SRC TR 88-60

TECHNICAL RESEARCH REPORT

Aspects Of Optimization-Based CADCS

by

A.L. Tits, M.K.H. Fan, and E.R. Panier

SYSTEMS RESEARCH CENTER

UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND 20742

ASPECTS OF OPTIMIZATION-BASED CADCS

A.L. Tits, M.K.H. Fan, E.R. Panier

Electrical Engineering Department and Systems Research Center

University of Maryland, College Park, MD 20742

Abstract

With the recent dramatic increase in available computing power, numerical optimization has become an attractive tool for the design of complex engineering systems. Yet, generalized use of numerical optimization techniques in design has been hindered by (i) the difficulty to translate in a faithful manner the actual design problem into any kind of rigid mathematical optimization problem, (ii) the inability of classical optimization tools to efficiently take into account the many distinctive features of optimization problems arising in a design context, and (iii) the unavailability of software tools offering to the designer a powerful as well as congenial environment supporting such capabilities.

In this paper, some aspects of these questions are touched upon and avenues are suggested to address them. In particular, a recently proposed interaction driven design methodology is briefly described and numerical optimization schemes satisfying two specific requirements of many design problems are sketched. As an example, the design of a controller for a copolymerization reactor using the Maryland developed CONSOLE system is considered.

Keywords. Computer-aided system design; man-machine interaction; man-machine systems; optimization; computational methods; nonlinear programming; semi-infinite programming; iterative methods; chemical variables control; industrial control.

INTRODUCTION

The most challenging task one faces when designing a complex engineering system is that of coming up with an appropriate system "structure". This task calls extensively upon the engineer's ingenuity, creativity, intuition and experience. After a structure has been (maybe temporarily) selected, it remains to determine the "best" value of a number of "design parameters". The engineer's input is still essential here, as multiple tradeoffs are

bound to appear. However, except in the simplest cases, achieving anything close to optimal would be impossible without the support of numerical optimization.

With the recent dramatic increase in available computing power, numerical optimization has indeed become an attractive tool for the design of complex engineering systems (see, e.g., [1,2]). A prime reason for this is the large variety of design specifications that it can potentially accommodate. Yet, generalized use of numerical optimization techniques in design has been hindered by (i) the difficulty to translate in a faithful manner the actual design problem into any kind of rigid mathematical optimization problem, (ii) the inability of classical optimization tools to efficiently take into account the many distinctive features of optimization problems arising in a design context, and (iii) the unavailability of software tools offering to the designer a powerful as well as congenial environment supporting such capabilities.

Recently, an optimization-based design methodology was proposed, that accommodates the inherent trade-off driven character of most design problems [3,4]. This methodology calls extensively upon the designer's knowledge and experience and emphasizes man-machine interaction. Initially the designer is not required to express in any precise manner what he would consider an optimal design. Rather, he is first requested to merely provide rough guidelines to the optimization process. After a corresponding approximate solution has been approached, he has the opportunity to interactively refine his definition of optimality, thus progressively "steering" the design toward the "true" optimal solution. Between successive interventions of the designer, the optimization process is faced with the task of tackling a constrained minimax problem. Specialized techniques have been proposed to efficiently account for special needs of such problems in a design context, such as that of dealing with approximate function values [5–7] and in particular with functional constraints [8,9], of generating feasible iterates [10–14], and of allowing for specific types of nondifferentiable functions [15–21]. Finally, software packages are now emerging that combine sophisticated interactive capabilities with the power of advanced optimization techniques [22–27].

In this paper some aspects of these issues are discussed, mostly in the context of work currently in progress at the University of Maryland. The methodology put forth in [3] and [4] is first outlined. Several requirements of optimization problems arising in design are then pointed out and specific techniques are suggested for (i) handling functional constraints and

(ii) generating a sequence of feasible iterates. Finally, a design example —a controller for a copolymerization reactor—using the recently developed CONSOLE package is presented.

METHODOLOGY

Problem Formulation

In an attempt to better represent real world design problems, the methodology proposed in [3] allows for three qualitatively different types of design specifications. An *objective* is a specification of a quantity that should be minimized or maximized. Typically multiple competing objectives are present. A *hard constraint* is a specification of a quantity that must achieve a specified threshold, or the corresponding design has no or little value. A *soft constraint* is a specification of a quantity that should achieve or at least approach a specified threshold, i.e., should be minimized or maximized as long as this threshold is not achieved. Soft constraints can be thought of as intermediate between objectives and hard constraints.

A precise meaning is given to the optimization problem by means of *good* and *bad* values assigned by the designer to each objective and soft constraint, according to the following uniform satisfaction/dissatisfaction rule: having any of the various objectives or soft constraints achieve its corresponding good value should provide the same level of *satisfaction* to the designer, while having any of them achieve its bad value should provide the same level of *dissatisfaction*. Also, the good value of a soft constraint must be the corresponding target value. Each one of the objectives and soft constraints is then scaled using its good and bad values according to the formula

$$\text{scaled\_value} = \frac{\text{raw\_value} - \text{good\_value}}{\text{bad\_value} - \text{good\_value}} .$$

Thus, achieving the good value will yield a scaled value of 0 while achieving the bad value will yield a scaled value of 1. For possible use in phase 1 (see below) where the maximum hard constraint violation is forced to decrease, hard constraints are also assigned good values (which are the threshold values) and bad values (only required to be consistent across hard constraints). The resulting optimization problem is as follows

$$\text{(P)} \qquad \begin{aligned} \min_{x} \quad & \text{obj}_k(x) \quad \forall k \\ \text{s. t.} \quad & \text{soft}_i(x) \leq 0 \quad \forall i \\ & \text{hard}_j(x) \leq 0 \quad \forall j \end{aligned}$$

where obj_k, soft_i, hard_j are now *scaled values* of objectives, soft constraints, and hard constraints, respectively, and where the first set of inequalities is meant in a "soft" sense. Problem (P) is then assigned three different meanings, corresponding to three different cases (or *phases*) according to feasibility or infeasibility of x with respect to hard and soft constraints.

Phase 1: not all hard constraints are satisfied. (P) takes the form

$$\min_x \quad \max_j \quad \text{hard}_j(x).$$

Phase 2: all hard constraints are satisfied; not all (scaled) objectives and soft constraints are nonpositive. (P) takes the form

$$\begin{aligned} \min_x \quad & \max_{k,i} \quad \{\text{obj}_k(x),\ \text{soft}_i(x)\} \\ \text{s. t.} \quad & \text{hard}_j(x) \leq 0 \quad \forall j. \end{aligned}$$

Phase 3: all hard constraints are satisfied and all (scaled) objectives and soft constraints are nonpositive. (P) takes the form

$$\begin{aligned} \min_x \quad & \max_k \quad \text{obj}_k(x) \\ \text{s. t.} \quad & \text{soft}_i(x) \leq 0 \quad \forall i \\ & \text{hard}_j(x) \leq 0 \quad \forall j. \end{aligned}$$

Typically most of the optimization run and user interaction (see below) will take place in Phase 2 or Phase 3.

To keep the exposition simple, we have so far left out the question of *functional* specifications, i.e., functional objectives and functional constraints. These are specifications according to which some quantity which depends on some free parameter (e.g., time or frequency) must be made small or large for all values of this parameter. Such are, e.g., specifications on time or frequency responses of a dynamical system. Similar to *ordinary* (non-functional) specifications, these can be objectives, soft constraints or hard constraints. They are normalized using designer specified good and bad *curves* (i.e., functions of the free parameter), according to the formula (ω represents the free parameter)

$$\text{scaled\_valued}(\omega) = \frac{\text{raw\_value}(\omega) \ - \text{good\_value}(\omega)}{\text{bad\_value}(\omega) - \text{good\_value}(\omega)}$$

To complete this overview of the problem formulation, it remains to mention the concept of *nominal variation.* This is one of the means by which the optimization process can take advantage of the designer's "expert" knowledge. For each design parameter, the designer is requested to provide, besides an initial guess, a quantity indicating his degree of confidence in this guess, i.e., how close he believes it may be away from the optimal value. This nominal variation should be selected according to the *uniform parameter influence rule*: a modification of any design parameter by an amount equal to its nominal variation should influence the most binding objectives and constraints to roughly the same degree. Admittedly, this rule is not intuitive. Thus it is suggested to choose as nominal variation for a given design parameter the difference (in absolute value) between the initial guess and the next value the designer would try if he had to proceed "by hand". The nominal variations are used for the initial scaling of the parameter space according to the formula

$$\text{scaled\_value} = \frac{\text{raw\_value}}{\text{nominal\_variation}} .$$

The penalty for an improper choice of the nominal variation is slower initial convergence of the optimization process. (As explained below, faster convergence can be recovered after a few iterations by the use of "automatic" scaling.) If no nominal variation is provided, the default value of 1 (no scaling) will be used.

Interactive Refinement of Specifications

It has often been claimed that any manual intervention of the user in an optimization process is unnecessary if the problem is correctly posed and the optimization algorithm sufficiently sophisticated. In this subsection, we argue that, in an engineering design environment, interaction between user and optimization process is generally crucial as a mean for the designer to progressively refine the description of his optimality criterion, locally as increasingly better approximate solutions are approached.

Consider, to simplify matters, a problem involving only objectives and hard constraints, i.e., a problem of the form

$$\begin{aligned} \min_{x} \quad & f^k(x) \quad \forall k \\ \text{s. t.} \quad & g^j(x) \leq 0 \quad \forall j \end{aligned} \tag{1}$$

where it remains to specify what is meant by concurrently minimizing several functions. At this point, the choice of a minimax formulation for (1) may seem arbitrary. To investigate this, let us consider the well known concept of *Pareto optimality* (see, e.g., [28,29]).

Definition. A point $x^* \in \mathbb{R}^n$ is a *local Pareto point* (or *noninferior point*) with respect to the objective functions f^k and constraints g^j if, for every j, $g^j(x^*) \leq 0$, and there exists a positive ρ such that, for any $x \in B(x^*, \rho)$,

$$f^i(x) < f^i(x^*) \quad \text{for some } i$$

implies that either

$$f^k(x) > f^k(x^*) \quad \text{for some } k$$

or

$$g^j(x) > 0 \quad \text{for some } j.$$

□

Clearly, any point $x \in \mathbb{R}^n$ which is not a Pareto point cannot be considered optimal.

The following simple fact (straightforward extension of Theorem 13.2 of [28]) relates the concept of Pareto point to constrained minimax optimization.

Fact. Suppose x^* is a local Pareto point with respect to the objective functions f^k and constraints g^j. Then there exists (nonunique) scalars α^k and β^k such that x^* is a local solution for the constrained minimax problem

$$\min_x \quad \max_k \{ \frac{f^k(x) - \alpha_k}{\beta_k} \mid g^j(x) \leq 0 \quad \forall j \} .$$

□

In terms of the design methodology sketched earlier, this means that, no matter which Pareto point x^* the designer feels is optimal, there exists a set of good and bad values such that the corresponding scaled minimax problem will have x^* as a local solution. Yet, to the designer, the design problem may not be equivalent to a constrained minimax problem in any obvious way. Thus the originally chosen good and bad values will likely not yield the desired Pareto point. The designer may then wish to *interactively adjust* these good and bad values and curves, guided by their application-related interpretation, so as to "steer" the minimax solution to the desired solution. The alternative would be for the designer to provide at the outset a complete description of the relative merits of the various Pareto

points of the multiobjective optimization problem, in most cases a prohibitively complex task.

Such interactive specification refinement is an essential aspect of the methodology of [3,4]. Two interactive displays are introduced there, that help the user in assessing the current design as well as the possible tradeoffs. The Pcomb display (for performance comb, see [3]) indicates how the current design is performing with respect to the assigned good and bad values or curves corresponding to the various specifications (see examples in a latter section of this paper). The Ecomb display (for estimation comb, see [4]) provides information on the sensitivity of the minimax solution to changes in the various goods and bad values of the currently competing specifications.

NUMERICAL OPTIMIZATION TECHNIQUES

In the previous section, it was suggested that a design problem can often be formulated as a constrained minimax problem of the form

$$\min_{x} \quad \max_{k} f^k(x) \quad \text{s.t.} \quad g^j(x) \leq 0, \quad \forall j \tag{2}$$

or rather as a sequence of such problems. (The meaning of the f^k's and g^j's varies according to the "phase" of the optimization process and their scaling is subject to interactive adjustment.) In this section, we first point out a few distinctive features of such problems in an engineering design context. We then examine in some more details how two of these features can be dealt with.

The first issue to be addressed is thus that of solving optimization problems such as (2). While (2) is readily transcribed into the constrained minimization problem

$$\begin{aligned} \min_{x,\,\xi} \quad & \xi \quad \text{s.t.} \quad f^k(x) - \xi \leq 0 \quad \forall k \\ & \qquad\quad g^j(x) \leq 0 \qquad\quad \forall j \end{aligned} \tag{3}$$

where ξ is an additional scalar variable, it would likely be ill-advised to tackle (3) using a standard constrained minimization code. Yet there is no major conceptual difficulty in adapting many constrained minimization algorithms to take advantage of the particular structure of (3). For the case of smooth objective and constraints, such extensions have been proposed, e.g., in [30,31] and [32].

A more difficult challenge is that of accounting for the fact that, in an engineering system context, objective and constraint function values are often expensive to compute with high accuracy. This is of particular concern since execution of an optimization algorithm typically involves many such function evaluations. Such is the case, among other instances, when function evaluations involve numerical integration. An appealing idea would be to perform function evaluations with limited, even poor accuracy, while away from a solution of the optimization problem, and to progressively increase the accuracy as a solution is approached. The culprit is that initial low accuracy may prevent progress toward a solution. This question has been addressed in [5] and more recently in [6,7] in the context of steepest descent for unconstrained optimization, via the concept of "accumulated error". The issue is far from resolved in the general case. Much attention has been devoted to a particular case of this question: that in which some function evaluations entail a global maximization with respect to an auxiliary variable, i.e., problems involving functions of the type

$$\psi(x) = \max_{\omega \in \Omega} \ \phi(x, \omega)$$

where Ω is, say, a compact set. These problems are often referred to as semi-infinite. They occur frequently in the context of system design, where ω may indicate, e.g., time, frequency or modeling error. Unless global properties such as convexity are present, semi-infinite problems soon become untractable if $\Omega \subset \mathbb{R}^p$ with p more than a small integer. A practically important case, however, is that in which Ω is an interval of the real line (e.g., time or frequency range).

Next, it is clear that tradeoff exploration cannot meaningfully take place unless all hard constraints are satisfied. It is thus necessary that all hard constraints be satisfied after a few iterations and *remain satisfied thereafter.* It is also desirable that the design obtained after each iteration improve on the previous one. In terms of problem (2) one would want algorithms that construct sequences $\{x_i\}$ satisfying

$$g^j(x_i) \leq 0, \quad \forall j$$

and

$$\max_k f^k(x_{i+1}) < \max_k f^k(x_i).$$

Most popular optimization algorithms do not satisfy these requirements (even in the case of a single objective).

An important aspect of many control system design problems is that they include constraints on the largest (or smallest) eigenvalue or singular value of certain matrices. Such constraints are typically used for achieving certain desirable stability or robustness properties. They are nondifferentiable, but their specific structure can be advantageously exploited. Corresponding algorithms have been proposed in [15,17,21] and, more recently, in [18,19] and in [16].

In the remainder of this section, we discuss in some more detail two of the issues identified above: dealing with functional constraints and generating feasible iterates at which the objective function is monotonically improving.

Accommodating Functional Constraints

Algorithms for semi-infinite optimization have been proposed by many authors both in the linear case and in the general, nonlinear case (see, e.g., [8,9,33–40] and other papers in [41]). However most of the existing methods are not readily implementable. Indeed, while most approaches make use, at each iteration, of a set of local maximizers over the range of the independent parameter, the question of suitably approximating such maximizers is generally left aside. A notable exception is found in the work of Polak and co-workers [8, 9,20] who propose schemes based on an adaptively refined discretization of the interval of variation of the independent parameter. The algorithm in [8] makes use at each iteration of the gradients at all ϵ-active points, i.e., all mesh points ω at which $\phi(x_i,\omega) \geq \psi(x_i) - \epsilon$, where ϵ is a certain strictly positive number. The discretization is then progressively refined and the corresponding problem is solved to a progressively better accuracy. Convergence to stationary points is proven. The algorithm proposed in [9] achieves substantial computational savings by better exploiting the regularity properties of ϕ as a function of ω. The key observation is that, when the discretization is fine enough, the critical sensitivity information is essentially carried by the gradients of ϕ at the ϵ-active local maximizers. The proposed algorithm makes use of the latter only. It has been pointed out, however, that for a given mesh size, the gradients at the local maximizers for the current iterate do not always carry enough information on the local behavior of the constraints, and that convergence to non-stationary points can result [42]. The algorithm below, largely inspired from that in [9] avoids this drawback, by taking into account, in the computation of the search direction, values of ω that are not local maximizers at the current iterate.

For the sake of exposition, consider the simple semi-infinite problem involving a single functional objective and no constraints

$$\min_{x} \quad \psi(x) \tag{4}$$

where the function ψ: $\mathbb{R}^n \to \mathbb{R}$ is defined by

$$\psi(x) = \max_{\omega \in [0,1]} \phi(x, \omega).$$

Given a *discretization index* $q \in \mathbb{N}\backslash\{0\}$, the interval $[0,1]$ of variation of the independent parameter is discretized into

$$\Omega_q = \{\omega = \hat{\ell}/q, \quad \hat{\ell} \in \{0, 1, \ldots, q\}\} \tag{5}$$

and Problem (4) is approximated accordingly by

$$\min_{x} \quad \psi_q(x)$$

where

$$\psi_q(x) = \max_{\omega \in \Omega_q} \phi(x, \omega).$$

Given a precision $\epsilon > 0$ and a point $x \in \mathbb{R}^n$, the set $\Omega_{q,\epsilon}(x)$ of *ϵ-active local maximizers* at x is defined as the set of mesh points $\hat{\omega} \in \Omega_q$ such that $\phi(x, \hat{\omega}) \geq \psi_q(x) - \epsilon$, and such that $\phi(x, \hat{\omega}) \geq \phi(x, \omega)$ whenever ω is adjacent to $\hat{\omega}$ in Ω_q.

In an attempt to reduce the computational overhead, we compute a search direction based on gradients evaluated at a small critical subset $\hat{\Omega}_i$ of Ω_q. As mentioned earlier, selecting as $\hat{\Omega}_i$ the set $\Omega_{q,\epsilon}(x_i)$ at the current iterate x_i may lead to possible "jamming" at "corners" of ψ_q. To circumvent this difficulty, we include in $\hat{\Omega}_i$ the global maximizers ω at the last *unsuccessful* trial point of the previous line search, as well as certain critical values of ω used at recent iterations (see Step 5). As in [9], when a stationary point for the current discretization is approached, the discretization is refined (Step 6). We denote by X_k the major iterates, updated every time the discretization is refined (outer loop) and by x_i the subiterates, constructed at a fixed discretization level (inner loop).

Algorithm.

Parameters. $\delta > 0$; $\alpha,\ \beta \in (0,1)$; $\epsilon_0 > 0$; $N_0 > 0$; $q_0 \in \mathbb{N}\backslash\{0\}$.

Data. $X_0 \in \mathbb{R}^n$.

Step 0. Initialization of the outer loop. Set $k = 0,\ \epsilon = \epsilon_0,\ N = N_0,\ q = q_0$.

Step 1. Initialization of the inner loop. Set $i = 0,\ x_0 = X_k,\ \hat{\Omega}_0 = \Omega_{q,\epsilon}(x_0)$.

Step 2. Search direction computation. Compute (d_i, v_i), solution of the quadratic program in (d, v),

$$\begin{aligned} \min_{d} \quad & \frac{1}{2}\|d\|^2 + v \\ \text{s. t. } & \phi(x_i,\omega) + \langle \nabla_x \phi(x_i,\omega), d\rangle \leq v \quad \forall \omega \in \hat{\Omega}_i. \end{aligned}$$

Let τ_i be the optimal value of this quadratic program and let $\lambda_i(\omega),\ \omega \in \hat{\Omega}_i$ be the corresponding multipliers.

Step 3. Optimality test. If $\tau_i \geq -\delta\epsilon$ or $\|x_i\| > N$, go to Step 6. Otherwise, go to Step 4.

Step 4. Line search. Let t_i be the first number t in the sequence $\{1, \beta, \beta^2, \cdots\}$ satisfying

$$\psi_q(x_i + t d_i) - \psi_q(x_i) \leq -\alpha t \delta \epsilon.$$

Step 5. Updates. Set $x_{i+1} = x_i + t_i d_i$ and

$$\hat{\Omega}_{i+1} = \Omega_{q,\epsilon}(x_{i+1}) \cup \{\omega \in \hat{\Omega}_i \mid \lambda_i(\omega) \neq 0\} \cup \Omega_{q,0}(x_i + \beta^{-1} t_i d_i)$$

if $t_i < 1$ and

$$\hat{\Omega}_{i+1} = \Omega_{q,\epsilon}(x_{i+1}) \cup \{\omega \in \hat{\Omega}_i \mid \lambda_i(\omega) \neq 0\}$$

if $t_i = 1$.

Step 6. Discretization refinement. If $\tau_i \geq -\delta\epsilon$, set $\epsilon = \epsilon/2$. If $\|x_i\| > N$, set $N = 2\|x_i\|$. Set $q = 2q$. Set $X_{k+1} = x_i$. Set $k = k + 1$ and go back to Step 1.

□

It can be shown that, under mild assumptions, the sequence $\{X_k\}$ constructed by this algorithm is infinite and that its accumulation points are stationary points for (4).

This algorithm can easily be extended to problems involving multiple functional objectives and constraints (see [42] for the case of a single nonfunctional objective and a single constraint).

Often the designer has some knowledge of the general "shape" of ϕ as a function of ω. In such case it may be advantageous to replace discretization (5) by an appropriate nonuniform discretization, e.g., logarithmically spaced mesh points or higher density of mesh points around "critical" areas. Provided the discretization still grows dense as k tends to infinity, the convergence properties will remain unchanged. Using a dynamically adapted discretization is much more delicate, as it may result in oscillations (see [43] for a heuristic scheme).

The algorithm described above typically exhibits a relatively slow rate of convergence. Superlinearly convergent algorithms for handling functional constraints have been proposed in the literature see, e.g., [33,37,39,40,44,45]. These methods are based on the fact that, locally, many semi-infinite optimization problems are equivalent to problems with finitely many constraints obtained at the local maximizers of the functional specification. They usually combine techniques for identifying these local maximizers with locally superlinearly convergent algorithms for problems with finitely many constraints. Most of these methods may only be applied under some restrictive conditions [33,39,40,44] or are local, in the sense that convergence is ensured only if the initial iterate is close enough to the solution [37,45]. Globalization schemes have been attempted (see, e.g., [34,35,38]). Typically, a solution is approached using a first order method and, close to that solution, a second order method is used. Nevertheless, it is not clear at this point, when to switch from the first to the second order method. Oscillations between the first and second order method may result if a poor switching mechanism is used, slowing down the overall convergence.

Generating Feasible Iterates

It was pointed out above that, when interactively using optimization techniques to solve design problems, it is often very desirable that some critical constraints be satisfied at each iteration, and that the overall objective function monotonically decrease from iteration to iteration. First order methods of feasible directions [12,14] enjoy such properties and have been used extensively in optimization-based design (see e.g., [1,2]). An important drawback of these methods, however, is their slow rate of convergence (linear at best). The first of the two properties sought could be almost satisfied by other existing algorithms if the corresponding constraints were scaled so as to considerably amplify constraint violations. This, however, would have the undesirable side effect of introducing ill-conditioning. In

[46] and [47], we proposed algorithms that satisfy the above mentioned properties while exhibiting superlinear convergence. The algorithm in [46] makes use of a modified Successive Quadratic Programming (SQP) iteration. The algorithm in [47], while still related to SQP, does not involve the solution of any quadratic program. Both of these algorithms are of the quasi-Newton type and thus tend, after a few iterations, to optimally scale the design parameters; this complements the initial scaling based on the designer's directives (nominal variations). While, as presented, these algorithms can only handle problems involving a single objective and constraints that are all hard (and non-functional), there is little conceptual difficulty in extending them to our more general framework (see, e.g., [32]). In this subsection we briefly sketch a simple algorithm, based on ideas from both [46] and [47].

Consider the problem

$$\min_{x} \; f(x) \quad \text{s.t.} \quad g^j(x) \leq 0, \quad \forall j \tag{6}$$

where the functions $f: \; \mathbb{R}^n \to \mathbb{R}$ and $g^j: \; \mathbb{R}^n \to \mathbb{R}$ are smooth.

Algorithm.

Parameters. $\nu > 1$; $\tau \in (2,3)$; $\alpha, \; \beta \in (0,1)$.

Data. $x_0 \in \{x \mid g(x) \leq 0\}$, $H_0 \in \mathbb{R}^{n \times n}$, symmetric positive definite.

Step 0. Initialization. Set $i = 0$.

Step 1. Computation of a search arc.

i. Compute d_i^0 by solving the quadratic program

$$\begin{aligned} \min_{d} \quad & \frac{1}{2}\langle d, H_i d\rangle + \langle \nabla f(x_i), d\rangle \\ \text{s.t.} \quad & g^j(x_i) + \langle \nabla g^j(x_i), d\rangle \leq 0, \quad \forall j \end{aligned}$$

If $d_i^0 = 0$ stop.

ii. Compute d_i^1 by solving the linear program

$$\begin{aligned} \min_{d} \quad & \max\{\langle \nabla f(x_i), d\rangle; \; \max_j g^j(x_i) + \langle \nabla g^j(x_i), d\rangle\} \\ \text{s.t.} \quad & \|d\|_\infty \leq 1. \end{aligned}$$

iii. Set $d_i = (1 - \rho_i) d_i^0 + \rho_i d_i^1$, with $\rho_i = \min\left(\frac{1}{2}, \|d_i^0\|^\nu\right)$.

iv. Compute $\tilde{d}_i$ by solving the linear least squares problem

$$\begin{aligned} \min_{d} \quad & \frac{1}{2}||d||^2 \\ \text{s.t.} \quad & g^j(x_i + d_i) + \langle \nabla g^j(x_i), d \rangle = -||d_i^0||^\tau, \quad \forall j \in I_i \end{aligned}$$

where

$$I_i = \{j \mid g^j(x_i) + \langle \nabla g^j(x_i), d_i^0 \rangle = 0\}.$$

If there is no solution or if $||\tilde{d}_i|| > ||d_i||$, set $\tilde{d}_i = 0$.

Step 2. Arc search. Compute t_i, the first number t in the sequence $\{1, \beta, \beta^2, \cdots\}$ satisfying

$$f(x_i + td_i + t^2\tilde{d}_i) \leq f(x_i) + \alpha t \langle \nabla f(x_i), d_i \rangle$$

$$g^j(x_i + td_i + t^2\tilde{d}_i) \leq 0, \quad \forall j.$$

Step 3. Updates. Compute a new symmetric positive definite approximation H_{i+1} to the Hessian matrix. Set $x_{i+1} = x_i + t_i d_i + t_i^2 \tilde{d}_i$. Set $i = i + 1$. Go back to Step 1.

□

The idea behind the computation of the search arc is as follows. Starting from a feasible point x_i, first obtain the standard SQP direction d_i^0. This can be shown to be a direction of descent for f but, if x_i is on the boundary of the feasible set, d_i^0 may be tangent to this boundary and not be a feasible direction. Thus, one then computes a "first order" feasible direction d_i^1 which is also a descent direction for f. Direction d_i is a convex combination of d_i^0 and d_i^1, thus always a feasible descent direction, which becomes arbitrarily close to the SQP direction d_i^0 as a Kuhn-Tucker point is approached. Finally a correction $\tilde{d}_i$ is computed to ensure feasibility of $x_i + d_i + \tilde{d}_i$ close to a Kuhn-Tucker point, thus avoiding any Maratos-like effect. The condition $\tau \in (2,3)$ is needed for this to hold without losing the superlinear convergence property of the SQP iteration. The "linear" search is then performed along the arc $x_i + td_i + t^2\tilde{d}_i$, i.e., $x_i + d_i + \tilde{d}_i$ is attempted first but for small t the search direction is close to d_i.

Under mild assumptions it can be shown that the sequence $\{x_i\}$ constructed by this algorithm converges superlinearly to a Kuhn-Tucker point of (6).

EXAMPLE OF IMPLEMENTATION: CONSOLE

Introduction

A designer using an optimization-based CAD package typically goes through two phases while designing a system. In the first phase, he formulates the problem; some time is spent checking and debugging this problem formulation. In the second phase the solution of the problem is sought. The first phase is very comparable to writing a classic C or FORTRAN program and debugging it. During the second phase the user-package interaction is more thorough and qualitatively different, typically involving graphics. Accordingly, CONSOLE is composed of two main programs: CONVERT and SOLVE. Figure 1 illustrates the structure of CONSOLE. The Problem Description File is the input to CONVERT, which checks the description for all possible syntax errors and some logic errors and then generates two files. One of these files is a data file which contains, among other things, the names of design parameters and specifications, as well as *good* and *bad* values or curves to be used for tradeoff exploration. The other file is an object file. It contains a compiled version of the various specifications (objectives, functional objectives, constraints and functional constraints). Both of these files are input to SOLVE, together with any object files required by a simulator (or simulators) the user wishes to use. SOLVE then iterates together with the user to obtain a solution with maximum or sufficient satisfaction.

The range of problems that can be solved efficiently using a CAD tool depends very much on the ability of this tool to be interfaced with user-supplied simulators. For instance, when designing a control system one makes use of the characteristics of the plant, and therefore a model of the plant under study has to be made available to the CAD tool. CONSOLE allows for an easy interfacing of almost any simulator written in C, FORTRAN or PASCAL, the user has available.

CONSOLE borrows many ideas from previously developed optimization-based CAD packages. In particular, the problem description language and the interactive command language are strongly inspired from those of DELIGHT.SPICE [48] and DELIGHT.MaryLin [22] (see also [25,26]).

Example Design: a Copolymerization Reactor Controller

In this subsection we demonstrate the operation of CONSOLE by means of an example: the optimal selection of the temperature and feed flow rate profiles in a copolymerization

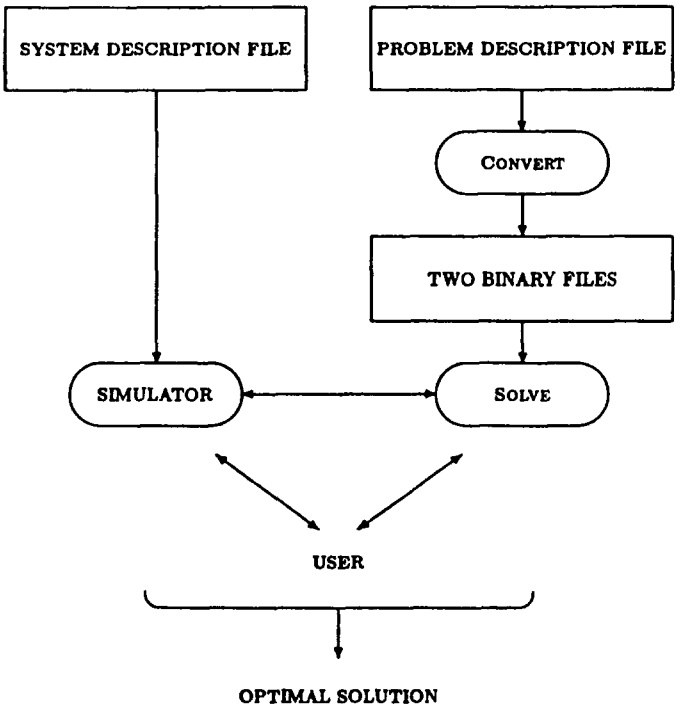

Figure 1: Structure of the CONSOLE CAD tandem

reactor. We divide the discussion into three parts. First we give some details on the reactor. Then we discuss the problem description. Finally, we demonstrate how SOLVE can be used on this problem. For a detailed discussion, the reader is referred to [49].

The copolymerization reactor

Polymerization is a group of chemical chain reactions that links a large number (more than a thousand) of "monomer" molecules together. The product, called polymer, can consist of long strings, entangled in each other, but can also be branched or even have a three dimensional structure. Copolymerization is polymerization using two different monomers at the same time. The group of materials popularly identified as *plastics* mainly consists of polymers and copolymers with a carbon skeleton. Plastics show a vast variety of physical properties (strength, color, elasticity, etc.). The giant polymer molecule structure and the location of the chemically active groups on this molecule determine the properties of the product.

It is important for the product to have the correct properties. Products that are even only slightly *off-spec* will be refused by the clients and are absolutely worthless. Therefore designing a control strategy to achieve the appropriate properties is necessary. In this example the objective of obtaining a certain molecular copolymer structure will be simplified to that of obtaining a certain molecular weight and a given ratio of both monomers in the product. The molecular weight is a measure of the average number of monomer molecules chained together in each copolymer molecule. The ratio of both monomers in the product is often referred to as the copolymer composition. The manipulated variables are the feed flowrate and the reactor temperature which are both functions of time. The selection of a suitable profile for those quantities will be performed using CONSOLE based on a certain model of the reactor in the form of a simulator: *copoly*. The latter, written in FORTRAN, simulates an existing copolymerization reactor which is part of the experimental equipment of the Department of Chemical and Nuclear Engineering of the University of Maryland at College Park. It predates CONSOLE.

Problem description. We give here (piece by piece) the entire contents of the Problem Description File. First, the time span of interest and the distance between two consecutive time points at which the simulation outputs should be recorded are specified by

```
final_time = 5.0      /* in hours */
mesh_size = 0.25      /* in hours */
```

Then, the design parameters need to be declared. CONSO-LE cannot solve problems with infinite dimensional parameter space. Therefore, an approximate problem is solved instead. Namely, it is assumed that the temperature and feed flowrate profiles are polynomial in time. This will result in a suboptimal solution. However, if the order of the polynomials is sufficient, the performance obtained will be very close to the true optimal. Here, temperature and feed flowrate are expressed as

$$\text{temperature } = a_1 + a_2 t + a_3 t^2 + a_4 t^3$$

and

$$\text{feed flowrate } = b_1 + b_2 t + b_3 t^2 + b_4 t^3.$$

The design parameters are the a_i's and b_i's for $i = 1, 2, 3, 4$.

```
design_parameter a1  init=333    variation=5.0    min=323.0   max=368.0
design_parameter a2  init=10     variation=1.0    min=-10.0   max=10.0
design_parameter a3  init=0.1    variation=0.01   min=-1.0    max=1.0
design_parameter a4  init=0.1    variation=0.01   min=-0.1    max=0.1
design_parameter b1  init=20     variation=1.0    min=5.0     max=50.0
design_parameter b2  init=-1     variation=0.1    min=-2.0    max=2.0
design_parameter b3  init=-0.1   variation=0.01   min=-0.2    max=0.2
design_parameter b4  init=-0.01  variation=0.001  min=-0.05   max=0.05
```

The initial values, variations, minimum and maximum values of the design parameters are obtained either from a previous design or from physical considerations (for temperature expressed in degree K and feed flowrate expressed in liters/minute).

Every call to the simulator *copoly* will be made through an interface routine *interface*. This routine has many input arguments (among which the design parameters) and many output arguments (simulation outputs). As the calling sequence is practically identical for many specifications, the following define will be used.

```
define call_interface  "\
   import a1,a2,a3,a4,b1,b2,b3,b4;                                    \
   import final_time, mesh_size;                                      \
                                                                      \
   double molecular_weight_t,  /* molecular weight at time 'time' */ \
          composition_t,       /* composition at time 'time' */      \
          final_volume;        /* final volume */                    \
                                                                      \
   interface (a1,a2,a3,a4,b1,b2,b3,b4,     /* input arguments */     \
              final_time,time,mesh_size,                              \
              &molecular_weight_t,         /* output arguments */    \
              &composition_t,                                         \
              &final_volume)"
```

There will be two design objectives, both functional: the maximum (over time) square deviation from the desired molecular weight and the maximum (over time) square deviation from the desired composition should both be made as small as possible. The former is required only after transients have died out. The formulation is as follows.

```
desired_molecular_weight = 3e4
bad_Mn_sqerr = 5e3**2
desired_molecular_weight = 3e4
functional_objective "(Mn-Mns)^2"   /* square of error in molecular weight */
   for time from 3 to final_time by mesh_size
   minimize {
      double diff;
      import desired_molecular_weight;
      import call_interface;   /* so that the define can be used here */

      call_interface;
      diff = molecular_weight_t - desired_molecular_weight;
      return (diff*diff);
      }
   good_curve = {return 0.0;}
   bad_curve  = {import bad_Mn_sqerr;
                 return bad_Mn_sqerr;}
```

```
desired_composition = 1  /* proportion of monomers 1 and 2 */
bad_CC_sqerr = 0.225**2
functional_objective "(CC-CCs)^2" /* square of error in copolymer composition */
   for time from 0 to final_time by mesh_size
   minimize {
      double diff;
      import desired_composition;
      import call_interface;

      call_interface;
      diff = composition_t - desired_composition;
      return (diff*diff);
      }
   good_curve = {return 0.0;}
   bad_curve  = {import bad_CC_sqerr;
                 return bad_CC_sqerr;}
```

The volume of product at the end of the batch is limited by the size of the tank. This is expressed by the following design constraint.

```
constraint "final volume" hard
   {  import call_interface;
      double time;          /* here time is dummy; but it must be declared */

      call_interface;
      return final_volume;
      }
   <= good_value = 4.0  /* liters */
      bad_value  = 4.1  /* liters */
```

Constraints on operating variables are necessary for safety and reasonable operation of the reactor. Based on the requirements for reaction rate, heat transfer limitations and

reactor safety, upper and lower bounds on reactor temperature are defined as constraints. Similarly, in order to avoid negative flowrate and to limit the maximum flowrate which can be handled by the reactor system, upper and lower bounds on feed flowrate are defined as constraints. Since the constraints are time dependent, they are given as functional constraints as follows.

```
functional_constraint "upper temperature" hard  /* upper bound on temperature */
   for time from 0 to final_time by mesh_size
   {  import a1,a2,a3,a4;
      return a1+a2*time+a3*pow(time,2.0)+a4*pow(time,3.0);
      }
   <= good_curve = {return 363.0;}  /* degrees K */
      bad_curve  = {return 364.0;}  /* degrees K */
```

```
functional_constraint "lower temperature" hard  /* lower bound on temperature */
   for time from 0 to final_time by mesh_size
   {  import a1,a2,a3,a4;
      return a1+a2*time+a3*pow(time,2.0)+a4*pow(time,3.0);
      }
   >= good_curve = {return 328.0;}  /* degrees K */
      bad_curve  = {return 323.0;}  /* degrees K */
```

```
functional_constraint "upper flowrate" hard   /* upper bound on feed flowrate */
   for time from 0 to final_time by mesh_size
   {  import b1,b2,b3,b4;
      return b1+b2*time+b3*pow(time,2.0)+b4*pow(time,3.0);
      }
   <= good_curve = {return 0.07;}   /* liters per minute */
      bad_curve  = {return 0.075;}  /* liters per minute */
```

```
functional_constraint "lower flowrate" hard   /* lower bound on feed flowrate */
   for time from 3 to final_time by mesh_size
   {  import b1,b2,b3,b4;
      return b1+b2*time+b3*pow(time,2.0)+b4*pow(time,3.0);
      }
   >= good_curve = {return 0.0;}    /* liters per minute */
      bad_curve  = {return -0.005;} /* liters per minute */
```

This completes the Problem Description File.

Coupling CONSOLE with a simulator typically requires one or more interface routines. Clearly, these interface routines should not be problem dependent, but merely simulator dependent. They may have to be quite sophisticated. This is so, among other instances, when the simulator itself is interactive, or when it needs to be initialized or reset at ap-

propriate times. Also, it is often convenient that values of design parameters be passed "automatically" to the simulator whenever they are modified. CONSOLE includes an interfacing scheme that makes such tasks relatively straightforward. For our purpose, the simulator *copoly* can be viewed as "unsophisticated". Thus the interface used here acts as a mere buffer, avoiding redundant calls to *copoly*.

Running the problem. Let "copoly-design" be the name of the Problem Description File listed above. One first "compiles" this file by typing

```
convert copoly-design
```

which prompts the output

```
Welcome to CONVERT, Version 1.0 (Released 9/1/87)

                    Copyright (c) 1987, 1988
                              by
  Michael K.H. Fan, Li-Sheng Wang, Jan Koninckx, and Andre L. Tits
                      All Rights Reserved.

[processing copoly-design]
[compiling copoly-design.c]
[writing copoly-design.d]
```

Then SOLVE is invoked

```
solve copoly-design copoly.o interface.o
```

yielding

```
Welcome to SOLVE, Version 1.0 (Released 9/1/87)

                    Copyright (c) 1987, 1988
                              by
  Michael K.H. Fan, Li-Sheng Wang, Jan Koninckx, and Andre L. Tits
                      All Rights Reserved.

[loading/reading copoly-design.o and ...]
[reading copoly-design.d]
[calling simulator initialization (if any)]
[calling problem initialization (if any)]

type "help" for help
type "help info" for information
type "help news" for local news

<0>
```

Issuing the command

```
run 0 print pcomb
```

displays the initial values and the corresponding Pcomb

```
Name        Value         Variation  wrt 0   Prev   Iter=0
a1          3.33000e+02   5.0e+00
a2          1.00000e+01   1.0e+00
a3          1.00000e-01   1.0e-02
a4          1.00000e-01   1.0e-02
b1          2.00000e+01   1.0e+00
b2         -1.00000e+00   1.0e-01
b3         -1.00000e-01   1.0e-02
b4         -1.00000e-02   1.0e-03
```

```
Pcomb (Iter=  0) (Phase 1) (MAX_HARD= 35)

SPECIFICATION     PRESENT   GOOD                G          B              BAD
FO1  (MN-MNs)^2   1.57e+08  0.00e+00 ================================>  2.50e+07
FO2  (CC-CCs)^2   4.75e-02  0.00e+00 ====================*|              5.06e-02
C1   final vol    5.91e+00  4.00e+00 -------------------------------->  4.10e+00
FC1  upper temp   3.98e+02  3.63e+02 -------------------------------->  3.64e+02
FC2  lower temp   3.33e+02  3.28e+02 *--------------------------------  3.23e+02
FC3  upper flow   2.00e-02  7.00e-02 <--        |          |            7.50e-02
FC4  lower flow   1.12e-02  0.00e+00 <---------------------------------- -5.00e-03
```

The Pcomb (see METHODOLOGY section above) shows that, for the given initial values of the design parameters, the molecular weight (FO1), copolymer composition (FO2), final volume of the product (C1) and temperature (FC1) are all unsatisfactory (see [3] for details). Plots of the functional specifications can be also seen by the *plot* command. Typing

```
run 22
```

causes SOLVE to run 22 iterations of the optimization algorithm (this may take some time ...), until finally the prompt

```
<22>
```

appears on the screen. The command

```
pcomb
```

yields

```
Pcomb (Iter=  22) (Phase 2) (MAX_COST_SOFT= 0.0766327)

SPECIFICATION    PRESENT   GOOD              G          B            BAD
FO1  (MN-MNs)^2  1.92e+06  0.00e+00 ==========*         |           2.50e+07
FO2  (CC-CCs)^2  3.88e-03  0.00e+00 ==========*         |           5.06e-02
C1   final vol   3.47e+00  4.00e+00 <--       |         |           4.10e+00
FC1  upper temp  3.53e+02  3.63e+02 <--       |         |           3.64e+02
FC2  lower temp  3.45e+02  3.28e+02 <---------------------------- 3.23e+02
FC3  upper flow  9.70e-03  7.00e-02 <--       |         |           7.50e-02
FC4  lower flow  6.00e-03  0.00e+00 <---------------------------- -5.00e-03
```

All hard constraints are now satisfied. Objectives FO1 and FO2 have almost reached their good values and seem to be competing against each other. If, in the designer's opinion, the current molecular weight is acceptable while a composition closer to the desired would be much preferred, a suitable action would now be to type

```
setgb FO1 = 2e6, 2.5e7
```

setting the good and bad curves for FO1 to the constraints 2×10^6 and 2.5×10^7, respectively. After a few more iterations, e.g.,

```
run 5
```

the designer may be satisfied. He would then type

```
print
```

to see the "optimal" design parameters

```
Name        Value          Variation    wrt 0     Prev    Iter=27
a1           3.53188e+02   5.0e+00        6%       0%
a2          -3.19240e+00   1.0e+00      -68%       0%
a3           9.13515e-02   1.0e-02        9%       0%
a4           5.31340e-02   1.0e-02       47%       0%
b1           9.69793e+00   1.0e+00       52%       0%
b2          -3.74885e-01   1.0e-01       63%       0%
b3          -4.07314e-02   1.0e-02       59%       0%
b4          -6.40500e-03   1.0e-03       36%       0%
```

then

```
store "copoly_optimal_design_parameters"
```

to save them and finally

```
quit
```

Some Other Applications

CONSOLE has already been used successfully in many other applications, including the design of controllers for a flexible arm [50] and a robotic manipulator [51] and the solution of a parameter selection problem for a neural network [52] (all under P.S. Krishnaprasad at the Univ. of Maryland at College Park), the design of an RC controller for a radar antenna (under F. Emad at the Univ. of Maryland at College Park), and the design of power filters [53] (at the Westinghouse Defense and Electronics Center). In the case of the neural network application, CONSOLE was coupled to the nonlinear system simulator SIMNON [54] by means of a sophisticated interface (see [55] for details).

Future Enhancements

CONSOLE was developed within a time span of a few months only. While it has been used on many different types of design problems and has given very satisfactory results, there is room for improvement, both in aspects of the user interface and in computational aspects. Here we mention enhancements that are currently being implemented or are planned for the near future.

In the current version, interaction is essentially alphanumerical or pseudo-graphical (Pcomb). A fully graphical Pcomb display, similar to the one suggested in [3] and already available in DELIGHT.MaryLin [22] and DELIGHT.SPICE [48] is currently being implemented. The CONSOLE implementation will take full advantage of a window type environment. Also being implemented is the Ecomb display [4], a display that provides in-

formation on the sensitivity of the overall design to changes in good and bad values of the various specifications. Development of tools for graphical input (e.g., to modify a good/bad curve) is being undertaken.

The optimization algorithm implemented in the current version of SOLVE makes use of a first order feasible direction algorithm with fixed discretization. Implementation of faster algorithms, including schemes such as those discussed earlier in this paper, is underway. Also, schemes are being investigated for taking advantage of the possible capability of simulators to efficiently compute gradients (currently, gradients are approximated by finite differences). One such scheme has been proposed in [56].

CONCLUDING REMARKS

While application of optimization techniques to system design appears to be full of promises, it should not be considered a panacea. One important limitation of optimization techniques is that in the absence of convexity properties, they often yield a local rather than global optimum. Yet, even in such cases, optimization is valuable if merely considered as a tool for improving on a given design. It is also essential to make use of other available application-specific design tools whenever they are available. In fact, by a clever choice of design parameters, it is often possible to use optimization concurrently with such tools. For example, following [57], one can combine numerical optimization with the technique of Q-parametrization of all stabilizing compensators. One would, e.g., express the Q matrix as

$$Q = \sum_{i=1}^{n} x_i Q_i$$

where the Q_i's are suitably selected, and use the x_i's as design parameters.

REFERENCES

[1] E. Polak, D.Q. Mayne & D.M. Stimler, "Control System Design via Semi-Infinite Optimization: A Review," *IEEE Proc.* 72 (1984), 1777–1794.

[2] R.K. Brayton, G.D. Hachtel & A.L. Sangiovanni-Vincentelli, "A Survey of Optimization Techniques for Integrated Circuit Design," *IEEE Proc.* 69 (1981), 1334–1362.

[3] W.T. Nye & A.L. Tits, "An Application-Oriented, Optimization-Based Methodology for Interactive Design of Engineering Systems," *Internat. J. Control* 43 (1986), 1693–1721.

[4] A.L. Tits & Z. Ma, "Interaction, Specification Refinement, and Tradeoff Exploration in Optimization-Based Design of Engineering Systems," in *Proceedings of the 1985 IFAC Workshop on Control Applications of Nonlinear Programming and Optimization*, G. Di Pillo, ed., Pergamon Press, 1986, 189–194.

[5] R. Klessig & E. Polak, "An Adaptive Precision Gradient Method for Optimal Control," *SIAM J. on Control* 11 (1973), 80–93.

[6] Y.Y. Wardi, "Adaptive-Precision Steepest Descent Optimization Algorithms," School if Electrical Engineering, Georgia Institute of Technology, Technical Report, Atlanta, Georgia, 1986.

[7] Y.Y. Wardi, "A Stochastic Steepest Descent Algorithm," School if Electrical Engineering, Georgia Institute of Technology, Technical Report, Atlanta, Georgia, 1986.

[8] E. Polak & D.Q. Mayne, "An Algorithm for Optimization Problems with Functional Inequality Constraints," *IEEE Trans. Automat. Control* 21 (1976), 184–193.

[9] C. Gonzaga, E. Polak & R. Trahan, "An Improved Algorithm for Optimization Problems with Functional Inequality Constraints," *IEEE Trans. Automat. Control* AC-25 (1980), 49–54.

[10] P. Huard, "Feasible Variable Metric Method for Nonlinearly Constrained Problems," in *Optimization and Control*, A. Auslender, W. Oettli & J. Stoer, eds., Lecture Notes in Control and Information Sciences #30, Springer Verlag, Berlin-Heidelberg-New York-Tokyo, 1981, 43–49.

[11] O. Pironneau & E. Polak, "Rate of Convergence of a Class of Methods of Feasible Directions," *SIAM Journal of Numerical Analysis* 10 (1968), 161–274.

[12] E. Polak, *Computational Methods in Optimization*, Academic Press, New York, N.Y., 1971.

[13] D.M. Topkis & A. Veinott, "On the Convergence of some Feasible Directions Algorithms for Nonlinear Programming," *SIAM J. on Control* 5 (1967), 268–279.

[14] G. Zoutendijk, *Methods of Feasible Directions*, Elsevier Scientific Publishing Company, Amsterdam, The Netherlands, 1960.

[15] J. Cullum, W.E. Donath & P. Wolfe, "The Minimization of Certain Nondifferentiable Sums of Eigenvalues of Symmetric Matrices ," in *Nondifferentiable optimization*, M. Balinski & P. Wolfe, eds., Mathematical programming study #3, North-Holland, 1975.

[16] C.J. Goh & K.L. Teo, "On Minimax Eigenvalue Problems via Constrained Optimization," *J. Optim. Theory Appl.* 57 (1988), 59–68.

[17] D.Q. Mayne & E. Polak, "Algorithms for the Design of Control Systems Subject to Singular Value Inequalities," in *Algorithms and Theory in Filtering and Control*, D.C. Sorensen & R.J-B. Wets, eds., Mathematical Programming Study #18, North Holland, 1982.

[18] M.L. Overton, "On Minimizing the Maximum Eigenvalue of a Symmetric Matrix," Centre for Mathematical Analysis, Australian National University , Report CMA-R03-87, Canberra, Australia, January, 1987, to appear in SIAM J. Matrix Anal. Appl..

[19] M.L. Overton & R.S. Womersley, "On Minimizing the Spectral Radius of a Nonsymmetric Matrix Function - Optimality Conditions and Duality Theory," 1987, to appear in SIAM J. Matrix Anal. Appl..

[20] E. Polak, "On the Mathematical Foundations of Nondifferentiable Optimization ," *SIAM Rev.* 29 (1987), 21–89.

[21] E. Polak & Y. Wardi, "Nondifferentiable Optimization Algorithm for the Design of Control Systems Subject to Singular Value Inequalities," *Automatica*, 18 (1982), 267–283.

[22] M.K.H. Fan, W.T. Nye & A.L. Tits, "DELIGHT.MaryLin User's Guide," Systems Research Center, Technical Report TR-85-9, University of Maryland, College Park, Maryland 20742, 1985.

[23] M.K.H. Fan, L.S. Wang, J. Koninckx & A.L. Tits, "CONSOLE User's Manual," Systems Research Center, University of Maryland, Technical Report TR-87-212, College Park, Maryland 20742, 1987.

[24] M.K.H. Fan, L.S. Wang, J. Koninckx & A.L. Tits, "CONSOLE: A CAD Tandem for Optimization-Based Design Interacting with User-Supplied Simulators," *Proc. of the American Control Conf.*, Atlanta, Georgia (June 1988).

[25] W.T. Nye, E. Polak, A. Sangiovanni-Vincentelli & A.L. Tits, "DELIGHT: An Optimization-Based Computer-Aided Design System," *Proceedings of the 1981 IEEE International Symposium on Circuits and Systems* (April 1981).

[26] W.T. Nye, *DELIGHT: An Interactive System for Optimization-Based Engineering Design*, Ph.D. Thesis, Department EECS, University of California, Berkeley, California, 1983.

[27] E. Polak, P. Siegel, T. Wuu, W.T. Nye & D.Q. Mayne, "DELIGHT.MIMO: An Interactive, Optimization-Based Multivariable Control System Design Package," in *Computer-Aided Control Systems Engineering*, M. Jamshidi & C.J. Herget, eds., Elsevier Science Publishers B.V. (North-Holland), Amsterdam, New York, Oxford, 1985, 139–147.

[28] R. K. Brayton & R. Spence, *Sensitivity and Optimization*, Elsevier Scientific Publishing Company, Amsterdam, The Netherlands, 1980.

[29] Y. Sawaragi, H. Nakayama & T. Tanino, *Theory of Multiobjective Optimization*, Academic Press, 1985.

[30] S.P. Han, "Superlinear Convergence of a Minimax Method," Department of Computer Science, Cornell University, TR78-336, 1978.

[31] S.P. Han, "Variable Metric Methods for Minimizing a Class of Nondifferentiable Functions," *Math. Prog.* 20 (1981), 1–13.

[32] E.R. Panier & A.L. Tits, "A Superlinearly Convergent Method of Feasible Directions for Optimization Problems Arising in the Design of Engineering Systems," in *Proc. of the Seventh International Conference on Analysis and Optimization of Systems — Antibes, June 25-27, 1986*, A. Bensoussan & J.L. Lions, eds., Lecture Notes in Control and Information Sciences #83, Springer Verlag, Berlin-Heidelberg-New York-Tokyo, June 1986, 65–73.

[33] I.D. Coope & G.A. Watson, "A Projected Lagrangian Algorithm for Semi-Infinite Programming," *Math. Programming* 32 (1985), 337–356.

[34] S.A. Gustafson, "On the Computational Solution of a Class of Generalized Moment Problems," *SIAM J. Numer. Anal.* 7 (1970), 343–357.

[35] S.A. Gustafson, "A Three-Phase Algorithm for Semi-Infinite Programs," in *Semi-Infinite Programming and Applications*, A.V. Fiacco & K.O. Kortanek, eds., Lecture Notes in Economics and Mathematical Systems #215, Springer Verlag, 1983, 138–157.

[36] R. Hettich, "An Implementation of a Discretization Method for Semi-Infinite Programming," *Math. Programming* 34 (1986), 354–361.

[37] R. Hettich & W. van Honstede, "On Quadratically Convergent Methods for Semi-Infinite Programming," in *Semi-Infinite Programming*, R. Hettich, ed., Lecture Notes in Control and Information Sciences #15, Springer Verlag, 1979, 97–111.

[38] E. Polak & A.L. Tits, "A Recursive Quadratic Programming Algorithm for Semi-Infinite Optimization Problems," *Appl. Math. Optim.* 8 (1982), 325–349.

[39] G.A. Watson, "Globally Convergent Methods for Semi-Infinite Programming," *BIT* 21 (1981), 362–373.

[40] G.A. Watson, "Numerical Experiments with Globally Convergent Methods for Semi-Infinite Programming Problems ," in *Semi-Infinite Programming and Applications*, A.V. Fiacco & K.O. Kortanek, eds., Lecture Notes in Economics and Mathematical Systems #215, Springer Verlag, 1983, 193–205.

[41] R. Hettich, *Semi-Infinite Programming*, Lecture Notes in Control and Information Sciences #15, Springer Verlag, 1979.

[42] E.R. Panier & A.L. Tits, "A Globally Convergent Algorithm with Adaptively Refined Discretization for Semi-Infinite Optimization Problems Arising in Engineering Design," *IEEE Trans. Automat. Control* (1988), to appear.

[43] H. Parsa & A.L. Tits, "Nonuniform, Dynamically Adapted Discretization for Functional Constraints in Engineering Design Problems," *Proceedings of the 22nd IEEE Conference on Decision and Control* (December 1983).

[44] H. Gfrerer, J. Guddat, H. Wacker & W. Zulehner, "Globalization of Locally Convergent Algorithms for Nonlinear Optimization problems with Constraints," in *Semi-Infinite Programming and Applications*, A.V. Fiacco & K.O. Kortanek, eds., Lecture Notes in Economics and Mathematical Systems #215, Springer Verlag, 1983, 128–137.

[45] R. Hettich, "A Newton Method for Nonlinear Chebyshev Approximation," in *Approximation Theory*, Schaback & Scherer, eds., Springer Verlag, 1976, 222–236.

[46] E.R. Panier & A.L. Tits, "A Superlinearly Convergent Feasible Method for the Solution of Inequality Constrained Optimization Problems," *SIAM J. Control Optim.* 25 (1987), 934–950.

[47] E.R. Panier, A.L. Tits & J.N. Herskovits, "A QP-Free, Globally Convergent Locally Superlinearly Convergent Algorithm for Inequality Constrained Optimization," *SIAM J. Control Optim.* 26 (1988), 788–811.

[48] W.T. Nye, D.C. Riley, A.L. Sangiovanni-Vincentelli & A.L. Tits, "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits," *IEEE Trans. CAD Integ. Circuits and Systems* CAD-7 (1988), 501–520.

[49] D. Butala, K.-Y. Choi & M.K.H. Fan, "Multiobjective Dynamic Optimization of Semi-batch Free Radical Copolymerization Process with Interactive CAD Tools," Systems Research Center, University of Maryland, Technical Report TR 87-166, College Park, Maryland 20742, 1987.

[50] L.S. Wang, "Control System Design for a Flexible Arm," University of Maryland, M.S. Thesis TR-87-164, College Park, MD 20742, 1987.

[51] X. Chen, "Object-Oriented Manipulator Design Exerciser," University of Maryland, M.S. Thesis, College Park, MD 20742, 1987.

[52] Y.C. Pati, D. Friedman, P.S. Krishnaprasad, C.T. Yao, M. Peckerar & C.K. Marian, "Neural Networks for Tactile Perception," *Proc. of the 1988 IEEE Robotics and Automation Conference*, Philadelphia, PA (April 1988).

[53] C.C. Glover & C. Walrath, Westinghouse Defense and Electronics Center, Progress Report #95, Baltimore, Maryland, October 16, 1987.

[54] H. Elmqvist, "SIMNON, An Interactive Simulation Program for Nonlinear Systems: User's Manual," Department of Automatic Control, Lund Institute of Technology, Report 7502, April 1975.

[55] M.K.H. Fan, "Optimization-Based Design of Nonlinear Systems Using CONSOLE and SIMNON," Systems Research Center, University of Maryland, TR 87-204, College park, Maryland 20742, 1987.

[56] W.T. Nye, "Techniques for Using SPICE Sensitivity Computations in DELIGHT.SPICE Optimization," *Proc. of the 1986 IEEE International Conference on Computer-Aided Design* (1986).

[57] S.P. Boyd, V. Balakrishnan, C.H. Barratt, N.M. Khraishi, X. Li, D.G. Meyer & S.A. Norman, "A New CAD Method and Associated Architectures for Linear Controllers," *IEEE Trans. Automat. Control* AC-33 (1988), 268–283.