Appears in **AI***EDAM* 11(3), 1997

Intelligent Gradient-Based Search of Incompletely Defined Design Spaces

Mark Schwabacher
National Institute of Standards and Technology, Gaithersburg, MD 20899

Andrew Gelsey
Department of Computer Science, Rutgers University, Piscataway, NJ 08855

Abstract

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 tested the resulting system in the domain of conceptual design of supersonic transport aircraft, and found that using rule-based gradients can decrease the cost of design space search by **one or more orders of magnitude**.

Keywords: Optimization, gradients, sequential quadratic programming, rule-based systems.

1 Introduction

Automated search of a space of candidate designs seems an attractive way to improve the traditional engineering design process. Each step of such automated search requires evaluating the quality of candidate designs, and for complex artifacts (e.g., aircraft, our main example), this evaluation must be done by computational simulation.

Gradient-based optimization methods, such as sequential quadratic programming (see Section 6), are reasonably fast and reliable when applied to search spaces that satisfy their assumptions. They generally assume that the objective function and constraint functions are continuous, smooth, and defined everywhere. Unfortunately, realistic simulators tend to violate these assumptions. Non-gradient-based optimization methods, such as simulated annealing and genetic algorithms, are better able to deal with search spaces that violate these assumptions, but they tend to require many more runs of the simulator than do the gradient based methods. We therefore seek to enhance gradient-based methods so that they can be used in the presence of these assumption violations.

One way in which realistic simulators violate the assumptions of gradient-based methods is that, 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 reduces its step size until it finds an evaluable point, as part of a gradient-based optimization system, produces optimization performance that is one or more orders of magnitude better (in CPU time) than that obtained without the rule-based gradients.

2 Related Work

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.

[Gelsey *et al.* 1996b] presents the model constraints method, a method that replaces unevaluable points with infeasible points, and that is described in Section 7. [Gelsey 1995] 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 does not discuss what an automated design system should do after detecting a low-quality result. [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 dealing with modeling failures in an automated design system are also not discussed.

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 problem of unevaluable points addressed in this article.

Simulated annealing (SA) [Press *et al.* 1986] and genetic algorithms (GA) [Goldberg 1989] 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. SA's and GA's do offer the ability to perform optimizations in search spaces with large numbers of local optima; they are more appropriate than gradient-based methods when the number of local optima is large. We argue that gradient-based optimization using rule-based gradients and multiple random starting points, as described in this article, is appropriate when the number of local optima is small, and the search space includes unevaluable points.

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 issue of unevaluable points.

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].

3 Unevaluable Points

In realistic engineering domains such as aircraft design, 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 many *pathologies*, such as unevaluable points, which make it difficult to use them within an automated design system. It may not be possible to determine *a priori* which parameter values will produce pathological results; it may only be possible to determine whether a simulation was successful after the simulation has been run. We would like to be able to use such a simulator inside an optimization loop, in order to rapidly produce better designs.

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, it may contain bugs that cause it 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.

Unevaluable points need to be distinguished from infeasible points. An *infeasible point* represents a design that violates the explicit nonlinear inequality constraints, while an unevaluable point is one for which the objective and constraint functions cannot even be evaluated, because, for example, the simulator crashes.

4 The Rules

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

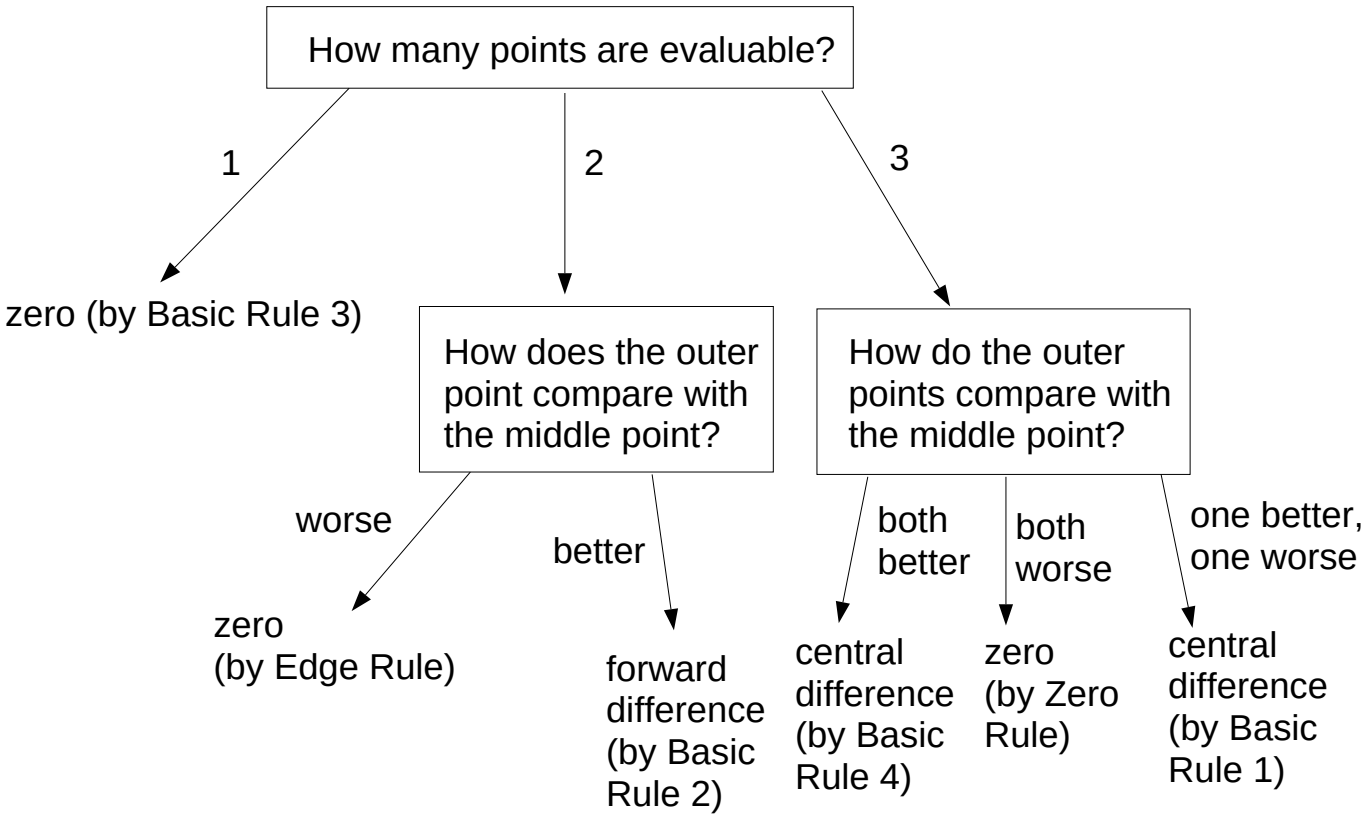
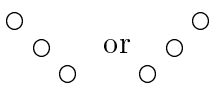


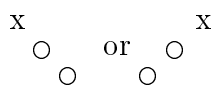
Figure 1: Flowchart of the rules used to determine how to compute a partial derivative from three points.

knowledge base an appropriate value of δ for each component of the gradient.[1] 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 a set of rules. In the following list of rules, the pictures illustrate the sets of three points corresponding to each rule. Each circle represents an evaluable point, and each "x" represents an unevaluable point. The pictures are based on the assumption that the optimizer is minimizing the objective function.



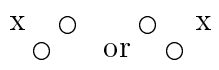
Basic Rule 1: If all three points are evaluable, one outer point is worse than the middle point, and the other outer point is better than the middle point, use the central difference formula. This rule should produce an accurate partial derivative.

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. Further, it always produces a gradient that points away from the unevaluable region.

[1] Choosing an appropriate value of δ is very important (as it is whenever derivatives are computed using finite differences [Press *et al.* 1986]). In the experiments that follow, we used a convergence study to select an appropriate value of δ for each component of the gradient.



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. In this situation, the current point is at the edge of the unevaluable region, and the partial derivative, if it were computed by forward differences using the two evaluable points, would cause the gradient to point into the unevaluable region. By returning a zero partial derivative instead of computing a gradient that would point into the unevaluable region, 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 1. It should be emphasized that these rules only affect how the gradient is computed, and not how the current point or the points encountered during a line search are evaluated. After a gradient is computed using the rules, the optimizer does a line search. The line search first attempts to take the full Newton step—the step that would minimize the quadratic approximation of the nonlinear programming problem. If taking the full Newton step results in a point that is unevaluable, or evaluable and worse than the current point, then the line search tries a smaller step. It continues to try smaller steps, until it finds a step size that results in a new point that is evaluable and better than the current point. This new point becomes the current point, and the optimization algorithm goes back to computing a new gradient. The optimizer will therefore always have an evaluable point as the "best point found so far," and there is no risk that the optimizer will return as the "optimum" a point which the simulator could not evaluate.

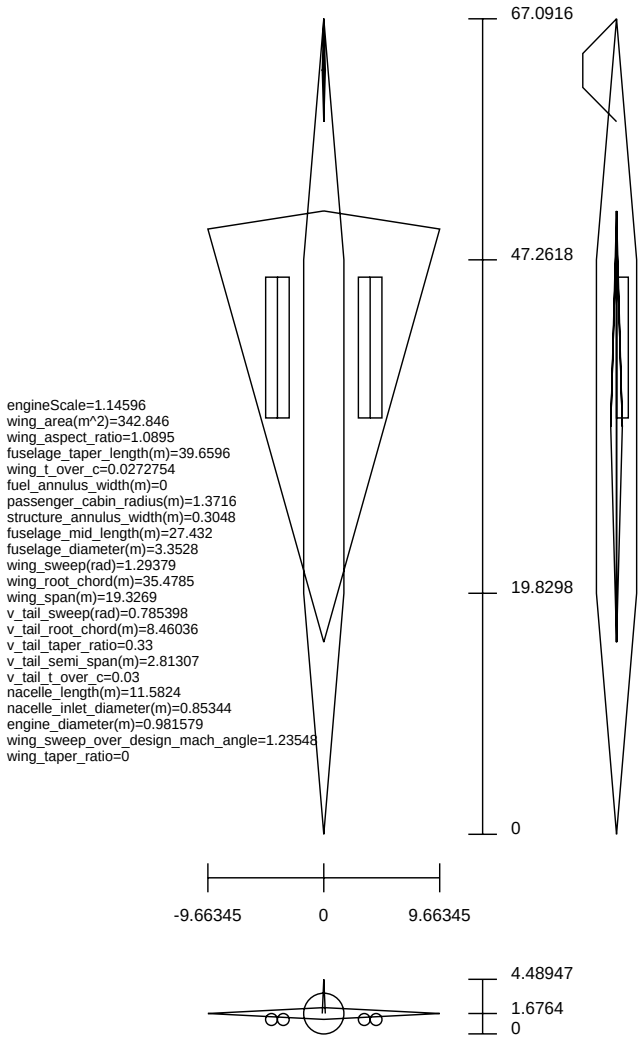


Figure 2: Supersonic transport aircraft designed by our system (dimensions in meters)

5 Aircraft Design

We have pursued our investigation in the domain of the conceptual design of supersonic transport aircraft [Gelsey and Smith 1996]. Figure 2 shows a diagram of a typical airplane automatically designed by our software system to fly the mission shown in Table 1. Figure 3 shows a block diagram of our automated conceptual design system. The design system has two major components: the Design Associate (DA), which searches the space of candidate designs, and the Model/Simulation Associate (MSA), which the DA uses to evaluate the quality of candidate designs. In our system, the optimizer 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 optimizer evaluates candidate designs using a multidisciplinary[2] simulator. In our current

[2]We call the simulator *multidisciplinary* because it contains code to evaluate the aircraft using several engineering disciplines, including weights, aerodynamics, and propulsion.

Phase	Mach	Altitude		Duration (min)	comment
		m	ft		
1	0.227	0	0	5	"takeoff"
2	0.85	12 192	40 000	50	subsonic cruise (over land)
3	2.0	18 288	60 000	225	supersonic cruise (over ocean)

capacity: 70 passengers.

Table 1: Mission specification for aircraft in Figure 2

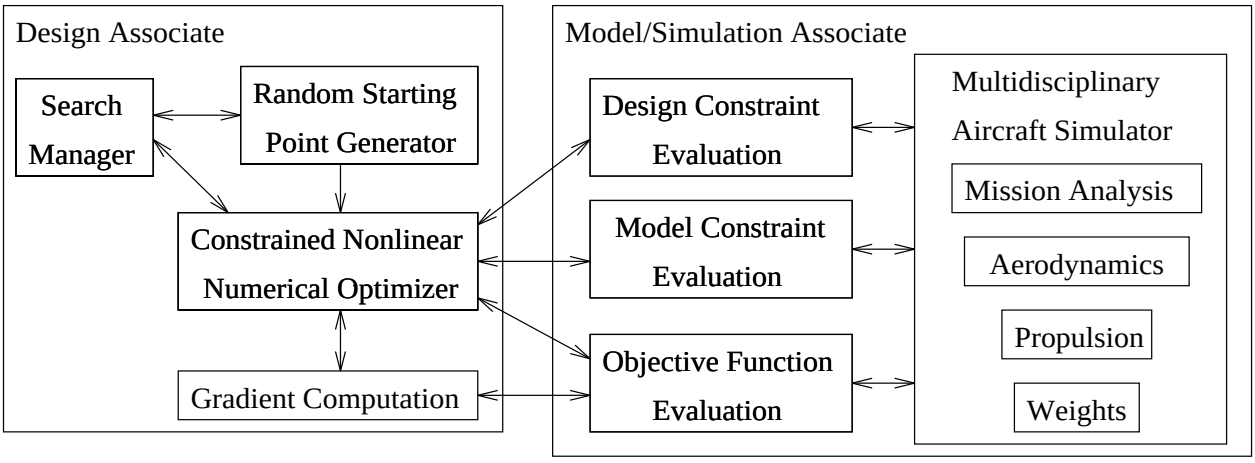


Figure 3: Automated design system block diagram

implementation, the optimizer'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 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.

A complete mission simulation requires about 1/4 second of CPU time on a DEC Alpha 250 4/266 desktop workstation. Note that the above procedure is deterministic; application of our technique to simulators with stochastic elements is discussed in Section 8.

6 Search Procedure

In this article we will focus on search of a space of candidate designs using numerical optimization methods which vary a set of continuous parameters to minimize[3] a nonlinear objective function subject to a set of nonlinear equality and inequality constraints. The numerical optimizer used in the experiments described in this article is CFSQP [Lawrence *et al.* 1995], a state-of-the-art implementation of the Sequential Quadratic Programming method.[4] Sequential Quadratic Programming is a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programming problems[5] to it, and then solving each of these problems using a quadratic programming method. When started from an infeasible point (one which violates at least one of the constraints), CFSQP first does an optimization in which it minimizes the maximum of the constraint violations.

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

[4] Earlier we tried doing optimization in this domain using several different optimization packages, and found that we obtained the best results when using CFSQP.

[5] A quadratic programming problem consists of a quadratic objective function to be optimized, and a set of linear constraints.

This first optimization continues until a feasible point (one which satisfies all of the constraints) is found. Then CFSQP does the main optimization, in which it minimizes the objective function, while keeping all of the constraints satisfied.

Because the search space has multiple local optima, we use a technique that we call "random multistart" to attempt to find the global optimum. In an n-point random multistart, the engineer first chooses a box which he or she believes will contain all reasonable designs.[6] The system randomly generates starting points using a uniform distribution within this box until it finds n evaluable points[7], and then performs a gradient-based optimization from each of these points. The best design found in these n optimizations is taken to be the global optimum.

7 Model Constraints

Previously, we presented a technique called *model constraints* for dealing with certain unevaluable points [Gelsey *et al.* 1996b]. Some unevaluable points are caused by designs that violate the modeling assumptions made by the designers of the simulator. The approach is to implement a set of model constraint functions that measure how much each of the modeling assumptions is violated, and to pass these model constraints to the optimizer along with the existing design constraints. The result is that some of the unevaluable points are replaced with infeasible points,[8] which greatly improves optimization performance.[9] In the cases described in this article, we had sufficient understanding of the limitations of the simulators so that we were able to implement model constraints that describe many of these limitations. We were not, however, able to capture all of these limitations in the model constraints, so some unevaluable points remained, which were handled by the rule-based gradients. We found that using rule-based gradients together with model constraints produced much greater optimization performance (better designs at lower CPU cost) than model constraints alone. Rule-based gradients can also be used in situations where it is not feasible to implement any model constraints.

8 Experimental 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).

[6] Choosing an appropriate box is important, and depends on the knowledge of the engineer. If the box is excessively large, then the optimization process will take an unreasonable amount of time to find the global optimum. If the box is too small, then the true global optimum might be missing. If the apparent global optimum produced by the search procedure is on the edge of the box, that is an indication that the box may be too small.

[7] 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.

[8] Recall that an infeasible point is one that violates the explicit constraints.

[9] See Section 6 for a description of how the optimizer handles infeasible points.

Design Parameter	low	high
engine size	0.1	5
wing area	46.45 m^2 (500 ft^2)	1858 m^2 (20 000 ft^2)
wing aspect ratio	0.5	3
fuselage taper length	15.24 m (50 ft)	91.44 m (300 ft)
effective structural thickness over chord	0.5	10
wing sweep over design mach angle	0	1.45
wing taper ratio	0	0.1
fuel annulus width	0	2.438 m (8 ft)

Table 2: Subset of design space explored

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 (less CPU time needed to obtain a given design quality) 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.

Because the rule-based gradients method is an heuristic method that is applied to simulators which can have arbitrary properties, it is not possible to mathematically prove that it will work. Instead, we must rely on empirical tests. For our primary set of tests of rule-based gradients, 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 nondimensional measure of wing thickness)

6. wing sweep over design mach angle (a nondimensional 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 chose these parameters to be independent of one another, in the sense that changing one parameter does not result in a change in any other parameter, i.e., $\forall i, j \frac{\partial p_i}{\partial p_j} = 0$. Gradient-based optimization methods such as CFSQP work better if the variables are independent

Design Parameters:	
engine size	1.146
wing area	342.8 m^2 (3690 ft^2)
wing aspect ratio	1.089
fuselage taper length	39.65 m (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	134 800 kg

Table 3: Best design found for mission of Table 1

in this sense. The design parameters are dependent in the sense that changing one variable can result in changes in the way that another variable affects the objective function, takeoff mass, i.e., $\exists i, j \frac{\partial m_t}{\partial p_i \partial p_j} \neq 0$. If the design parameters were independent in this second sense, then they could be optimized one at a time, which would be much less expensive.

Table 2 shows the subset we explored in the design space defined by these eight design parameters.

We compared four optimization strategies:

- **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 article.
- **Model Constraints:** This strategy replaces some of the unevaluable points with infeasible points, using the model constraints strategy described in [Gelsey *et al.* 1996b].
- **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.[10] 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 Table 1. (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

[10] 74 is not a "magic number"; we ran optimizations until we ran out of disk space.

were unevaluable — that is, they violated the underlying assumptions of the simulator, which caused the simulator to fail. With model constraints, only 4% of the points encountered were unevaluable, suggesting that the model constraints captured most of the underlying assumptions of the simulator. The best design found for this mission in all the experiments is shown in Table 3, and a diagram of this aircraft appears in Figure 2.

The performance of the strategy combinations is shown in a graph in Figure 4. 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.[11] A value of one indicates that the method has found the apparent 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. It 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).[12]

The data in Figure 4 indicates that the strategy of combining rule-based gradients with model constraints is one or more orders of magnitude better (in CPU time) 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 apparent 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 Table 4. Table 5 shows the parameters 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; the experimental data appears in Figure 5. For this mission, the strategy that combines rule-based gradients with model constraints is the only strategy that got to the apparent global optimum. The other strategies were orders of magnitude worse, consistent with our hypothesis.

[11] We do not know with certainty what the global optimum is. Finding the global optimum of an arbitrary nonlinear function is undecidable [Schwabacher 1996]. However, we have performed many optimizations from different random starting points, and a large number of them have converged to the same point. We call this point "the apparent global optimum."

[12] 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 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.

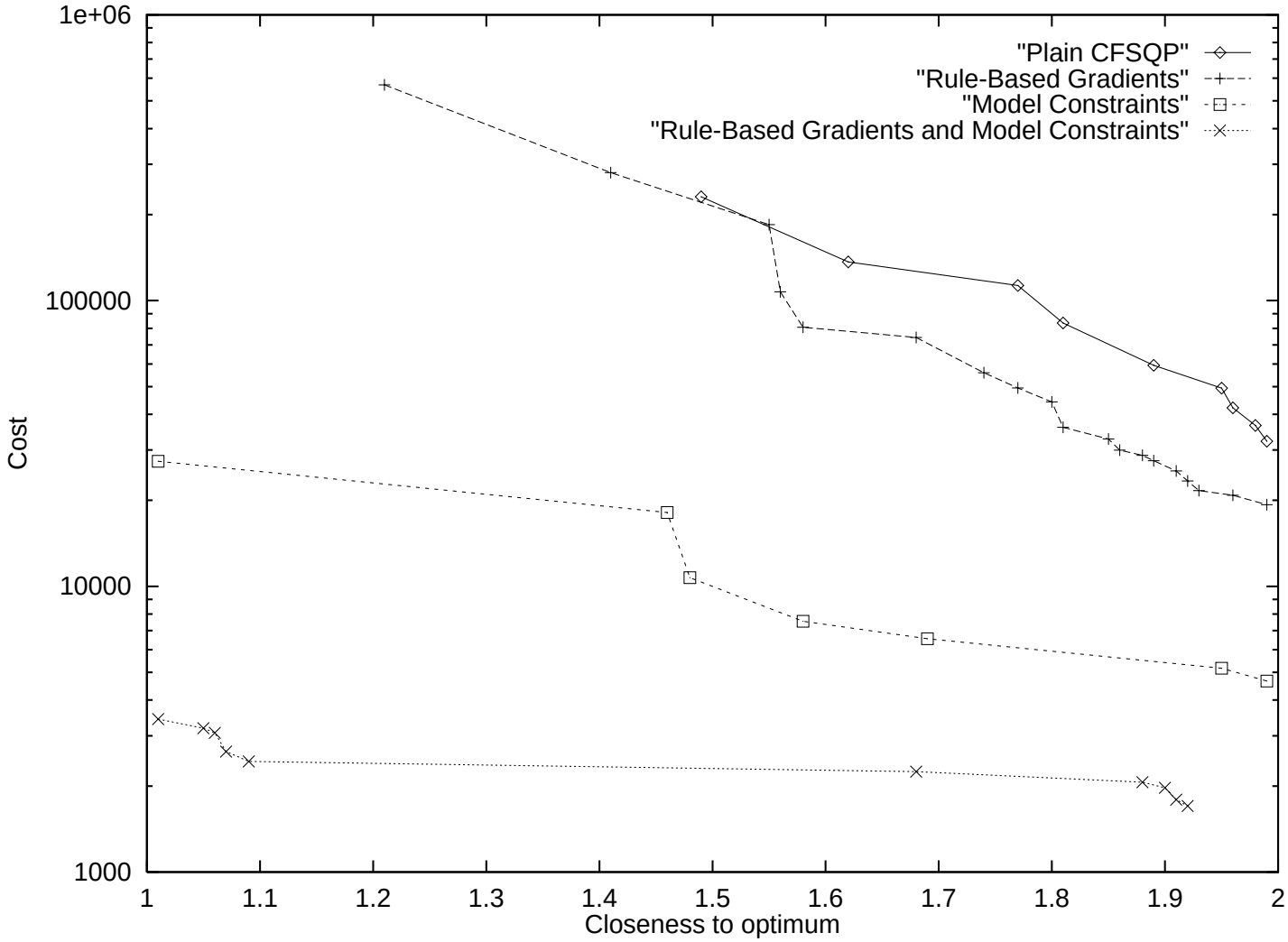


Figure 4: Performance of the various strategies. Optimization performance increases as one moves down (lower cost) and to the left (closer to the apparent global optimum). The cost shown is the estimated number of simulations needed to have a 99% chance of getting within the specified fraction of the optimum.

Phase	Mach	Altitude		Duration	comment
		m	ft	(min)	
1	0.227	0	0	5	"takeoff"
2	0.85	12 192	40 000	85	subsonic cruise (over land)
3	2.0	18 288	60 000	180	supersonic cruise (over ocean)

capacity: 70 passengers.

Table 4: Another mission specification

Design Parameters:	
engine size	1.532
wing area	432.2 m^2 (4652 ft^2)
wing aspect ratio	1.570
fuselage taper length	36.97 m (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	167 400 kg

Table 5: Best design found for the 2nd mission (Table 4)

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

Table 6: 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

Table 7: Optimization performance using subsets of the rule-based gradient rules, for the second mission.

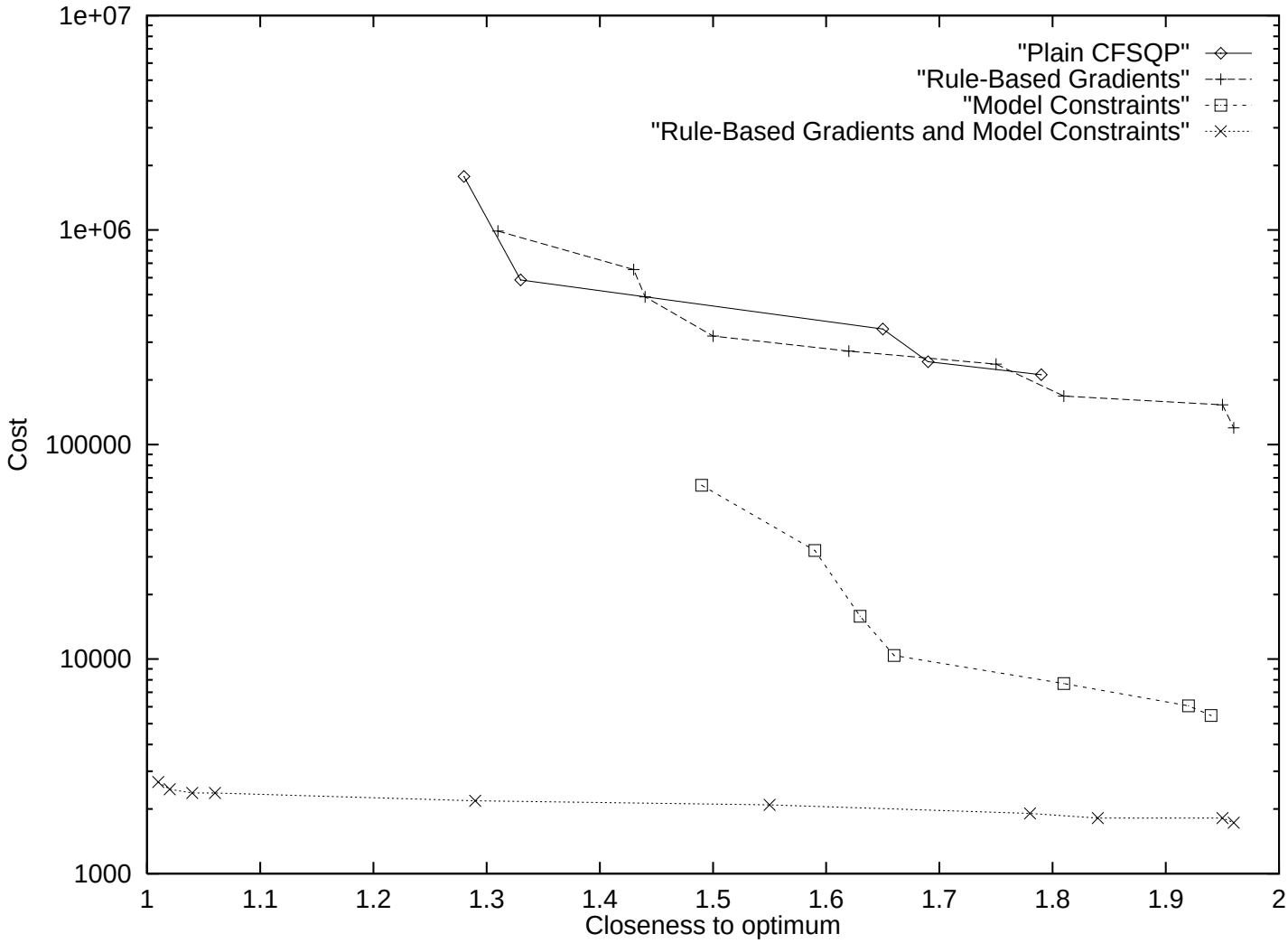


Figure 5: Performance of the various strategies for a different mission. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum). The cost shown is the estimated number of simulations needed to have a 99% chance of getting within the specified fraction of the optimum.

Of the rules listed in Section 4, 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, for each of the two missions, with just the "basic rules" (the rules listed in Section 4 other than these two), and with each of these two rules added individually. Table 6 and Table 7 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.

The edge rule is intended to improve optimization performance in cases where the unevaluable points exist in large unevaluable regions. Our aircraft design example is such a case; hence, the edge rule improves optimization performance in this domain. In a domain in which the unevaluable points exist as "islands" or "patches" scattered throughout the space, we would not expect the edge rule to help optimization performance—rather, we would expect it to hurt optimization performance slightly. In a domain that has scattered patches of unevaluable points, without any large unevaluable regions, it would be better to disable the edge rule. Scattered patches of unevaluable points might occur if the simulator contained a stochastic element. The missile inlet domain, described in the next section, has both patches of unevaluable points, and large unevaluable regions. We found the edge rule to be helpful 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.

At first we knew very little about the distribution of unevaluable points in the aircraft design search space. We used our Search Space Toolkit (SST) [Gelsey *et al.* 1996a] to help us determine the distribution of unevaluable points in this search space. Because our aircraft design search space is eight-dimensional, it is extremely difficult to visualize the shape of the entire eight-dimensional valid region, so we instead visualized various cross-sections of the space. We determined from these cross-sections that the unevaluable points occur in large unevaluable regions. However, we are not able to describe the shape of the eight-dimensional valid region, or the path followed by the optimizer within this eight-dimensional space. In a two-, three-, or even four-dimensional space, one can use visualization tools such as our SST to look at the shape of the evaluable region, and then use the insight gained from the visualization to implement additional model constraints to capture more of the unevaluable points. In a higher-dimensional space such as our aircraft design space, it is more difficult to understand the properties of the space, and therefore more likely that rule-based gradients

will be necessary. If the engineer does not know how the unevaluable points are distributed in the search space, he or she can use a tool such as the SST to visualize the search space, or he or she can simply use all of our rules, which should produce reasonably good optimization performance for a variety of distributions of unevaluable points.

9 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 [Ellman *et al.* 1993, Schwabacher *et al.* 1994, Schwabacher *et al.* 1996], the design of exhaust nozzles for supersonic jets [Gelsey *et al.* 1996a], the design of inlets for hypersonic jets [Gelsey *et al.* 1995, Shukla *et al.* 1996, Shukla *et al.* 1997], and the design of inlets for supersonic missiles [Zha *et al.* 1996, Zha *et al.* 1997]. 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.

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 6. Using rule-based gradients allows the optimizer to get much closer to the optimum, at a significantly lower cost, supporting our hypothesis.

We compared optimization performance with and without rule-based gradients in two domains—the airframe domain and the supersonic missile inlet domain. These comparisons, reported in this article, strongly suggest that rule-based gradients are helpful. When we performed optimizations using more-expensive simulators in the yacht domain and the hypersonic inlet domain, we did not have enough CPU time available to do the optimizations both with and without rule-based gradients. We knew that these domains did have unevaluable points. Our confidence in rule-based gradients was high enough that we decided to run the optimizations with rule-based gradients. We obtained good optimization results in these domains (results which in the case of hypersonic inlets have been published in the aerodynamics engineering literature [Shukla *et al.* 1997]), and we believe that the results would have been less good if we had not used rule-based gradients, although we have not been able to experimentally test this belief.

10 Limitations, Future Work, and Conclusions

Most of our experiments have been performed in the airframe domain, in which the global optimum has a fairly large "basin of attraction," so that a local optimization method like

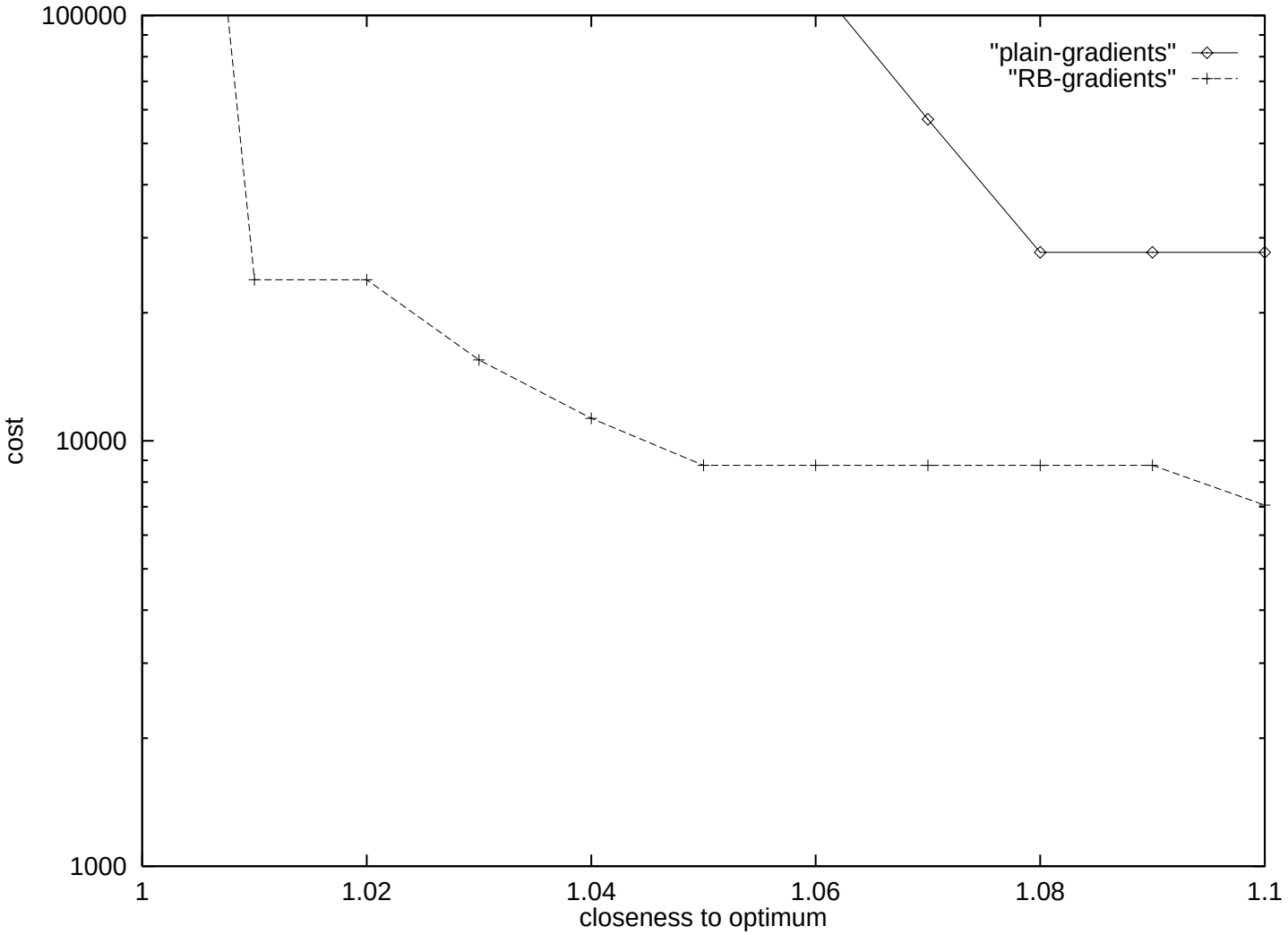


Figure 6: Performance of rule-based gradients for the missile inlet domain. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum). The cost shown is the estimated number of simulations needed to have a 99% chance of getting within the specified fraction of the optimum.

Sequential Quadratic Programming will give a high confidence of finding the global optimum if started from a small number of random starting points, assuming the unevaluable points are handled properly. For domains in which this property fails to hold, global optimization methods such as Simulated Annealing and Genetic Algorithms will often be preferable. Such methods do not use gradients, and therefore do not need rule-based gradients.

We have shown that using rule-based gradients produces much better optimization performance (less CPU time needed to obtain a particular design quality) 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 jet engine exhaust nozzle design. These points violate certain assumptions that were made by the authors of the simulator, but they do not cause the simulator to fail completely, although they may cause it to produce less-accurate results. In the nozzle 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. For example, there are model constraints specifying that the simulator should not extrapolate outside the tables of experimental data that it uses. If a particular design requires extrapolation, then the simulator does extrapolate, but one or more model constraints are violated, indicating that the results are probably inaccurate. The new rule causes the optimizer to compute gradients without using these points that require extrapolation, if possible. This new rule requires further testing, and other new rules could be developed.

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.

Acknowledgments

We thank our aircraft design expert, Gene Bouchard of Lockheed, for his invaluable assistance in this research. We also thank all members of the HPCD project, especially Tom Ellman, Keith Miyake, and Don Smith. We thank Ron Giachetti and Alex Meystel of NIST for reviewing this manuscript. This research was partially supported by NASA under grant NAG2-817 and was also part of the Rutgers-based HPCD (Hypercomputing and Design) project supported by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DABT 63-93-C-0064. Mark Schwabacher was supported by a National Research Council Postdoctoral Research Associateship for a portion of the time that

he worked on this article.

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, instruments, or materials are identified in this report in order to facilitate understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

References

[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 Paper 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.

[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*, pages 594–599, Washington, DC, 1993. MIT Press, Cambridge, MA.

[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*, pages 1798–1805, Montreal, Quebec, Canada, 1995. MIT Press.

[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.* PhD thesis, Stanford University, Stanford, CA, 1994.

[Gelsey and Smith 1996] Andrew Gelsey and Don Smith. Computational environment for exhaust nozzle design. *Journal of Aircraft*, 33(3):470–476, 1996.

[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] A. Gelsey, D. Smith, M. Schwabacher, K. Rasheed, and K. Miyake. A search space toolkit: SST. *Decision Support Systems*, 18:341–356, 1996.

[Gelsey *et al.* 1996b] 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, Dordrecht, The Netherlands, 1996.

[Gelsey 1995] 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.

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

[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.

[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 Paper 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, College Park, MD, August 1995.

[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. MIT Press, Cambridge, MA.

[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, Los Altos, CA.

[Powell 1990] D. Powell. *Inter-GEN: A Hybrid Approach to Engineering Design Optimization*. PhD thesis, Rensselaer Polytechnic Institute Department of Computer Science, Troy, NY, December 1990.

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

[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*, pages 56–62, San Antonio, Texas, 1994.

[Schwabacher *et al.* 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, editors, *Artificial Intelligence in Design '96*, pages 447–462. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[Schwabacher 1996] Mark Schwabacher. The use of artificial intelligence to improve the numerical optimization of complex engineering designs. Technical Report HPCD-TR-45, Department of Computer Science, Rutgers University, New Brunswick, NJ, October 1996. Ph.D. Thesis. http://www.cs.rutgers.edu/∼schwabac/thesis.html.

[Shukla *et al.* 1996] 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.* 1997] Vijay Shukla, Andrew Gelsey, Mark Schwabacher, Donald Smith, and Doyle D. Knight. Automated design optimization for the P2 and P8 hypersonic inlets. *AIAA Journal of Aircraft*, 34(2):228–235, March 1997.

[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 Paper 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.

[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 Paper 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, West Lafayette, IN, 1988.

[Zha *et al.* 1996] G.-C. Zha, D. Smith, M. Schwabacher, K. Rasheed, A. Gelsey, D. Knight, and M. Haas. High performance supersonic missile inlet design using automated optimization. In *6th AIAA/NASA/USAF Multidisciplinary Analysis & Optimization Symposium*, Bellevue, WA, September 1996. AIAA Paper 96-4142.

[Zha *et al.* 1997] G.-C. Zha, D. Smith, M. Schwabacher, K. Rasheed, A. Gelsey, D. Knight, and M. Haas. High performance supersonic missile inlet design using automated optimization. *AIAA Journal of Aircraft*, 34(6):697–705, November 1997.