Computing in Aerospace 10, San Antonio, TX, March 1995 **AIAA-95-1016-CP**

A Computational Environment for Exhaust Nozzle Design

Andrew Gelsey and **Don Smith**
(gelsey@cs.rutgers.edu, dsmith@cs.rutgers.edu)
Computer Science Department
Rutgers University
New Brunswick, NJ 08903

Abstract

The Nozzle Design Associate (NDA) is a computational environment for the design of jet engine exhaust nozzles for supersonic aircraft. NDA may be used either to design new aircraft or to design new nozzles that adapt existing aircraft so they may be reutilized for new missions. NDA was developed in a collaboration between computer scientists at Rutgers University and exhaust nozzle designers at General Electric Aircraft Engines and General Electric Corporate Research and Development. The NDA project has two principal goals: to provide a useful engineering tool for exhaust nozzle design, and to explore fundamental research issues that arise in the application of automated design optimization methods to realistic engineering problems.

Introduction

The Nozzle Design Associate (NDA) is a computational environment for the design of jet engine exhaust nozzles for supersonic aircraft. NDA may be used either to design exhaust nozzles for new aircraft or to design new nozzles that adapt existing aircraft so they may be reutilized for new missions. NDA was developed in a collaboration between computer scientists at Rutgers University and design engineers at General Electric and Lockheed. The NDA project has two principal goals: to provide a useful engineering tool for exhaust nozzle design, and to explore fundamental research issues that arise in the application of automated design optimization methods to realistic engineering problems.

Figure 1 shows the NDA software architecture. The search space contains the possible nozzle designs whose performance is evaluated by the sim-

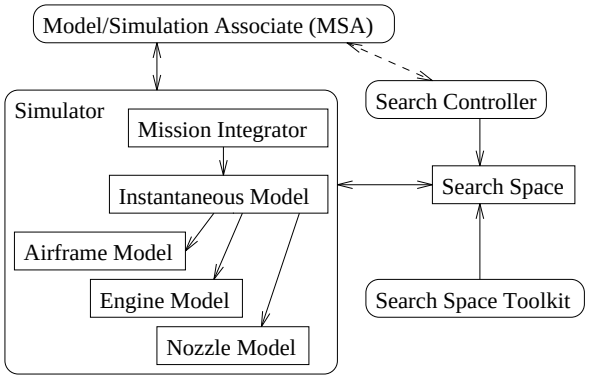

Figure 1: NDA software architecture

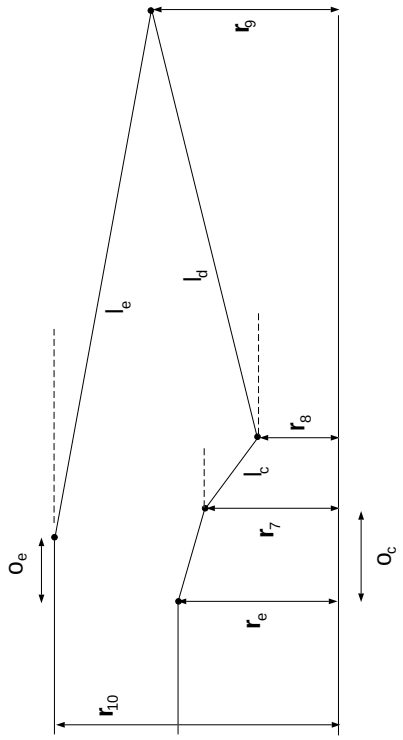

Figure 2: Axisymmetric Scheduled Convergent-Divergent Exhaust Nozzle

ulator using its models of relevant physics and with the help of the Model/Simulation Associate (MSA). The search controller looks for good nozzle designs in the search space using various optimization algorithms, and the search space toolkit is used to investigate the structure of the search space. The next four sections of this paper describe these four components of the NDA in more detail.

Simulator

Figure 2 shows the class of nozzles supported by the current NDA, the axisymmetric scheduled convergent-divergent exhaust nozzles often found in supersonic aircraft. [Mattingly *et al.* 1987] In Figure 2, r_{10}, r_e, and r_7 are fixed radii, and r_8 and r_9 are radii which are mechanically varied during aircraft operation. r_{10} is the outer radius of the engine to which the nozzle is attached, r_e is the radius of the duct leaving the engine, r_7 is the radius of the duct at the beginning of the movable convergent section of the nozzle, r_8 is the (variable) radius of the nozzle throat, and r_9 is the (variable) nozzle exit radius. Mechanically, this nozzle is a four-bar linkage, with three movable links labeled in Figure 2 by their lengths l_c, l_d, and l_e. During aircraft operation, the linkage is moved to change r_8 so that the cross-sectional area at the nozzle throat will produce desired engine performance. Since a four-bar linkage has one degree of freedom, setting r_8 also sets r_9. The job of NDA is to choose values for the parameters in Figure 2 to give optimal performance for a particular aircraft and flight mission.

In the current version of NDA, the design parameters defining the search space are the lengths of the convergent, divergent, and external nozzle flaps (l_c, l_d, and l_e in Figure 2). NDA optimizes the nozzle design under the constraint that the aircraft must be able to complete its designated mission, and with the goal that cost should be minimized. NDA currently uses gross takeoff mass as an approximation for cost, as takeoff mass is a rough combination of both acquisition cost (approximated by dry mass) and operating cost (approximated by fuel mass).

The NDA mission simulator is used both to verify that the aircraft can complete its designated mission and thus that the constraint is satisfied, and also to compute the total fuel mass consumed during the mission. NDA computes the fuel mass used during the mission by numerically solving the nonlinear 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. To compute the rate of fuel consumption, the mission simulator must determine the aircraft control settings (currently, throttle and angle of attack) at each point in the mission. The simulator chooses the control settings by solving the system of nonlinear equations

$$\mathbf{a}(\mathbf{c}) = \mathbf{a}_{\text{mission}}(t)$$

where $\mathbf{a}$ is the current acceleration vector (horizontal acceleration, vertical acceleration), $\mathbf{c}$ is the current control vector (throttle, angle of attack), and $\mathbf{a}_{\text{mission}}(t)$ is the acceleration vector required for the current time in the mission.

For each control setting, forces and fuel consumption rate are determined using the airframe, engine, and nozzle models. Presently the NDA uses airframe and nozzle models based on one-dimensional gas dynamics heavily supplemented by experimental data tables, and an engine model based on thermodynamic cycles with correction factors.

Model/Simulation Associate

Computational simulations of physical systems are traditionally run by human experts who can recognize simulation problems and deal with them. In contrast, the NDA simulator is invoked automatically by the NDA search controller, and the input and output of an NDA simulation may never been seen by a human. As a result, the NDA simulator architecture necessarily includes nontraditional enhancements which automatically detect and monitor simulation problems and thus allow the simulator to serve as a reliable subsystem. The NDA simulator presently includes three types of enhancements, which we call "spies", "saboteurs", and "selective backtracking". These simulator enhancements all work by communicating with a separate "intelligent software agent" called the MSA (Model/Simulation Associate, see Figure 1).

"Spies" are supplementary procedure calls added at particular points in an existing simulator which do nothing but transmit information to the MSA. Spies have no side effects of any kind within the simulator, and therefore do not change

the simulator's flow of control at all. If an existing simulator has been validated, and a source code comparison shows that the only change to the simulator has been the addition of spy calls, then the simulator will behave just as before and does not need to be revalidated.

"Saboteurs" are supplementary procedure calls added to the simulator which first ask the MSA what to do and then either do nothing or completely abort the simulation. For a validated simulator, the addition of saboteurs will not make revalidation necessary, since any successful simulation run will produce exactly the result it would have if no saboteurs were present. However, it is necessary that any person or program using the simulator must recognize that if a simulation aborts, then it has not returned meaningful information. The MSA instructs a saboteur to abort a simulation if the MSA determines, using information received from its spies, that the simulation will not be able to produce a valid result.

"Selective backtracking" is a somewhat more complex simulator enhancement which modifies the simulator's behavior at choice points, for example when the simulator chooses a starting point for a numerical iteration. Selective backtracking allows the simulator to try additional choices under control of the MSA if its original choice fails. Determining whether selective backtracking affects the validation of a simulator requires a more careful examination of the details of the particular simulator than the other enhancements, but in some cases it can be shown that adding selective backtracking does not mean that a simulator must be revalidated. We show such an example later in this section.

The MSA is an intelligent agent in an automated design system for handling tasks involving models of physical systems and the simulations which "execute" those models. One of the most important capabilities for the MSA is to monitor and control computational simulation and to recognize when the quality of simulation output is inadequate.

[Gelsey 1994] provides an extensive list of methods for automatically evaluating the quality of simulation output. For the purposes of this paper, we will focus on simulator problems arising from assumption violations. Models of physical systems always involve approximations and simplifying assumptions. If a simulator is given input which violates the assumptions underlying the model which the simulator is based on, then the simulator's output will not be trustworthy.

The MSA must have methods for processing the information received from its spies within a simulator to determine if the simulator's modeling assumptions have been violated. If the MSA detects a model violation, then it must either use a saboteur to abort the simulation or use selective backtracking to try to correct the model violation.

The models shown in Figure 1 are based on numerous assumptions, and spies have been inserted within the NDA simulator to send the MSA the information it needs to determine whether the assumptions are satisfied. Recovery from some model violations is impossible, so saboteurs have been put in the NDA simulator to abort simulations having those model violations. For example, if the search controller invokes the NDA simulator to evaluate a nozzle in which l_e is much longer than $l_c + l_d$, then it will not be possible to connect this nozzle to form a four-bar linkage. A basic assumption of the NDA simulator is that the nozzle is a four-bar linkage, so if this assumption is violated the simulator cannot give valid output. When the MSA detects this model violation, it instructs a saboteur within the NDA simulator to abort the simulation. The NDA search controller (see Figure 1) recognizes that it should ignore data from an aborted simulation and should not attempt to consider that parameter combination.

In some cases, the MSA can use selective backtracking to allow the NDA simulator to recover from model violations. For example, the NDA simulator solves the system of nonlinear equations

$$\mathbf{a}(\mathbf{c}) = \mathbf{a}_{\text{mission}}(t)$$

using Newton's method. Each step of the Newton iteration does function evaluations by calling the NDA physics models. If the NDA simulator chooses the wrong initial guess for the iterative Newton algorithm, the physics models may be invoked with input which violates their underlying assumptions. For example, the engine model assumes that flow through the nozzle becomes supersonic. If the initial guess for the Newton iteration has too low a throttle setting, it may be impossible for the nozzle to make the flow supersonic. This causes a model violation for the engine model. However, it is important to note that this model violation applies to the initial guess, and not necessarily to the true solution to the system of equations, which in fact may be fully consistent will all modeling assumptions. So the MSA uses selective backtracking to restart the Newton iteration with a different initial guess. Often this selective backtracking leads to a sound solution to the equations, which would have been missed if

the initial model violation had caused an aborted simulation.

In our MSA implementation, the MSA capabilities can be invoked selectively, which is convenient for running comparative experiments. To test the impact of the MSA on the nozzle design process, we ran a comparison study to determine the effects of MSA-controlled selective backtracking on numerical optimizations. We ran one set of forty optimizations with MSA-controlled selective backtracking enabled, and then ran the same set of optimizations with selective backtracking disabled. We used four different numerical optimization methods: the Fletcher-Reeves, Pollack-Ribiere, and Powell methods from [Press *et al.* 1992], and sequential quadratic programming in the CFSQP program from the University of Maryland. The same set of ten randomly chosen starting points were used for each optimization method.

We found that one of the optimization methods (CFSQP) proved very reliable when MSA-controlled selective backtracking was used by the simulator which the optimization method called to do function evaluations. All of the CFSQP runs from the ten different starting points found designs whose takeoff mass was within 1/4% of the best takeoff mass found by any run. On the other hand, without MSA-controlled selective backtracking, none of the optimization methods worked reliably: only 9 out of the 40 optimization runs found designs that were within 1% of the best design found.

Search Controller

Though automated design optimization has been applied to some engineering tasks for over three decades [Vanderplaats 1984], the majority of engineering design still relies on traditional trial-and-error techniques. This backwardness is partly due to cultural and educational factors, but in fact there are significant technical barriers that make the application of automated design optimization to realistic engineering problems a far-from-trivial task. Numerical optimization algorithms have traditionally been developed using abstract, mathematically well-behaved objective functions. However, the objective functions needed for engineering design are typically embodied in complex simulators such as the NDA mission simulator described above. These simulators tend to produce output which is not at all mathematically well-behaved, with problems ranging from nonsmooth transitions in the values computed for neighboring points in a design space to the common practice of aborting a simulation whenever design parameters violate the simulator's underlying assumptions.

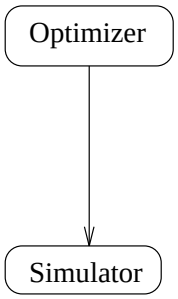

Figure 3: Simple simulate/modify loop

NDA addresses this problem by MSA-controlled simulator enhancements (as described in the previous section), by systematic investigation of search space structure (see the description of the search space toolkit in the next section), and by optimizer enhancements. NDA includes a number of extensions to traditional optimization algorithms to make them more robust. For example, NDA algorithms for numerical differentiation recognize when a simulator aborts trying to compute a function value. NDA then automatically attempts to recover by computing the derivative of the function using different step sizes thus avoiding the current "bad point".

Search Space Toolkit

The Search Space Toolkit (SST) is a suite of tools for investigating the properties of continuous search spaces. The search spaces which SST explores differ significantly from the discrete search spaces that typically arise in artificial intelligence research, and properly searching such spaces is a fundamental AI research area. Our SST research has focused on the problem of designing complex engineering artifacts and the analysis of the associated search spaces. Evaluation of points within these search spaces requires significant computation by a numerical simulator.

Figure 3 shows what might be called a "naive" approach to design automation: simply combine a standard optimizer with a simulator capable of evaluating candidate designs. Unfortunately, simulators are typically written with the assumption that they will be invoked by experienced human users, and making them robust enough for use in an automated environment like Figure 3 can be demanding. Even when some software engineering has been done to make the simulator and optimizer capable of working together, optimization results

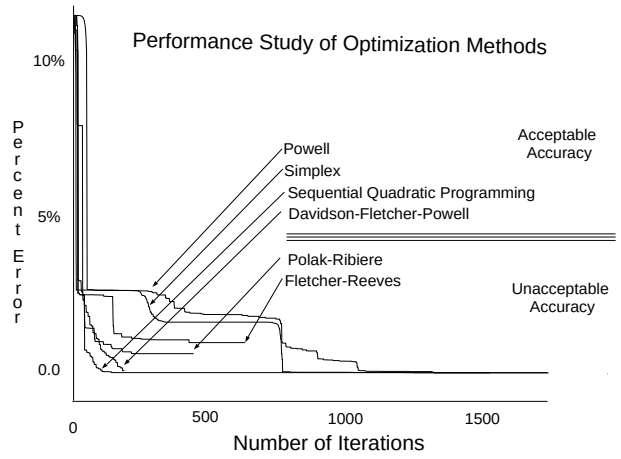

Figure 4: Experimental data showing quality of termination points of various optimization methods

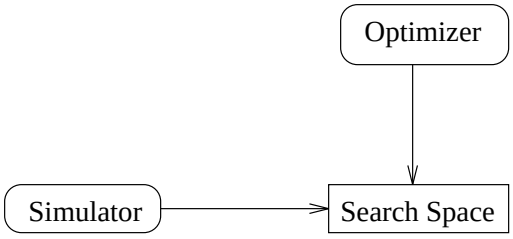

Figure 5: Optimizer searches space induced by simulator

tend to vary widely, as illustrated by the example in Figure 4, in which an exhaust nozzle simulator is combined with a number of different optimization algorithms from [Press *et al.* 1992]. In Figure 4, each optimizer was started at the same point and run until it could find no further design improvement. The horizontal axis shows the number of iterations the optimization method required to find its best point, and the vertical axis shows the deviation of the design quality of each termination point from the best point found by any method. In Figure 4, we sort the optimization methods into groups: those whose deviation was small enough to be "acceptable" for the current design goals, and those with larger, unacceptable deviation.

Figure 5 illustrates an alternative way of looking at the problem of automated design optimization. The viewpoint here is that the simulator implicitly defines a search space, which in turn is searched by the optimizer. The premise of our SST research is that automated design optimization has a much better chance of success if this search space is treated as a distinct entity whose geometry and topology should be investigated by

- number of local optima
- convexity
- "depth" of local optima
- smoothness; continuity of n^{th} derivative
- local properties in piecewise smooth regions
- evaluability of objective function
- topology/geometry of evaluable region
- constraints: explicit, implicit
- ridges; valleys
- plateaus
- noise

Figure 6: Search space properties

a variety of computational tools, rather than as a "black box" buried in the interface between an optimizer and a simulator.

Figure 6 lists a number of search space properties that are likely to be important in searching the space for an acceptable design. [Gelsey and Smith 1995] explains these properties in more detail, but in these paper we will just discuss how we have used SST to investigate the structure of the nozzle design search space.

The number of local optima in a search space is a critical property. Unfortunately, for an objective function defined by a large numerical simulation program, the information we are able to obtain about the number of local optima will generally be statistical in nature, rather than the subject of a mathematical proof. SST uses a Monte-Carlo-like multistart method for estimating the number of local optima: the algorithm repeatedly chooses random combinations of design parameters, uses the resulting design as a starting point for a numerical optimizer, and sorts the termination points of the optimizations into bins. (A byproduct of this process may be the identification of a global optimum, which is the best of the local optima.)

Properly classifying the termination points of the optimizations is a nontrivial task. For numerical reasons, even if two numerical optimizations end up at "the same" local optimum, they will typically stop at slightly different points due to numerical tolerances, etc. SST can do a line search between any two "close" termination points to determine whether they are in fact at the same local optimum and thus belong in the same bin.

A more important problem in classifying the termination points of the optimizations is whether the termination points are local optima at all. If the objective function includes complexity such as

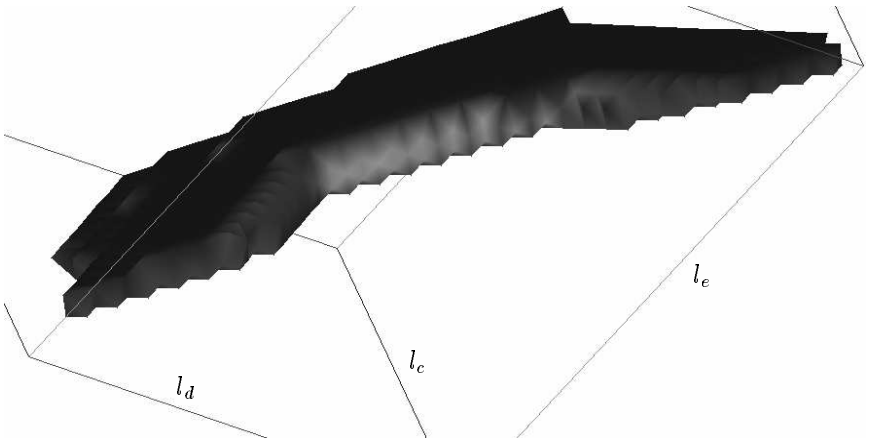

Figure 7: The "slab-shaped" evaluable region

ridges and unevaluable regions, then optimizers will often stop at points which are not true local optima. SST includes a local property analysis capability which can be applied at "interesting" local points, such as optimization stopping points. The SST local property analyzer generates a family of other points surrounding the point of interest by adjusting each design parameter by $+m\Delta h_i$ and $-m\Delta h_i$, where m is a small number (4, say), and Δh_i is an appropriate step size for numerical differentiation of the objective function with respect to design parameter i. The local property analyzer then computes the gradient numerically at each point in this family, and if the gradients are not all effectively the same, it partitions the neighborhood into piecewise smooth components. In each piecewise smooth component the Hessian (matrix of second partial derivatives of the objective function with respect to pairs of design parameters) is computed and diagonalized. If the neighborhood has only one component, the gradient is zero, and the Hessian is positive definite, then the point is identified as a true local optimum. Otherwise, further analysis may be required, particularly at points where an optimization terminated.

SST currently addresses the issue of objective function evaluability by a fixed grid sampling technique, both on large regions and on selected subregions of a search space. In the NDA exhaust nozzle search space, if SST imposes a grid on a large section of the search space spanning the "reasonable" range of values for the design parameters, sampling the grid points reveals that only about 4% of the grid points are evaluable. These grid points are contiguous, and form a "slab-shaped" evaluable region. Figure 7 graphically portrays the appearance of this slab using scientific visualization software. In the current NDA, the space of possible nozzle designs is three dimensional, since we only allow NDA to vary the three parameters l_c, l_d, and l_e, the lengths of the movable nozzle flaps. SST does not detect internal pockets of unevaluable points within this slab, suggesting that the evaluable region in this space is simply connected. The boundaries of this slab are implicit constraints on the acceptable combinations of design parameters for this problem.

Several of the SST tools have led us to the conclusion that the gross structure of the NDA nozzle design space is that of a valley. The SST fixed grid sampling reveals that the slab-like evaluable region has a thin surface running midway between the flat boundaries of the slab which contains designs much better than their neighbors closer to the outside of the slab. The optimizations run by the Monte-Carlo-like multistart techniques tend to stop on this central surface, though most of the stopping points are not local minima. If the Hessian matrix for a point on this central surface is diagonalized, one eigenvalue is much larger than the others, and its corresponding eigenvector is normal to the central surface. This data gather by SST strongly suggests that the central surface running through the middle of the slab is a higher dimensional analog of a ridge. The nozzle design objective function is an approximation of cost, which should be minimized, so we refer to this ridge as a "valley" and we refer to the central surface in the slab as the "valley floor". Optimizers tend to stop soon after finding the valley floor because the gradients driving the optimization towards the valley floor are very strong and tend to mask the much weaker gradients along the valley floor.

Figure 8 shows how the objective function varies

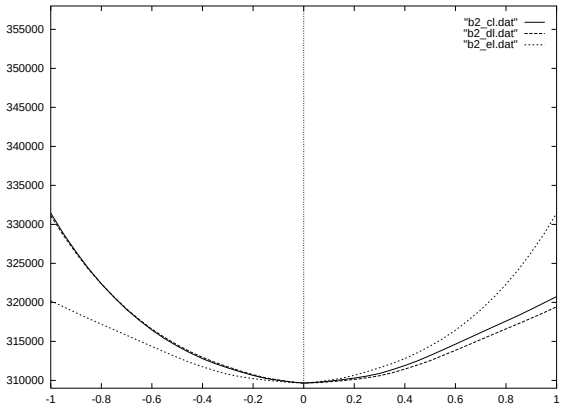

Figure 8: Objective as function of l_c, l_d, and l_e

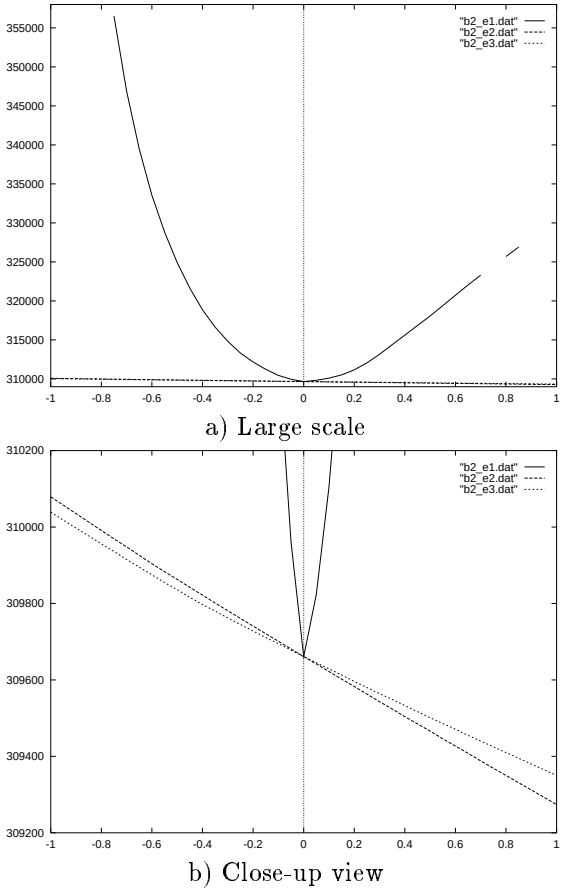

Figure 9: Objective function variation in directions of Hessian eigenvalues

about a "typical" optimization stopping point as a function of design space parameters (l_c, l_d, and l_e in Figure 2), and indicates that the gradient is approximately zero and the second derivatives positive, so that it was "legitimate" for the optimizer to stop here. Figure 9 shows how the objective function varies about the same optimization stopping point as a function of combinations of parameters in the direction of the eigenvectors of the Hessian at this point. Here we see that there may in fact be a "downhill" direction which is a linear combination of the eigenvectors corresponding to the two smaller eigenvalues, but that the other much larger eigenvalue is "masking" this possibility for improvement.

There is more to a search space than its gross structure. If gross analysis reveals that a search space is a valley, the next natural question is "what is the structure of the valley floor?". To investigate this issue, SST includes a tool we call "dimension reduction". The floor of a valley can be considered a search space in its own right, but a search space of dimensionality one less than that of the primary search space. Though the valley floor space has fewer dimensions, it may still have a very complex structure. To ascertain the properties of this subspace without having them masked by the strong gradients in the rest of the primary search space, SST must limit its evaluations to points exactly on the valley floor.

The SST dimension reduction algorithm works by projecting the desired subspace onto a hyperplane tangent to the subspace at some point. (A limitation of our current version of this algorithm is that it works poorly for subspaces with high curvature.) Linear algebra gives a coordinate system for the hyperplane with one less dimension than the primary space. This coordinate systems then serves as a coordinate system for the subspace by identifying each point P_s in the subspace with the nearest point P_h on the hyperplane. (I.e., the line defined by P_s and P_h is normal to the hyperplane.) Thus each function evaluation in the reduced dimension subspace requires a search in the primary space along the line normal to the corresponding point on the hyperplane in order to find the intersection with the subspace and evaluate the point of intersection. If the subspace is the valley floor in the nozzle search space, then each function evaluation requires solving a one-dimensional minimization problem, because the line normal to the hyperplane (just a plane in this case) will have its minimum value of the objective function where it intersects the valley floor.

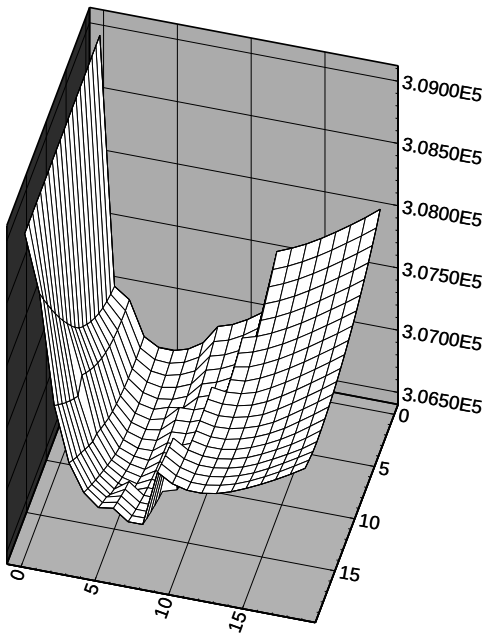

Figure 10: Valley floor structure for NDA nozzle design space

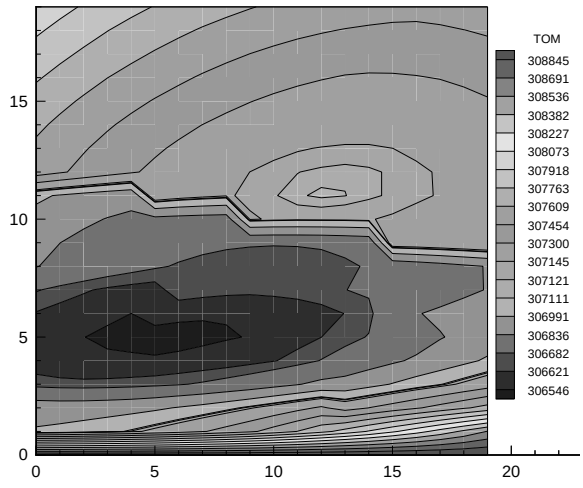

Figure 11: Valley floor contour plot

The SST dimension reduction algorithm has been applied to the NDA nozzle design search space. By combining the dimension reduction algorithm with our fixed grid sampling technique, we were able to determine the structure of the valley floor for the nozzle design space, as shown in Figure 10 (Figure 11 shows a contour plot of the same data). Note the apparent presence of two local optima, especially in the contour plot of Figure 11. The fixed grid sampling technique is too coarse to conclusively demonstrate that these apparent local optima are "real", but the "big picture" given by the sampling technique is very useful for suggesting points in the space where optimization algorithms and gradient and Hessian analysis may be able to identify local optima.

Related Work

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], though not much has been published about the particular difficulties of attempting to optimize functions defined by large "real-world" numerical simulators. Search has been a key focus of AI research from the field's beginning [Charniak and McDermott 1987], but most of the attention has been on discrete rather than continuous objective functions. A number of research efforts have combined AI techniques with numerical optimization [Tong *et al.* 1992, Bouchard *et al.* 1988, Bouchard 1992, Sobieszczanski-Sobieski *et al.* 1985, Agogino and Almgren 1987, Williams and Cagan 1994], but automated identification of search space properties has not been a focus in this work.

Conclusion

NDA combines automated optimization, computational simulation, and enhancements to each which allow them to work effectively together. The automation of the NDA computational environment enables a faster and more robust exhaust nozzle design process. It produces potentially superior designs by a combination of systematic optimization, which is less likely to overlook promising designs, and mission-oriented simulation, which produces a more balanced evaluation of candidate designs. NDA is the result of an active academic-industrial collaboration, and is a significant step in the direction of a new and better design methodology for exhaust nozzles and other aircraft components.

Acknowledgments

This research depended critically on our collaboration with Ron Luffy and Steve Scavo of General Electric Aircraft Engines and Gene Bouchard of Lockheed. We thank Gerard Richter, Mark Schwabacher, Khaled Mohamed Rasheed Shehata, and Keith Miyake for valuable contributions to the research described in this paper. This research is 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.

References

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

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.

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

Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Mass., 1987. Reprinted with corrections January, 1987.

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

Andrew Gelsey. Intelligent automated quality control for computational simulation. Technical Report CAP-TR-21, Department of Computer Science, Rutgers University, August 1994.

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

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

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

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

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C : the Art of Scientific Computing.* Cambridge University Press, Cambridge [England] ; New York, 2nd edition, 1992.

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

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.

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

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.