maple Publishing, 1992.

[Choy and Agogino 1986] J. Choy and A. Agogino. Symon: Automated symbolic monotonicity analysis system for qualitative design optimization. In *Proceedings ASME International Computers in Engineering Conference*, 1986.

[Clark and Matwin 1993] P. Clark and S. Matwin. Using qualitative models to guide inductive learning. In *Proceedings of the tenth international machine learning conference*, pages 49–56. Kaufmann, 1993.

[Dvorak and Kuipers 1991] Daniel Dvorak and Benjamin Kuipers. Process monitoring and diagnosis. *IEEE Expert*, 6(3):67–74, June 1991.

[Ellman and Schwabacher 1993] T. Ellman and M. Schwabacher. Abstraction and decomposition in hillclimbing design optimization. Technical Report CAP-TR-14, Department of Computer Science, Rutgers University, January 1993.

[Ellman *et al.* 1992] T. Ellman, J. Keane, and M. Schwabacher. The Rutgers CAP Project Design Associate. Technical Report CAP-TR-7, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1992. ftp://ftp.cs.rutgers.edu/pub/technical-reports/cap-tr-7.ps.Z.

[Ellman *et al.* 1993] T. Ellman, J. Keane, and M. Schwabacher. Intelligent model selection for hillclimbing search in computer-aided design. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993.

[Forbus and Falkenhainer 1990] Kenneth D. Forbus and Brian Falkenhainer. Self-explanatory simulations: An integration of qualitative and quantitative knowledge. In *Proceedings, Eighth National Conference on Artificial Intelligence*, pages 380–387, Boston, MA, 1990.

[Forbus and Falkenhainer 1992] Kenneth D. Forbus and Brian Falkenhainer. Self-explanatory simulations: Scaling up to large models. In *Proceedings, Tenth National Conference on Artificial Intelligence*, San Jose, CA, 1992.

[Forbus and Falkenhainer 1995] Kenneth D. Forbus and Brian Falkenhainer. Scaling up self-explanatory simulations: Polynomial-time compilation. In *Proceedings, Fourteenth Internationnal Joint Conference on Artificial Intelligence*, Montreal, Quebec, Canada, 1995.

[Gage *et al.* 1995] P. Gage, I. Kroo, and I. Sobieski. Variable-complexity genetic algorithm for topological design. *AIAA Journal*, 33(11):2212–2217, 1995.

[Gage 1994] P. Gage. New approaches to optimization in aerospace conceptual design. Ph.D. Thesis, Stanford University, 1994.

[Gelsey and Smith 1995] Andrew Gelsey and Don Smith. A search space toolkit. In *Proceedings, 11th IEEE Conference on Artificial Intelligence Applications*, pages 117–123, Los Angeles, CA, February 1995.

[Gelsey *et al.* 1995] Andrew Gelsey, Doyle D. Knight, Song Gao, and Mark Schwabacher. NPARC simulation and redesign of the NASA P2 hypersonic inlet. In *31st Joint Propulsion Conference*, San Diego, CA, July 1995. AIAA-95-2760.

[Gelsey *et al.* 1996a] Andrew Gelsey, Mark Schwabacher, and Don Smith. Using modeling knowledge to guide design space search. In J.S. Gero and F. Sudweeks, editors, *Artificial Intelligence in Design '96*, pages 367–385. Kluwer Academic Publishers, The Netherlands, 1996.

[Gelsey *et al.* 1996b] Andrew Gelsey, Don Smith, Mark Schwabacher, Khaled Rasheed, and Keith Miyake. A search space toolkit. *Decision Support Systems, special issue on Unification of Artificial Intelligence with Optimization*, ?(?):?–?, 1996. To appear.

[Gelsey 1991] Andrew Gelsey. Using intelligently controlled simulation to predict a machine's long-term behavior. In *Proceedings, Ninth National Conference on Artificial Intelligence*, pages 880–887, Cambridge, MA, July 1991.

[Gelsey 1995a] Andrew Gelsey. Automated reasoning about machines. *Artificial Intelligence*, 74(1):1–53, March 1995.

[Gelsey 1995b] Andrew Gelsey. Intelligent automated quality control for computational simulation. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, 9(5):387–400, November 1995.

[Gill *et al.* 1981] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, London ; New York, 1981.

[Goldberg 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[Haas *et al.* 1992] M. Haas, R. Elmquist, and D. Sobel. NAWC Inlet Design and Analysis (NIDA) Code, Final Report. UTRC Report R92-970037-1, 1992.

[Hoeltzel and Chieng 1987] D. Hoeltzel and W. Chieng. Statistical machine learning for the cognitive selection of nonlinear programming algorithms in engineering design optimization. In *Advances in Design Automation*, Boston, MA, 1987.

[IYRU 1985] *The Rating Rule and Measurement Instructions of the International Twelve Metre Class.* International Yacht Racing Union, 1985.

[Knight 1994] D. Knight. Survey of high-speed inlet design methodology. Technical Report ARPA Inlet Design Group Report No. 1, Department of Mechanical and Aerospace Engineering, Rutgers University, New Brunswick, NJ, March 1994. http://www.cs.rutgers.edu/hpcd/Area_II.1/report1.ps.Z.

[Kolodner 1993] J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[Kroo *et al.* 1994] Ilan Kroo, Steve Altus, Robert Braun, Peter Gage, and Ian Sobieski. Multidisciplinary optimization methods for aircraft preliminary design. In *Fifth AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, Florida, September 1994. AIAA 94-4325.

[Lawrence *et al.* 1995] C. Lawrence, J. Zhou, and A. Tits. User's guide for CFSQP version 2.3: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical Report TR-94-16r1, Institute for Systems Research, University of Maryland, August 1995.

[Letcher *et al.* 1987] J. Letcher, J. Marshall, J. Oliver, and N. Salvesen. Stars and Stripes. *Scientific American*, 257(2), August 1987.

[Letcher 1991] J. Letcher. *The Aero/Hydro VPP Manual.* Aero/Hydro, Inc., Southwest Harbor, ME, 1991.

[Lowry and McCartney 1991] M. Lowry and R. McCartney, editors. *Automating Software Design.* AAAI Press, Menlo Park, CA, 1991.

[Mattingly *et al.* 1987] Jack D. Mattingly, William H. Heiser, and Daniel H. Daley. *Aircraft Engine Design.* AIAA education series. American Institute of Aeronautics and Astronautics, New York, N.Y., 1987.

[Moré and Wright 1993] Jorge J. Moré and Stephen J. Wright. *Optimization Software Guide.* SIAM, Philadelphia, 1993.

[Murthy *et al.* 1994] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[Orelup *et al.* 1988] M. F. Orelup, J. R. Dixon, P. R. Cohen, and M. K. Simmons. Dominic II: Meta-level control in iterative redesign. In *Proceedings of the National Conference on Artificial Intelligence*, pages 25–30, St. Paul, MN, 1988.

[Papalambros and Wilde 1988] P. Papalambros and J. Wilde. *Principles of Optimal Design.* Cambridge University Press, New York, NY, 1988.

[Peressini *et al.* 1988] Anthony L. Peressini, Francis E. Sullivan, and J. J. Uhl, Jr. *The Mathematics of Nonlinear Programming.* Springer-Verlag, New York, 1988.

[Powell and Skolnick 1993] D. Powell and M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, Univeristy of Illinois at Urbana-Champaign, July 1993. Morgan Kaufmann.

[Powell 1990] D. Powell. Inter-GEN: A hybrid approach to engineering design optimization. Technical report, Rensselaer Polytechnic Institute Department of Computer Science, December 1990. Ph.D. Thesis.

[Press *et al.* 1986] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes.* Cambridge University Press, New York, NY, 1986.

[Quinlan 1990] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Quinlan 1993] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Ramachandran *et al.* 1992] N. Ramachandran, N. Langrana, L. Steinberg, and V. Jamalabad. Initial design strategies for iterative design. *Research in Engineering Design*, 4:159–169, 1992.

[Rogers and Adams 1990] D. Rogers and J. Adams. *Mathematical elements for computer graphics.* McGraw-Hill, 2nd edition, 1990.

[Rogers 1989] J. L. Rogers. A knowledge-based tool for multilevel decomposition of a complex design problem. Technical Report NASA Technical Paper 2903, Langley Research Center, National Aeronautics and Space Administration, Hampton, VA, 1989.

[Sacerdoti 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115 – 135, 1974.

[Sacks 1991] Elisha P. Sacks. Automatic analysis of one-parameter ordinary differential equations by intelligent numeric simulation. *Artificial Intelligence*, 48(1):27–56, February 1991.

[Schwabacher and Gelsey 1996a] M. Schwabacher and A. Gelsey. Intelligent gradient-based search of incompletely defined design spaces. Technical Report HPCD-TR-38, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1996. ftp://ftp.cs.rutgers.edu/pub/technical-reports/hpcd-tr-38.ps.Z.

[Schwabacher and Gelsey 1996b] M. Schwabacher and A. Gelsey. Multi-level simulation and numerical optimization of complex engineering designs. In *6th AIAA/NASA/USAF Multidisciplinary Analysis & Optimization Symposium*, Bellevue, WA, September 1996. AIAA-96-4021.

[Schwabacher *et al.* 1994] M. Schwabacher, H. Hirsh, and T. Ellman. Learning prototype-selection rules for case-based iterative design. In *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications*, San Antonio, Texas, 1994.

[Schwabacher *et al.* 1995] M. Schwabacher, H. Hirsh, and T. Ellman. Inductive learning for engineering design optimization. In Aha, D., and Riddle, P. (Eds.), *Working Notes for Applying Machine Learning in Practice: A Workshop at the Twelfth International Machine Learning Conference* (Technical Report AIC-95-023). Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence. http://www.aic.nrl.navy.mil/~aha/imlc95-workshop/, 1995.

[Schwabacher *et al.* 1996a] M. Schwabacher, T. Ellman, and H. Hirsh. Inductive learning for engineering design optimization. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 10(2):179–180, 1996. Research Abstract.

[Schwabacher *et al.* 1996b] M. Schwabacher, T. Ellman, H. Hirsh, and G. Richter. Learning to choose a reformulation for numerical optimization of engineering designs. In J.S. Gero and F. Sudweeks, editors, *Artificial Intelligence in Design '96*, pages 447–462. Kluwer Academic Publishers, The Netherlands, 1996.

[Shannon and Weaver 1949] C. Shannon and W. Weaver. *The mathematical theory of communication.* University of Illinois Press, Urbana, IL, 1949.

[Shukla *et al.* 1996a] Vijay Shukla, Andrew Gelsey, Mark Schwabacher, Donald Smith, and Doyle D. Knight. Automated redesign of the NASA P8 hypersonic inlet using numerical optimization. In *AIAA Joint Propulsion Conference*, 1996.

[Shukla *et al.* 1996b] Vijay Shukla, Andrew Gelsey, Mark Schwabacher, Donald Smith, and Doyle D. Knight. Automated design optimization for hypersonic inlets. *AIAA Journal of Aircraft*, 1996. Submitted; currently being reviewed.

[Simon 1981] H. Simon. *The Sciences of the Artificial.* MIT Press, Cambridge, MA, 1981.

[Sobieszczanski-Sobieski and Haftka 1996] Jaroslaw Sobieszczanski-Sobieski and Raphael T. Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. In *34th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, January 1996. AIAA-96-0711.

[Sobieszczanski-Sobieski *et al.* 1985] J. Sobieszczanski-Sobieski, B. B. James, and A. R. Dovi. Structural optimization by multilevel decomposition. *AIAA Journal*, 23(11):1775–1782, November 1985.

[Sobieszczanski-Sobiesky 1982] J. Sobieszczanski-Sobiesky. A linear decomposition method for large optimization problems—blueprint for development. Technical Report NASA TM-83248, National Aeronautics and Space Administration, 1982.

[Sycara and Navinchandra 1992] K. Sycara and D. Navinchandra. Retrieval strategies in a case-based design system. In C. Tong and D. Sriram, editors, *Artificial Intelligence in Engineering Design (Volume II)*, pages 145 – 164. Academic Press, New York, NY, 1992.

[Tong *et al.* 1992] Siu Shing Tong, David Powell, and Sanjay Goel. Integration of artificial intelligence and numerical optimization techniques for the design of complex aerospace systems. In *1992 Aerospace Design Conference*, Irvine, CA, February 1992. AIAA-92-1189.

[Tong 1988] S. S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. In *International Association of Mathematics and Computers in Simulation Conference on Expert Systems for Numerical Computing*, Purdue University, 1988.

[Tufte 1983] E. Tufte. *Visual display of quantitative information.* Graphics Press, Cheshire, Conn., 1983.

[Vanderplaats 1984] Garret N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design : With Applications.* McGraw-Hill, New York, 1984.

[Weiss and Kulikowski 1991] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann, San Mateo, CA, 1991.

[Williams and Cagan 1994] B. Williams and J. Cagan. Activity analysis: The qualitative analysis of stationary points for optimal reasoning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[Yip 1991] Kenneth Yip. Understanding complex dynamics by visual and symbolic reasoning. *Artificial Intelligence*, 51(1–3):179–221, October 1991.

[Zha *et al.* 1996] G.-C. Zha, D. Smith, M. Schwabacher, A. Gelsey, and D. Knight. High performance supersonic missile inlet design using automated optimization. In *AIAA Symposium on Multidisciplinary Design*, 1996. AIAA Paper 96-4142.

[Zhao 1994] Feng Zhao. Extracting and representing qualitative behaviors of complex systems in phase space. *Artificial Intelligence*, 69(1-2):51–92, 1994.

Vita

Mark Schwabacher

1986 Graduated from The Dalton School, New York, New York.

1986-90 Attended Rice University, Houston, Texas. Triple majored in Honors Computer Science, Mathematical Sciences, and Mathematical Economic Analysis.

1990 B.A., Rice University.

1991-96 Graduate Assistant, Department of Computer Science.

1992 M.S. in Computer Science, Rutgers, The State University of New Jersey.

1993 T. Ellman, J. Keane, and M. Schwabacher. Intelligent Model Selection for Hillclimbing Search in Computer-Aided Design. *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C.

1994 M. Schwabacher, H. Hirsh, and T. Ellman. Learning Prototype-Selection Rules for Case-Based Iterative Design. *Proceedings of the Tenth IEEE Conference on Artificial Intelligence for Applications.* San Antonio, Texas.

1995 A. Gelsey, D. Knight, S. Gao, and M. Schwabacher. NPARC Simulation and Redesign of the NASA P2 Hypersonic Inlet. American Institute of Aeronautics and Astronautics Joint Propulsion Conference.

1996 A. Gelsey, D. Smith, M. Schwabacher, K. Shehata, and K. Miyake. A Search Space Toolkit. *Decision Support Systems*, special issue on Unification of Artificial Intelligence with Optimization for Decision Support: Toward Unified Programming. To appear.

1996 A. Gelsey, M. Schwabacher, and D. Smith. Using Modeling Knowledge to Guide Design Space Search. In J.S. Gero and F. Sudweeks (eds.), *Artificial Intelligence in Design '96.* Kluwer Academic Publishers.

1996 M. Schwabacher, T. Ellman, and H. Hirsh. Research abstract: Inductive learning for engineering design optimization. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 10:179-180.

1996 M. Schwabacher, T. Ellman, H. Hirsh, and G. Richter. Learning to choose a reformulation for numerical optimization of engineering designs. In J.S. Gero and F. Sudweeks (eds.), *Artificial Intelligence in Design '96.* Kluwer Academic Publishers.

1996 Ph.D. in Computer Science.