A Transformation System for Interactive Reformulation of Design Optimization Strategies*

Thomas Ellman John Keane Arunava Banerjee George Armhold
Department of Computer Science, Hill Center for Mathematical Sciences
Rutgers University, Piscataway, New Jersey 08855
{ellman,keane,arunava,armhold}@cs.rutgers.edu

Abstract

Automatic design optimization is highly sensitive to problem formulation. The choice of objective function, constraints and design parameters can dramatically impact the computational cost of optimization and the quality of the resulting design. The best formulation varies from one application to another. A design engineer will usually not know the best formulation in advance. In order to address this problem, we have developed a system that supports interactive formulation, testing and reformulation of design optimization strategies. Our system includes an executable, data-flow language for representing optimization strategies. The language allows an engineer to define multiple stages of optimization, each using different approximations of the objective and constraints or different abstractions of the design space. We have also developed a set of transformations that reformulate strategies represented in our language. The transformations can approximate objective and constraint functions, abstract or reparameterize search spaces, or divide an optimization process into multiple stages. The system is applicable in principle to any design problem that can be expressed in terms of constrained optimization; however, we expect the system to be most useful when the design artifact is governed by algebraic and ordinary differential equations. We have tested the system on problems of racing yacht design and jet engine nozzle design. We report experimental results demonstrating that our reformulation techniques can significantly improve the performance of automatic design optimization. Our research demonstrates the viability of a reformulation methodology that combines symbolic program transformation with numerical experimentation. It is an important first step in a research program aimed at automating the entire strategy formulation process.

*Fully accepted to *Research in Engineering Design*.

1 Introduction

Numerical design optimization is a notoriously unreliable process. Optimization programs often take excessive time to reach termination. Furthermore, upon termination, optimization programs often fail even to reach locally optimal designs. These difficulties typically arise if the constraints or objective functions are expensive to evaluate; the problem contains many design parameters; or the search space contains pathologies, such as ridges, discontinuities, plateaus or non-evaluable regions. Design engineers can in principle overcome these difficulties, by carefully formulating the inputs to numerical optimization codes. In particular, by carefully choosing search spaces, design parameters, and approximations of the objective and constraint functions, a design engineer can dramatically reduce the duration of the optimization process, and improve the quality of the resulting design. Unfortunately, a design engineer may not know the best formulation in advance of attempting to set up and run a design optimization process. The best formulation may vary from one application domain to another, and from one problem to another within a given application domain.

Design problems can often be cast in the form of constrained optimization, defined formally in Figure 1. Given a set of problem instance parameters, and a set of design parameters to be varied, one seeks to optimize an objective function, while satisfying equality and inequality constraints [Peressini *et al.*, 1988]. A variety of numerical algorithms have been developed for attacking constrained optimization problems [Press *et al.*, 1986]. These numerical tools unfortunately provide only limited means of overcoming the difficulties described above. In particular, they take the objectives, constraints and design parameters as givens, provided in advance by a human user, and remaining fixed during solution of the problem. Furthermore, they treat objective and constraint functions as "black boxes". They can be evaluated; however, their internal structure can neither be seen nor modified by the numerical optimization algorithm. Our research is based on the hypothesis that more robust design optimization methods can be constructed by combining existing numerical algorithms with techniques for reasoning about and manipulating the mathematical structure of the objective and constraint functions.

In order to illustrate our hypothesis, let us consider how approximations are used in numerical optimization algorithms. An illustrative example is CFSQP, a state of the art code for sequential quadratic programming [Lawrence *et al.*, 1995]. CFSQP is a "Quasi-Newton" method. Taking a seed design as input, it approximates the objective function at the seed point using a quadratic function. It solves the resulting quadratic program to find a new design. CFSPQ repeats the approximation-solution process until satisfying convergence criteria. CFSQP uses purely numerical methods to approximate the objective function. The quadratic approximation results from evaluating the objective function, treated as a black box, at a series of points along the search path. We believe that better approximations can be constructed by exploiting the internal structure of the objective and constraint functions. An important type of approximation involves intermediate quantities that are computed inside the objective and constraint functions, but which do not appear as final results. For example, if the value or gradient of an intermediate quantity does not change much during the optimization process, the quantity may be replaced by a constant or linear approximation. In general, we expect that such *internal approximations* can be more cost-effective than approximations constructed by treating objective and constraint functions as black boxes.

As another illustration of our hypothesis, consider how design parameter coordinate systems are chosen by numerical optimization algorithms. An illustrative example is Powell's direction set method [Press *et al.*, 1986]. Powell's method operates by computing a linear transformation of the

Given:

Problem Instance Parameters: $\bar{p} = (p_1, \ldots, p_m)$

Design Parameters to be Varied: $\bar{d} = (d_1, \ldots, d_n)$

Optimize an Objective Function: $f(\bar{d}, \bar{p})$

Subject to Constraints:

Inequality Constraints: $\bar{g}(\bar{d}, \bar{p}) \leq \bar{0}$

Equality Constraints: $\bar{h}(\bar{d}, \bar{p}) = \bar{0}$

Figure 1: The Constrained Optimization Problem

design parameters to find a set of conjugate directions. It then uses the transformed parameters to find an optimum. The transformation-solution process is repeated until convergence criteria are satisfied. Powell's method uses purely numerical methods to change the coordinates of the design parameters, i.e., the transformation of coordinates is computed by evaluations of the objective function, treated as a black box. We believe that better changes of parameters can be constructed by exploiting the internal structure of the objective and constraint functions. For example, intermediate quantities may be used to construct non-linear changes of coordinates that align the design parameters with ridges or discontinuities that create problems for numerical optimization. These observations suggest a methodology of simultaneously formulating search spaces, objective functions and constraint functions in combination with the design optimization strategies that use them.

Stochastic techniques are sometimes useful for optimizing pathological objective functions. Examples of stochastic techniques include simulated annealing [Kirkpatrick *et al.*, 1983] and genetic algorithms [Mitchell, 1996]. Stochastic methods usually require a large number of evaluations of the objective and constraint functions in order to generate reliable results. They are often impractical if these functions are computationally expensive to evaluate. This situation arises quite often in engineering design problems, for example, when the functions exercise computational fluid dynamics (CFD) codes. Stochastic methods may also be impractical even when the objective and constraint functions require fairly small amounts of CPU time. For example, during the early phases of a design process, an engineer may wish to use a fast approximation of the objective function to carry out a large number of optimizations, each under a different set of assumptions about the operating environment of the artifact. The time available for each optimization may be too small for the large number of function evaluations needed for stochastic optimization, even though each evaluation itself requires only a small amount of CPU time. Both of these considerations suggest a need for reliable optimization techniques that use fewer function evaluations than are typically required by stochastic methods. Our work is based on the belief that fast and reliable optimization techniques can be constructed from gradient-based algorithms, such as $CFSQP$, by carefully formulating search spaces, objective functions and constraints. We also expect that careful formulation can improve the performance of stochastic optimization methods like simulated annealing and genetic algorithms; however, we have not investigated this possibility.

An effective optimization strategy must often include multiple stages, each using a different search space, a different objective function or different constraints. There are several reasons for this. First, suppose the design space includes pathologies such as ridges or discontinuities. Such pathologies can sometimes be removed by abstracting the original space. Abstraction replaces the original design space by a new space with lower dimension and fewer pathologies. Optimization proceeds in stages: first in the abstract space, to find an approximate solution, and then in the original space, to find a true solution. Now suppose that the objective or constraint functions are expensive to evaluate. In principle the exact functions need only be used in the neighborhood of a final solution, to verify local optimality and satisfaction of constraints. The search for a solution may be guided by approximations of the objective function and constraints. The optimization process may therefore be divided into stages. Early stages use approximations. Exact objective and constraint functions are used only at the end.

2 The DA-MSA Architecture

We have developed a system called the "Design and Modeling/Simulation Associate" (DA-MSA). The DA-MSA is an environment that supports interactive formulation, testing and reformulation of design optimization strategies. A "strategy" is a description of a process for solving a design optimization problem. A strategy may use multiple search spaces, each at a different level of abstraction. A strategy may also use multiple versions of objective and constraint functions, each at a different level of approximation. Finally, a strategy may involve multiple stages of optimization, using different search spaces, different objective functions or different constraints at each stage.

The architecture of the DA-MSA is shown in Figure 2. A human engineer uses a graphical interface to build a constraint network that represents a specification of a design problem. The specification includes a description of the physics of the design domain, in terms of algebraic and ordinary differential equations. It also includes a description of an optimization problem (design parameters, objective, constraints) that represents the ultimate, true problem the engineer wishes to solve. In the first phase of operation, the DA-MSA uses a technique known as "deductive program synthesis" to create an initial, default optimization strategy. Our research on this portion of the system is reported in [Ellman and Murata, 1996].

In the second phase of operation, the user interacts with the DA-MSA to reformulate the initial strategy into a more robust and efficient one. He does so by using the interface illustrated in Figure 3. Optimization strategies are represented visually to the user as data-flow graphs. A portion of a data-flow graph appears on the right side of Figure 3. Nodes in data-flow graphs represent operations on data. Arcs represent flow of data from one operation to another. Our data-flow graphs represent "second-order" programs, i.e., they include second-order operations like optimization, integration and root extraction, which take functions as arguments. The user reformulates his optimization strategy by drawing upon a catalog of transformations. Each transformation converts one data-flow graph into another. The transformations can approximate objective and constraint functions, abstract or reparameterize the search space, or divide an optimization into multiple stages, among other things. By applying various combinations of transformations the user can explore a large space of strategies.

As the user explores the strategy space, the system maintains a record of the strategies he has generated. A portion of such a record appears on the left side of Figure 3. The record is organized

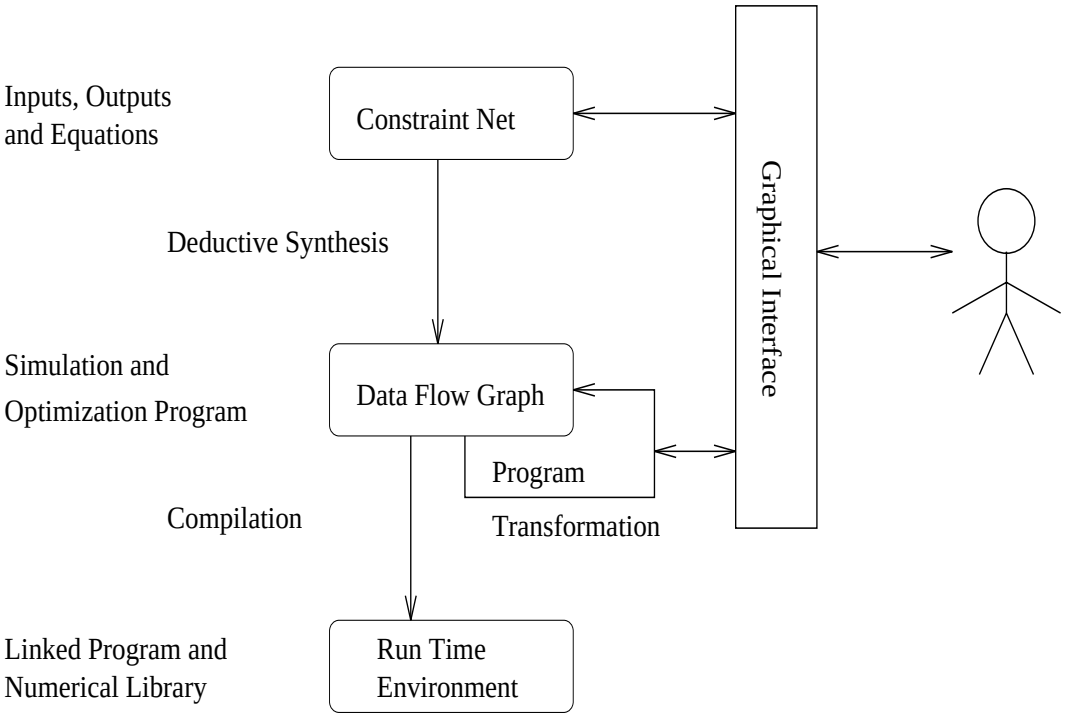

Figure 2: Architecture of the DA-MSA

as a tree. Each node in the tree holds a description of an optimization strategy. The user can move around in the tree to backtrack to previous strategies and try new alternatives. He can run the strategy in any tree node on a set of test problems and store the results in a database associated with that node. He can also generate plots that describe the behavior of strategies on test problems, e.g., a plot of the path through a design space, or a plot of the evolution of an objective, constraint, or intermediate quantity. Finally, the user can annotate the current strategy node with his own observations and conclusions about the behavior of the strategy.

The DA-MSA provides several different kinds of support to an engineer who is attempting to develop an effective design optimization strategy. To begin with, it frees him from the time-consuming process of writing in a conventional programming language each time he wants to change the current strategy. It does so by allowing him to specify a new strategy as a transformation applied to an existing strategy. In addition, the system provides some conditional guarantees about the behavior of strategies that are derived using the catalog of transformations. In particular, under certain clearly identifiable conditions, derivations constructed in our system are guaranteed to preserve the correctness and convergence properties of the initial strategy. This stands in contrast to the situation that obtains when programs written in a conventional language are modified in an unrestricted manner. Furthermore, the tree structured record of the derivation serves as a kind of documentation. It does so by indicating the sequence of transformations used to derive the final strategy. In addition, the record may facilitate maintenance and modification of optimization strategies. When the initial strategy is modified to suit changing circumstances, it may be possible to replay the sequence of transformations in order to modify the derived strategy accordingly;

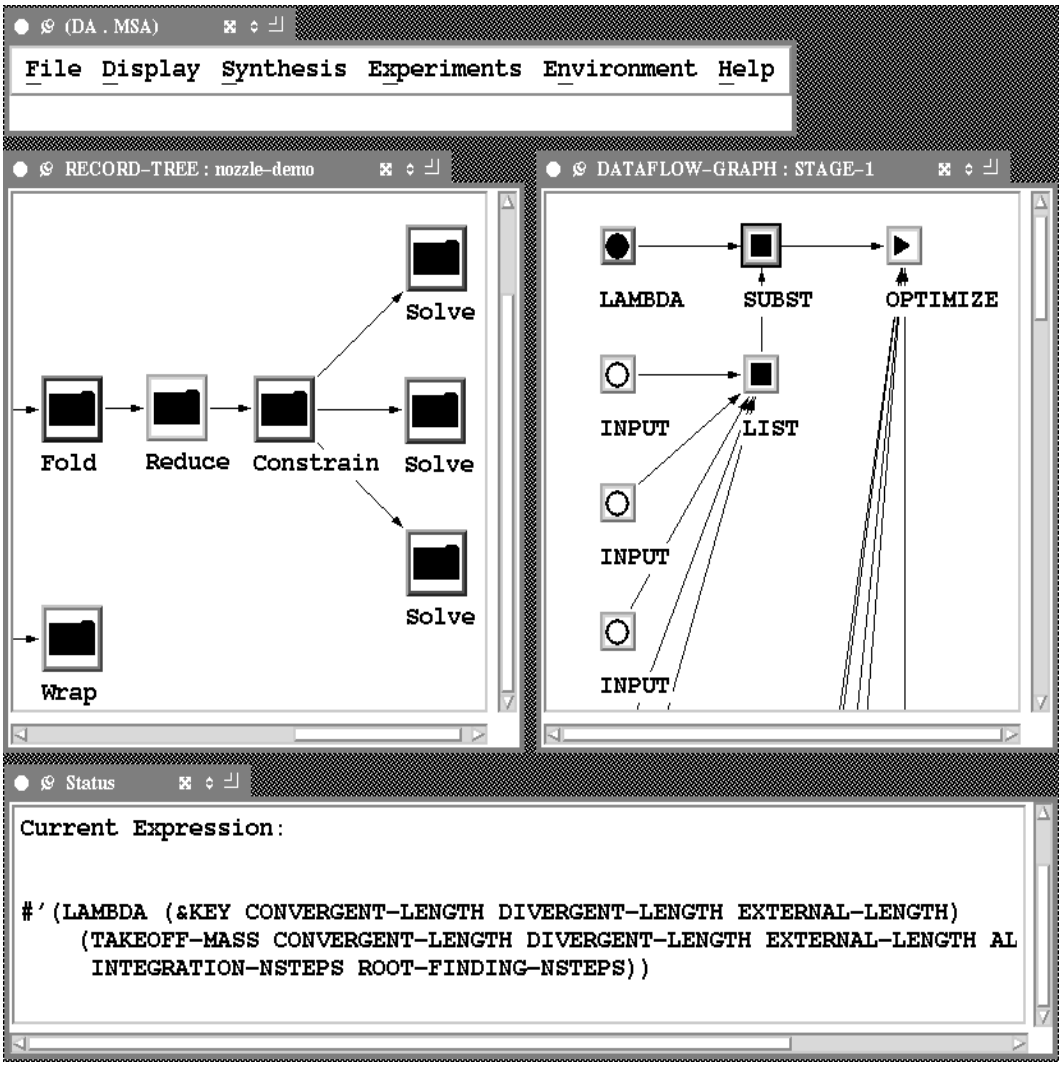

Figure 3: User Interface of the DA-MSA

however, we have not experimentally investigated this possibility. Our system thus provides the standard benefits of transformational programming [Partsch and Steinbruggen, 1983], [Mostow, 1989] to engineers developing design optimization strategies.

We shall begin by describing two design applications on which we have tested our system: sailing yachts and jet engine nozzles (Section 3). Next, we shall describe our language for representing design optimization strategies (Section 4). After that, we shall describe the catalog of transformations that convert one strategy into another (Section 5). Next, we shall investigate the degree to which our transformations preserve the correctness and convergence properties of the initial strategy supplied to the system (Section 6). In the following section, we shall describe derivations of several strategies for optimizing sailing yacht designs and jet engine nozzle designs (Section 7). Next, we present results of experimentally testing yacht and nozzle strategies generated using our system (Section 8). Finally, we shall discuss related work (Section 10), summarize our contributions (Section 11), and outline our plans for future work (Section 12).

3 Testbed Domains

3.1 Design of Sailing Yachts

Our research on yacht design has attempted to reconstruct the design process that led to the "Stars and Stripes '87", the sailing yacht that won the America's Cup race in 1987 [Letcher *et al.*, 1987]. The yacht design problem involves determining values of the major dimensions of the hull of a racing sailboat. These include the length l, beam b and depth d of the "canoe-body", the height h of the "keel", and the span s of "winglets" attached to the keel. These dimensions must be chosen to maximize the steady-state velocity the yacht achieves under racing conditions specified by the velocity w of the wind and the angular heading β at which the yacht sails relative to the wind. The steady state velocity is computed by solving simultaneous balance of force and torque equations for the velocity v and the heel angle ϕ at which the yacht will be in steady state motion. The equation $f(v, \phi, w, \beta, l, b, d, h, s) = 0$ specifies that the thrust generated by the wind is balanced by the air and water resistance on the yacht. Likewise, the equation $\tau(v, \phi, w, \beta, l, b, d, h, s) = 0$ specifies that the heeling torque generated by the wind and water is balanced by the gravitational and buoyant torques acting on the yacht. The functions f and τ are specified by explicit algebraic formulae in combination with functions that interpolate tabular data describing the aerodynamic properties of the sail and the hydrodynamic properties of the hull. A yacht design problem *instance* is defined by the heading β and windspeed w in which the yacht sails. The yacht problem *class* is thus parameterized by β and w. A real yacht race course includes a series of "legs" each defined by a heading and a distance. We are dealing with a simplified version of the racing yacht design problem. For the experiments reported in this paper, we used an implementation of the velocity objective function that requires about 1.0 CPU seconds per evaluation, on a Sun Microsystems Sparc 5 workstation. We intend this version of the objective function to be representative of the sort of fast approximation that an engineer might use during early phases of a yacht design process. In related work [Ellman *et al.*, 1997] we used a more accurate and computationally expensive implementation, requiring about 1.0 CPU hours per evaluation. This implementation represents the sort of high fidelity code that an engineer might use during later stages of a yacht design process.

The yacht problem presents a challenge to automated design optimization technology for two reasons: the presence of pathologies in the design space and the computational expense of evaluating some versions of the objective function. Pathologies arise in the yacht domain due to the mathematical properties of the balance of force equations. In particular, the net force f is not a smooth function of the design parameters. Non-smoothness results from the fact that the sizes of the yacht sails are computed dynamically, inside the net force function, to be as large as possible while satisfying a "rating rule" imposed by a yacht racing association. The rating rule provides a formula giving the largest legal sail area as a function of the major dimensions of the hull. The formula includes conditional expressions that cause the sail area, and therefore the net force, to be a non-smooth function of the design parameters. The non-smoothness in net force presents a potential problem for design optimization by generating ridges in the velocity objective function. Non-smoothness provides a motivation for reformulating the search space, as described below. The computational expense of evaluation is potentially an important issue when using high fidelity versions of the velocity objective function, or when running a large number of optimizations, each using different assumptions about the operating conditions of the yacht. The cost of evaluation

provides a motivation for constructing and utilizing approximate versions of objective functions or constraints, as described below.

3.2 Design of Jet Engine Nozzles

The nozzle design problem involves determining the lengths of three flaps that regulate the jet-engine exhaust flow in a supersonic aircraft: the convergent flap l_c, the divergent flap l_d, and the external flap l_e. These flap lengths must be chosen to minimize the total fuel consumption over a specified flight mission. A mission is specified by an altitude h, a velocity v at which the aircraft flies, and the duration d of the flight. Fuel consumption is computed by integrating a differential equation governing instantaneous fuel mass m flow: $\dot{m} = f(m, h, v, l_c, l_d, l_e)$. The equation is integrated backwards in time, starting with the empty mass of the aircraft at the end of the mission, and ending with the "takeoff mass" of the fully fueled aircraft at the start of the mission. This differential equation cannot be solved in closed-form. It must be solved numerically. Furthermore, the derivative function f is not given explicitly. Instead it must be computed by solving three simultaneous equations for the instantaneous values of three control parameters that are continuously varied by the aircraft control systems: the angle of attack α, the throttle setting t, and the nozzle convergent flap angle γ. The three equations specify that vertical and horizontal forces on the aircraft be in balance (for steady state motion) and that the air/fuel mixture have a velocity of mach 1.0 at the narrowest point in the nozzle – a setting known to achieve maximum fuel efficiency. One of these equations can be solved in closed form. The other two can be decomposed yielding the equations: $a_v(\alpha, m, h, v, l_c, l_d, l_e) = 0$ and $a_h(t, \alpha, m, h, v, l_c, l_d, l_e) = 0$ asserting that vertical and horizontal acceleration are each zero. Each must then be solved numerically. These equations are specified by explicit algebraic formulae in combination with functions that interpolate tabular data describing the flight characteristics of the aircraft and the engine(s). A nozzle design problem *instance* is defined by the altitude h, velocity v and duration d of the mission. The nozzle problem *class* is thus parameterized by h, v and d. Our research on jet engine nozzle design is described more fully in [Gelsey *et al.*, 1996]. For the experiments reported in this paper, we used an implementation of the takeoff mass objective function that requires about 23.1 CPU seconds per evaluation, on a Sun Microsystems Sparc 5 workstation. We intend this version of the objective function to be representative of the sort of fast approximation that an engineer might use during early phases of an aircraft design process. In related work on aircraft design [Shukla *et al.*, 1997], our colleagues have used objective functions requiring $1 - 2$ CPU hours per evaluation. These represent the sort of high fidelity codes that an engineer might use during later stages of an aircraft design process.

The nozzle problem presents a challenge to automated design optimization technology for two reasons: the presence of pathologies in the design space and the computational expense of evaluating the objective function. Pathologies arise in the nozzle domain because the evaluation code fails entirely for many points in the design space. Crashes result from table lookups going out of bounds, failure of numerical root extractors to find roots, and violation of geometric constraints on the flap lengths, among other things. The design optimization process is protected from crashes of the objective function by wrapping the objective function with a routine that gains control in the event of such failures. The wrapper keeps the optimization process running by returning an extremely bad value for the objective function. Unfortunately, the wrapper also introduces discontinuities and plateaus into the objective function. These pathologies often cause gradient-based optimization

to get stuck at points that are not even locally optimal. Discontinuities and plateaus provide a motivation for reformulating the search space, as described below. The computational expense of evaluation is potentially an important issue when using high fidelity versions of the takeoff mass objective function, or when running a large number of optimizations, each using different assumptions about the operating conditions of the aircraft. The cost of evaluation provides a motivation for constructing and utilizing approximate versions of objective functions or constraints, as described below.

4 Specification of Optimization Strategies

Our language of optimization strategies is presented in Figure 4. The language is described by a BNF grammar. Each sentence generated by the grammar is a LISP expression that represents an optimization strategy. The language enables a design engineer to specify strategies that involve multiple stages of optimization. Stages may involve different approximations for objective and constraint functions, different search spaces, different coordinate systems or different optimization algorithms. The language also enables an engineer to specify decompositions of search spaces, methods of using and periodically re-calibrating approximate objective and constraint functions, and methods of generating starting points for optimization algorithms based on local search.

A single-stage optimization strategy may be generated by using rule (1) to expand the start symbol S. The resulting expression describes a strategy that starts with a "seed" design (S), and optimizes a real-valued objective function (OBJ) over a hyper-rectangular region subject to inequality constraints and equality constraints. The hyper-rectangle is specified by lists of upper (UBs) and lower (LBs) bounds. The equality constraints ($EQNs$) are designated by a list of real-valued functions whose values must be zero. The inequality constraints ($INEQNs$) are designated by a list of real-valued functions whose values must be zero or less. The strategy expression also includes a symbol ($OptMETH$) designating a numerical optimization method, which may be any one of a variety of gradient-based or stochastic algorithms. The seed (S) may be generated by an expression described by any of the rules for expanding the non-terminal symbol S. For example, the seed may be simply a list of design parameters – expanding S using rule (7). It may also be a design that is randomly generated within a hyper-rectangle specified by lists of upper and lower bounds – expanding S using rule (6). In addition, the seed may itself be generated by a complex optimization strategy – expanding S using any of rules (1) through (5). Nested strategies, involving multiple stages of optimization, may thus be specified by exploiting the recursive structure of the grammar.

A simple multi-stage strategy results from using rule (1) twice to construct a strategy with two stages of optimization, perhaps using different objectives, constraints, design parameters or optimization methods in each stage. Strategies based on multiple starting points are generated using rule (2). This rule generates an expression describing a process that applies an optimization strategy function ($\lambda(d)S$) to each of several seed designs ($list\ \ S^*$). It compares the results, using a boolean-valued function BF, and selects the best. Each of the starting points in ($list\ \ S^*$) may be generated using a different strategy. Strategies involving decomposition are generated using rule (3). This rule generates an expression describing a design composed of several partial designs. Each partial design may be found using a different strategy.

Strategies involving approximation of constraints or objective functions may be generated by

$$
\begin{array}{rrcll}
(Strategy) & S & \rightarrow & (optimize\ S\ OBJ\ EQNs\ INEQNs\ LBs\ UBs\ OptMETH) & (1)\\
 & S & \rightarrow & (select\ (list\ S^*)\ (\lambda(d)S)\ BF) & (2)\\
 & S & \rightarrow & (compose\ (list\ S^*)) & (3)\\
 & S & \rightarrow & (let\ ((SYMBOL\ RV)^*)\ S) & (4)\\
 & S & \rightarrow & (converge\ S\ (\lambda(d)S)\ BF) & (5)\\
 & S & \rightarrow & (randomize\ (list\ RV^*)\ (list\ RV^*)) & (6)\\
 & S & \rightarrow & (list\ RV^*) & (7)\\
 & S & \rightarrow & (insert\ S\ INDEX\ RV) & (8)\\
 & S & \rightarrow & (delete\ S\ INDEX) & (9)\\
 & OBJ & \rightarrow & RF & (10)\\
 & EQNs & \rightarrow & (list\ RF^*) & (11)\\
 & INEQNs & \rightarrow & (list\ RF^*) & (12)\\
(LowerBounds) & LBs & \rightarrow & (list\ RV^*) & (13)\\
(UpperBounds) & UBs & \rightarrow & (list\ RV^*) & (14)\\
(Real\ Function) & RF & \rightarrow & (\lambda\ (d)\ RV) & (15)\\
(Real\ Value) & RV & \rightarrow & 1ST\ \mid\ 2ND \mid\ (BLACKBOX\ RV^*) & (16)\\
(Boolean\ Function) & BF & \rightarrow & (\lambda\ (d_1 d_2)\ BV) & (17)\\
(Boolean\ Value) & BV & \rightarrow & (RELATION\ RV\ RV) & (18)\\
(First\ Order\ Real) & 1ST & \rightarrow & ARITHMETIC \mid CONDITIONAL \mid INTERPOLATION & (19)\\
(Second\ Order\ Real) & 2ND & \rightarrow & ROOT\ \mid\ INTEGRAL & (20)\\
 & ROOT & \rightarrow & (root\ (list\ RF^*)\ (list\ RV^*)\ RtMETH) & (21)\\
 & INTEGRAL & \rightarrow & (integrate\ (list\ RF^*)\ (list\ RV^*)\ RV\ RV\ IntMETH) & (22)\\
 & RtMETH & \rightarrow & newtonraphson\ \mid\ bisection\ \mid\ \ldots & (23)\\
 & IntMETH & \rightarrow & rungekutta\ \mid\ \ldots & (24)\\
 & OptMETH & \rightarrow & cfsqp\ \mid\ dfp\ \mid\ simplex\ \mid\ genetic\ \mid\ annealing\ \mid\ \ldots & (25)
\end{array}
$$

Figure 4: Design Optimization Strategy Grammar

using rule (4). This rule allows placing an optimization strategy inside the scope of variables that hold fitting coefficients that are computed prior to the start of the strategy. Iterative strategies are generated using rule (5). This rule generates an expression specifying a convergence process. The process starts with a seed design and repeatedly applies a strategy function $(\lambda(d)S)$ until a boolean-valued comparison function BF applied to successive iterates indicates that a convergence criterion has been reached. When rules (4) and (5) are used in combination, the result is a strategy that periodically re-calibrates approximate objective or constraint functions as it moves through a design space.

The remaining rules are used to express objective and constraint functions in terms of first-order operations (arithmetic, interpolation, conditionals) as well as second-order operations (integration and root extraction). The grammar rules also allow for domain-specific "black box" codes to be included in optimization strategies. Black boxes can be used to evaluate objective or constraint functions, or portions thereof. The strategy expressions generated by our grammar are all executable

in the DA-MSA runtime environment. Complex numerical operations, such as *optimize*, *integral* and *root* are actually implemented by C routines taken from [Press *et al.*, 1986], or public domain sources [Lawrence *et al.*, 1995]. The C routines are wrapped by LISP functions that provide the interfaces described here [Keane, 1996].

Each construct in the strategy language plays an important role in design optimization. Nested strategies are useful for applying a sequence of optimization methods (e.g., downhill simplex followed by sequential quadratic programming) in succession, because neither one alone is expected to reliably reach an optimum. Nested strategies are also useful when an early stage of optimization is expected to produce a sub-optimal result, perhaps because it uses an approximate objective or constraint function, or searches only an abstract version of the original design space. The early stage serves to generate a seed design for later stages of optimization. Decomposition is useful when the objective and constraint functions are decomposable (e.g., a sum of functions defined on the factor spaces) or nearly decomposable. It is also useful when the best strategy for one factor space is different from the best strategy for another (e.g., when approximations suitable for one factor space differ from approximations suitable for another factor space). Multiple starting points are useful because many search methods get stuck on local optima, plateaus, ridges or discontinuities. By using multiple starting points, one increases the likelihood of finding a true local, or even global, optimum. Approximation is useful when most of the values computed internally by an objective or constraint function are not expected to change much when moving from point to point in the design space. Such quantities can be approximated by constants or linear functions prior to the start of an optimization strategy. The constants or linear functions may be periodically re-calibrated as the strategy is iterated to a fixed point.

Several aspects of our strategy language are worth emphasizing. To begin with, notice that the language is highly orthogonal. It allows primitives like *optimize*, *select*, *compose* and *converge* to be combined with each other systematically. In addition, notice that the language includes constructs for defining optimization processes (in the first nine rules) along with constructs for defining simulation and analysis processes (in the remaining sixteen rules). Both optimization and simulation/analysis are described within the same language. This feature is important when it comes to writing transformations that operate on expressions in our language. In particular, many important types of transformation work by simultaneously changing simulation/analysis functions and the optimization process in which they are embedded. Our strategy language provides a uniform level of representation in terms of which such transformations can be defined.

5 Transformation of Optimization Strategies

The reformulation process begins with an initial, default strategy. The initial strategy may be constructed by the deductive-synthesis component of the DA-MSA. It may also be hand-coded directly by the user. (The initial strategies used in the experiments reported in this paper were all coded by hand.) The strategy is represented visually on the screen as a data-flow graph. It is also represented internally as an expression of the form:

$$(\lambda(d)\ \ (optimize\ \ d\ \ OBJ\ \ EQNs\ \ INEQNs\ \ LBs\ \ UBs\ \ OptMETH))$$

This expression describes a function that takes a seed design d and uses a numerical algorithm $OptMETH$ to optimize an objective function OBJ over a hyper-rectangular region defined by

lower LBs and upper UBs bounds, subject to sets of equality constraints $EQNs$ and inequality constraints $INEQNs$. The initial strategy normally includes all potentially relevant design parameters, and the "exact" versions of the objective and constraint functions. The user modifies the current strategy by drawing upon a catalog of transformations. First, he selects a transformation from the catalog. In many cases, the transformation can be applied to the current strategy in more than one way. In such cases, the system asks the user which instantiation of the transformation he wants to apply. Then the system applies the selected transformation and displays the revised strategy to the user, symbolically as a LISP expression and visually as a data-flow graph. The revised strategy then becomes the current strategy. The user may proceed to transform the current strategy once again, or he may back up and try alternative transformations.

Our system includes five groups of transformations. Transformations that reformulate the search space are described in Figure 5. Transformations that introduce approximations into objective and constraint functions are described in Figure 6. Transformations that construct nested optimization strategies are described in Figure 7. A fourth group includes transformations that insert wrappers that catch error conditions and allow optimization to continue. These are described in Figure 10. A fifth group includes equivalence preserving transformations that improve efficiency (e.g. transforms that remove duplicate or unreferenced sub-expressions) or that control the granularity of the user's view of strategies (e.g., transforms that fold or unfold function definitions).

5.1 Reformulating Search Spaces

Transforms that reformulate search spaces are described in Figure 5. These transforms are designed to use intermediate quantities appearing in the objective or constraint functions as a basis for reparameterizing or reducing the dimension of design spaces. Consider first how an intermediate quantity may be used to reparameterize a search space. The engineer begins by identifying an intermediate quantity $Q(x_1, \ldots, x_n)$. He then uses the transform "Parameterize Intermediate Quantity" to define a new design parameter y. The new parameter is made to equal the intermediate quantity by inserting an equality constraint: $y = Q(x_1, \ldots, x_n)$. The engineer then uses the transform "Solve Equality Constraint" to remove one of the original parameters x_i from the design space. This transform will solve $y = Q(x_1, \ldots, x_i, \ldots, x_n)$ for x_i symbolically (using Maple [Char *et al.*, 1992]) if possible. Otherwise, it will insert an expression that solves $y = Q(x_1, \ldots, x_i, \ldots, x_n)$ for x_i numerically (e.g., using Newton-Raphson) at the time the strategy is executed. The overall effect of these two transforms is to change the set of parameters that describe points in the design space. Now consider how an intermediate quantity may be used to reduce the dimension of a design space. The engineer first uses the transform "Constrain Intermediate Quantity" to install an equality constraint $Q(x_1, \ldots, x_n) = K$ on an intermediate quantity. He then uses the transform "Solve Equality Constraint" to remove one of the design parameters, x_i, by solving for x_i in $Q(x_1, \ldots, x_i, \ldots, x_n) = K$.

The choice of the intermediate quantity Q is obviously important for the success of the derived optimization strategy. We are investigating the following heuristic for choosing intermediate quantities. We define a "critical quantity" to be one that appears inside a conditional expression of the form $(if \ \ (\geq \ \ Q \ \ K) \ \ e_1 \ \ e_2)$. The value of the critical quantity Q governs which of e_1 and e_2 is returned. The conditional expression is potentially non-smooth or discontinuous when $Q = K$, resulting in a ridge or discontinuity in the objective or constraint function. When Q is used to reparameterize the search space, the coordinate axies of the new space are aligned with the ridge

Parameterize Intermediate Quantity: Given an optimization over parameters $(x_1, \ldots, x_n)$ and any expression $Q(x_1, \ldots, x_n)$ appearing as an intermediate quantity in an objective or constraint function: (1) Introduce a new design parameter y and add y to the argument list of each objective or constraint function; (2) Replace each appearance of $Q(x_1, \ldots, x_n)$ in a constraint or objective function with a reference to y; (3) Install an equality constraint requiring that $y = Q(x_1, \ldots, x_n)$; (4) Insert code mapping the seed $(x_1, \ldots, x_n)$ of the optimization to the point $(x_1, \ldots, x_n, Q(x_1, \ldots, x_n))$ in the expanded search space; (5) Insert code mapping the result $(x_1, \ldots, x_n, y)$ of the optimization to the point $(x_1, \ldots, x_n)$ in the original search space.

Solve Equality Constraint: Given an optimization over parameters $(x_1, \ldots, x_i, \ldots, x_n)$, and some constraint of the form $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$: (1) Remove x_i from the set of design parameters and drop x_i from the argument list of each constraint or objective function; (2) Arrange for each constraint or objective function to symbolically or numerically solve $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$ for x_i in terms of $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$; (3) Insert code mapping the seed $(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ of the optimization to the point $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ in the contracted search space; (4) Insert code mapping the result $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ of the optimization to the point $(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ in the original search space by solving for x_i in the equation $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$.

Constrain Intermediate Quantity: Given an optimization over parameters $(x_1, \ldots, x_n)$, and any expression $Q(x_1, \ldots, x_n)$ appearing as an intermediate quantity computed by some constraint or objective function: Introduce a new constraint asserting that $Q(x_1, \ldots, x_n) \leq K$, $Q(x_1, \ldots, x_n) = K$, or $Q(x_1, \ldots, x_n) \geq K$, etc., for some specified bound K.

Figure 5: Transforms to Reformulate Search Space

or discontinuity – an arrangement we expect will prevent optimization codes from getting stuck on the ridge or discontinuity. When Q is used to reduce the dimension of the search space, the discontinuity or non-smoothness is removed entirely. Furthermore, topological considerations suggest that optimal designs are likely to lie on the non-smooth or discontinuous subspace defined by $Q(x_1, \ldots, x_n) = K$. Thus our heuristic recommends using critical quantities to reparameterize or reduce the dimension of search spaces.

Expand Root Expression: Replace an expression for numerically finding the root of a vector-valued function $\bar{f}(\bar{x})$ starting at a seed $\bar{s}$ with the result of applying the Newton-Raphson iteration formula $\bar{x}_{i+1} = \bar{x}_i - (\nabla \bar{f}(\bar{x}_i))^{-1} \bar{f}(\bar{x}_i)$ a fixed number of times with $\bar{x}_0 = \bar{s}$.

Expand Integral Expression: Replace an expression for numerically integrating a system of differential equations specified by a vector-valued derivative function $\bar{f}(\bar{x}, t)$, with the result of using the fourth-order Runge-Kutta formula to compute the integral using four separate evaluations of $\bar{f}(\bar{x}, t)$.

Freeze Intermediate Quantity: Given an optimization over design parameters $x_1, \ldots, x_n$, starting from the seed point $(s_1, \ldots, s_n)$, and given any expression $Q(x_1, \ldots, x_n)$ appearing as an intermediate quantity computed by some constraint or objective function: Replace the expression $Q(x_1, \ldots, x_n)$ with a fixed constant $K = Q(s_1, \ldots, s_n)$ where K is computed prior to the start of the optimization.

Linearize Intermediate Quantity: Given an optimization over design parameters $x_1, \ldots, x_n$, starting from the seed point $(s_1, \ldots, s_n)$, and given any expression $Q(x_1, \ldots, x_n)$ appearing as an intermediate quantity computed by some constraint or objective function: Replace the expression $Q(x_1, \ldots, x_n)$ with a linear function $L(x_1, \ldots, x_n) = Q(s_1, \ldots, s_n) + \sum_{i=1}^{n} (x_i - s_i) \partial Q / \partial x_i (s_1, \ldots, s_n)$, where $Q(s_1, \ldots, s_n)$ and each $\partial Q / \partial x_i (s_1, \ldots, s_n)$ are computed prior to the start of the optimization.

Figure 6: Transforms to Approximate Objective and Constraint Functions

5.2 Approximating Objective and Constraint Functions

Transforms that approximate objective and constraint functions are described in Figure 6. These transforms are designed to exploit the internal structure of objective and constraint functions. They may be used to construct internal approximations that are more cost-effective than purely numerical approximations that treat objective and constraint functions as black boxes. The transforms "Expand Root Expression" and "Expand Integral Expression" are used to replace calls to numerical root extraction and integration routines with simple algebraic expressions approximating their behavior. Aside from speeding up the evaluation of constraint and objective functions (with some loss of accuracy) these transforms serve to convert second order expressions (roots and integrals) into first-order expressions. The transform "Freeze Intermediate Quantity" can then be used to approximate first-order sub-expressions with constants. The constants are calibrated to exact values before the optimization process begins and remain fixed during optimization. Likewise, the transform "Linearize Intermediate Quantity" can be used to approximate first-order sub-expressions with linear functions whose coefficients are calibrated when the optimization process begins and remain fixed during optimization.

Introduce Multi-Stage Optimization: Replace a single-stage optimization strategy with a nested series of two copies of the strategy, in which the first stage generates a seed design for the second stage.

$(optimize\ \ S\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots) \quad \Rightarrow$

$(optimize\ \ (optimize\ \ S\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots)\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots)$

Introduce Multi-Start Optimization: Replace a strategy starting from a single seed design with a new strategy that applies the original strategy to multiple randomly generated seeds and selects the best result.

$(optimize\ \ S\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots) \quad \Rightarrow$

$(select\ \ (list\ \ (random\ \ \ldots)^*)\ \ (\lambda(d)(optimize\ \ d\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots))\ \ BF)$

Introduce Convergence: Replace a single strategy with a convergence process that iteratively applies the original strategy to its own result until a convergence criterion is met.

$(optimize\ \ S\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots) \quad \Rightarrow$

$(converge\ \ S\ \ (\lambda(d)(optimize\ \ d\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots))\ \ BF)$

Introduce Decomposition: Replace a strategy working in a single design space with a combination of two or more strategies that work in factors of the original space, yielding partial designs that are composed to construct a complete design.

$(optimize\ \ S\ \ OBJ\ \ EQNs\ \ INEQNs\ \ \ldots) \quad \Rightarrow$

$(compose\ \ (list\ \ (optimize\ \ S_1\ \ OBJ_1\ \ EQNs_1\ \ \ldots)\ \ \ldots\ \ (optimize\ \ S_n\ \ OBJ_n\ \ EQNs_n\ \ \ldots)))$

Figure 7: Transforms to Nest Optimization Strategies

5.3 Nesting Optimization Strategies

Our transforms for constructing nested optimization strategies are described in Figure 7. These transforms are based on our belief that strategies involving multiple search spaces, multiple levels of approximation and multiple stages of optimization are an effective means of attacking complex design optimization problems. Considerations motivating the use of these transforms were presented above, in Section 4, in the context of the discussion of our strategy language grammar.

6 Preserving Correctness and Convergence

Problems of correctness and convergence may arise if our transforms are used in an unrestricted fashion. Notice that many of our transforms can change the mathematical model defining the user's optimization problem. Approximating transforms can modify the objective function or constraint functions that are specified in the initial strategy. Reformulating transforms can restrict the search

space that is specified in the original strategy. Both types of transform can be used to derive a new strategy that purports to solve an optimization problem that is quite different from the one solved by the initial strategy. A derived strategy may therefore return a design that is neither a local nor global optimum with respect to the original mathematical model. Furthermore, our reformulating transforms might introduce root extraction routines that fail to converge, causing objective or constraint functions to loop forever. Likewise, our approximating transforms might convert a bounded objective function into an unbounded one, causing the optimization code to go into an infinite loop. In the absence of any guidelines regarding safe and unsafe uses of our transforms, the user may derive strategies that return incorrect answers, or that fail to return at all!

We shall address these issues in two ways. First we shall examine the transformations in our catalog, one by one. For each transform, we shall ask whether an application of the transform is guaranteed preserve the correctness and convergence of the original strategy. This analysis will enable us to classify our transforms along several different dimensions. We shall subsequently present a set of guidelines for using our catalog of transformations. We shall show that any strategy derived in accordance with the guidelines will have the same correctness and convergence properties as the original strategy.

6.1 Impact of Individual Transforms

We distinguish between two separate but related notions of correctness. One the one hand, we would like our analysis to help us determine when a derivation preserves the solution set of the mathematical model encoded in the initial strategy. In order to make this notion precise, we shall assume that each occurrence of the *optimize* primitive, in any strategy S, is implemented by an idealized numerical optimization code. When given an objective function $f(d)$, inequality constraints $g(d) \leq 0$ and equality constraints $h(d) = 0$, as inputs, each occurrence of this idealized *optimize* primitive is guaranteed to return a solution satisfying the Karush-Kuhn-Tucker conditions [Peressini *et al.*, 1988] for local optimality with respect to the objective function $f(d)$ and constraint functions $g(d)$ and $h(d)$. If more than one such solution exists, the *optimize* primitive makes a non-deterministic choice. If no such solution exists, the *optimize* primitive returns no value at all. Using this idealization, we define the "mathematical solution set" encoded in a given strategy S to be the set of all possible values that may be non-deterministically returned by S when the strategy is implemented using the idealized *optimize* primitive. Using this definition, a transform that converts a current strategy S into a revised strategy S' will be said to "preserve the mathematical solution set", whenever the set of possible return values of the revised strategy S' is equal to the set of possible return values of the current strategy S.

On the other hand, we would like our analysis to help us determine when a derivation preserves the actual behavior of the initial strategy. In this context, we shall take into account the fact that changes in the search space, objectives or constraints may impact the reliability of an occurrence of the *optimize* primitive, even when the mathematical solution set remains the same. In order to address this issue, we shall not assume that every occurrence of the *optimize* primitive behaves in an ideal fashion. We shall assume only that the particular formulation of the search space, objective function and constraints appearing in the *initial* strategy behaves in an ideal fashion. In order to make this notion precise, we must say what it means for a strategy to behave in an ideal fashion. Assume that the initial strategy optimizes the objective function $f(d)$ with respect to inequality

constraints $g(d) \leq 0$ and equality constraints $h(d) = 0$. We shall say that a strategy S has ideal behavior if the following condition is satisfied: Given any input seed design d, the strategy S returns a design d' satisfying the Karush-Kuhn-Tucker conditions with respect to the objective $f(d)$ and constraints $g(d)$ and $h(d)$ appearing in the *initial* strategy, whenever some such design exists. With this definition in hand, we can now say what it means for a transform to preserve ideal behavior. Suppose a transform converts a current strategy S into a revised strategy S'. Suppose further that S has ideal behavior. If this assumption logically implies ideal behavior for S', then the transform "preserves ideal strategy behavior".

The convergence properties of our strategies will depend, in part, on the underlying numerical algorithm that is used to implement the *optimize* primitive. Different algorithms have different conditions on the objective and constraint functions that provide guarantees about when the algorithm will converge. Instead of considering the convergence conditions of specific algorithms, we focus instead on several properties of objective and constraint functions that influence convergence: convexity, boundedness, continuity and smoothness [Peressini *et al.*, 1988], [Gill *et al.*, 1981]. When considering a transform that converts a current strategy S into a revised strategy S', we assume that each occurrence of the *optimize* primitive appearing in S is well-behaved in terms of these properties, i.e., the objective function is bounded (from above/below for maximization/minimization), the constraint functions are bounded from above, and all of the functions are continuous and smooth throughout the (closed) bounding hyper-rectangle defined by the LBs and UBs parameters supplied to the *optimize* primitive. We also assume that these functions define a convex programming problem, i.e., optimization of a convex function subject to convex constraints. This means that the objective function is convex/concave for minimization/maximization, the inequality constraint functions are convex, and the equality constraints are linear. We then consider which of these properties must remain satisfied in the revised strategy S' that results from applying the transform under consideration.

The results of our correctness analysis are shown in Figure 8. The results of our convergence analysis are shown in Figure 9. The reasoning behind these results is outlined in detail in the Appendix. In most cases our analysis provides a clear "yes" or "no" answer for each transform. In some cases we identify a few potentially important exceptions. Such cases are marked with an asterisk. An entry of "Yes*" means that preservation of the property is guaranteed in general, except for situations that we explicitly enumerate in our analysis. An entry of "No*" means that preservation of the property is not guaranteed in general, except for situations that we explicitly enumerate in our analysis.

Special issues of correctness and convergence arise in the context of strategies involving the "converge" primitive, shown in Figure 4. This primitive is useful for specifying strategies that iteratively calibrate and optimize an approximate objective function. In the most general case, the user has no guarantee that the iteration process will actually converge. When it does converge, the user has no general guarantee that the answer returned is in fact a solution to the original problem. We examine these questions in detail for problems of unconstrained optimization in our paper [Ellman *et al.*, 1997]. Under suitable assumptions of convexity, continuity and smoothness of the objective function, and assumptions about the approximation technique, we are able to make some conditional guarantees. Let the function $I(s)$ represent the inner-loop of the calibration/optimization process. This function takes a seed point s as input. It calibrates an approximate objective function at the seed point s. Then it optimizes the approximate objective function to find a new point $I(s)$. Our analysis shows that convergence is guaranteed when the function $I(s)$ is a contraction mapping. Our

Transform	Guaranteed to Preserve Mathematical Solution Set	Ideal Strategy Behavior
Introduce Multi-Stage	Yes	Yes
Introduce Multi-Start	Yes	Yes
Introduce Convergence	Yes	Yes
Introduce Decomposition	No*	No
Parameterize Intermediate Quantity	Yes	No
Solve Equality Constraint	No*	No
Constrain Intermediate Quantity	No*	No
Expand Root Expression	No	No
Expand Integral Expression	No	No
Freeze Intermediate Quantity	No	No
Linearize Intermediate Quantity	No	No

Figure 8: Impact of Transforms on Mathematical Model and Strategy Behavior

Transform	Guaranteed to Preserve Convexity	Boundedness	Continuity	Smoothness
Introduce Multi-Stage	Yes	Yes	Yes	Yes
Introduce Multi-Start	Yes	Yes	Yes	Yes
Introduce Convergence	Yes	Yes	Yes	Yes
Introduce Decomposition	Yes	Yes	Yes	Yes
Parameterize Intermediate Quantity	No*	Yes*	Yes*	Yes*
Solve Equality Constraint	Yes	No*	No*	No*
Constrain Intermediate Quantity	No*	Yes*	Yes*	Yes*
Expand Root Expression	No	No	No	No
Expand Integral Expression	No	No	No	No
Freeze Intermediate Quantity	No	No	No	No
Linearize Intermediate Quantity	No	No	No	No

Figure 9: Impact of Transforms on Properties Governing Convergence

analysis also identifies a condition on the approximation technique that guarantees that $I(s)$ will have the contraction property, and we identify some specific approximation techniques that satisfy the condition. Finally, our analysis shows that correctness is guaranteed when $I(s)$ uses a first-order approximation of the objective function, i.e., the approximation and its derivative are equal to the exact function and its derivative at the calibration point. Under these assumptions, we prove that any fixed point of $I(s)$ is a true local optimum of the exact objective function. Unfortunately, for complex engineering design problems, these conditions are often difficult to verify in advance.

6.2 Guidelines for Preserving Correctness and Convergence

We now present a set of guidelines that guarantee preservation of the correctness and convergence properties of the initial strategy. Our guidelines require the user to begin each derivation by creating a two-stage strategy consisting of two successive copies of the original strategy. For this purpose,

the user must apply the transform "Introduce Multi-Stage Optimization" to the initial strategy, which generates an expression of the following form:

$$(\lambda(d)\ (optimize\ (optimize\ d\ OBJ\ EQNs\ INEQNs\ \ldots)\ OBJ\ EQNs\ INEQNs\ \ldots))$$

This expression describes two stages of optimization. The inner optimization expression represents the first stage. The outer optimization expression represents the second stage. The design resulting from the first stage is the seed for the second stage. After constructing this two-stage strategy, the user may apply transforms only to the first stage of optimization, i.e., the inner optimization expression. The transforms may expand the first stage into several new ones, approximate objectives and constraints, or reformulate search spaces. Throughout the subsequent derivation the user must not modify the final stage, i.e., the outer optimization expression. Our guidelines also require the user to put a "wrapper" around the first stage of optimization, represented by the inner optimization expression above. (Transforms that install such wrappers are described informally in Figure 10.) The wrapper detects infinite loops by running the optimization stage in a separate thread, periodically checking the system clock. When a time limit expires, the wrapper aborts the optimization stage, and returns the seed design as the result value. The derivation will end with a strategy in the following form:

$$(\lambda(d)\ (optimize\ (wrap\ InitialStages\ \ldots)\ OBJ\ EQNs\ INEQNs\ \ldots))$$

The expression $(wrap\ InitialStages\ \ldots)$ represents the result of wrapping and transforming the inner optimization expression. Notice that every derived strategy will include the original strategy as the final stage of optimization.

Our guidelines guarantee preservation of correctness in both senses defined above: They preserve the mathematical solution set encoded in the initial strategy. They also preserve the ideal behavior of the initial strategy. In both cases, the reasoning hinges on the fact that every derived strategy uses the initial strategy itself as the final stage of optimization. First consider preservation of the mathematical solution set: The solution set of the initial strategy depends only on the search space, objective function and constraints appearing in the initial strategy. It does not depend on the seed. Since the same search space, objective function and constraints appear in the final stage of the derived strategy, the solution set of the derived strategy is the solution set of the initial strategy. Now consider preservation of ideal strategy behavior: If we assume the initial strategy has ideal behavior, then it returns a design satisfying the Karush-Kuhn-Tucker conditions with respect to the search space, objective function and constraints appearing in the initial strategy. It does so regardless of the seed given to the initial strategy. The same space, objective and constraints appear in the final stage of the derived strategy. Therefore the derived strategy returns a design satisfying the same Karush-Kuhn-Tucker conditions, even though it uses a potentially different seed. Therefore the derived strategy has ideal behavior as well.

Now consider preservation of convergence. In this analysis, we take "convergence" to mean "termination for any seed". The reasoning hinges on the guideline requiring placement of a wrapper around the initial stages of optimization, as well as the fact that every derived strategy uses the initial strategy itself as the final stage of optimization. Suppose we assume that the initial strategy terminates, regardless of the seed it receives as input. We reason about the derived strategy as follows: The initial stages are guaranteed to terminate because they are surrounded by a wrapper with a timeout mechanism. The final stage is guaranteed to terminate regardless of the seed it

Wrap Optimization: Place an optimization expression inside a wrapper that acquires control in the event of errors and infinite loops and returns the optimization seed as a result value.

Wrap Root Extraction: Place a root extraction expression inside a wrapper that acquires control in the event of errors and infinite loops, and transfers control to the wrapper of the innermost surrounding optimization expression.

Figure 10: Transforms That Introduce Wrappers

receives from the initial stages, because the final stage is the same as the initial strategy, and because the initial strategy terminates for any seed. Therefore the derived strategy terminates for any seed.

Our system could easily be modified to enforce the guidelines outlined above. For this purpose, we would automatically transform the initial strategy into a two-stage strategy, at the start of the derivation. We would also modify the preconditions of all transformations to prevent modification of the final stage. We would also modify the multi-staging transforms to automatically insert suitable wrappers to detect and abort infinite loops. Nevertheless, we have refrained from making such modifications. A user may not always want to follow the guidelines, in practice. For example, in some situations, strategies that violate the guidelines will give good results. Our experiments in the yacht and nozzle domains will provide some evidence of this fact. The guidelines are therefore not enforced by our system. This approach is true to the spirit of our research. We have not focused on providing *a priori* guarantees about the performance of strategies generated by our system. We have focused on developing tools that enable an engineer to easily develop complex strategies, test them on sample problems, and determine which strategies perform well in practice.

7 Derivations of Yacht and Nozzle Design Strategies

Derivations of strategies for yacht optimization are presented pictorially in Figure 11. The initial yacht design strategy specifies an optimization of the five design parameters (l, b, d, h, s) (length, beam, draft, keel-height and winglet-span described in Section 3) to maximize the velocity objective function, subject to one inequality constraint requiring the yacht to be stable. Strategy Y_1 is constructed by applying the transform "Introduce Multi-Start" to the initial strategy. Strategy Y_1 operates by generating a set of random seed designs, optimizing each with the initial strategy, and selecting the best result. Strategies Y_2, Y_3 and Y_4 are derived directly or indirectly from strategy Y_1, and thus inherit the multi-start process.

Strategy Y_2 is derived from strategy Y_1, by reparameterizing the design space. Reparameterization results from identifying four critical quantities, and using "Parameterize Intermediate Quantity" and "Solve Equality Constraint" to convert them into new design parameters, d_1, d_2, d_3 and d_4, replacing the original parameters b, d, h, and s. The four critical quantities are non-linear functions of the original design parameters. All four quantities appear in the portion of the velocity objective function devoted to computing maximum sail area, based on the limits imposed by the "rating rule" described above. Thus strategy Y_2 operates with design parameters obtained by a

non-linear transformation on the original parameters.

Strategies Y_3 and Y_4 result from using "Expand Root" and "Freeze Intermediate Quantity" to form an approximation of the series that defines the Newton-Raphson root extraction algorithm for solving the balance of force and torque equations governing the motion of the yacht. The transform "Introduce Convergence" is used to set up a process that periodically recalibrates the coefficients of the series expansion. Strategy Y_3 involves a process that repeatedly calibrates the approximation to fit the exact values of velocity v and heel ϕ of the seed design. The calibrated values are used for $\bar{x}_0$ in the Newton-Raphson expansion inside the optimization process. Strategy Y_4 involves a process that repeatedly calibrates the approximation to fit the exact values of velocity v and heel ϕ of the seed design, and the equilibrium values of the Jacobians of net force and net torque functions of v and ϕ. The precomputed values are then used for $\bar{x}_0$ and $\nabla \bar{f}(\bar{x}_0)$ in the Newton-Raphson expansion inside the objective function.

All of the remaining strategies, Y_5, Y_6, Y_7, Y_8 and Y_9, are constructed in a derivation that begins as follows: First the transform "Introduce Multi-Stage" is used to construct a two-stage strategy, consisting of two successive copies of initial strategy. Next the transform "Introduce Multi-Start" is applied to the first of these two stages. Finally, the transforms "Constrain Intermediate Quantity" and "Solve Equality Constraint" are used to reduce the dimension of the search space used in the first stage of optimization. The transform "Constrain Intermediate Quantity" is used to assert four equality constraints that force the critical quantities to lie at points of non-smoothness. The transform "Solve Equality Constraint" is used to remove the four equality constraints and the parameters b, d, h, and s from the set of design parameters. As a result, strategies Y_5, Y_6, Y_7, Y_8 and Y_9 each operate in a non-linear one-dimensional subspace of the original space, including only the length l of the yacht. This reduced space does not exhibit the non-smoothness present in the original space. Strategies Y_5, Y_6, Y_7, Y_8 and Y_9 all inherit the multi-start process and the reduced-dimension search space in the first stage of optimization.

Strategy Y_5 results from deleting the second stage of optimization. This strategy carries out a single stage of optimization in a one-dimensional design space. Strategies Y_6 and Y_8 use the same approximation and recalibration techniques as used in strategies Y_3 and Y_4; however, strategies Y_6 and Y_8 include two stages of optimization. They use a one-dimensional space and the approximation/calibration process in the first stage of optimization. They use the initial strategy in the second stage of optimization. Finally, strategies Y_7 and Y_9 are respectively derived from strategies Y_6 and Y_8 by deleting the second stage of optimization.

Derivations of strategies for nozzle optimization are presented pictorially in Figure 12. The initial nozzle design strategy specifies an optimization of the three flap lengths (l_c, l_d, l_e) (convergent length, divergent length and external length described in Section 3) to minimize fuel consumption over a prescribed mission. The transform "Introduce Multi-Start" is used to construct the strategy N_1. Strategy N_1 operates by generating a set of random seed designs, optimizing each, and selecting the best. Strategies N_2, N_3 and N_4 are derived directly or indirectly from strategy N_1, and thus inherit the multi-start process.

Strategy N_2 is derived from N_1 by identifying a critical quantity $g(l_c, l_d, l_e)$ and using "Parameterize Intermediate Quantity" and "Solve Equality Constraint" to convert it into a new design parameter, replacing the original design parameter l_e, the external flap length. The intermediate quantity appears in a conditional expression testing whether the flaps are geometrically connectable. When g goes negative, the nozzle is unrealizable, evaluation fails, and an extremely bad value is returned, causing a discontinuity in the evaluation function. Since g is a non-linear function of the

design parameters, strategy N_2 operates with design parameters obtained by a non-linear transformation on the original parameters.

Strategies N_3 and N_4 are derived from N_1 using the approximating transform "Expand Integral Expression" to remove the call to a numerical integration routine and using the approximating transform "Expand Root Expression" to remove calls to numerical root extractors. Strategy N_3 results from using "Freeze Intermediate Quantity" and "Introduce Convergence" to set up a process of periodically re-calibrating the seeds in the expanded root expressions. Strategy N_4 results from using "Freeze Intermediate Quantity" and "Introduce Convergence" to set up a process of periodically re-calibrating the seeds and Jacobians in the expanded root expressions.

All of the remaining strategies, N_5, N_6 and N_7 are constructed in a derivation that begins as follows: First the transform "Constrain Intermediate Quantity" is used to assert an inequality constraint placing a lower bound (zero) on $g(l_c, l_d, l_e)$. Then the transform "Introduce Multi-Stage" is used to construct a two-stage strategy, consisting of two successive copies of the initial strategy along with the new lower bound constraint. Finally, the transform "Introduce Multi-Start" is applied to the first stage of optimization. Strategies N_5, N_6 and N_7 are all derived directly or indirectly from this point, and thus inherit the two stages of optimization, the multi-start process in the first stage, and the new lower bound constraint in both stages. Strategy N_5 then results from deleting the second stage of optimization entirely. Strategy N_6 is derived by retaining the second stage, and instead applying the transform "Constrain Intermediate Quantity" to the first stage of optimization. This transform asserts an equality constraint forcing $g(l_c, l_d, l_e)$ to have a fixed, small, positive value. The transform "Solve Equality Constraint" is then used to remove the equality constraint and the parameter l_e (external length) from the design space. Strategy N_6 thus operates in a two-dimensional non-linear subspace, including only l_c (convergent length) and l_d (divergent length) during the first stage of optimization. It operates in the full three-dimensional design space and enforces a lower bound constraint on $g(l_c, l_d, l_e)$, during the second stage of optimization. Strategy N_7 is derived from N_6 by deleting the final stage of optimization.

8 Experimental Results

Results of testing yacht and nozzle design strategies are presented in Figures 13 and 14. Each yacht strategy was tested on a set of 9 design problems, characterized by all combinations of three different headings (60, 120 and 180 degrees) and three different windspeeds (8, 12 and 16 knots). Each nozzle strategy was tested on a set of 6 test problems, with various combinations of mission altitude (60,000 feet and 30,000 feet), velocity (mach 2.0 and mach 1.414), and duration (2 hours and 1/2 hour). Each strategy used five randomly selected seed designs as starting points for optimization. CFSQP was used to implement each occurrence of the *optimize* primitive in each strategy. For each (problem, strategy) combination we recorded the CPU time used in the run and the value of the objective function (yacht velocity or nozzle takeoff mass) of the resulting design. We normalized CPU time for each (problem, strategy) combination, dividing by the average CPU time for the default strategy, yielding "Average Relative CPU Time" shown in the tables: Y_1 required an average of 5:15 CPU minutes, carrying out an average of 308 objective function evaluations per problem; N_1 required an average of 55:17 CPU minutes, carrying out an average of 144 objective function evaluations per problem – all on a Sun Microsystems Sparc 5 workstation. We normalized the objective value for each (problem, strategy) combination, dividing by the objective value of the

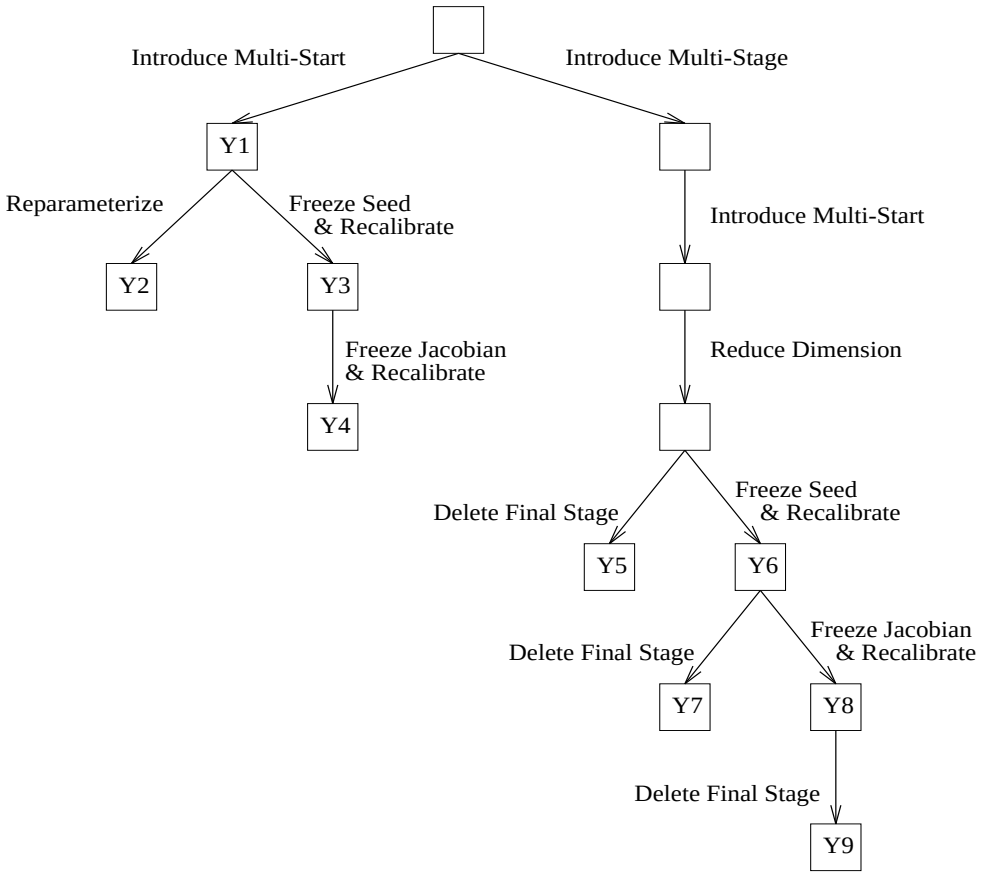

Figure 11: Derivation Relations among Yacht Optimization Strategies

"optimal" design for that problem, i.e., the best design found by any strategy, yielding "Average Relative Objective" shown in the table. Finally, for each strategy, we recorded the best and worst relative objective over all problems in the test set. A typical yacht velocity is about 10 knots, so a 1% variation in the objective function amounts to about a tenth of a knot. A typical value for takeoff mass is about 175000 kilograms. A 1% variation in the objective function amounts to 1750 kilograms of fuel.

The effect of approximation and re-calibration can be seen in the yacht domain by comparing Y_1 with Y_3 and Y_4, and in the nozzle domain by comparing N_1 with N_3 and N_4. Approximation speeds optimization by as much as a factor of three in the yacht domain with only a small (less than 0.4%) loss in average quality. Approximation speeds optimization by a factor of more than two in the nozzle domain, but with a larger (nearly 4.0%) loss in average quality. We believe that additional speedup is possible by approximating, and re-calibrating the net-torque functions used in strategies Y_3 and Y_4 and the net-vertical-acceleration and net-horizontal-acceleration functions used in strategies N_3 and N_4. The loss of quality may be remedied by adding additional stages of optimization.

The effect of reparameterization can be seen by comparing Y_1 with Y_2 in the yacht domain and by comparing N_1 with N_2 in the nozzle domain. We had expected reparameterization to improve design quality, by aligning coordinate axies with ridges or discontinuities and preventing CFSQP from getting stuck. Our expectation was not borne out in the yacht domain: Y_2 does not improve

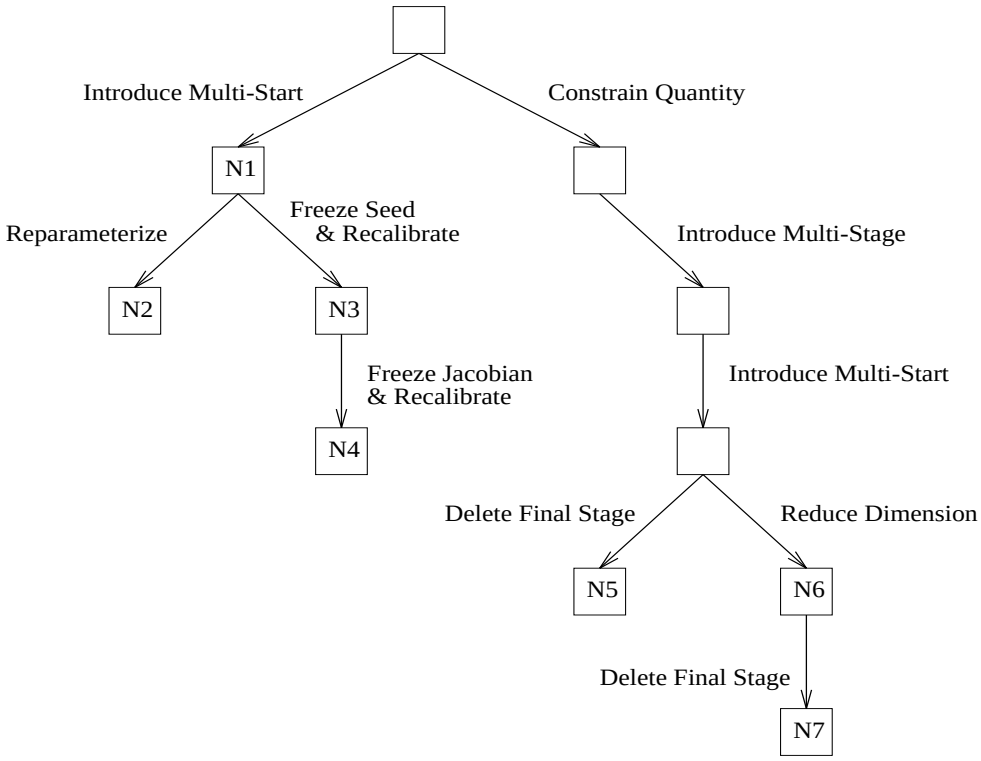

Figure 12: Derivation Relations among Nozzle Optimization Strategies

the average quality compared to Y_1. Our expectation did hold up in the nozzle domain: N_2 does improve quality compared to N_1, with a modest penalty in CPU time. The additional CPU time apparently results from the fact that CFSQP does not terminate prematurely but continues to improve design quality.

Strategy	Rel. CPU Time	Avg. Rel. Objective	Best Rel. Objective	Worst Rel. Objective
Y_1	100.00%	99.90%	100.00%	99.73%
Y_2	93.36%	99.81%	100.00%	98.70%
Y_3	49.15%	99.68%	99.88%	99.38%
Y_4	33.45%	99.63%	99.88%	98.65%
Y_5	28.31%	99.36%	100.00%	97.60%
Y_6	37.73%	99.75%	100.00%	99.27%
Y_7	24.44%	99.36%	100.00%	97.60%
Y_8	30.37%	99.50%	99.98%	97.53%
Y_9	17.72%	99.01%	100.00%	97.42%

Figure 13: Performance of Yacht Optimization Strategies

Strategy	Rel. CPU Time	Avg. Rel. Objective	Best Rel. Objective	Worst Rel. Objective
N_1	100.00%	102.90%	100.23%	108.77%
N_2	112.04%	101.95%	100.47%	104.18%
N_3	84.85%	103.81%	101.69%	108.77%
N_4	42.65%	103.81%	101.69%	108.77%
N_5	99.31%	102.16%	100.47%	104.69%
N_6	116.83%	100.00%	100.00%	100.00%
N_7	84.98%	100.65%	100.00%	102.95%

Figure 14: Performance of Nozzle Optimization Strategies

The effect of dimension reduction can be seen by comparing Y_1 with Y_5 in the yacht domain. Strategy Y_5 runs considerably faster than strategy Y_1 while suffering a small (less than 0.7%) loss in average quality. On at least one problem, Y_5 found an "optimal" design; however on at least one other problem, Y_5 found a design that was significantly below optimal. These results demonstrate that our dimension reducing transformations can significantly speed up optimization; however, they are not guaranteed to preserve optimality. It would be useful to develop methods for predicting which problems within a problem class have their solutions in reduced dimension subspaces.

The effect of dimension reduction can also be seen in the nozzle domain by comparing N_1 with N_7. Strategy N_7 speeds optimization by a small amount, compared to N_1. In addition N_7 achieves a dramatic improvement (2.25%) in design quality. We believe this improvement results from the fact that N_1 gets stuck on ridges or discontinuities that are removed in the reduced dimension space searched by N_7. Notice that merely imposing an inequality constraint on the critical quantity (as in strategy N_5) does not work as well. Strategy N_5 does improve design quality in comparison to N_1; however, the improvement is much smaller than that achieved by N_7.

Finally the effect of multi-stage optimization may be seen in the yacht domain by comparing Y_7 with Y_6 and comparing Y_9 with Y_8. Strategies Y_6 and Y_8 include a final optimization stage, in the full search space, using the exact objective function to remedy possible ill effects of dimension reduction and approximation. These multi-stage strategies improve average quality while incurring a small additional computational cost. Likewise, the effect of multi-stage optimization may be seen

in the nozzle domain by comparing N_7 with N_6 (and with all the other strategies). Strategy N_6 is the same as N_7, but includes a final stage of optimization in the full search space, using the exact objective function, and one inequality constraint. It finds the best design on all six test problems. We expect that further improvements can be attained by combining dimension reduction (strategies N_7 and N_6) with approximation (N_3 and N_4).

Some of our yacht strategies perform quite well, despite violating our guidelines for deriving strategies that are guaranteed to terminate and return correct solutions. In particular, strategies Y_2, Y_3, Y_4, Y_5, Y_7 and Y_9 all violate the guidelines. Each of these strategies involves a final stage of optimization that is different from the initial strategy, either by using an approximation of the objective function, a reparameterized search space, or a reduced-dimension search space. Nevertheless, in our experiments these strategies always terminated and returned designs whose quality is nearly identical to the designs found by the initial strategy. Our results thus provide some support for our decision to refrain from strictly enforcing the guidelines in our system.

After reviewing our yacht and nozzle domain results, the reader will naturally ask whether any of the derived strategies can be said to exhibit better overall performance than the default strategies. The answer is fairly clear for the yacht domain. The data in Figure 13 show that several derived yacht strategies (Y_4, Y_5, Y_6, Y_7, Y_8 and Y_9) lead to a significant speedup (i.e., factors of three to five) in comparison to the default strategy (Y_1). Furthermore, these derived strategies suffer very little (less than 1%) loss in average design quality. From these results, we conclude that our transforms can significantly improve performance in the yacht domain. The answer is less clear for the nozzle domain. The data in Figure 14 shows significant variations in both running time and design quality. Some strategies are faster than the default strategy (N_1) but lead to lower average quality (e.g., N_4). Others are slower but result in better average quality (e.g., N_6). Further analysis is needed to determine whether any of the derived nozzle strategies represents an overall improvement, in comparison to the default.

CPU time and design quality are somewhat interchangeable. For a given strategy, one may convert low CPU time into high design quality by running the strategy with a larger number of starting points. Likewise one may convert high design quality into low CPU time by running the strategy with a smaller number of starting points. These observations suggest a statistical method of comparing strategies on a common scale. We run each strategy from m randomly selected starting points. We keep track of the design quality of each result, along with the average CPU time per starting point. Let $q(i)$ be the quality of the ith best result obtained using this procedure. Let $p(i, n, m)$ represent the probability that a random selection (with replacement) of n values from $\{q(i)|i = 1 \ldots m\}$ will yield $q(i)$ as the best quality value. We can then compute a statistic $EQ(n)$ that represents the expected quality of the best design found after running the strategy with n randomly selected starting points. Using elementary probability theory:

$$\begin{aligned} EQ(n) &= \sum_{i=1}^{m} p(i,n,m)q(i) \\ &= \sum_{i=1}^{m} \left[\left(\frac{m-(i-1)}{m} \right)^n - \left(\frac{m-i}{m} \right)^n \right] q(i) \end{aligned}$$

We can also compute a statistic $ET(n)$ representing the expected CPU time needed to run the strategy using n starting points: $ET(n) = nC$, where C is the average CPU time per starting point.

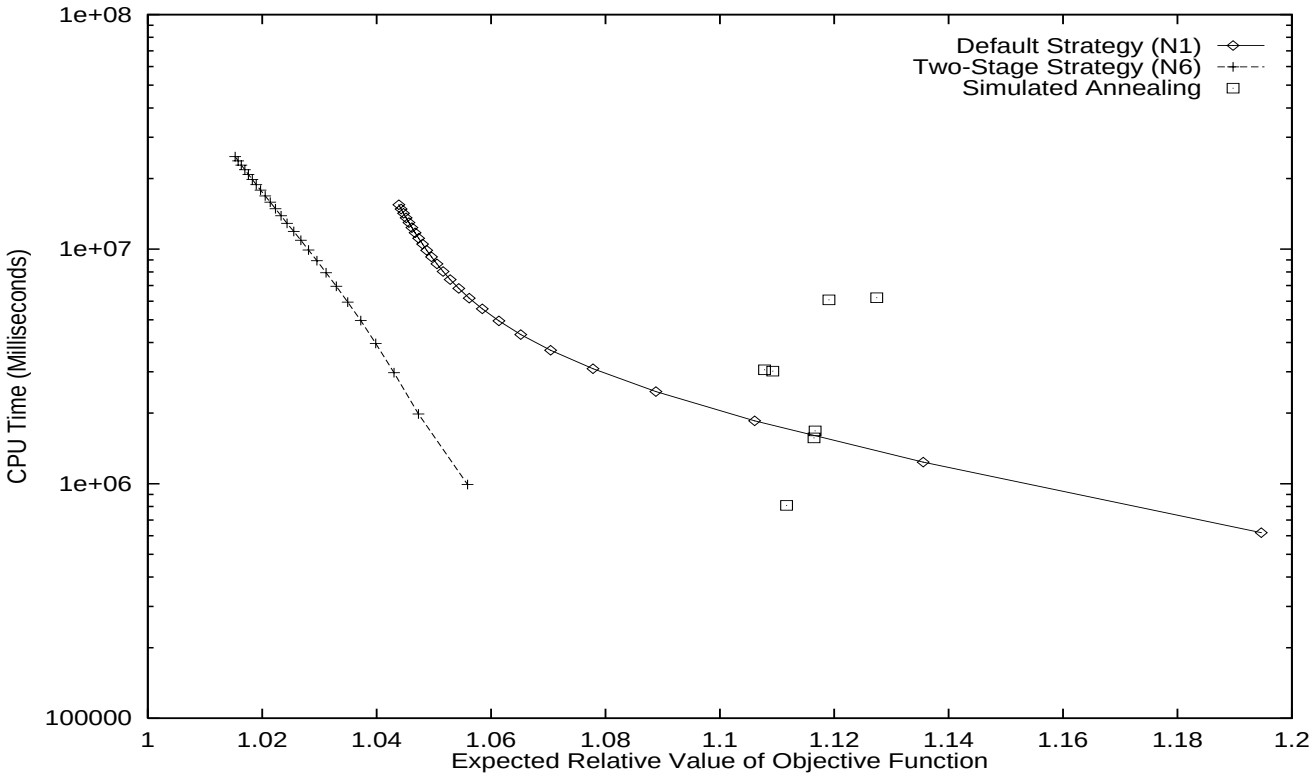

Figure 15: Comparison of Nozzle Design Strategies (Problem P1)

Tradeoffs between CPU time and design quality are shown in Figures 15-20 for two of the nozzle design strategies: N_1, the default strategy, and N_6, the two-stage strategy using dimension reduction in the first stage. These data were generated using the procedure outlined above with $m = 100$ starting points and plotting $EQ(n)$ versus $ET(n)$ for $n = 1 \ldots 25$. These graphs allow one to compare strategies in two different ways: For a given amount of CPU time, the graphs show that strategy N_6 yields a better expected value for design quality than strategy N_1. On the other hand, for a given level of design quality, the graphs show that strategy N_6 requires less CPU time than strategy N_1. In particular, for most levels of quality, strategy N_6 is faster by almost an order of magnitude. From these results, we conclude that our transforms can significantly improve performance in the nozzle domain, as well as the yacht domain.

We also carried out a series of experiments comparing our nozzle design strategies to a stochastic algorithm. In particular, we compared nozzle strategies N_1 and N_7 to the *Amebsa* algorithm taken from [Press *et al.*, 1986]. *Amebsa* is a combination of simulated annealing and the downhill-simplex algorithm. It uses the simplex method for defining the kinds of changes to the current design(s) that can be made when moving through the design space. It uses the simulated annealing as the control structure for deciding what changes to make and when to make them. In order to use this algorithm, the user must supply a "cooling schedule", i.e., an algorithm for initializing and decreasing the "temperature" used by the *Amebsa* algorithm. We experimented with a variety of different cooling schedules. In particular, we varied the initial temperatures (ranging from $100°K$ to $800°K$) and the number of different temperature levels (ranging from 4 to 16). The final temperature for each schedule was $12.5°K$. We ran the *Amebsa* algorithm multiple times with each schedule.

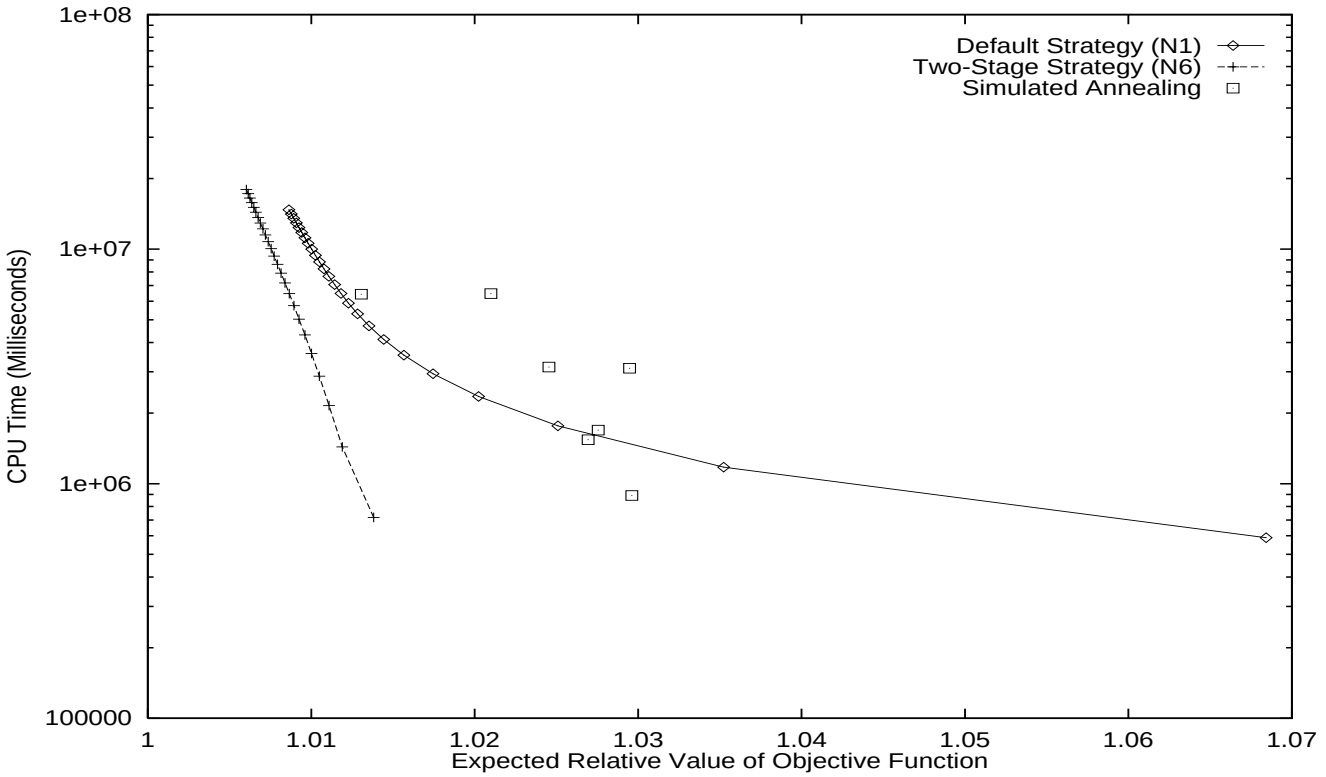

Figure 16: Comparison of Nozzle Design Strategies (Problem P2)

The number of runs ranged from 6 times for the slowest schedule to 50 times for the fastest schedule. Averaging the results over each run, we obtained expected values of design quality and CPU time for each schedule. These results are shown in Figures 15-20. Notice that the two-stage nozzle strategy N_6 outperforms *Amebsa* regardless of the cooling schedule, i.e., N_6 achieves better design quality for a given amount of CPU time, or uses less CPU time to achieve a given level of design quality. Of course we cannot rule out the possibility that a better cooling schedule would change this result. The difficulty of finding the best cooling schedule is a major limitation of simulated annealing algorithms. In the absence of an effective method for finding a better schedule, we conclude that our derived nozzle strategies are more cost-effective in the nozzle domain than the *Amebsa* simulated annealing algorithm.

The choice of an optimization strategy will ultimately depend on context. During preliminary stages of the design process, an engineer may want to use a strategy that is fast but possibly unreliable. On the other hand, during later stages of the design process, an engineer may want to use a more reliable optimization strategy that requires greater CPU time. Before making the commitment to invest a large sum of money in construction of an artifact, an engineer will likely want to obtain the greatest possible reliability, perhaps by using a combination of several different deterministic or stochastic strategies. The choice of an appropriate optimization strategy will therefore vary over the product design cycle.

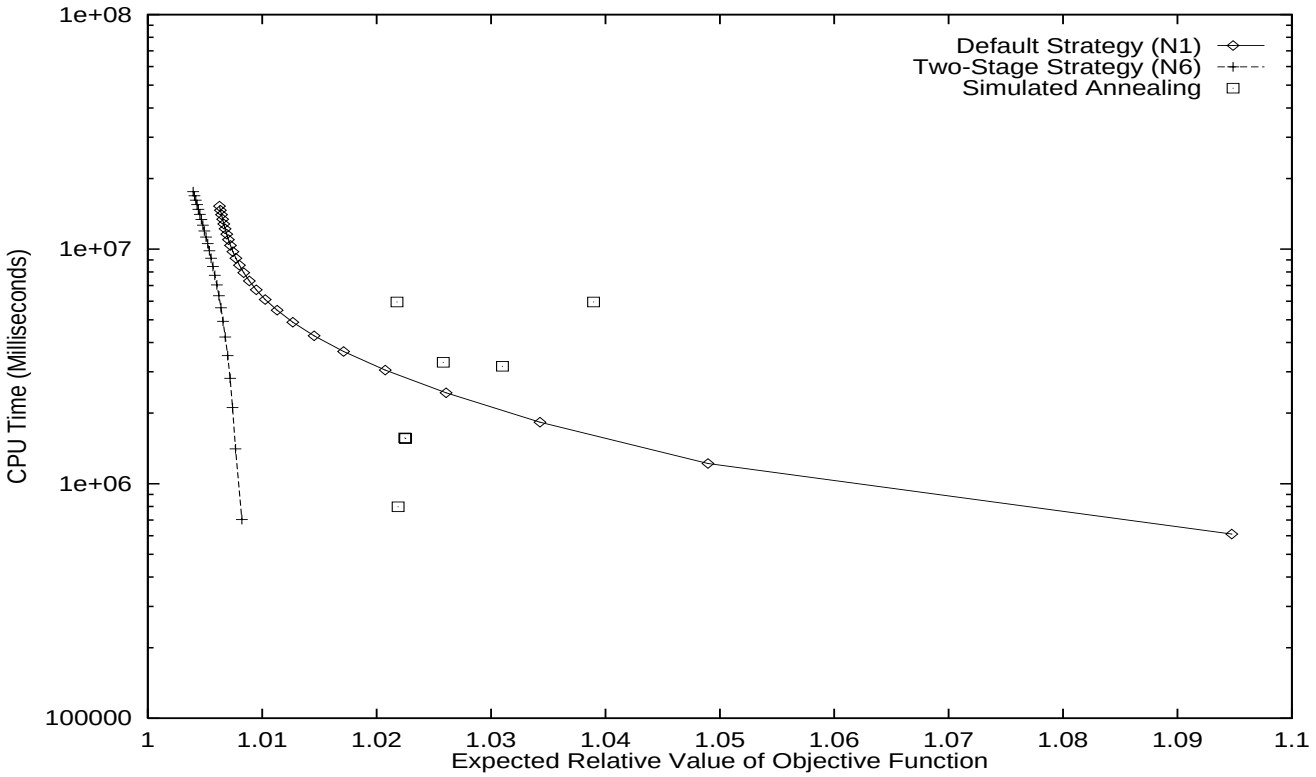

Figure 17: Comparison of Nozzle Design Strategies (Problem P3)

9 Domain Independence of Results

Our strategy language and transformation system were developed with the intention of handling any engineering design problem that can be formulated in terms of constrained optimization, as outlined in Figure 1. We have now demonstrated useful capabilities of our system in the domains of yacht and nozzle design. Nevertheless, the reader will naturally wonder whether our system would really be useful in practice in other engineering design applications. Two different questions should be considered in this connection: (1) What types of design optimization problems can actually be implemented in our system? (2) What types of optimization problems are most amenable to the kinds of optimization strategies (using approximation, recalibration, reparameterization, dimension reduction and multi-stage optimization) which our system is able to construct? We shall consider each of these issues in turn.

In order to use our transformation system, an engineer must encode an initial strategy in terms of the functional, data-flow language defined by the grammar shown in Figure 4. The language includes primitives for root extraction and integration that make it especially well-suited to design of artifacts that are governed by algebraic and ordinary differential equations. In addition, the language includes hooks for interfacing black boxes that implement arbitrary functionality. Therefore in principle *any* constrained optimization problem can be implemented as an initial strategy for use in our system. On the other hand, many of our transforms require access to intermediate quantities appearing in the objective or constraint functions. Expressions representing such quantities are manipulated by transforms that introduce approximations and transforms that reformulate search

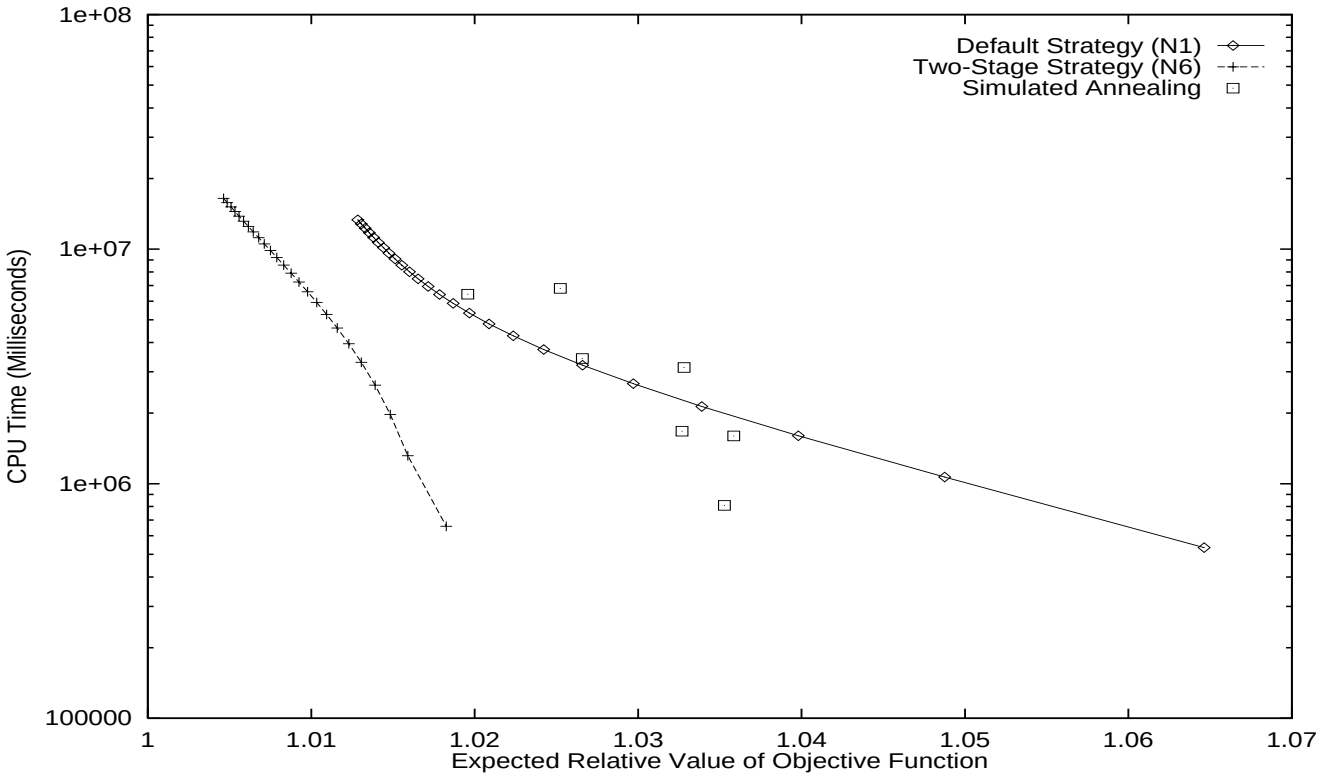

Figure 18: Comparison of Nozzle Design Strategies (Problem P4)

spaces. In order for our full catalog of transforms to be applicable to a design optimization problem, the initial strategy must provide access to intermediate quantities. This may present a problem when objective and constraint functions are implemented by legacy codes written in conventional programming languages, e.g., C or Fortran. When such legacy codes are interfaced with our system as monolithic black boxes, the intermediate quantities generated inside them cannot be manipulated by our transforms. Nevertheless, this problem is substantially mitigated by the following considerations: Many or most numerical codes can be decomposed into modules with identifiable inputs and outputs. The modules can be encoded as separate black boxes in the initial strategy. Quantities passed between modules can be manipulated by our system and exploited for purposes of approximation or reformulation. In a related body of work [Keane and Ellman, 1996], we have developed a suite of techniques and tools that facilitate integration of modular legacy codes into our system.

One portion of our transform catalog is aimed at developing optimization strategies that use approximation and periodic recalibration of objective or constraint functions. Strategies based on approximation and recalibration are potentially useful when portions of the objective or constraint functions are expensive to compute, or when running a large number of optimizations, each under different assumptions about the operating environment of the artifact. Our approximating transforms allow an engineer to construct *internal approximations* of selected portions of objective or constraint functions, while leaving the rest of each function intact. We now consider when such approximations will likely result in strategies with superior performance. Precise predictions are difficult to make. Instead, we shall try to identify the factors that tend to favor

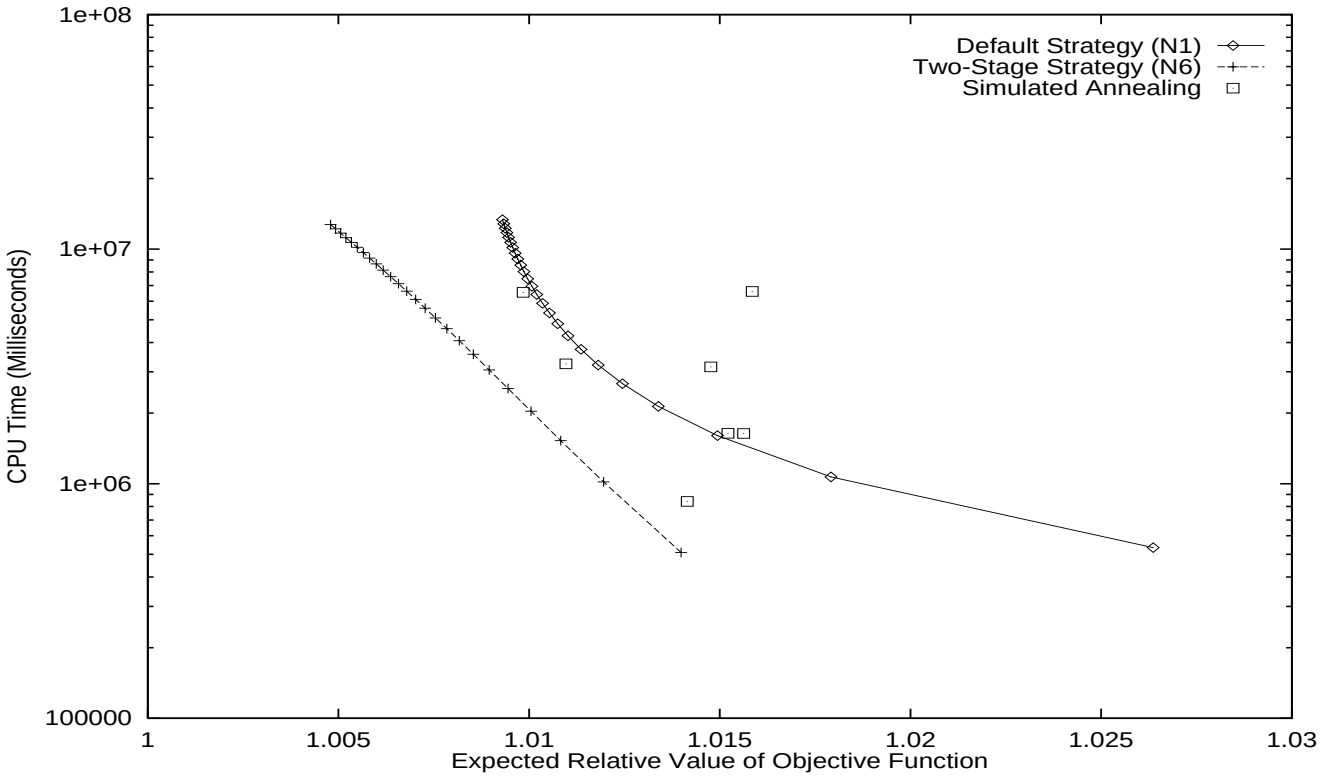

Figure 19: Comparison of Nozzle Design Strategies (Problem P5)

strategies of approximation and recalibration. As an extreme example, consider an objective function $f(x_1, \ldots, x_n) = e(x_1, \ldots, x_n, Q(x_1, \ldots, x_n))$ involving an intermediate quantity $Q(x_1, \ldots, x_n)$. Suppose that the computation of Q is expensive in comparison to the computation of the rest of the objective function. Suppose further that the value of Q changes slowly from point to point in the design space. Finally, suppose that the objective function f is relatively insensitive to the value of Q, i.e., the partial derivative $\partial e/\partial Q$ is small in magnitude. A linear approximation of Q could be calibrated at the cost of only $n + 1$ evaluations of Q, i.e., roughly the cost of computing one gradient of the objective function. The linear approximation of Q would be much less expensive to evaluate than Q itself. The approximation would be accurate over a wide region. Errors in Q would have only a small effect on the location of the optimum of the objective function. In this extreme case, we would expect that a strategy of repeated approximation, optimization and recalibration would perform better than direct optimization with the exact objective function. This example suggests that an engineer should consider the following factors in deciding whether to use a strategy of approximating and recalibrating an intermediate quantity: the cost of computing the intermediate quantity in comparison to the cost of the rest of the objective or constraint function; the cost savings that result from using the approximation; the cost of calibrating the approximation; the range of the design space over which a calibrated approximation is accurate; and the sensitivity of the overall objective or constraint function to errors in the approximated intermediate quantity. A more comprehensive discussion of these issues is contained in our paper [Ellman *et al.*, 1997].

Another portion of our transform catalog is aimed at developing optimization strategies that use dimension reduction and multi-stage optimization. Strategies based on dimension reduction and

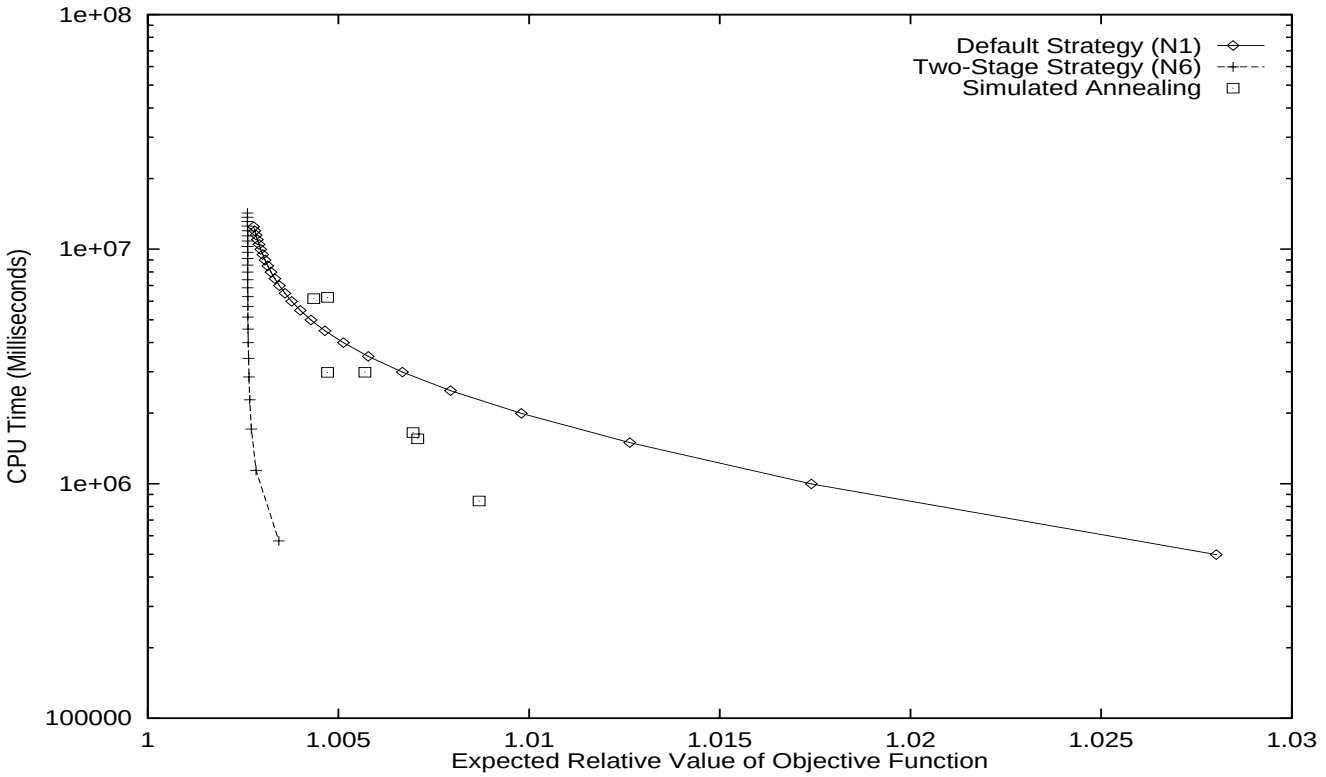

Figure 20: Comparison of Nozzle Design Strategies (Problem P6)

multi-stage optimization are potentially useful for dealing with pathological objective or constraint functions. For example, in the nozzle domain, this type of strategy overcame problems that resulted from the presence of a region of the design space in which the objective function was not evaluable. In general, one may ask what sorts of pathologies are amenable to this approach. In order to address this question, we defined a family of synthetic, pathological objective functions. We then ran a series of experiments comparing the behavior of single-stage strategies to the behavior of multi-stage, dimension reduction strategies, on these pathological problems. We considered the following types of pathologies: ridges, unevaluable regions and broken ridges (described below). Our investigation of ridges was motivated in part by our experience in the yacht domain. Ridges appeared in the velocity objective function as a result of non-smoothness in the "rating rule". Unevaluable regions were motivated in part by our experience in the nozzle domain. Failures occurred in the takeoff mass objective function in regions of the design space where the nozzle design was not geometrically realizable. Broken ridges were motivated by a desire to find a type of pathology that would present a challenge to strategies involving dimension reduction and two-stage optimization.

Our family of pathological objective functions was constructed in the following way. We began with a well-behaved objective function, i.e., a simple convex quadratic function of two variables: $e(x,y) = ax^2 + by^2 + cxy + dx + ey + f$. We then defined a pathological objective function: $f(x,y) = e(x,y) + p(x,y)$, in which $p(x,y)$ represents a pathology. Ridges were constructed by defining $p(x,y) = K|b(x,y)|$, where K is a large positive constant. Points on the ridge satisfy the condition $b(x,y) = 0$. Points off the ridge satisfy the condition $b(x,y) > 0$. Broken ridges were constructed by defining $p(x,y) = K|b(x,y)|$, if $b(x,y) \leq 0$ and $p(x,y) = K|b(x,y)| + L$, if $b(x,y) > 0$,

where L is another large positive constant. In one set of experiments we chose $b(x, y)$ to be linear in x and y resulting in a "linear ridge" or "linear broken ridge". In another set of experiments, we chose $b(x, y)$ to be quadratic, resulting in "non-linear ridge" or "non-linear broken ridge". In all cases we arranged for the ridge or broken ridge to pass through the optimum of the non-pathological objective function $e(x, y)$. Unevaluable regions were constructed by defining $f(x, y) = e(x, y)$ if $b(x, y) \leq 0$ and $f(x, y) = K$ if $b(x, y) > 0$ where K is a large positive constant. This definition creates a discontinuous function in which the non-pathological function remains unchanged on one side of the discontinuity, and a plateau is present on the other side of the discontinuity. In one set of experiments we chose $b(x, y)$ to be linear in x and y resulting in a "linear unevaluable" region. In another set of experiments, we chose $b(x, y)$ to be quadratic, resulting in a "non-linear unevaluable" region. We also varied the location of the boundary relative to the optimum of the non-pathological objective function $e(x, y)$. In one set of experiments, the optimum (x_{opt}, y_{opt}) of $e(x, y)$ was chosen to satisfy $b(x_{opt}, y_{opt}) = 0$, so that the pathological function $f(x, y)$ and the non-pathological function $e(x, y)$ had the *same optimum* point. In another set, the optimum (x_{opt}, y_{opt}) of $e(x, y)$ was chosen to satisfy $b(x_{opt}, y_{opt}) > 0$, so that the pathological function $f(x, y)$ and the non-pathological function $e(x, y)$ had *different optimum* points, and so that the non-pathological optimum was located well within the plateau defined by the pathological objective function.

We used our interactive transformation system to construct two strategies for optimizing these pathological objective functions: a one-stage strategy and a two-stage strategy. The one-stage strategy operates by generating a random (evaluable) seed inside the bounding hyper-rectangle, and then using CFSQP to optimize the pathological objective function $f(x, y)$ starting from that seed point. The two-stage strategy operates by generating a random (evaluable) seed (x_0, y_0) inside the bounding hyper-rectangle, and solving the equation $b(x_0, y) = 0$ numerically for y to find a new point (x_0, y_b) lying on the ridge or the boundary the evaluable region. It then uses CFSQP to optimize $f(x, y)$ along the ridge or boundary, starting at the point (x_0, y_b), resulting in the point (x_1, y_1). Finally, it uses CFSQP to optimize $f(x, y)$ in the full two-dimensional space, starting at the point (x_1, y_1).

Results of our experiments on these synthetic pathological problems are shown in Figure 21. We ran each of the two strategies 100 times on each pathological problem. We recorded the optimum quality level found by each of the 100 runs, along with the number of evaluations of $f(x, y)$ used by the strategy in each run. We then computed a statistic $N(p, \epsilon)$ that represents the number of seed points the strategy requires in order to have probability p of finding a solution whose quality is within a fraction ϵ of the best solution found. Using elementary probability theory:

$$N(p, \epsilon) = \frac{log(1 - p)}{log(1 - r(\epsilon))}$$

where $r(\epsilon)$ is the probability that an optimization from a single seed results in a solution within ϵ of the best solution. We estimated the value of $r(\epsilon)$ from the empirical histogram of solution quality levels. Finally, we obtained an estimate of the required CPU time, $T(p, \epsilon)$, by forming the product of $N(p, \epsilon)$ and the average number of objective function evaluations per seed used by the strategy. In some cases, the strategy required only one run from a single seed to achieve the designated level of reliability. Such cases are marked with an asterisk.

Our results show that the one-stage strategy was mildly unreliable in solving the "Linear Ridge" problem. On the other hand, it was extremely unreliable in solving the "Non-Linear Ridge" problem. It appears that CFSQP can determine the direction of a linear ridge and then move in a fixed

Pathology	One-Stage Strategy T(0.99,0.01)	Two-Stage Strategy T(0.99,0.01)
Linear Ridge	136	20*
Non-Linear Ridge	17110	24*
Linear Unevaluable (Same Optimum)	38*	21*
Non-Linear Unevaluable (Same Optimum)	41*	26*
Linear Unevaluable (Different Optima)	597	77*
Non-Linear Unevaluable (Different Optima)	961	90*
Linear Broken Ridge	5253	224
Non-Linear Broken Ridge	32138	476

Figure 21: Performance of Single and Multi-Stage Strategies on Pathological Problems

direction along the ridge to find the optimum; however, CFSQP apparently cannot move along a ridge that constantly changes its direction. In contrast to this, the two-stage strategy was quite reliable in solving problems with either linear or non-linear ridges. The one-stage strategy was quite reliable in solving the "Linear Unevaluable (Same Optimum)" and "Non-Linear Unevaluable (Same Optimum)" problems, i.e., problems in which the optimum of the non-pathological function is the optimum of the pathological function. On the other hand, the one-stage strategy was somewhat less reliable in solving the "Linear Unevaluable (Different Optima)" and "Non-Linear Unevaluable (Different Optima)" problems, i.e., problems in which the optimum of the non-pathological function is different from the optimum of the pathological function, and lies within the plateau defined by the pathological function. It appears that the separation between the pathological and non-pathological optima causes CFSQP to be misled about the location of the true optimum. In contrast to this, the two-stage strategy was quite reliable in solving the "Linear Unevaluable" and "Non-Linear Unevaluable" problems, regardless of whether the pathological and non-pathological objective functions had the same optimum points. Finally, both the one-stage and two-stage strategies had some difficulty handling "Linear Broken Ridge" and "Non-Linear Broken Ridge" problems; however, the two-stage strategy performed much better on these problems than the one-stage strategy.

We have not undertaken an exhaustive experimental study of the performance of single-stage and multi-stage strategies in the presence of pathological objective and constraint functions. In particular, there are many more ways we could vary the mathematical properties of the pathologies and objective functions in the examples described above. These variations might well influence the behavior of either the single-stage or multi-stage strategies. For this reason, we shall refrain from attempting to draw general conclusions about the relative performance of single and multi-stage strategies on pathological problems. Nevertheless, we believe our results demonstrate the potential for obtaining significant performance improvements from using our strategy language and transformation system on problems with ridges, discontinuities or other pathologies.

10 Related Work

Several other investigators have used knowledge-based techniques to improve the performance of automatic design algorithms. The DOMINIC-II system is one example [Orelup *et al.*, 1988]. DOMINIC-II was motivated by the observation that hill-climbing search does not reliably solve design problems – a motivation similar to our own. In DOMINIC-II, this problem was addressed by using a meta-level controller to dynamically switch between different hill-climbing search strategies. The meta-level controller would monitor the hill-climbing process. Upon detecting an impasse, it would switch to a new strategy. DOMINIC-II is similar to our system in the sense that it uses a multi-stage, multi-strategy design process. Nevertheless, our concepts of "strategy" are rather different from each other. In DOMINIC-II, a strategy is a search control method. A choice of strategy in DOMINIC-II is analogous to a choice of a numerical optimization algorithm (e.g., CFSQP) in our system. In contrast to this, a strategy in our system includes definitions of search spaces, objective functions and constraints, among other things. These problem formulation issues are not addressed in the DOMINIC-II research. It might be useful to use some ideas from the DOMINIC-II research in our system. For example, our system might benefit from methods used in DOMINIC-II for monitoring and diagnosing problems with hill-climbing search.

The ENGINEOUS [Tong, 1990] and INTERGEN [Powell, 1990] systems also use knowledge-based methods to improve the performance of design optimization. This work was motivated, in part, by the problem of dealing with failures and other pathological features of complex objective and constraint functions – a motivation that is similar to our own. ENGINEOUS and INTERGEN both combine numerical methods (e.g., hill-climbing, simulated annealing, genetic algorithms) with symbolic reasoning techniques. These systems include knowledge-bases with rules that recommend design modifications and rules that switch between numerical optimization methods. Both systems adhere to a paradigm in which the design parameters, objective functions and constraint functions are provided in advance and are treated as fixed "black boxes" during the design process. In contrast to this, our work has focused on methods of reformulating the design parameters, constraints and objective functions themselves.

A symbolic reasoning technique called "activity analysis" is presented in [Williams and Cagan, 1994]. Activity analysis reasons about algebraic properties of constraints and objective functions in order to identify opportunities for dimension reduction. Activity analysis is a generalization of a previous technique, called "monotonicity analysis", that also reduced the dimension of design spaces [Choy and Agogino, 1986], [Papalambros and Wilde, 1988]. This line of research applies to situations in which dimension reduction is provably locally optimal. In contrast to this, we have focused on engineering applications in which the complexity of the constraint and objective functions precludes such proof. In such complex domains, computational experiments appear to provide the best means of identifying opportunities for dimension reduction. We have therefore developed a framework in which engineers can quickly and easily formulate, test and reformulate reduced design spaces. In any case, activity analysis and monotonicity analysis do immediately apply to our domains, because reduced subspaces are defined by critical quantities, rather than active constraints. Furthermore, in our domains, optimal solutions do not always lie in the reduced-dimension subspaces. Dimension reduction by itself would sometimes lead to solutions that are not locally optimal. It is therefore useful mainly as the first stage of a multi-stage design process.

Methods of intelligently constructing and selecting approximate models of physical systems have been presented in [Falkenhainer and Forbus, 1991], [Nayak, 1994]. Most of these methods reason

qualitatively about the behavior of models in order to choose a suitable one. Considerations of relevance, causality and monotonicity are used to decide which models might solve the problem at hand. They do not provide a framework for choosing among models that are qualitatively similar but have differing degrees of numerical accuracy. In contrast, our system provides a framework in which an engineer can carry out experiments that support a cost/benefit analysis of alternative approximations.

11 Contributions

We consider our work to be the first step in a research program aimed at automating the entire strategy formulation process. We have developed a language for expressing a wide variety of optimization strategies. We have also developed a catalog of transforms that map one strategy into another. The strategy language and transform catalog include many features that will seem familiar to design engineers. Approximation, re-calibration, dimension-reduction, and multi-stage optimization are common techniques that appear in the repertoire of most experienced engineers. The strategy primitives and transformations are therefore not new contributions in and of themselves. Nevertheless, our system does help to automate the use of these primitives and transformations in constructing optimization strategies. The strategy primitives provide the engineer with a specialized, high-level programming language. The transformations relieve the engineer from the burden of rewriting programs each time he modifies an optimization strategy. The transformations also enforce a discipline on the engineer. They encourage him to be more systematic in exploring the strategy space than he might otherwise be. In addition, the transformations come with guarantees regarding conditions under which they preserve the correctness and convergence properties of the initial strategy. We also believe that our system facilitates maintenance and modification of optimization strategies by maintaining a record of the derivation process; however, we have not experimentally tested this possibility.

Our work also contributes to the theory and methodology of engineering design. To begin with, our strategy language formalizes the hitherto informal notion of an optimization strategy. It specifies exactly what strategies are syntactically meaningful. It clarifies how distinct stages of optimization may differ from each other (i.e., different design spaces, different design parameters, different approximations of objective and constraint functions). It clarifies how distinct stages of optimization may be combined with each other (i.e., using the primitives *optimize*, *select*, *compose* and *converge*). In addition, our transform catalog enables one to understand how complex, multi-stage strategies may be seen as transformations on simpler strategies. Finally, our experiments demonstrate that the strategy language and transform catalog can be used to derive strategies that significantly improve the efficiency and robustness of the design optimization process in two realistic application domains. Nevertheless, we do not consider our current strategy language or transform catalog to be complete. Research on different types of design problems will likely reveal the need to add new primitives to our strategy language and new transforms to our catalog.

Finally, we also believe that our work provides a framework for organizing and codifying the results of future research. For example, consider an expression E generated by the grammar of our strategy language. If E contains non-terminal symbols, it describes a whole class of strategies sharing a common structure. One may use expressions of this sort to define research problems: "Are strategies of class E ever useful in practice?"; "Is a class E_i strategy always better than a

class E_j strategy?"; "Does the efficacy of a class E strategy depend on the application domain?". Once a strategy class is determined to be useful, one may seek to determine the reason for success: "What difficulties does a strategy of class E overcome?"; "What features of the design problem predict the success of a class E strategy?". Our system may thus serve as a framework for posing research questions and stating experimental or theoretical results.

12 Future Work

Further progress will require developing tools that help an engineer to select suitable transformations from among the many that are applicable. We are currently engaged in two approaches to this problem. In our first approach, we are developing techniques for producing graphical visualizations that will assist engineers in diagnosing problems with the optimization process. For example, one visualization technique would animate the series of gradient computations and line searches that is invoked by a sequential quadratic programming code, in the course of an optimization. This sort of animation may help engineers to recognize situations when the search process is getting stuck on a ridge or discontinuity. Another visualization technique would illustrate the evolution of intermediate quantities over the course of an optimization. This may help engineers to decide what parts of objective and constraint functions are nearly invariant and might therefore be approximated during early stages of an optimization process.

Our second approach aims to automate the sort of imagistic reasoning that takes place in the eye and mind of an engineer examining graphical visualizations like the ones described above. This work is inspired by research using Artificial Intelligence and Machine Vision techniques to extract information from simulations of physical systems [Abelson *et al.*, 1989], [Sacks, 1991], [Yip and Zhao, 1996]. We are developing a suite of tools for automatically observing and analyzing the behavior of trial optimizations. Each tool will be designed to extract information that can in principle be used to decide whether and how to apply a particular type of transformation. For example, in order to automate the decision of whether to apply a transform that reparameterizes the search space, it would help to have a tool that can automatically detect ridges or discontinuities. One such tool would run a trial optimization to obtain the stopping point and then find the eigenvalues of the Hessian of the objective function. A wide range of eigenvalues would suggest the presence of a ridge. An alternative tool would run multiple optimizations to obtain a set of stopping points and compute the fractal dimension of this set. A low fractal dimension would again suggest the presence of a ridge. Likewise, in order to automate the decision of whether to apply transforms that approximate objective or constraint functions, it would help to have tools that automatically measure the cost-effectiveness of various approximations. One such tool would run a trial optimization and measure the sensitivity of the final design with respect to errors in intermediate quantities. Notice that our strategy language and transform catalog play an important role in focusing this research. They define the decisions that must be made in the course of formulating a strategy. Once the decisions are defined, it becomes possible to think concretely about the types of experiments and data analysis tools that would help to automate the decision. For this reason, we believe that our work to date provides a useful framework for organizing future research.

13 Acknowledgments

Our research is supported by the National Aeronautics and Space Administration through NASA Grants NCC-2-802 and NAG2-817 and by the Hypercomputing and Design Project (HPCD), sponsored by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DABT 63-93-C-0064. It has benefited from discussions with Saul Amarel, Gene Bouchard, Andrew Gelsey, Haym Hirsh, Rich Keller, John Letcher, Takahiro Murata, Gerry Richter, Mark Schwabacher, Don Smith and Lou Steinberg, and the advice of the anonymous referees of this paper.

A Analysis of Individual Transforms

A.1 Preservation of Correctness

A.1.1 Transforms that Nest Optimization Strategies

Introduce Multi-Stage Optimization: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$, into a revised strategy S'. In the revised strategy S', the expression e is replaced by a revised expression e', where $e' = (optimize\ (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)\ OBJ\ EQNS\ INEQNS \ldots)$. The expression e' defines two stages of optimization. Each stage in e' is a copy of e. The result of the first stage in e' is the seed of the second stage in e'.

- Preservation of Mathematical Solution Set: The second stage of the revised expression e' has exactly the same search space, objective function and constraints as the original expression e. Only the seeds are different. The possible return values of our idealized *optimize* primitive depend only on the search space, objective function and constraints. They do not depend on the seed. Therefore the revised expression e' has the same set of possible return values as the original expression e. Therefore this transform is guaranteed to preserve the mathematical solution set.

- Preservation of Ideal Strategy Behavior: Suppose that the current strategy S has ideal behavior. Then every outermost optimization expression (i.e., an expression of the form $(optimize \ldots)$) must also have ideal behavior. Let α be the unique outermost optimization expression containing e. If $e = \alpha$, then the expression e has ideal behavior. Then e returns a design satisfying the Karush-Kuhn-Tucker conditions, regardless of the value of $SEED$. Since the second stage of e' simply executes e on a potentially different seed, the expression e' also returns a design satisfying the Karush-Kuhn-Tucker conditions. Therefore the expression e' also has ideal behavior. On the other hand, if $e \neq \alpha$, then replacing e with e' changes only the seed of α, which preserves the ideal behavior of α. Therefore, in either case, this transform preserves the ideal behavior of every outermost optimization expression. Therefore this transform is guaranteed to preserve ideal strategy behavior.

Introduce Multi-Start Optimization: This transform converts a current strategy S, containing a subexpression $e = (optimize\ \ SEED \ldots)$ into a revised strategy S' in which e is replaced by $e' = (select\ (list\ (random\ \ \ldots)^*)\ (\lambda(d)(optimize\ \ d \ldots))\ \ \ldots)$. The revised expression e' generates a random set of seeds, applies e to each of them, and returns the best result.

- Preservation of Mathematical Solution Set: The revised expression e' uses the same search space, objective function and constraints as the original expression e. The only difference is that e' runs the optimization multiple times from multiple seeds, whereas e runs the optimization once, from single seed. The possible return values of our idealized *optimize* primitive depend only on the search space, objective function and constraints. They do not depend on the seed. Therefore the revised expression e' has the same set of possible return values as the original expression e. Therefore this transform is guaranteed to preserve the mathematical solution set.

- Preservation of Ideal Strategy Behavior: Suppose that the current strategy S has ideal behavior. Then every outermost optimization expression (i.e., an expression of the form (*optimize* ...)) must also have ideal behavior. Let α be the unique outermost optimization expression containing e. If $e = \alpha$, then the expression e has ideal behavior. Then each time e' invokes e, it returns a design satisfying the Karush-Kuhn-Tucker conditions, provided such a design exists. Since e' simply returns the best result from all invocations of e, the expression e' will return a design satisfying the Karush-Kuhn-Tucker conditions, provided such a design exists. Therefore expression e' has ideal behavior. On the other hand, if $e \neq \alpha$, then replacing e with e' changes only the seed of α, which preserves the ideal behavior of α. Therefore, in either case, this transform preserves the ideal behavior of every outermost optimization expression. Therefore this transform is guaranteed to preserve ideal strategy behavior.

Introduce Convergence: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED \ldots)$ into a revised strategy S' in which e is replaced by $e' = (converge\ SEED\ (\lambda(d)(optimize\ d \ldots)) \ldots)$. The revised expression e' starts with a seed design $d_0 = SEED$ and repeatedly generates a new design d_i, where d_i results from applying e to d_{i-1}. The revised expression e' terminates when successive designs d_{i-1} and d_i satisfy a convergence condition.

- Preservation of Mathematical Solution Set: Each iteration of the revised expression e' uses the same search space, objective function and constraints as the original expression e. The only difference is that e obtains its seed as an input, whereas in e' each iteration other than the first one obtains its seed from the previous iteration. The possible return values of our idealized *optimize* primitive depend only on the search space, objective function and constraints. They do not depend on the seed. Therefore each iteration of the revised expression e' has the same set of possible return values as the original expression e. Therefore if the revised strategy e' terminates, it must return a value that is a possible return value of the original expression e. Furthermore if we assume that $d_1 = d_2$ satisfies the convergence test of the revised expression e', then each possible return value of e is a possible return value of e' after two iterations. Therefore this transform is guaranteed to preserve the mathematical solution set.

- Preservation of Ideal Strategy Behavior: Suppose that the current strategy S has ideal behavior. Then every outermost optimization expression (i.e., an expression of the form (*optimize* ...)) must also have ideal behavior. Let α be the unique outermost optimization expression containing e. If $e = \alpha$, then the expression e has ideal behavior. Then e returns a design satisfying the Karush-Kuhn-Tucker conditions, regardless of the value of $SEED$. Since the last iteration of e' simply executes e on a potentially different seed, the expression e'

also returns a design satisfying the Karush-Kuhn-Tucker conditions. Therefore the expression e' also has ideal behavior. On the other hand, if $e \neq \alpha$, then replacing e with e' changes only the seed of α, which preserves the ideal behavior of α. Therefore, in either case, this transform preserves the ideal behavior of every outermost optimization expression. Therefore this transform is guaranteed to preserve ideal strategy behavior.

Introduce Decomposition: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED \ldots)$ into a revised strategy S' in which e is replaced by $e' = (compose\ (list \ldots (optimize\ S_1 \ldots) \ldots (optimize\ S_n \ldots)))$. The revised expression e' operates by solving several optimization problems, each in a factor of the search space of the original expression e', and composing the results.

- Preservation of Mathematical Solution Set: The revised expression e' uses search spaces that are subspaces of the search space used in the original expression e. It uses objective and constraint functions that are restrictions of the objective and constraints used in e. Therefore this transform does not generally preserve the mathematical solution set. Nevertheless, in some special situations, syntactic analysis of the objective and constraint functions may be used to determine that the mathematical solution set is preserved after all. For example, the solution set will be preserved if the objective function is a sum of several terms, and no term or constraint references design variables in two different factor spaces. Therefore in the general case, this transform is not guaranteed to preserve the mathematical solution set; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

- Preservation of Ideal Strategy Behavior: In the revised strategy S', the underlying numerical optimization code is supplied with objective and constraint functions that are restrictions of the ones used in the initial strategy S. Even when the solution set is preserved, the different objective and constraint functions may result in different strategy behavior, since the reliability of the numerical optimization code may be sensitive to such details. Therefore this transform is not guaranteed to preserve ideal strategy behavior.

A.1.2 Transforms that Reformulate Search Spaces

Parameterize Intermediate Quantity: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED'\ OBJ'\ EQNS'\ INEQNS' \ldots)$. The revised expression e' has a new design parameter y and a new constraint, $y = Q(x_1, \ldots, x_n)$, where $Q(x_1, \ldots, x_n)$ is an intermediate quantity appearing in an objective or constraint function of the original expression e. The objective and constraint functions of e' are obtained from those of e by replacing each occurrence of $Q(x_1, \ldots, x_n)$ with the new variable y.

- Preservation of Mathematical Solution Set: The feasible points in the search space of e may be placed in one-to-one correspondence with the feasible points of the search space of e', by the mappings $(x_1, \ldots, x_n) \leftrightarrow (x_1, \ldots, x_n, Q(x_1, \ldots, x_n))$. Furthermore, the objective function of e agrees with the objective function of e' at corresponding feasible points. Therefore this transform is guaranteed to preserve the mathematical solution set.

- Preservation of Ideal Strategy Behavior: The performance of the numerical optimization code may be sensitive to the difference between incorporating the expression $Q(x_1, \ldots, x_n)$ directly into the objective and constraint functions (as in S) and handling the equation $y = Q(x_1, \ldots, x_n)$ as an explicit equality constraint (as in S'). Therefore this transform is not guaranteed to preserve ideal strategy behavior.

Solve Equality Constraint: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED'\ OBJ'\ EQNS'\ INEQNS' \ldots)$. The original expression e has an equality constraint $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$. In the revised expression e', a variable x_i has been removed from the set of design parameters, and the equality constraint has been dropped. The transform constructs a function $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ that solves $Q(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = 0$ for x_i in terms of $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. The objective and constraint functions of e' are obtained by composing $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ with the objective and constraint functions of e.

- Preservation of Mathematical Solution Set: We distinguish between cases in which the equality constraint has a unique solution for x_i in terms of $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ and cases in which no unique solution exists. When a unique solution exists, $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is well defined. The feasible points in the search space of the original expression e can be placed in one-to-one correspondence with the feasible points in the search space of the revised expression e', by the mappings $(x_1, \ldots, x_{i-1}, F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), x_{i+1}, \ldots, x_n) \leftrightarrow (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. The objective function of e agrees with the objective function of e' at corresponding feasible points. In this case, the transform is guaranteed to preserve the mathematical solution set. When the solution to the equality constraint is not unique, there exist points $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, α and β, such that $Q(x_1, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_n) = 0$, $Q(x_1, \ldots, x_{i-1}, \beta, x_{i+1}, \ldots, x_n) = 0$, and $\alpha \neq \beta$. Either $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \alpha$ or $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \beta$, but not both. The transform must therefore remove either $(x_1, \ldots, x_{i-1}, \alpha, x_{i+1}, \ldots, x_n)$ or $(x_1, \ldots, x_{i-1}, \beta, x_{i+1}, \ldots, x_n)$ from the search space, one of which may be a solution to the optimization problem. In this case, the transform is not guaranteed to preserve the mathematical solution set. Therefore in the general case, this transform is not guaranteed to preserve the mathematical solution set; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

- Preservation of Ideal Strategy Behavior: This transform changes the search space, objective function and constraints that are supplied to the numerical optimization code, regardless of whether the equality constraint has a unique solution. The behavior of the code may be sensitive to these changes, even if the mathematical solution set is preserved. Therefore this transform is not guaranteed to preserve ideal strategy behavior.

Constrain Intermediate Quantity: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED\ OBJ\ EQNS'\ INEQNS' \ldots)$. The expression $Q(x_1, \ldots, x_n)$ appears as an intermediate quantity in an objective or constraint function of the original expression e. The revised expression has a new constraint of the form $Q(x_1, \ldots, x_n) \leq K$, $Q(x_1, \ldots, x_n) = K$, or $Q(x_1, \ldots, x_n) \geq K$, for some specified bound K.

- Preservation of Mathematical Solution Set: The feasible points defined by the revised expression e' may be a proper subset of the feasible points defined by the original expression e, resulting in a mathematical model with a solution set different from the original one. Nevertheless, in some situations, it may be possible to ascertain in advance that the solution set will be preserved. For example, if the new constraint simply enforces a condition that is necessary for the objective function to evaluate without failure, the solution set will be preserved. Likewise, if the new constraint is merely a necessary condition on another more computationally expensive constraint, the solution set will also be preserved. Therefore in the general case, this transform is not guaranteed to preserve the mathematical solution set; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

- Preservation of Ideal Strategy Behavior: The behavior of the numerical optimization code may be sensitive to the presence of the new constraint, even when the new constraint does not change the set of feasible points. Therefore this transform is not guaranteed to preserve ideal strategy behavior.

A.1.3 Transforms that Approximate Objective and Constraint Functions

The approximating transforms "Expand Root Expression", "Expand Integral Expression", "Freeze Intermediate Quantity" and "Linearize Intermediate Quantity" have the potential to make arbitrarily large changes to the objective and constraint functions. These changes may move the location of a local optimum or change the number of local optima. Therefore these transforms are not guaranteed to preserve either the mathematical solution set or ideal strategy behavior.

A.2 Preservation of Convergence

A.2.1 Transforms that Nest Optimization Strategies

The transforms "Introduce Multi-Stage Optimization", "Introduce Multi-Start Optimization" and "Introduce Convergence" merely make copies of the current strategy, including its objective and constraint functions, or merely invoke the current strategy repeatedly with different seeds. Since the objective and constraint functions are not modified at all, the properties of convexity, boundedness, continuity and smoothness are trivially preserved by these three transforms. The transform "Introduce Decomposition" does change the objective and constraint functions; however, it merely restricts them to various linear, axis-parallel, subspaces of the original design space. The restricted functions will be convex, bounded, continuous and smooth if the original, unrestricted functions have the corresponding properties. These properties are therefore preserved by the decomposition transform as well.

A.2.2 Transforms that Reformulate Search Spaces

Parameterize Intermediate Quantity: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED'\ OBJ'\ EQNS'\ INEQNS' \ldots)$. The revised expression e' has a new design parameter y and a new constraint, $y = Q(x_1, \ldots, x_n)$, where

$Q(x_1, \ldots, x_n)$ is an intermediate quantity appearing in an objective or constraint function of the original expression e. The objective and constraint functions of e' are obtained from those of e by replacing each occurrence of $Q(x_1, \ldots, x_n)$ with the new variable y.

- Preservation of Convexity: This transform preserves convexity whenever the intermediate quantity Q is a linear function of the design parameters $(x_1, \ldots, x_n)$. Whenever the intermediate quantity is a non-linear function of the design parameters, the transform will not preserve convexity. Therefore in the general case, this transform is not guaranteed to preserve convexity; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

- Preservation of Boundedness, Continuity and Smoothness: In general, the boundedness, continuity and smoothness of the original objective or constraint function, from which $Q(x_1, \ldots, x_n)$ is extracted, implies the corresponding property in $Q(x_1, \ldots, x_n)$ itself. Exceptions may occur in special cases. For example, original objective or constraint function may actually be invariant with respect to changes in the value of $Q(x_1, \ldots, x_n)$. In addition, a discontinuity or non-smoothness in $Q(x_1, \ldots, x_n)$ may fortuitously be removed by a discontinuity or non-smoothness in another portion of the original objective or constraint function. Finally, when $Q(x_1, \ldots, x_n)$ is extracted from a branch of a conditional expression, the region of the design space over which $Q(x_1, \ldots, x_n)$ is evaluated may be enlarged. This may activate an unbounded, discontinuous or nonsmooth feature in $Q(x_1, \ldots, x_n)$ that was previously never evaluated. Nevertheless, we consider these exceptions to be rare events. Therefore in the general case, this transform is guaranteed to preserve boundedness, continuity and smoothness; however, in some special cases syntactic analysis of the objectives and constraints can determine that the guarantee does not apply.

Solve Equality Constraint: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED'\ OBJ'\ EQNS'\ INEQNS' \ldots)$. The original expression e has an equality constraint $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$. In the revised expression e', a variable x_i has been removed from the set of design parameters, and the equality constraint has been dropped. The transform constructs a function $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ that solves $Q(x_1, \ldots, x_i, \ldots, x_n) = 0$ for x_i in terms of $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. The objective and constraint functions of e' are obtained by composing $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ with the objective and constraint functions appearing in e.

- Preservation of Convexity: The convexity of the original problem implies that the equality constraint is linear. The inverse is also linear. The composition of the linear inverse with any convex function is also convex. Therefore this transform is guaranteed to preserve convexity.

- Preservation of Boundedness, Continuity and Smoothness: We distinguish between cases in which the equality constraint has a unique solution, and cases in which no unique solution exists. Suppose that a unique solution exists. Suppose further that the function $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ maps the bounding hyper-rectangle (defined by the LBs and UBs parameters supplied to the *optimize* primitive) into itself. Then the transform preserves

all three of the properties. Boundedness is preserved because the overall effect of this transform is simply to restrict each objective or constraint function to the subspace of the original space over which the equality constraint is satisfied. Continuity and smoothness are preserved because the function $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ inherits these properties from the original function, and because composition preserves continuity and smoothness of the components [Rudin, 1964]. On the other hand, suppose the inverse fails to exist, is not unique, or has a range lying outside the bounding hyper-rectangle. In this case, the transform may fail to preserve any of the three properties. For example, continuity and smoothness may be lost if a numerical root extraction method is used to implement the inverse function. When multiple solutions are present, the numerical root extractor may unpredictably switch from one root to another. The function $F(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ may therefore be discontinuous and non-smooth. Furthermore, boundedness may be lost if the range of the inverse lies outside the bounding hyper-rectangle. In this case, when the remaining functions are composed with the inverse, they will be evaluated at points outside the region where they were assumed to be bounded. Therefore in the general case, this transform is not guaranteed to preserve boundedness, continuity and smoothness; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

Constrain Intermediate Quantity: This transform converts a current strategy S, containing a subexpression $e = (optimize\ SEED\ OBJ\ EQNS\ INEQNS \ldots)$ into a revised strategy S' in which e is replaced by $e' = (optimize\ SEED\ OBJ\ EQNS'\ INEQNS' \ldots)$. The expression $Q(x_1, \ldots, x_n)$ appears as an intermediate quantity in an objective or constraint function of the original expression e. The revised expression has a new constraint of the form $Q(x_1, \ldots, x_n) \leq K$, $Q(x_1, \ldots, x_n) = K$, or $Q(x_1, \ldots, x_n) \geq K$, for some specified bound K.

- Preservation of Convexity: The three types of new constraint may be respectively rewritten in the forms $Q \leq 0$, $Q = 0$ and $Q \geq 0$. The new problem will be convex if and only if the function $Q(x_1, \ldots, x_n)$ is respectively convex, linear or concave. Therefore in the general case, this transform is not guaranteed to preserve convexity; however, a guarantee may be obtained from syntactic analysis of the objectives and constraints in some special circumstances.

- Preservation of Boundedness, Continuity and Smoothness: In general, the boundedness, continuity and smoothness of the original objective or constraint function, from which $Q(x_1, \ldots, x_n)$ is extracted, implies the corresponding property in $Q(x_1, \ldots, x_n)$ itself. Exceptions may occur in special cases. For example, the original objective or constraint function may actually be invariant with respect to changes in the value of $Q(x_1, \ldots, x_n)$. In addition, a discontinuity or non-smoothness in $Q(x_1, \ldots, x_n)$ may fortuitously be removed by a discontinuity or non-smoothness in another portion of the original objective or constraint function. Finally, when $Q(x_1, \ldots, x_n)$ is extracted from a branch of a conditional expression, the region of the design space over which $Q(x_1, \ldots, x_n)$ is evaluated may be enlarged. This may activate an unbounded, discontinuous or nonsmooth feature in $Q(x_1, \ldots, x_n)$ that was previously never evaluated. Nevertheless, we consider these exceptions to be rare events. Therefore in the general case, this transform is guaranteed to preserve boundedness, continuity and smoothness; however, in some special cases syntactic analysis of the objectives and constraints can determine that the guarantee does not apply.

A.2.3 Transforms that Approximate Objective and Constraint Functions

The approximating transforms "Expand Root Expression" "Expand Integral Expression" "Freeze Intermediate Quantity" and "Linearize Intermediate Quantity", have the potential to make arbitrarily large changes to intermediate quantities appearing in the objective and constraint functions. These changes may convert a convex objective or constraint function into a non-convex function. They may also have the effect of moving singularities, discontinuities or nonsmoothness into the bounding hyper-rectangle, where such pathologies previously lay entirely outside. These transforms therefore are not guaranteed to preserve any of the properties of convexity, boundedness, continuity or smoothness.

References

[Abelson *et al.*, 1989] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, Sussman J., J. Wisdom, and K. Yip. Intelligence in scientific computing. *Communications of the ACM*, 32, 1989.

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

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

[Ellman and Murata, 1996] T. Ellman and T. Murata. Deductive synthesis of numerical simulation programs from networks of algebraic and ordinary differential equations. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, Syracuse, NY, 1996.

[Ellman *et al.*, 1997] T. Ellman, J. Keane, M. Schwabacher, and K. Yao. Multi-level modeling for engineering design optimization. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 11(5), 1997.

[Falkenhainer and Forbus, 1991] B. Falkenhainer and K. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51:95–144, 1991.

[Gelsey *et al.*, 1996] 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.

[Gill *et al.*, 1981] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, London, England, 1981.

[Keane and Ellman, 1996] J. Keane and T. Ellman. Knowledge-based re-engineering of legacy programs for robustness in automated design. In *Proceedings of the Eleventh Knowledge-Based Software Engineering Conference*, Syracuse, NY, 1996.

[Keane, 1996] J. Keane. Extensions to Franz, Inc.'s Allegro Common Lisp foreign function interface. Technical Report HPCD-TR-41, Department of Computer Science, Rutgers University, 1996.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

[Letcher *et al.*, 1987] J. Letcher, J. Marshall, J. Oliver, and N. Salvesen. Stars and Stripes. *Scientific American*, 257(2):24–32, August 1987.

[Mitchell, 1996] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.

[Mostow, 1989] J. Mostow. Design by derivational analogy: Issues in the automated replay of design plans. *Artificial Intelligence*, 40:119–184, 1989.

[Nayak, 1994] P. Nayak. Causal approximations. *Artificial Intelligence*, 70:277–334, 1994.

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

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

[Partsch and Steinbruggen, 1983] H. Partsch and R. Steinbruggen. Program transformation systems. *Computing Surveys*, 15(3), 1983.

[Peressini *et al.*, 1988] A. Peressini, F. Sullivan, and J. Uhl. *The Mathematics of Nonlinear Programming.* Springer-Verlag, New York, NY, 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.

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

[Rudin, 1964] W. Rudin. *Principles of Mathematical Analysis.* McGraw-Hill, 1964.

[Sacks, 1991] E. Sacks. Automatic analysis of one parameter planar ordinary differential equations by intelligent numeric simulation. *Artificial Intelligence*, 48, 1991.

[Shukla *et al.*, 1997] V. Shukla, A. Gelsey, M. Schwabacher, D. Smith, and Knight D. Automated design optimization for the p2 and p8 hypersonic inlets. *Journal of Aircraft*, 34(2), 1997.

[Tong, 1990] S. S. Tong. Coupling symbolic manipulation and numerical simulation for complex engineering designs. In *Intelligent Mathematical Software Systems*, pages 241–252. North-Holland, New York, NY, 1990.

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

[Yip and Zhao, 1996] K. Yip and F. Zhao. Spatial aggregation: Theory and applications. *Journal of Artificial Intelligence Research*, 5, 1996.