# Intelligent Gradient-Based Search of Incompletely Defined Design Spaces

Mark Schwabacher schwabac@cs.rutgers.edu
Andrew Gelsey gelsey@cs.rutgers.edu
Computer Science Department
Rutgers University
New Brunswick, NJ 08903
USA
(908) 445–3213, FAX –5691

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

HPCD-TR-38

**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. This research was partially supported by NASA under grant NAG2-817 and is 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.

# 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 5), 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. 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 knowledge-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 knowledge-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 knowledge-based gradients.

# 2 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.

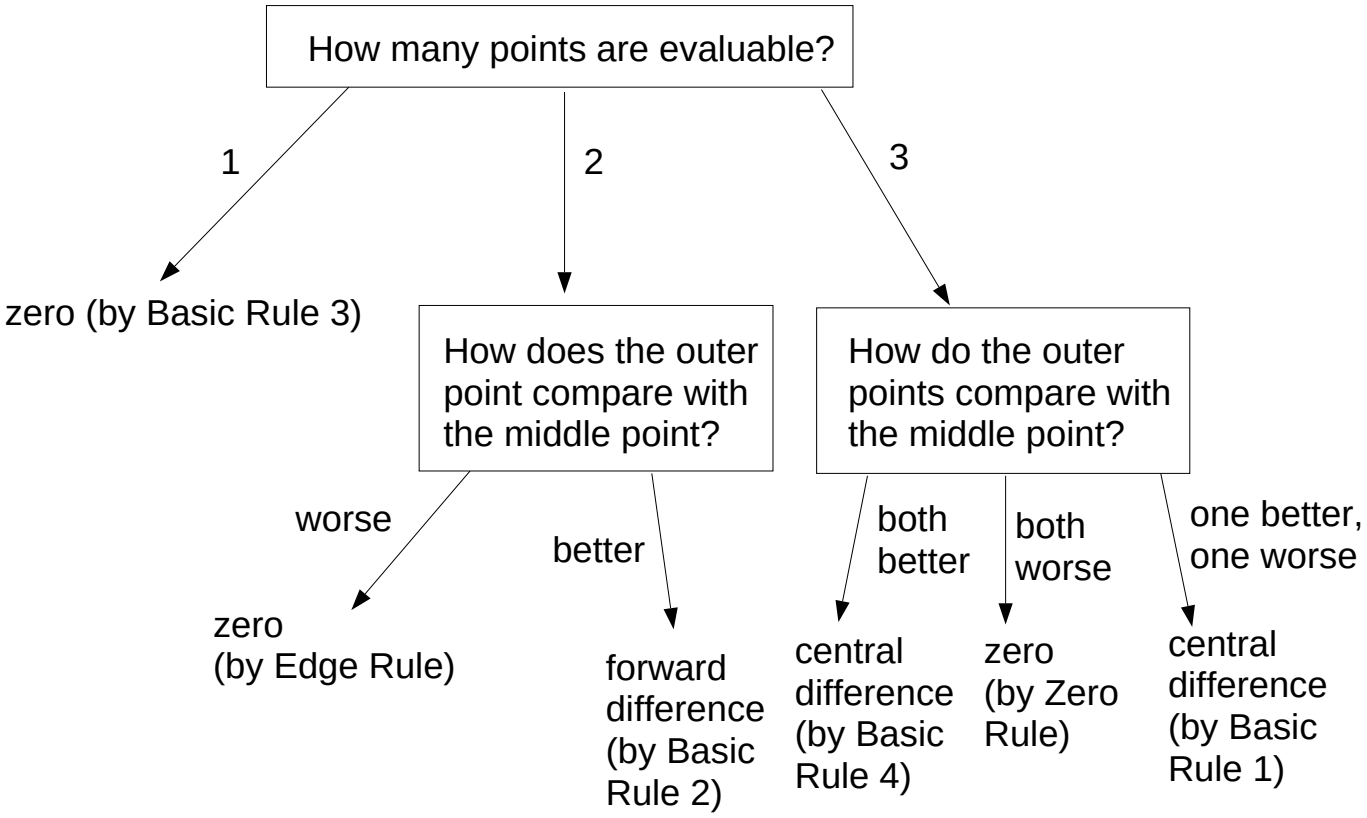


Figure 1: Flowchart of the rules used to determine how to compute a partial derivative from three points.

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.

# 3 The Rules

For each component of the gradient, the knowledge-based gradient computation method evaluates three points: the current point, and two points produced by changing the specified design parameter by plus or minus $\delta$, where $\delta$ is a small step size. The method has stored in its knowledge base an appropriate value of $\delta$ 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.
- **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 1.

# 4 Aircraft Design

We have pursued our investigation in the domain of conceptual design of supersonic transport aircraft. Figure 2 shows a diagram of a typical airplane automatically designed by our software system to fly the mission shown in Figure 3, and Figure 4 shows a block diagram of the system's software architecture. The search controller attempts to find a good aircraft conceptual design for a particular mission by varying major aircraft parameters such as

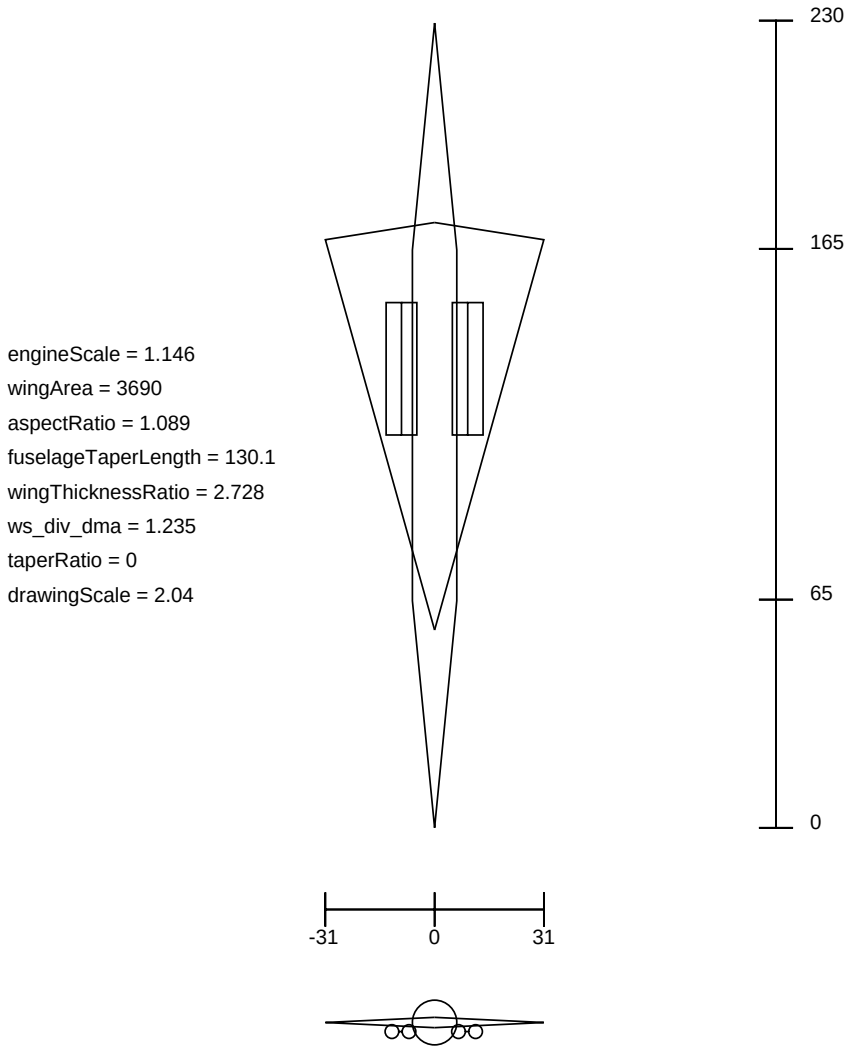


Figure 2: Supersonic transport aircraft designed by our system (dimensions in feet)

| 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 3: Mission specification for aircraft in Figure 2

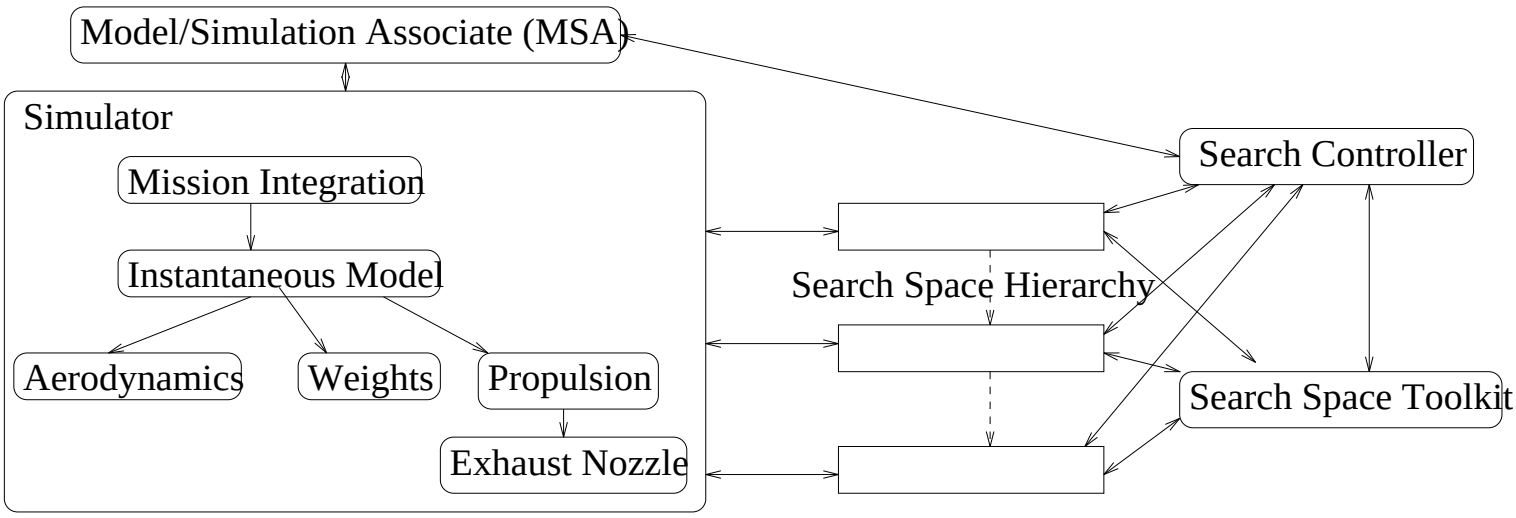


Figure 4: Software architecture block diagram

wing area, aspect ratio, engine size, etc. using a numerical optimization algorithm. The search controller evaluates candidate designs using a multidisciplinary simulator with which it communicates via the Model/Simulation Associate (MSA). 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 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.

The software architecture in Figure 4 also includes a "search space toolkit" for determining the design space structure, which is described in [Gelsey and Smith 1995, Gelsey *et al.* 1996b] and therefore will not be discussed further in this paper.

A complete mission simulation requires about 1/4 second of CPU time on a DEC Alpha 250 4/266 desktop workstation.

# 5 Search Procedure

In this paper we will focus on search of a space of candidate designs using numerical optimization methods which vary a set of continuous parameters to minimize[1] a nonlinear objective function subject to a set of nonlinear equality and inequality constraints. The numerical optimizer used in this paper is CFSQP [Lawrence *et al.* 1995], a state-of-the-art implementation of the Sequential Quadratic Programming method. Sequential Quadratic Programming is a quasi-Newton method that solves a nonlinear constrained optimization problem by fitting a sequence of quadratic programs[2] to it, and then solving each of these problems using a quadratic programming method.

# 6 Model Constraints

Previously, Gelsey, Schwabacher, and Smith presented a technique called *model constraints* for dealing with certain unevaluable points [Gelsey *et al.* 1996a]. 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. This technique replaces some of the unevaluable points with infeasible points, which greatly improves optimization performance. We found that using knowledge-based gradients together with model constraints produced much greater optimization performance than model constraints alone. Knowledge-based gradients can also be used in situations where it is not feasible to implement a set of model constraints.

# 7 Experimental Results

To experimentally test knowledge-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)

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

[2] A quadratic program consists of a quadratic objective function to be optimized, and a set of linear constraints.

| Design Parameter | low | high |
|---|---|---|
| engine size | 0.1 | 5 |
| wing area (sq. ft.) | 500 | 20000 |
| wing aspect ratio | 0.5 | 3 |
| fuselage taper length (ft.) | 50 | 300 |
| 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 (ft.) | 0 | 8 |

Figure 5: Subset of design space explored

| 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 |

Figure 6: Best design found for mission of Figure 3

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)

Figure 5 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.

- **Knowledge-based gradients:** This strategy handles the unevaluable points using the knowledge-based gradient computation method described in this paper.
- **Model Constraints:** This strategy replaces some of the unevaluable points with infeasible points, using the model constraints strategy described in [Gelsey *et al.* 1996a].
- **Knowledge-based gradients and model constraints:** This strategy replaces some of the unevaluable points with infeasible points, and uses knowledge-based gradients to handle the remaining unevaluable points.

For each strategy, our system randomly chose points until it found 74 evaluable points.[3] 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 3. (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 6, and a diagram of this aircraft appears in Figure 2.

The performance of the strategy combinations is shown in a graph in Figure 7. 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 obtain a particular closeness to the optimum.[4] It is computed by multiplying the average cost per optimization times $\log(1-P_{\text{desired}})/\log(1-P_{\text{success}})$, where $P_{\text{desired}}$ is the desired probability of getting within the specified fraction of the global optimum (99% in this case) and $P_{\text{success}}$ 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).

The data in Figure 7 indicates that the strategy of combining knowledge-based gradients with model constraints is one or more orders of magnitude better than any of the other strategies. Without model constraints, the optimizer fails to even get within 30% of the optimum. With model constraints, but without the knowledge-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 the effect of the design goal on our conclusions, we repeated our experiments with a different goal. The goal was to design the best aircraft for the mission shown in Figure 8. Figure 9 and Figure 10 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 11. For this mission, the strategy that combines knowledge-based gradients with model constraints

[3] 74 is not a "magic number"; we ran optimizations until we ran out of disk space.

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

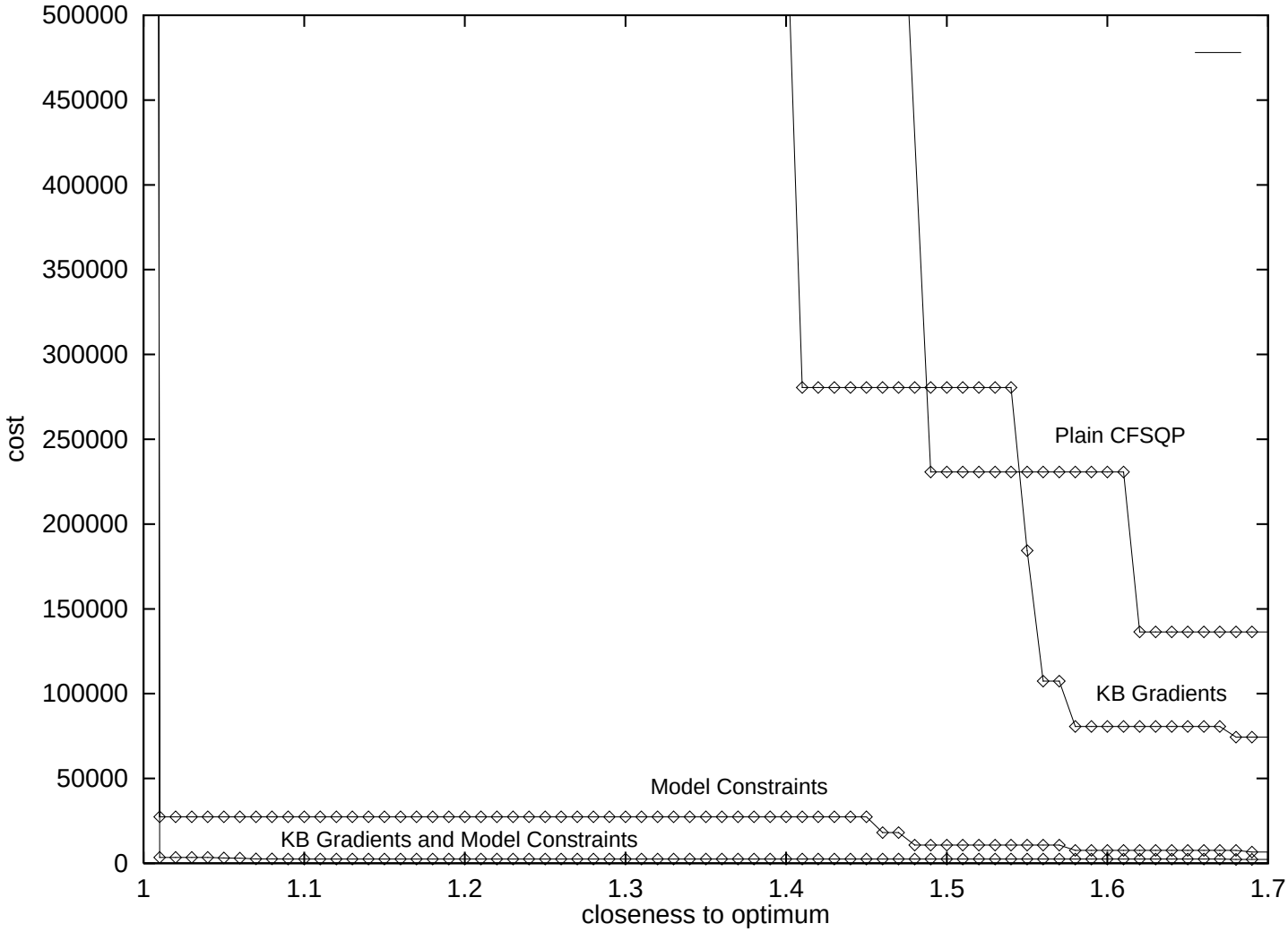


Figure 7: Performance of the various strategies. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

| 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 8: Another mission specification

| | |
|---|---|
| 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 |

Figure 9: Best design found for the 2nd mission (Figure 8)

is the only strategy that got to the global optimum. The other strategies were orders of magnitude worse.

Of the rules listed in Section 3, 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 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 3 other than these two), and with each of these two rules added individually. Figure 12 and Figure 13 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. 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.

# 8 Other Domains

We have found that using knowledge-based gradients results in good optimization performance in several other design domains. These domains include the design of racing yachts

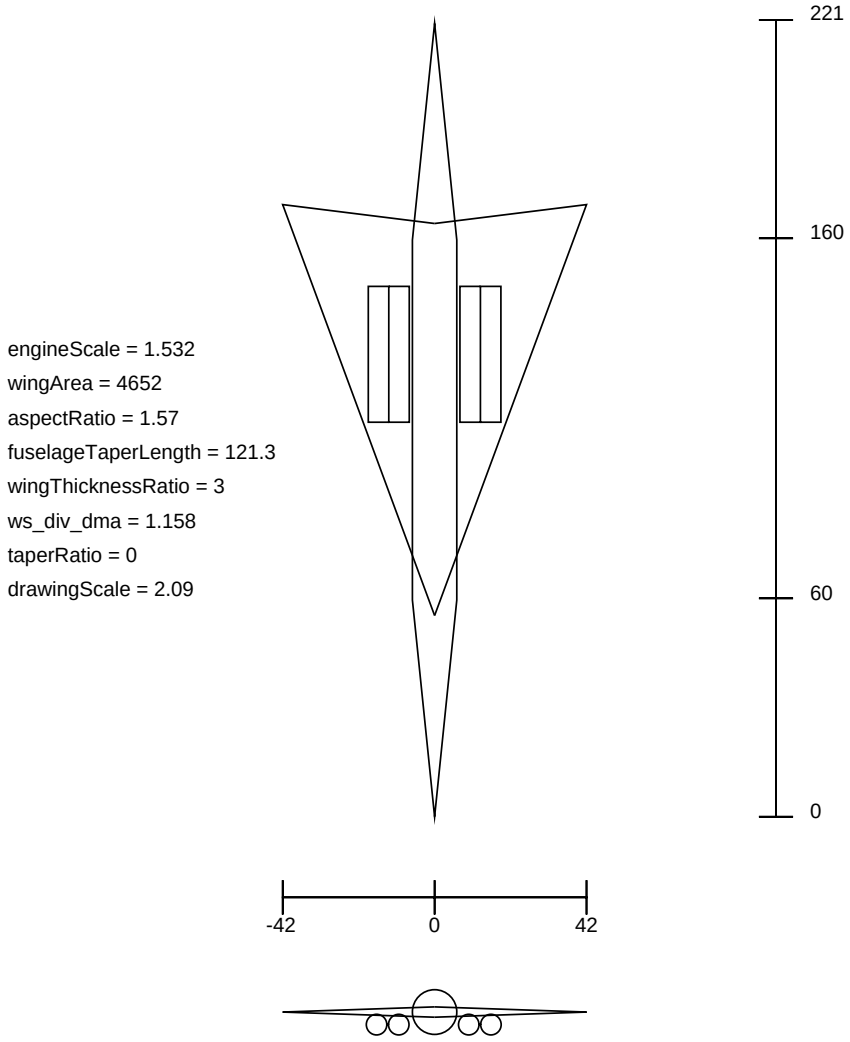


Figure 10: Supersonic transport aircraft designed by our system for the second mission (dimensions in feet)

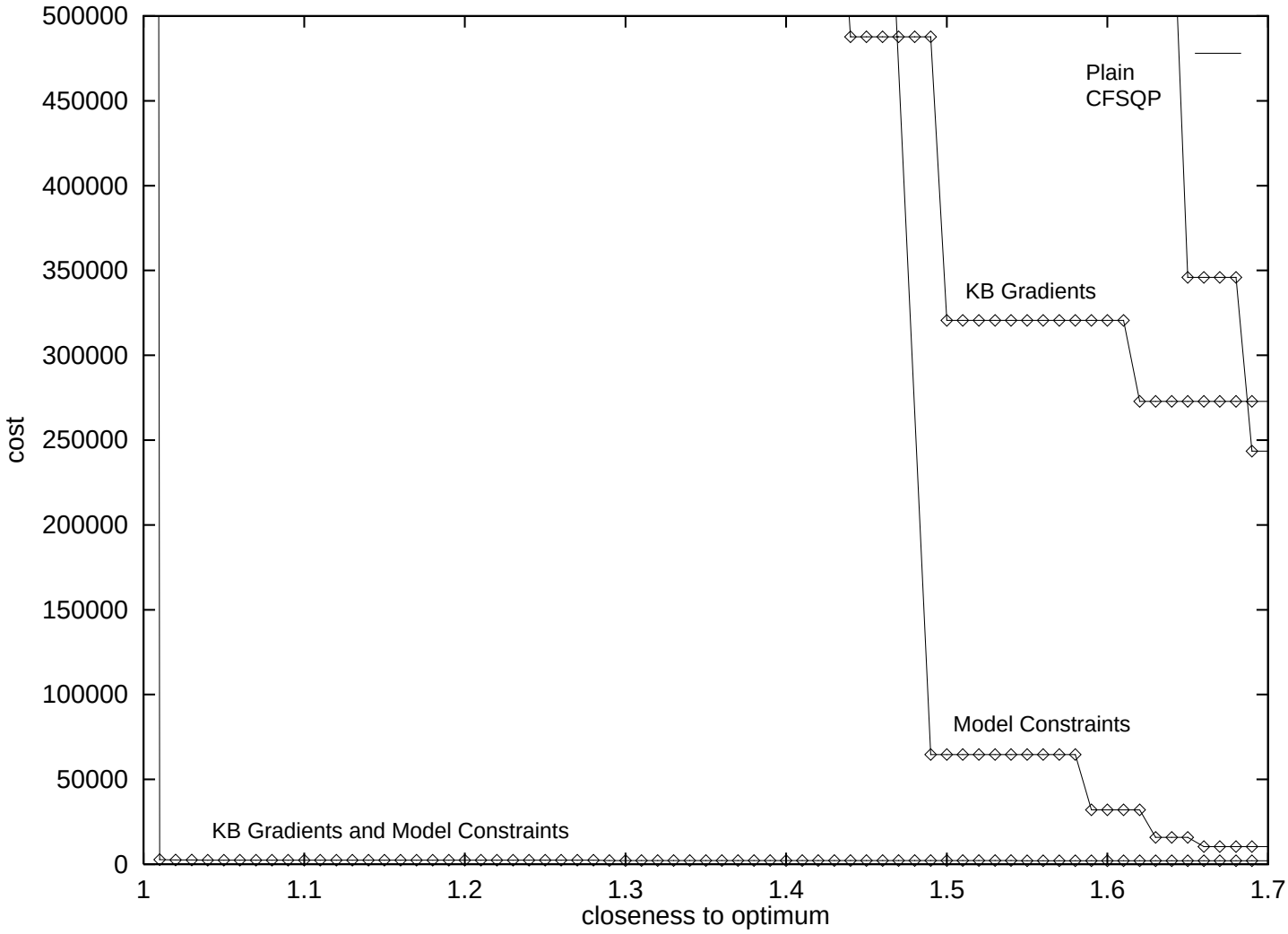


Figure 11: Performance of the various strategies for a different mission. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

| 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 12: Optimization performance using subsets of the knowledge-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 13: Optimization performance using subsets of the knowledge-based gradient rules, for the second mission.

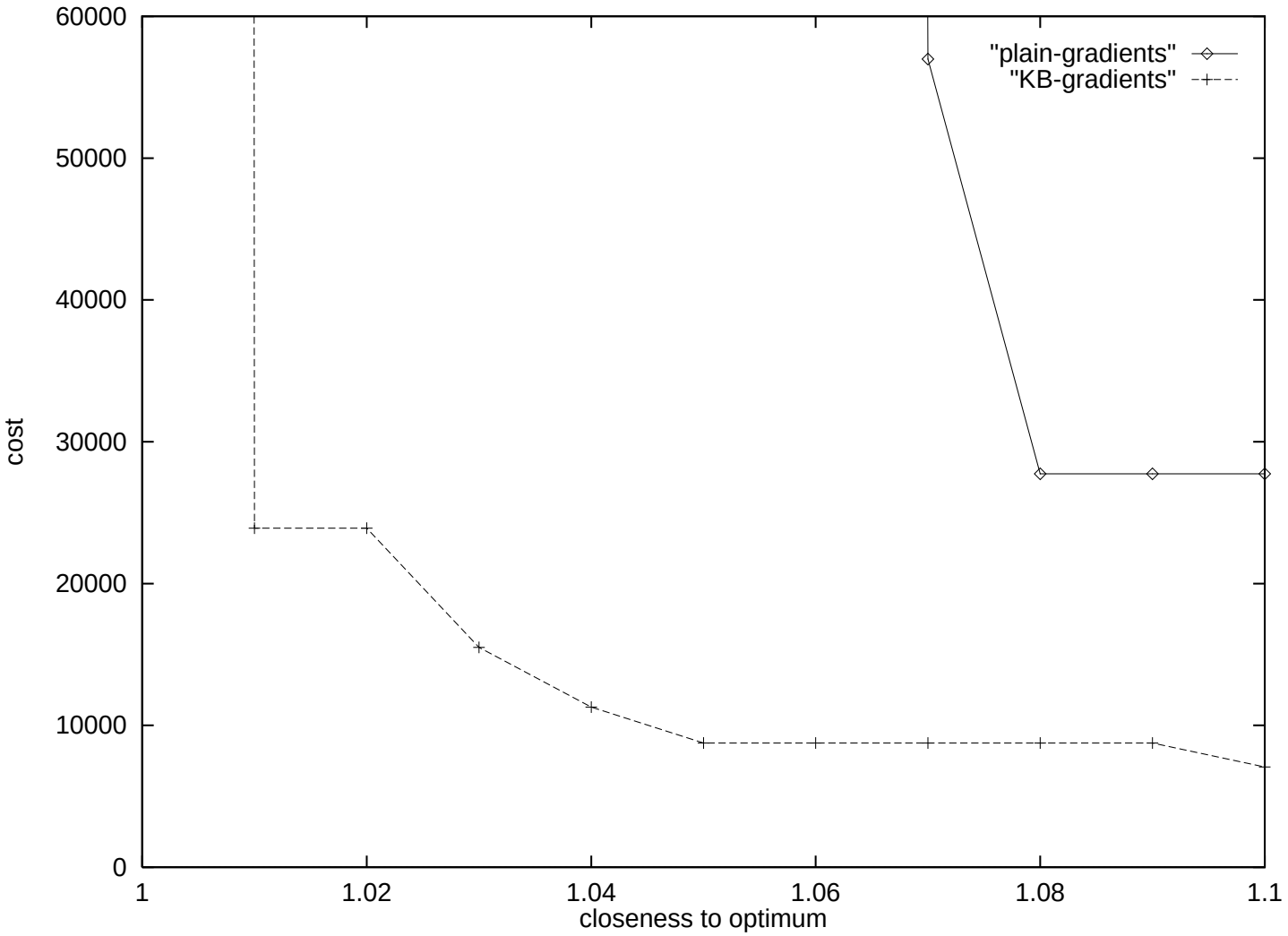


Figure 14: Performance of knowledge-based gradients for the missile inlet domain. Optimization performance increases as one moves down (lower cost) and to the left (closer to optimum).

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

We have preliminary results demonstrating the value of knowledge-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 knowledge-based gradients, we ran 20 CFSQP optimizations with and without knowledge-based gradients. The optimization performance is shown in Figure 14. Using knowledge-based gradients allows the optimizer to get much closer to the optimum, at a significantly lower cost.

# 9 Related Work

[Gelsey *et al.* 1996a] presents the model constraints method, a method that replaces unevaluable points with infeasible points, and that is described in Section 6. [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 great deal of work has been done in the area of numerical optimization algorithms [Gill *et al.* 1981, Vanderplaats 1984, Peressini *et al.* 1988, Moré and Wright 1993, Papalambros and Wilde 1988], though not much has been published about 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, Schwabacher *et al.* 1994, Schwabacher *et al.* 1996, Tong *et al.* 1992, Powell 1990, Bouchard *et al.* 1988, Bouchard 1992, Sobieszczanski-Sobieski *et al.* 1985, Agogino and Almgren 1987, Williams and Cagan 1994, Hoeltzel and Chieng 1987, Cerbone 1992], but have not addressed the issue of performing optimization in the presence of unevaluable points.

# 10 Limitations and Future Work

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 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 knowledge-based gradients.

We have shown that using knowledge-based gradients produces much better optimization performance than not using knowledge-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.

# 11 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 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 have implemented and tested knowledge-based gradients in several realistic design domains. Our experimental results show that using knowledge-based gradients can decrease the cost of design space search by one or more orders of magnitude.

# References


[Agogino and Almgren 1987] A. M. Agogino and A. S. Almgren. Techniques for integrating qualitative reasoning and symbolic computing. *Engineering Optimization*, 12:117–135, 1987.

[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.

[Bouchard 1992] E. E. Bouchard. Concepts for a future aircraft design environment. In *Proceedings, 1992 Aerospace Design Conference*, Irvine, CA, February 1992. AIAA-92-1188.

[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.

[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.

[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 *Artificial Intelligence in Design Conference*. Kluwer Academic Publishers, June 1996. To appear.

[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 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.

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

[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.

[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.

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

[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 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.

[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.* 1996] M. Schwabacher, T. Ellman, H. Hirsh, and G. Richter. Learning when reformulation is appropriate for iterative design. In *Artificial Intelligence in Design Conference*, June 1996. To appear.

[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. To appear.

[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 *Proceedings, 1992 Aerospace Design Conference*, Irvine, CA, February 1992. AIAA.

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

[Williams and Cagan 1994] Brian C. Williams and Jonathan Cagan. Activity analysis: the qualitative analysis of stationary points for optimal reasoning. In *Proceedings, 12th National Conference on Artificial Intelligence*, pages 1224–1230, Seattle, Washington, August 1994.

[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. Submitted, under review.