Reformulation of Design Optimization Strategies

Thomas Ellman John Keane Takahiro Murata Arunava Banerjee
Department of Computer Science, Hill Center for Mathematical Sciences
Rutgers University, Piscataway, New Jersey 08855
{ellman,keane,murata,arunava}@cs.rutgers.edu
HPCD-TR-43

January 1996

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. We have therefore developed a system that supports interactive formulation, testing and reformulation of design optimization strategies. Our system includes an executable, second-order data-flow language for representing optimization strategies. The language allows an engineer to define multiple stages of optimization, and to specify the design parameters, constraints and objectives to be handled at each stage. We have also developed a set of transformations that reformulate strategies represented in our language. The transformations can approximate objective and constraint functions, define new constraints, and re-parameterize or change the dimension of the search space, among other things. The system is applicable to design problems in which the artifact is governed by algebraic or ordinary differential equations. We have tested the system on problems of racing yacht and jet engine nozzle design. We report experimental results demonstrating that our reformulation techniques can significantly improve the performance of automatic design optimization.

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 will 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 operation is not visible to the numerical optimization algorithm. Our research is based on the hypothesis that more powerful design optimization methods can be constructed by combining existing numerical methods with techniques for reasoning about the mathematical structure of the objective and constraint functions.

Consider first 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, which operates by approximating the objective with a quadratic function and solving the resulting quadratic program. The approximation-solution process is repeated until a fixed point is reached. CFSQP uses a purely numerical method to approximate the objective function. We believe that more cost-effective approximations of objective (and constraint) functions can be constructed exploiting the internal structure of these functions, in particular, by identifying intermediate quantities whose values do not change dramatically during an optimization process. Consider also how design parameters are used in numerical optimization algorithms. An illustrative example is Powell's direction set method [Press *et al.*, 1986], a conjugate-gradient algorithm. Powell's method operates by computing a linear transformation of the design parameters to find "conjugate directions" and then uses transformed parameters to find an optimum. The transformation-solution process is repeated until a fixed point is reached. Powell uses purely numerical methods to change the design parameters. We believe that more useful changes of parameters can be constructed by using intermediate quantities to construct non-linear transformations that align design parameters with ridges and discontinuities that would otherwise create problems for numerical optimization. In general, we believe that the best results can be obtained by a methodology of simultaneously formulating search spaces, objective functions and constraint functions in combination with the design optimization strategies that use them to solve problems.

In order to test our hypotheses, we have developed a software environment that supports interactive formulation, testing and reformulation of design optimization strategies. A "strategy" in our system is a description of a process that (potentially) uses multiple search spaces, multiple levels of approximation, and multiple stages of optimization, in order to solve design problems. Strategies are represented declaratively as second-order functional programs. Reformulations of strategies are implemented as transformations that rewrite second-order functional programs. Our system permits the user to interactively generate a space of strategies, and experimentally evaluate the performance of each on sets of test problems.

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:

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

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

Figure 1: The Constrained Optimization Problem

2 Testbed Domains

2.1 Design of Sailing Yachts

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.

The yacht problem presents a challenge to automated design optimization technology due to mathematical properties of the balance of force equations. Of particular interest is the fact that 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 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 objective function. Non-smoothness also provides an opportunity to reformulate the design optimization strategy as described below.

2.2 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 aircraft mass m flow: $\dot{m} = f(m, h, v, l_c, l_d, l_d)$. 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: $a_v(\alpha, m, h, v, l_c, l_d, l_d) = 0$ and $a_h(t, \alpha, m, h, v, l_c, l_d, l_d) = 0$ asserting that vertical and horizontal acceleration are each zero. These must then each 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.

The nozzle problem presents a challenge to automated design optimization technology largely due to the difficulty of computing the objective function. Each evaluation involves a numerical integration wrapped around numerical root finders wrapped around table interpolation functions, among other things. The evaluation code fails entirely for many points in the design space, due to table lookups going out of bounds, failure of numerical root finders to find roots, and violation of geometric constraints on the flap lengths. The design optimization process is protected from crashes of the objective function by wrapping the objective function with routines that gain control in the event of such failures. The wrappers keep the optimization process running by returning extremely bad values for the objective and constraint functions. Unfortunately, the wrappers also introduce discontinuities and non-smoothness into the objective function. These anomalies often cause optimization algorithms based on local search to get stuck at points that are not even locally optimal, much less globally optimal.

3 Specification of Optimization Strategies

A grammar for a language of optimization strategies is presented in Figure 2. The language enables a design engineer to specify strategies that involve multiple stages of optimization. Stages may involve different models 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 approxi-

$$
\begin{aligned}
S &\rightarrow (optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ OptMETH) && (1)\\
S &\rightarrow (select\ \ (list\ \ S^*)\ \ (\lambda(d)S)\ \ BF) && (2)\\
S &\rightarrow (compose\ \ (list\ \ S^*)) && (3)\\
S &\rightarrow (let\ \ ((SYMBOL\ \ RV)^*)\ \ S) && (4)\\
S &\rightarrow (converge\ \ S\ \ (\lambda(d)S)\ \ BF) && (5)\\
S &\rightarrow (randomize\ \ (list\ \ RV^*)\ \ (list\ \ RV^*)) && (6)\\
S &\rightarrow (list\ \ RV^*) && (7)\\
RF &\rightarrow (\lambda\ \ (d)\ \ RV) && (8)\\
BF &\rightarrow (\lambda\ \ (d_1 d_2)\ \ BV) && (9)\\
RV &\rightarrow 2ndORDER\ \ |\ \ 1stORDER && (10)\\
BV &\rightarrow (RELATION\ \ RV\ \ RV) && (11)\\
1stORDER &\rightarrow ARITHMETIC\ \ |\ \ INTERPOLATION\ |\ \ CONDITIONAL && (12)\\
2ndORDER &\rightarrow INTEGRAL\ \ |\ \ ROOT\ |\ \ OPTIMUM && (13)\\
INTEGRAL &\rightarrow (integrate\ \ (list\ \ RF^*)\ \ (list\ \ RV^*)\ \ RV\ \ RV\ \ IntMETH) && (14)\\
ROOT &\rightarrow (root\ \ (list\ \ RF^*)\ \ (list\ \ RV^*)\ \ RtMETH) && (15)\\
OPTIMUM &\rightarrow (apply\ \ RF\ \ S) && (16)\\
OptMETH &\rightarrow cfsqp\ \ |\ \ dfp\ |\ \ simplex\ |\ \ \ldots && (17)\\
RtMETH &\rightarrow newtonraphson\ \ |\ \ bisection\ |\ \ \ldots && (18)\\
IntMETH &\rightarrow rungekutta\ \ |\ \ \ldots && (19)
\end{aligned}
$$

Figure 2: Design Optimization Strategy Grammar

mate 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 starting with rule (1). The resulting expression describes a strategy starting from a single "seed" design (S), optimizing a real-valued objective function (RF) subject to inequality constraints and equality constraints designated by lists of real-valued functions, using a designated numerical method. The seed S may be a fixed-value design, (expanding S using rule 7), or a design that is randomly generated within a hypercube specified by lists of upper and lower bounds, (expanding S using rule 6). Strategies based on multiple starting points are generated using rule (2). The resulting expression describes a process that applies an optimization strategy function ($\lambda(d)S$) to each of several seeds ($list\ \ S^*$), compares them using a boolean-valued function BF and selects the best. Seed designs may also be generated using any optimization strategies described by the grammar, including those resulting from rules (1) through (5). Thus, nested strategies, involving multiple stages of optimization, may be specified by exploiting the recursive structure of the grammar.

Strategies using decomposition may be generated using rule (3). This rules generates an expression describing a design that is composed of multiple partial designs. Each partial design is found using a separate strategy. Strategies using approximation of constraint and objective functions may

be generated 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 may be 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 and/or constraint functions as it moves through a design space. Finally, rules (8) through (19) allow for expressing objective and constraint functions in terms of first-order operations (arithmetic, interpolation, conditionals) as well as second order operations (integration, root-finding and optimization). Notice that optimization can be placed inside objective and constraint functions. This allows the values of some design parameters to be optimized dynamically during the process of evaluating the rest of the design.

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 a subspace of the complete 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 product of unimodal 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 and discontinuities. Use of multiple starting points increases the likelihood of finding a true 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 have their values calibrated and "frozen" prior to the start of an optimization strategy, and periodically re-calibrated as the strategy is iterated to a fixed point. The grammar presented in Figure 2 is a useful means of describing the range of strategies that may be constructed in our system; however, the grammar itself is not used to generate strategies. Instead, the user generates strategies using a catalog of transformations that operate on ground sentences of the strategy language, as described below.

4 Generating a Space of Optimization Strategies

4.1 User Interaction

The user begins by preparing an initial design optimization strategy to serve as the starting point of the strategy reformulation process. The initial strategy is formulated as an expression of the form:

$$(\lambda(d\ \ p)\ \ (optimize\ \ d\ \ RF[p]\ \ (list\ \ RF[p]^*)\ \ (list\ \ RF[p]^*)\ \ OptMETH))$$

This expression describes a strategy function that takes a seed design d and a set of problem instance parameters p and uses a numerical algorithm $OptMETH$ to optimize an objective function $RF[p]$

subject to sets of equality and inequality constraints ($listRF[p]^*$) that reference p, the problem instance parameters. The initial strategy should include all potentially relevant design parameters, and the "exact" versions of the objective and constraint functions. After preparing the initial strategy, the user selects a transformation from a menu and applies it to the initial strategy, to create a new, derived strategy. In many cases, a given transformation can be applied in more than one way. In such cases, the system asks the user which instantiation of the transformation he wants to apply. The system then applies the selected transformation and displays the revised strategy to the user. Users who are LISP literate may view the strategies in the form of LISP expressions. Users may also view strategies drawn automatically on the screen as data-flow graphs.

As the user interacts with the strategy transformation system, he generates a tree in the space of design optimization strategies. The tree is maintained by the system and displayed on request to the user. Each node in the tree holds a description of an optimization strategy. The user can move up and down the tree at will to backtrack to previous strategies to try new alternatives. He can run the strategy in any node on a set of test problems and store the results in a database associated with that node. He can also generate plots of various kinds of data that describe the behavior of strategies on test problems, e.g., a plot of the path through a search space, or a plot of the evolution of an objective, constraint, or intermediate quantity. The user can also annotate the current strategy node with his own observations and conclusions about the behavior of the strategy.

4.2 The Catalog of Transformations

Our system includes four groups of transformations. Transformations that reformulate the search space are described in Figure 3. Transformations that introduce approximations into objective and constraint functions are described in Figure 4. Transformations that construct nested optimization strategies are described in Figure 5. A fourth 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). We focus here on describing the key transforms used in deriving yacht and nozzle design strategies.

4.2.1 Reformulating Search Spaces

The transforms for reformulating search spaces (Figure 3) were designed to use intermediate quantities of the objective and constraint functions to re-parameterize and/or reduce the dimension of design spaces. Consider first how an intermediate quantity may be used to re-parameterize 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 forced to equal the intermediate quantity Q by assertion of an equality constraint: $y = Q(x_1, \ldots, x_n)$. He then uses the transform "Solve Equality Constraint" to remove one of the original parameters x_i by solving $y = Q(x_1, \ldots, x_i, \ldots, x_n)$ for x_i symbolically (e.g. using Maple [Char *et al.*, 1992] at strategy construction time) or numerically (e.g., using Newton-Raphson at strategy execution time). The effect of these two transformations is to change the set of parameters used to 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 impose an equality constraint $Q(x_1, \ldots, x_n) = K$ on an intermediate

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 computed by some constraint or objective function: (1) Introduce a new design parameter y, add y as an additional argument to each constraint or objective function; (2) Impose an equality constraint requiring that $y = Q(x_1, \ldots, x_n)$.

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 $Q(x_1, \ldots, x_n) = K$, $Q(x_1, \ldots, x_n) \leq K$, or $Q(x_1, \ldots, x_n) \geq K$, etc., for some specified bound K.

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; (2) Drop x_i from the argument list of each constraint or objective function; (3) 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 $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

Figure 3: Transforms to Reformulate Search Space Spaces

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 to serve as a basis for re-parameterization and dimension reduction. 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 re-parameterize the search space, the coordinate axies of the new space are aligned with the ridge 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 $Q(\bar{d}) = K$. Thus our heuristic recommends using critical quantities to re-parameterize or reduce the dimension of search spaces.

4.2.2 Approximating Objective and Constraint Functions

The transforms for approximating objective and constraint functions (Figure 4) were designed to exploit the internal structure of objective and constraint functions to construct more cost-effective approximations than are available with purely numerical approaches to design optimization. The transforms "Expand Root Expression" and "Expand Integral Expression" are used to replace calls to numerical root-finding 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 inte-

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 $x_1, \ldots, x_n$, and any expression $Q(x_1, \ldots, x_n)$ appearing as an intermediate quantity computed by some constraint or objective function: Replace $Q(x_1, \ldots, x_n)$ with a fixed value computed outside the scope of the optimization, and obtained by computing Q for the seed values of parameters $(x_1, \ldots, x_n)$.

Figure 4: Transforms to Approximate Objective and Constraint Functions

grals) 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.

4.2.3 Nesting Optimization Strategies

Our transforms for constructing nested optimization strategies are described in Figure 5. 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 3, in the context of the discussion of our strategy language grammar.

4.3 Derivations of Yacht and Nozzle Design Strategies

Derivations of strategies for yacht optimization are presented pictorially in Figure 6. The initial yacht design strategy Y_0 specifies an optimization of the five design parameters (l, b, d, h, s) (length, beam, draft, keel-height and winglet-span described in Section 2) to maximize the velocity objective function, subject to one inequality constraint requiring the yacht to be stable. The transform "Introduce Multi-Start" is then used to construct strategy Y_1 that operates by generating a set of random seed designs, optimizes each, and selects the best. (All of the remaining yacht strategies, Y_2 through Y_9 use the same multi-start process.) Strategy Y_2 is derived from Y_1 by 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. Thus strategy Y_2 operates with design parameters obtained by a non-linear transformation on the original parameters. Strategy Y_3 is derived from Y_1 using "Constrain Intermediate Quantity" to assert four equality constraints that force the critical quantities to lie at points of non-smoothness, and

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.

$(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots) \quad \Rightarrow$

$(optimize\ \ (optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots)\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots)$

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

$(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots) \quad \Rightarrow$

$(select\ \ (list\ \ S^*)\ \ (\lambda(d)(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots))\ \ BF)$

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

$(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots) \quad \Rightarrow$

$(compose\ \ (list\ \ (optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots)^*))$

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.

$(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots) \quad \Rightarrow$

$(converge\ \ S\ \ (\lambda(d)(optimize\ \ S\ \ RF\ \ (list\ \ RF^*)\ \ (list\ \ RF^*)\ \ \ldots))\ \ BF)$

Figure 5: Transforms to Nest Optimization Strategies

then using "Solve Equality Constraint" to remove the four equality constraints and the parameters b, d, h, and s from the set of design parameters. Thus strategy Y_3 operates 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.

Strategy Y_8 is derived from Y_1 using "Expand Root Expression" to replace the numerical routine for solving the balance of force equations with a single step of the Newton-Raphson iteration formula: $x_1 = \bar{x}_0 - (\nabla \bar{f}(\bar{x}_0))^{-1} \bar{f}(\bar{x}_0)$. In this formula, $\bar{x}$ is the velocity and heel vector; $\bar{f}$ is the net force and net torque vector; and $\nabla \bar{f}$ is the jacobian of the force and torque vector w.r.t. velocity and heel. The transform "Freeze Intermediate Quantity" is then used to set up a process that precomputes the exact values of velocity v and heel ϕ of the seed design, and uses these for $\bar{x}_0$ in the Newton-Raphson expansion inside the optimization process. Finally, the transform "Introduce Convergence" is used to construct an iterative strategy that periodically re-calibrates $\bar{x}_0$ in between successive phases optimization. Strategy Y_9 is derived from Y_8 using "Freeze Intermediate Quantity" to do with the jacobian $\nabla \bar{f}(\bar{x}_0)$ what Y_8 does with the seed $\bar{x}_0$. Thus Y_9 uses fixed values of the velocity and heel seed and fixed values of the net force and net torque jacobian inside the optimization. It periodically

re-calibrates these during successive phases of optimization.

Strategy Y_4 is derived from Y_3; however, it may be understood as a hybrid that combines the dimension reductions of Y_3 with the approximations of Y_8. Likewise, strategy Y_6 is derived from Y_4; however, it may be understood as a combination of the dimension reductions of Y_3 and the approximations of Y_9. Finally, strategies Y_5 and Y_7 are respectively derived from Y_4 and Y_6. Each includes an additional, final stage of optimization that uses the original objective function and original five design parameters. The final stage is introduced as an attempt to remedy any sub-optimality due to the use of dimension reduction or approximation during the earlier phases of optimization.

Derivations of strategies for nozzle optimization are presented pictorially in Figure 7. The initial nozzle design strategy N_0 specifies an optimization of the three flap lengths (l_c, l_d, l_e) (convergent length, divergent length and external length described in Section 2) to minimize fuel consumption over a prescribed mission. The transform "Introduce Multi-Start" is then used to construct strategy N_1 that operates by generating a set of random seed designs, optimizes each, and selects the best. (All of the remaining nozzle strategies, N_2 through N_6, use the same 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, 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 extreme 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.

Strategy N_5 is derived from N_1 using "Constrain Intermediate Quantity" to assert an inequality constraint placing a lower bound (zero) on $g(l_c, l_d, l_e)$. Strategy N_6 is derived from N_5 using "Constrain Intermediate Quantity" to assert an equality constraint forcing $g(l_c, l_d, l_e)$ to have a fixed, small, positive value, and then using "Solve Equality Constraint" to remove the equality constraint and the parameter l_e (external length) from the design space. Thus strategy N_6 operates in a two-dimensional non-linear subspace, including only l_c (convergent length) and l_d (divergent length). Strategy N_7 is derived from N_6. It includes an additional, final stage of optimization which includes all three design parameters, l_c,l_d, and l_e, but preserves the inequality constraint placing a lower bound on $g(l_c, l_d, l_e)$. The final stage is an attempt to remedy any sub-optimality due to dimension reduction during the earlier phases of optimization.

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 finders. 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 these same transforms to set up a process of periodically re-calibrating the seeds and jacobians in the expanded root expressions.

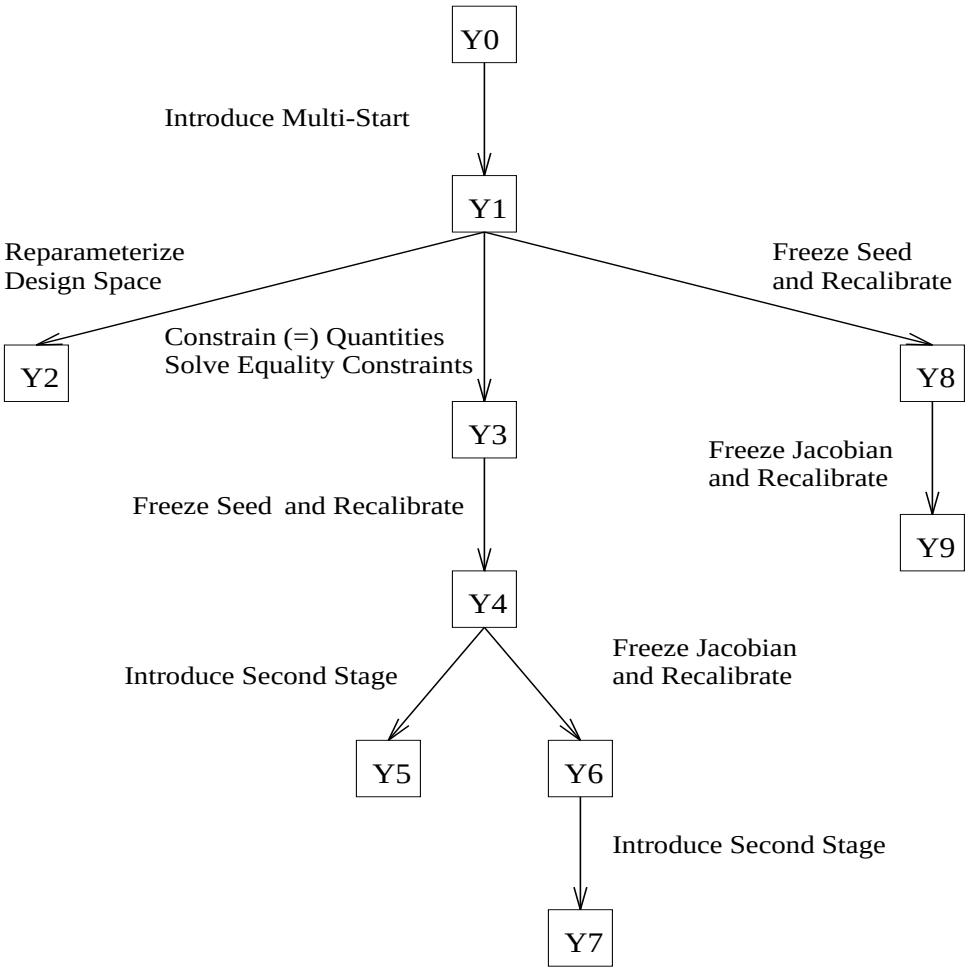

Figure 6: Derivation Relations among Yacht Optimization Strategies

5 Experimental Results

Results of testing yacht and nozzle design strategies are presented in Figures 8 and 9. 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 as the underlying numerical optimization algorithm in each strategy, and each stage of optimization. For each (problem,strategy) combination we recorded the CPU time, and the velocity or 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; N_1 required an average of 55:17 CPU minutes - all on a Sparc 5 workstation. We normalized velocity or takeoff mass for each (problem,strategy) combination, dividing by the velocity or takeoff mass of the "optimal" design for that problem - i.e., the best design found by any strategy, yielding "Average Relative Quality" shown in the table. Finally, for each strategy, we recorded the best and worst relative

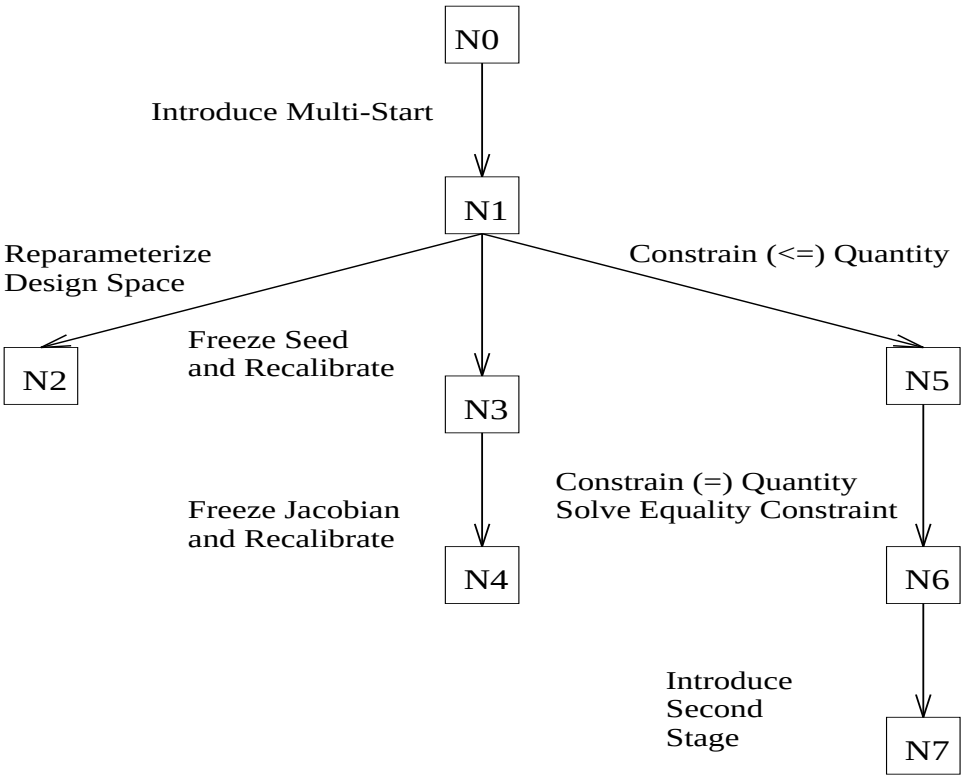

Figure 7: Derivation Relations among Nozzle Optimization Strategies

quality over all problems in the test set.

A typical yacht velocity is about 10 knots, so a 1% variation in quality amounts to about a tenth of a knot: an amount that is significant in a yacht race. A typical value for takeoff mass is about 175000 kilograms. A 1% variation in quality amounts to 1750 kilograms of fuel an amount that is significant in nozzle design. Note that the importance of finding the exactly optimal design will vary over the product design cycle. Preliminary designs presumably need not be as carefully optimized as final designs. Note also that quality and CPU time are somewhat interchangeable. Strategies that run faster can use more starting points in fixed CPU time and presumably achieve higher quality results.

The effect of approximation and re-calibration can be seen in the yacht domain by comparing Y_1 with Y_8 and Y_9, 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, and a factor of more than two in the nozzle domain - in all cases incurring a modest loss of quality. We believe that additional speedup is possible by approximating, and re-calibrating the net-torque functions used in strategies Y_8 and Y_9 and the net-vertical-acceleration and net-horizontal-acceleration acceleration functions used in strategies N_3 and N_4. The loss of quality may be remedied by adding additional stages of optimization. These experiments are in progress.

The effect of re-parameterization 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 re-parameterization to 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

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

Figure 8: Performance of Yacht Optimization Strategies

Strategy	Rel. CPU Time	Avg. Rel. Quality	Best Rel. Quality	Worst Rel. Quality	Rel. Std. Deviation
N_1	100.00%	102.90%	100.23%	108.77%	3.34%
N_2	112.04%	101.95%	100.47%	104.18%	2.56%
N_3	84.85%	103.81%	101.69%	108.77%	2.76%
N_4	42.65%	103.81%	101.69%	108.77%	2.78%
N_5	99.31%	102.16%	100.47%	104.69%	3.77%
N_6	84.98%	100.65%	100.00%	102.95%	1.62%
N_7	116.83%	100.00%	100.00%	100.00%	

Figure 9: Performance of Nozzle Optimization Strategies

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 cost apparently results from the fact that CFSQP does not terminate prematurely but continues to improve design quality.

The effect of dimension reduction can be seen by comparing Y_1 with Y_3 in the yacht domain Strategy Y_3 runs considerably faster than strategy Y_1 while suffering a small loss in average quality. On at least one problem, Y_3 found an "optimal" design; however on at least one other problem, Y_3 found a design that was significantly below optimal. This results demonstrate that our dimension reducing transformations can significantly speed up optimization; however, they are not guaranteed to preserve optimality. These results also suggest that 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_6. Strategy N_6 speeds optimization by small amount, compared to N_1. In addition N_6 achieves a dramatic improvement in design quality. We believe this improvement results from the fact that N_1 gets stuck on ridges which are removed in the reduced dimension space searched by N_6. 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_6.

Finally the effect of multi-stage optimization may be seen in the yacht domain by comparing Y_4 with Y_5 and comparing Y_6 with Y_7. Strategies Y_5 and Y_7 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_6 with N_7 (and with all the other strategies). Strategy N_7 is the same as N_6, 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_6 and N_7) with approximation (N_3 and N_4). Experiments to test such hybrid strategies are in progress.

Additional insight can be obtained by examining the variation in design quality, shown in Figure 9 for the nozzle domain. The last column in this table shows for each strategy the relative standard deviation in design quality (standard deviation divided by mean), averaged over all test problems. This quantity is determined by looking at the final designs that result from each of the five randomly selected seed designs. Notice that N_2 (re-parameterization) achieves a modest reduction in relative variation, compared to the N_1, (the default). Furthermore, strategy N_6 (dimension reduction), achieves a dramatic reduction in variation. These data suggest that N_6 could make due with fewer starting points than the other strategies, with a considerable improvement in computational efficiency. Strategy N_7, which uses N_6 as its first stage, would likely be made more efficient as well.

6 Related Work

Several other investigators have attempted to use knowledge-based methods to improve the performance of numerical optimization, including [Orelup *et al.*, 1988; Tong, 1988; Powell, 1990]. This work has focused primarily on automating the choice of underlying numerical algorithm, and numerical parameters to that algorithm. It has also adhered to a paradigm in which the design parameters, objective functions and constraint functions are provided in advance and are treated as fixed "black boxes". In contrast to this, our work has focused on methods of reformulating the design parameters, constraints and objective functions themselves.

Williams and Cagan present a method called "Activity Analysis" to identify opportunities for dimension reduction in design optimization [Williams and Cagan, 1994]. Their method applies to situations in which dimension reduction is provably optimal. 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 and other reformulations. We have therefore developed a framework in which engineers can quickly and easily construct, test and reformulate various design optimization strategies. In any case, activity analysis does not immediately apply to our domains, because reduced subspaces are defined by critical quantities, rather than active constraints. Furthermore, in the nozzle domain, optimal solutions do not actually lie in the lower dimensional subspaces.

Methods of intelligently constructing and/or selecting approximate models of physical systems have been presented in [Weld, 1990], [Falkenhainer and Forbus, 1991], [Nayak, 1992], [Nayak *et al.*,

1992], [Farquhar, 1994], [Iwasaki and Levy, 1994], [Rickel and Porter, 1994]. Most of these methods reason qualitatively about the behavior of models in order to choose a suitable one. Considerations of relevance, causality and/or 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 provide differing degrees of numerical accuracy. Exceptions include [Ellman *et al.*, 1993] and [Falkenhainer, 1993]. Of these, Ellman's "Gradient Magnitude Model Selection" technique is closest to the approach we have used here. GMMS is a rule for dynamically selecting approximations during design optimization based on the magnitude of the local gradient of the objective function. GMMS provides no mechanism for generating approximate objective and constraint functions, as we have done. Furthermore, GMMS is not easily combined with arbitrary numerical optimization routines. It appears to require use of a specialized numerical algorithm. In contrast, our recalibration/convergence process can be interleaved with any numerical optimization method.

7 Future Work

Our system requires a human engineer to select suitable series of transformations in the course of formulating a design optimization strategy. The transformations relieve the engineer from the burden of re-coding optimization strategies, objective functions and constraint functions, each time he changes strategies. Furthermore, the transformations enforce a discipline on the engineer, helping him to be more systematic in his exploration of the strategy space than he would be otherwise. Nevertheless, in the future we would like to develop a more automated reformulation system. Portions of the strategy space can probably be searched automatically. For example, an engineer might specify a small set of re-parameterizations, dimension reductions and approximations he wants to consider, and let the system automatically construct all meaningful combinations and try them on test problems. We expect that Machine Learning techniques can be used to construct rules that recommend different design optimization strategies for different regions of the problem class parameter space. We expect that search of the strategy space can be further aided by specialized diagnostic tools that analyze the behavior of objective and constraint functions. For example, the choice of approximation can be guided by sensitivity analysis. Likewise, ridges in evaluation functions can often be identified by diagonalization of the Hessian of the objective function. A collection of such diagnostic tools is presented in [Gelsey *et al.*, 1996]. We expect that full automation of the strategy reformulation process will require combinations of such numerical diagnostics with the symbolic methods we have described above.

8 Summary

The performance of numerical design optimization can be significantly enhanced by symbolic reformulation techniques. Results from the yacht domain demonstrate significant speedup of design optimization, with small loss of design quality. Results from the nozzle domain demonstrate smaller speedup, but dramatic improvements in design quality. Reformulations that introduce recalibratable approximations, perform nonlinear changes of design parameters, reduce dimension of search spaces, and introduce multiple stages of optimization, all played a role in achieving the performance improvements. These results demonstrate the power of a methodology of simultaneously

formulating search spaces, objective functions and constraint functions in combination with the design optimization strategies that use them to solve problems.

9 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 has benefited from discussions with Saul Amarel, Gene Bouchard, Andrew Gelsey, Haym Hirsh, Rich Keller, John Letcher, Gerry Richter, Mark Schwabacher, Don Smith and Lou Steinberg.

References

[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.

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

[Falkenhainer and Forbus, 1991] B. Falkenhainer and K. Forbus. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 50, 1991.

[Falkenhainer, 1993] B. Falkenhainer. Ideal physical systems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993.

[Farquhar, 1994] A. Farquhar. A qualitative physics compiler. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington., 1994.

[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. To appear.

[Iwasaki and Levy, 1994] Y. Iwasaki and A. Levy. Automated model selection for simulation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington., 1994.

[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.

[Nayak *et al.*, 1992] P. Nayak, L. Joskowicz, and S. Addanki. Automated model selection using context-dependent behaviors. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, California., 1992.

[Nayak, 1992] P. Nayak. Causal approximations. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, California., 1992.

[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.

[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.

[Rickel and Porter, 1994] J. Rickel and B. Porter. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington., 1994.

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

[Weld, 1990] D. Weld. Approximation reformulations. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990. MIT Press.

[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.