AIAA JOURNAL
Vol. 31, No. 2, February 1993

Constrained Conjugate Directions Methods for Design Optimization of Large Systems

Jasbir S. Arora* and Guangyao Li†
University of Iowa, Iowa City, Iowa 52242

The concept of conjugate directions for unconstrained optimization is extended to solve general dense constrained minimization problems that have no special structure to exploit sparse matrix techniques. The constrained steepest descent (CSD) directions obtained as the solution of a quadratic programming (QP) subproblem are used to generate the constrained conjugate directions (CCD). The resulting algorithm is quite simple, and its finite convergence and other properties have been proved for convex quadratic programming problems. For general problems, restart procedures, step size determination, and potential constraint strategy are discussed. A set of 41 structural design test problems having 2–489 design variables and 7–1051 constraints, excluding the simple bounds, is used to evaluate the method and its variations. The new method performs much better than the CSD method; however, its performance is not as good as the sequential quadratic programming (SQP) method for moderate size problems. This result is expected and is quite similar to the one for the unconstrained counterpart of the method. For a large-scale problem, the new method performs better than both the CSD and the SQP methods. It is concluded that the basic concept of constrained conjugate directions is a viable approach for large-scale optimization problems. It needs to be fully developed and evaluated for such problems.

Nomenclature

$(\boldsymbol{a},\boldsymbol{b})$ = scalar product $\boldsymbol{a}^T\boldsymbol{b}$
$\boldsymbol{c}$ = gradient of $f(\boldsymbol{x})$, dimension n
D = displacement constraints indicator
$\boldsymbol{d}$ = solution of quadratic programming subproblem, dimension n
F = frequency constraints indicator
$f(\boldsymbol{x})$ = objective function
$\boldsymbol{G}^i$ = gradient of the ith constraint
$g_i(\boldsymbol{x})$ = ith constraint function
$\boldsymbol{H}$ = Hessian matrix
I_E = potential equality constraint index set
I_I = potential inequality constraint index set
m = total number of constraints
m' = number of equality constraints or total number of active constraints
n = number of variables
$\boldsymbol{p}$ = conjugate direction vector, dimension n
S = constraint set, stress constraints indicator
t = iteration number for restarting the algorithm
$\boldsymbol{v}$ = Lagrange multiplier vector
$\boldsymbol{x}$ = vector of variables of dimension n
x_{il} = lower bound on the ith variable
x_{iu} = upper bound on the ith variable
$\boldsymbol{y}$ = difference of gradients of the objective function at two points for the unconstrained problem, difference of descent directions at two points for the constrained problems
α = step size
β = constant used to form the conjugate directions
Φ = descent function
$\nabla\Phi$ = gradient of the descent function
$*$ = optimum value of a quantity

Superscript and subscript

k = superscript for vectors and matrices and subscript for scalars indicating iteration number

Received Feb. 24, 1992; presented as Paper 92-2500 at the AIAA/ASME/ASCE/AHS/ASC 33rd Structures, Strucutral Dynamics, and Materials Conference, Dallas TX, April 13–15, 1992; revision received July 6, 1992; accepted for publication July 9, 1992.

*Professor, Departments of Civil and Environmental Engineering, and Mechanical Engineering, Optimal Design Laboratory.

†Graduate Research Assistant, Optimal Design Laboratory.

I. Introduction

A MATHEMATICAL model for general constrained optimization problems is stated as problem P: minimize $f(\boldsymbol{x})$ for $\boldsymbol{x} \in S$, where the constraint set S is defined as

$$S = \{\boldsymbol{x} \in R^n : g_i(\boldsymbol{x}) = 0, \quad i = 1 \text{ to } m'; \quad g_i(\boldsymbol{x}) \le 0, \quad i = (m'+1) \text{ to } m \quad x_{il} \le x_i \le x_{iu}\} \tag{1}$$

with $f(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$, $i = 1$ to m to be continuously differentiable functions. It is important to note that in numerical calculations the explicit bound constraints in the set S are treated separately to effect efficiency. In the following presentation, however, details of this treatment are not discussed. In numerical examples, the special structure of these constraints is exploited. It is also important to note that no other structure for the problem that can be exploited for efficiency is assumed. In other words, gradients of the functions are full vectors, and Hessian of the Lagrangian is a full matrix. Such problems are classified as dense in the literature.

Several algorithms have been developed in the past to solve the dense problem P, and others are under active development. Most of the methods require substantial computational effort for large-scale problems having hundreds of variables and thousands of constraints. The most modern, efficient, and reliable methods are based on sequential quadratic programming (SQP).[1–3] Even though these methods are the best up to now, they have some drawbacks for large-scale dense problems: 1) use of the approximate Hessian results in a larger requirement for the computer memory, and 2) the large number of operations associated with the use and updating of this matrix results in inefficiency, numerical uncertainty, and instability.

An algorithm that does not have the foregoing drawbacks is needed for large-scale dense problems, so the basic purpose of this paper is to present such an algorithm and evaluate it using a set of structural design problems. It is well known in the unconstrained optimization literature that the conjugate directions methods, such as Fletcher-Reeves[4] and Polak and Ribière[5] methods, are good choices for large-scale problems. The methods turn out to be very

simple modifications of the basic steepest descent method and are quite easy to implement on the computer, but they perform much better than the steepest descent method. Therefore it is proposed to develop a constrained analog of the unconstrained conjugate directions method. This will be done by using the constrained steepest descent (CSD) directions to generate the conjugate directions in the constrained space. Note that this is not a study of the variable metric methods that are also known as the conjugate directions methods.

Optimization methods for large-scale problems can be divided into two categories: 1) methods for problems with special structure and 2) methods for problems that have no special structure. Methods in the first category exploit the special structure of the problem and use sparse matrix approaches to achieve efficiency and computer memory control. There are many applications that fall into this category and methods to treat such problems have been discussed.[6-9] These methods, however, require more complicated programming. In the second category, the optimization problem has no apparent special structure, so sparse matrix techniques are not only inappropriate by also inefficient. The proposed constrained conjugate directions method is intended for problems in the second category, so sparse matrix techniques are not discussed. It is noted, however, that, if an application has special structure, the proposed method can definitely exploit that structure while evaluating problem functions, gradients, and the search direction.

In the next section, the conjugate gradient (CG) and the CSD methods are summarized. In Sec. III, the proposed constrained conjugate directions (CCD) method is given, and its properties are discussed. Computational aspects, such as the starting point, step size determination, restart, etc., are discussed in Sec. IV. Section V describes the structural design test problems used in evaluating the algorithm and its variations. Results are presented and discussed in Sec. VI. In Sec. VII, summary and conclusions are presented.

II. Background and Basic Hypothesis

Since the proposed method is based on the CG method for unconstrained problems and the CSD method, it is prudent to summarize the basic steps of the two methods and discuss them. Also the sequential quadratic programming (SQP) method will be summarized to highlight its differences with the proposed method.

A. Conjugate Gradient Method

The CG method for minimizing a general function $f(\boldsymbol{x})$ without any constraints is summarized as follows:

Algorithm A: Conjugate Gradient Method

Step 1: Initialize $\boldsymbol{x}^{(0)}$. Set $\boldsymbol{p}^{(0)} = -\boldsymbol{c}^{(0)}$ and $k = 0$.

Step 2: Stop if convergence criteria are satisfied.

Step 3: Update the solution as $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$, where α_k is a step size determined by minimizing $f(\boldsymbol{x})$ in the direction $\boldsymbol{p}^{(k)}$.

Step 4: Calculate $\boldsymbol{c}^{(k+1)} = \nabla f[\boldsymbol{x}^{(k+1)}]$ and

$$\boldsymbol{p}^{(k+1)} = \boldsymbol{c}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)} \text{ with } \beta_k = \frac{[\boldsymbol{c}^{k+1}, \boldsymbol{c}^{k+1}]}{[\boldsymbol{c}^{(k)}, \boldsymbol{c}^{(k)}]} \tag{2}$$

Step 5: $k = k + 1$, go to Step 2.

The formula for β_k given in Eq. (2) was developed by Fletcher-Reeves,[4] and the corresponding method is called the Fletcher-Reeves method. Several other formulas for β_k (discussed later) have been developed in the literature; e.g., Polak-Ribière conjugate gradient method.[5] It is important to note that the conjugate gradient method for unconstrained problems has been thoroughly investigated in the literature.[10-12] These references should be consulted for more details.

The foregoing algorithm has been proven to converge to a global minimum point in n iterations with the exact line search when it is applied to convex quadratic functions $f(\boldsymbol{x})$.[4,13] For general problems, it needs to be restarted every nth iteration to achieve convergence.[12] Also, for convex functions, the sequence of iterates is exactly the same as for a quasi-Newton method. Therefore, for this class of problems, the CG method is classified as a quasi-Newton method.

B. Constrained Steepest Descent Method

The simplest method for general constrained problems is the CSD method that is an extension of the steepest descent method for unconstrained problems.[2] It uses a potential constraint strategy (where only a subset of all of the constraints is used in the search direction determination) that is particularly suitable for large-scale applications. It has been proven[14] that the solution point of the sequence $\{\boldsymbol{x}^{(k)}\}$ generated by the algorithm is a Karush-Kuhn-Tucker point for the problem P. In this method, the search direction $\boldsymbol{d}^{(k)}$ at the kth iteration is obtained as a solution of the quadratic programming (QP) subproblem.

Subproblem QPI: Minimize $(\boldsymbol{c}, \boldsymbol{d}) + 0.5\,(\boldsymbol{d}, \boldsymbol{d})$ subject to

$$g_i + (\boldsymbol{a}^i, \boldsymbol{d}) = 0, \quad i \in I_E \tag{3a}$$

$$g_i + (\boldsymbol{a}^i, \boldsymbol{d}) \le 0, \quad i \in I_I \tag{3b}$$

where all of the quantities are evaluated at the current point $\boldsymbol{x}^{(k)}$, I_E and I_I are the potential constraint index sets for equality and inequality constraints, respectively. $\boldsymbol{G}^i$ is the gradient of the ith constraint, and $\boldsymbol{c}$ is the gradient of the objective function. The explicit bound constraints are always imposed.

Note that I_E will generally contain all of the indices 1 to m', although the CSD algorithm allows for deletion of some of the equalities during the iterative process.[14] The algorithm is summarized as follows.

Algorithm B: Constrained Steepest Descent Method

Step 1: Initialize $\boldsymbol{x}^{(0)}$ and set $k = 0$.

Step 2: At $\boldsymbol{x}^{(k)}$, compute gradients of the objective function and the potential constraints. Define the subproblem QPI.

Step 3: Solve the subproblem QPI for the search direction $\boldsymbol{d}^{(k)}$.

Step 4: Check the convergence criterion and stop if it is satisfied, otherwise continue.

Step 5: Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{d}^{(k)}$, where α_k is a step size determined by minimizing a descent function along the search direction $\boldsymbol{d}^{(k)}$. Go to step 2.

The CSD method can also be viewed as a form of the gradient projection method. If the current point is feasible, then the search direction is a projection of the steepest descent direction onto the constraint surface defined by the active constraints. If the current point is infeasible, then the search direction is a composite direction consisting of the projected steepest descent direction and the constraint correction direction. It is important to note, however, that the CSD method is different from the traditional gradient projection method.[15] In the traditional method, the QP subproblem is not defined or solved. Instead, a system of linear equations that depends on an estimate of the active set of linearized constraints is solved to determine the projected direction. Since the active set is not known a priori, heuristics and complicated numerical procedures are used to compute the projected gradient directions. In this regards, the CSD method is much superior to the traditional gradient projection method and produces superior search directions.

Although the CSD method is theoretically guaranteed to converge to a local minimum point starting from an arbitrary point, its numerical behavior can be quite different.[10,16-18] Every iteration starts afresh, so no higher order information is accumulated. The method suffers from linear convergence when the solution is not fully constrained. Therefore performance is similar to that of the steepest descent method.

C. SQP Method

The CSD method has been extended to include approximate second-order information by generating and using a Hessian matrix with the potential constraint strategy.[2,3,16-18] The resulting SQP method has worked extremely well,[19] but, as noted earlier, it has some drawbacks for large-scale problems (defined in this paper as the ones having more than 99 design variables). The method can be derived with or without the use of a potential constraint strategy.[2,3] Here we will describe an SQP method that uses this strategy and is a simple modification of the CSD method. The algorithm solves a slightly modified subproblem QPI defined as follows.

Subproblem QP2: Minimize $(\boldsymbol{c}, \boldsymbol{d}) + 0.5\ (\boldsymbol{d}, \boldsymbol{Hd})$ subject to the same constraints as in subproblem QPI, where $\boldsymbol{H}$ is an approximation to the Hessian of the Lagrangian for the problem.

The SQP algorithm is now obtained by modifying the CSD algorithm B as follows: In steps 2 and 3, the subproblem QP2 is defined and solved instead of subproblem QPI; in step 6, the approximation to the Hessian matrix $\boldsymbol{H}$ is updated using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) procedure.[2,16]

III. Constrained Conjugate Directions Method

A. Proposed New Algorithm

Experience has shown that the CSD method is inefficient as well as less robust compared with the SQP method. Therefore, in practice the SQP method has replaced the CSD method. However, for large-scale problems, the SQP method requires enormous amounts of calculations, primarily because of using and updating the Hessian matrix $\boldsymbol{H}$, unless the problem structure can be exploited, or partial solutions to QP subproblems are used (similar to the truncated Newton method for unconstrained optimization). Therefore, for large-scale problems, it may be useful to avoid the use of $\boldsymbol{H}$ but still have an algorithm that is more efficient and as robust as the SQP method.

The proposed new method is an extension of the conjugate gradient method for unconstrained optimization problems to the constrained ones. It can also be viewed as a modification of the CSD method. The method is called the constrained conjugate directions (CCD) method that is stated first and then its properties are discussed.

Algorithm C: Constrained Conjugate Directions (CCD) Algorithm

Step 1: Select a starting point $\boldsymbol{x}^{(0)}$ and set $\boldsymbol{k} = 0$.

Step 2: Define and solve subproblem QPI at $\boldsymbol{x}^{(0)}$ for $\boldsymbol{d}^{(0)}$, and set $\boldsymbol{p}^{(0)} = \boldsymbol{p}^{(0)}$.

Step 3: Stop if convergence criteria are satisfied.

Step 4: Set $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{p}^{(k)}$, where α_k minimizes some descent function (to be discussed later).

Step 5: Check the restart criteria. If restart is required, set $\boldsymbol{x}^{(0)} = \boldsymbol{x}^{(k+1)}$ and go to step 2. Otherwise continue.

Step 6: Define and solve subproblem QPI at $\boldsymbol{x}^{(k+1)}$ for $\boldsymbol{d}^{(k+1)}$.

Step 7: Stop if convergence criteria are satisfied.

Step 8: Define the conjugate direction as $\boldsymbol{p}^{(k+1)} = \boldsymbol{d}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}$, where the scalar β_k may be calculated by using any of the formulas discussed later. Set $\boldsymbol{k} = \boldsymbol{k}+1$.

Step 9: If the descent condition for $\boldsymbol{p}^{(k)}$ is satisfied, go to step 4. If it is not satisfied or restart is necessary, set $\boldsymbol{p}^{(k)} = \boldsymbol{d}^{(k)}$ and $k = 0$, and go to step 4.

It can be seen that the only difference between the proposed method and the CSD method is in step 8, where the conjugate directions are generated. They are used as the search directions along which an appropriate step size is determined by minimizing a descent function. It is important to note that many programs are available to solve subproblem QPI efficiently.[2] Most linear programming (LP) codes also have an option to solve a quadratic programming problem. It is also important to note that the proposed method reduces to the usual conjugate gradient methods for unconstrained problems.

B. Properties of the CCD Method

It is customary to study the properties of a computational method for a convex quadratic programming problem with equality constraints and then generalize it for the nonlinear programming problem. This has been done for the CCD method, and the following properties have been proven with the exact line search and a feasible starting point[10] (if there are constraint violations, they can be corrected in one iteration using the constraint correction step 2):

1) $(\boldsymbol{d}^{(k)}, \boldsymbol{d}^{(i)}) = 0$ for $i < k$; i.e., the current CSD direction is orthogonal to all of the previous CSD directions.

2) $(\boldsymbol{d}^{(k)}, \boldsymbol{p}^{(i)}) = 0$ for $i < k$; i.e., the current CSD direction is orthogonal to all of the previous conjugate directions.

3) $(\boldsymbol{p}^{(k)}, \boldsymbol{Hp}^{(i)}) = 0$ for $i < k$; i.e., the directions $\boldsymbol{p}^{(k)}$ are conjugate with respect to the Hessian for the quadratic objective function.

4) $\boldsymbol{x}^* = \boldsymbol{x}^{(n-m')}$; i.e., global minimum is reached in $(n - m')$ iterations.

5) The CCD and SQP methods generate exactly same sequence of points.

The foregoing properties are quite analogous to those for the unconstrained counterpart of the algorithm. They provide insights for applications of the algorithm to general nonlinear programming problems. Note that when there are linear inequality constraints, the algorithm needs to be restarted whenever the active set changes. Once the correct active set has been identified, the algorithm converges in $(n - m')$ iterations, where m' is the total number of active constraints.

IV. Computational Considerations for General Problems

The properties of the CCD method for the convex quadratic programming problem do not hold for general nonlinear programming problems. Therefore we need to discuss various computational strategies to insure convergence of the algorithm for general applications. These include starting point determination, potential constraint strategy, descent function, search direction calculation, step size determination, restart procedure, and stopping criterion.

A. Starting Point

Since the method (algorithm C) is a generalization of the one for quadratic problems that is initiated from a feasible point, a feasible or nearly feasible point is desirable. However, it is impossible to maintain feasibility during the optimization process due to nonlinear constraints. Therefore, infeasible starting points or intermediate infeasible points need to be treated in the algorithm.

The starting point can affect performance of the algorithm. If it is too far away from the feasible region, violations should be corrected to a desirable level before switching to the CCD algorithm. This way, the algorithm may give a smoother performance and faster convergence. Therefore, if maximum constraint violation is larger than a threshold level at the first iteration [taken as 0.50 in the present study; note that all constraints are normalized[2] with respect to their limit values before transforming them to the standard form shown in Eq. (1)], the algorithm is switched to constraint corrections only. A simple modification of the subproblem QPI is used to correct constraints.[2] This strategy is also used in the program IDESIGN.[20]

B. Potential Constraint Strategy

As in the CSD and SQP methods, the potential constraint strategy is incorporated into the general CCD methods. This strategy is beneficial, since gradients of only the potential constraints are calculated and used in defining the QP subproblem. This is particularly true for large-scale problems with many inequality constraints where the evaluation of gradients of constraints is expensive. In this strategy, the index sets I_E and I_I used in the definition of the QP subproblems at the kth iteration are defined as

$$I_E = \{i: i = 1 \text{ to } m'\} \tag{4a}$$

$$I_I = \{i: g_i + \varepsilon \geq 0, i = (m'+1) \text{ to } m\} \tag{4b}$$

where ε is a small positive constant (taken as 0.10 in the present study).

C. Descent Function

For a linearly constrained quadratic problem, since constraints are always satisfied during the entire process, the objective function is treated as the descent function for step size determination. This cannot be done for general problems, however. Therefore, a descent function needs to be defined and minimized at each iteration to obtain the step size. For the general problem P, the objective function needs to be minimized in the constrained space. Therefore, the descent function has to take into account both the objective function and constraints to balance the decrease in the objective function and violation of the constraints.

The descent function will affect efficiency of the line search as well as progress of the optimization process. Many descent functions are available. Different descent functions give different step sizes even with the exact line search. A simple descent function that is continuous but not differentiable everywhere is defined as

$$\Phi(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m'} r_i g_i^2 + \sum_{(i=m')+1}^{m} v_i g_i^+ \tag{5}$$

where $r_i = |v_i/2g_i|$, v_i are the Lagrange multipliers with $v_i \geq 0$ for $i > m'$ and $g_i^+ = \max\{0, g_i\}$. The values of the Lagrange multipliers are obtained as a part of the solution of subproblem QP1. This descent function is very close to the Lagrange function, and its minimum point is the same as that for the problem P. It has been proven[10] that the search direction $\boldsymbol{p}^{(k)}$ is that of the descent for $\Phi(\boldsymbol{x})$ at $\boldsymbol{x}^{(k)}$. Other descent functions such as the augmented Lagrangian can also be used.

D. Search Direction

The search direction in the constrained conjugate directions methods at $(k+1)$th iteration is defined as $\boldsymbol{p}^{(k+1)} = \boldsymbol{d}^{(k+1)} + \beta_k \boldsymbol{p}^{(k)}$, where $\boldsymbol{d}^{(k+1)}$ is solution of the subproblem QP1. There are several alternative formulas for calculation of β_k, and each one gives a different search direction for the general constrained optimization problem. The best known formulas are Fletcher-Reeves[4] and Polak-Ribière.[5] In the numerical test of the CCD method, these two formulas and three other modifications are used:

Formula 1—Fletcher-Reeves:

$$\beta_k = \frac{(\boldsymbol{d}^{(k+1)}, \boldsymbol{d}^{(k+1)})}{(\boldsymbol{d}^{(k)}, \boldsymbol{d}^{(k)})} \tag{6}$$

Formula 2—Polak-Ribière:

$$\beta_k = \frac{(\boldsymbol{d}^{(k+1)}, \boldsymbol{y}^{(k)})}{(\boldsymbol{d}^{(k)}, \boldsymbol{d}^{(k)})}, \qquad \boldsymbol{y}^{(k)} = \boldsymbol{d}^{(k+1)} - \boldsymbol{d}^{(k)} \tag{7}$$

Formula 3:

$$\beta_k = \begin{cases} \beta_k^P & \text{if } 0 \leq \beta_k^P \leq 5\beta_k^F \\ 5\beta_k^F & \text{if } \beta_k^P \leq 5\beta_k^F \\ 0 & \text{if } \beta_k^P < 0 \end{cases} \tag{8}$$

Formula 4:

$$\beta_k = \begin{cases} \beta_k^P & \text{if } -5\beta_k^F \leq \beta_k^P \leq 5\beta_k^F \\ 5\beta_k^F & \text{if } \beta_k^P \geq 5\beta_k^F \\ -5\beta_k^F & \text{if } \beta_k^P < -5\beta_k^F \end{cases} \tag{9}$$

Formula 5:

$$\beta_k = \begin{cases} \beta_k^P & \text{if } -4\beta_k^F \leq \beta_k^P \leq 5\beta_k^F \\ 5\beta_k^F & \text{if } \beta_k^P \geq 5\beta_k^F \\ -4\beta_k^F & \text{if } \beta_k^P < -4\beta_k^F \end{cases} \tag{10}$$

In the last three formulas, β_k^P and β_k^F are the values calculated by the Polak-Ribère and the Fletcher-Reeves formulas, respectively. These formulas are based on the work of Gilbert and Nocedal.[12]

The descent condition

$$(\boldsymbol{p}^{(k)}, \nabla\Phi^{(k)}) < 0 \tag{11}$$

is a critical requirement for the search direction $\boldsymbol{p}^{(k)}$. If this condition is not satisfied, it is impossible to move closer to the optimum point along this direction from the point $\boldsymbol{x}^{(k)}$. Theoretically, if the search direction satisfies the descent condition (11), a finite step size can be calculated. Numerically, we should test the following inequality to insure that the search direction is sufficiently downhill and avoid calculation of $\nabla\Phi^{(k)}$:

$$(\boldsymbol{d}^{(k)}, \boldsymbol{p}^{(k)}) \geq c_1 \|\boldsymbol{d}^{(k)}\|^2 \tag{12}$$

where c_1 is a positive constant. If this inequality does not hold, restart is required. Note that a larger value for c_1 implies more frequent restart.

E. Step Size Determination

Evaluation of the objective and constraint functions and their gradients accounts for a significant part of the calculations in the entire optimization process. In addition, the CCD method needs a relatively accurate step size that needs more function evaluations. Although an accurate step size needs more calculations, fewer iterations may be needed to reach the solution. Inaccurate line search needs fewer function evaluations during an iteration but may need more overall iterations. How to implement the line search and when to terminate it are the key issues.

In the inexact line search algorithms that follow, curve-fitting strategy is used. Initial step size is set to one. Before the interval of uncertainty for the minimum point α_k^* is found, the next trial step size is calculated using the quadratic approximation for the descent function based on the least squares method. The minimum point of the quadratic approximation is the next trial step size. Once the interval of uncertainty is located, the descent function is approximated as a quadratic function passing through three points, and the minimum point of the quadratic function is set as the next trial step size. Several criteria to terminate the line search have been proposed. The following criterion that is practical for large-scale problems is used in the present study. At the kth iteration, let $\bar{\alpha}$ be such that

$$\Phi(\bar{\alpha}) = \min\{\Phi(\alpha_k^{(1)}), \Phi(\alpha_k^{(2)}), \ldots, \Phi(\alpha_k^{(j-1)})\} \tag{13}$$

If the next trial step size $\alpha_k^{(j)}$ a satisfies one of the conditions

$$\Phi(\alpha_k^{(j)}) - \Phi(\bar{\alpha}) \leq c_2\{\Phi(\boldsymbol{x}^{(k)}) - \Phi(\alpha)\} \quad \text{if } \Phi(\bar{\alpha}_k^{(j)}) \geq \Phi(\alpha) \tag{14a}$$

$$\Phi(\bar{\alpha}) - \Phi(\alpha_k^{(j)}) \leq c_2\{\Phi(\boldsymbol{x}^{(k)}) - \Phi(\bar{\alpha})\} \quad \text{if } \Phi(\alpha_k^{(j)}) \leq \Phi(\bar{\alpha}) \tag{14b}$$

then line search is terminated [i.e., change in the descent function from the best step size is less than the c_2 fraction of the reduction obtained from the point $\boldsymbol{x}^{(k)}$]. Here $0 < c_2 < 0.25$ is a specified constant. Note that a smaller value for c_2 produces more accurate step sizes.

F. Restart

The information from the previous iterations is accumulated in the search directions for the CCD methods. But the information far away from the current point may not be useful for defining the next search direction. It may even be detrimental to use such information for general nonlinear problems. Therefore, at some point, such information should be discarded and the optimization process restarted with the CSD direction.

Powell[21] has proposed a restart criterion for the unconstrained conjugate directions methods that can also be used in the CCD methods. We redefine it for the CCD methods as follows: at the kth iteration, if the inequality

$$|(\boldsymbol{d}^{(k-1)}, \boldsymbol{d}^{(k)})| \leq 0.2 \|\boldsymbol{d}^{(k)}\|^2 \tag{15}$$

does not hold, then restart is required (note that $[\boldsymbol{d}^{(k-1)}, \boldsymbol{d}^{(k)}]$ is supposed to be zero for the convex quadratic problem).

If the number of iterations exceeds a certain value, restart is necessary since finite termination cannot be proven for general nonlinear problems. This number should be smaller than $(n - m')$, where m' is the number of active constraints. Let $t = \min [c_3 n, (n - m'), c_4]$, where $0 < c_3 \leq 1$ and $0 < c_4 \leq n$. If $k \geq t$, the process is restarted with the CSD direction.

G. Stopping Criterion

If one of the following criteria is satisfied, the optimization process is stopped: $\|\boldsymbol{d}^{(k)}\| \leq c_5$ at a feasible point, or $k \geq k_{\max}$, where c_5 may be taken between 0.001 and 0.00001, and $k_{\max}$ is the maximum number of iterations allowed. The value for c_5 affects the total computational effort, so it should be selected judiciously for practical applications. A severe stopping criterion is used in the present study to compare the methods.

V. Test Problems

The new CCD method has been evaluated using a set of 115 small-scale mathematical programming test problems.[10] For that set, the method is more robust and efficient than the CSD method. It is slightly less robust and less efficient than the SQP method. These results are quite similar to the ones for the unconstrained counterparts of the algorithms; however, they cannot be generalized to large-scale problems and the method needs to be evaluated for such problems.

In the present study, a total of 41 structural design problems are used to test the CCD methods. Overall data for the problems are summarized in Table 1. Note that the total number of constraints (NC) given there does not include the explicit design variable bound constraints that are always imposed during each iteration.

Table 1 Summary of structural design test problems

Description	Problem no.	Constraints[a]	No. of constraints	No. of design variables	Cost
10-bar truss	1A	S,D	18	10	5060.85
	2B	S,D,F	19	10	4783.17
	3C	S	10	10	1664.53
	4D	S,D	18	10	676.460
Tail boom	5	S,D,F	181	42	105.576
200-bar tower	6A	S	600	29	2.545E4
	7B	S,D,F	1051	96	2.941E4
	8C	S	600	96	7466.39
22-bar truss	9	S,D	102	7	1023.89
25-bar truss	10	S,D	92	13	2168.73
25-bar tower	11A	S,D,F	87	7	590.723
	12B	S	50	7	91.132
	13C	S,D	86	7	545.036
	14D	S,D	43	7	1.257E5
	15E	S,D	43	7	1.354E4
3-bar truss	16A	S,D,F	16	2	22.970
	17B	S,D,F	16	3	21.049
	18C	S	9	2	21.111
	19D	S	9	3	15.968
	20E	S,D	15	2	22.970
	21F	S,D	15	3	20.545
4-bar truss	22A	S,D	7	4	676.460
	23B	S,D	7	4	725.084
47-bar truss	24A	S,D,F	88	47	3769.59
	25B	S	47	47	2994.64
	26C	S,D	87	47	3769.59
49-bar truss	27A	S	98	25	1234.67
	28B	S	147	25	1234.67
72-bar tower	29A	S,D,F	241	16	519.632
	30B	S	144	16	96.638
	31C	S,D	240	16	379.615
10-bar truss	32E	S,D	18	10	5.077
	33F	S,D	18	10	5.077
	34G	S,D	18	20	4.669
	35H	S,D,F	18	20	4.878
1-bay, 2-story frame	36A	S,D	84	16	7.756
	37B	S,D,F	85	16	7.756
47-bar truss	38D	S,D	87	47	3.825
2-bay, 6-story	39	S,D	264	36	22.799
200-bar tower	40D	S	600	29	25.450
COFS	41	S,F	490	489	275.000

[a]S: stress constraint; D: displacement constraint; F: frequency constraints.

A. Truss and Frame Structural Design Problems

The first 40 test problems are constructed using 11 trusses and 2 frames. These problems can be found in the literature (e.g., Refs. 15 and 22) and have been used in the past for testing and comparing optimization algorithms and software. These problems range from small scale to a 96 design variable problem. Cross-sectional areas are treated as design variables unless noted otherwise. The weight of the structure is minimized in all of the problems. A brief description of the problems is given next.

1) 10-Bar Truss[15]: This cantilever truss has been used extensively in the literature for studying various algorithms. Four problems are constructed for the structure with two load cases: case A—load case 1, stress and displacement constraints (no. 1); case B—load case 2, stress, displacement, and frequency constraints (no. 2); case C—load case 2, stress constraints only (no. 3); and case D—same as case A with different displacement constraints and initial point (no. 4).

2) Tail Boom Truss[15]: One design problem is studied with stress, displacement, and frequency constraints for this helicopter tail boom truss. Forty-two variables are used for 108 members (no. 5).

3) 200-Bar Tower[15]: Three problems are constructed for the structure with different numbers of design variables and constraints. Three loading conditions are used for all problems: condition A—29 design variables with stress constraints (no. 6); condition B—96 design variables with all constraints (no. 7); and condition C—96 design variables with stress constraints (no. 8).

4) 22-Bar Truss: Stress and displacement constraints are imposed (no. 9).

5) 25-Bar Truss: Stress and displacement constraints are imposed (no. 10).

6) 25-Bar Tower[15]: Five problems are constructed for the structure with different constraints and loading conditions (all problems have seven design variables: problem A—two loading conditions with stress, displacement, and frequency constraints (no. 11); problem B—two loading conditions with stress constraints (no. 12); problem C—same as problem A but without the frequency constraint (no. 13); problem D—one loading condition with stress and displacement constraints (no. 14); and problem E—same as problem D with a different loading condition (no. 15).

7) 3-Bar Truss[15]: Six problems are constructed for this truss with different numbers of design variables and different constraints: problem A—two variables, stress, displacement, and frequency constraints (no. 16); problem B—three variables, stress, displacement, and frequency constraints (no. 17); problem C—two variables, stress constraints only (no. 18); problem D—three variables, stress constraints only (no. 19); problem E—two variables, stress and displacement constraints (no. 20); and problem F—three variables, stress and displacement constraints (no. 21).

8) 4-Bar Truss: This truss consists of four bars. There is one loading condition, and stress and displacement constraints are imposed. Two problems are constructed for the truss due to the different requirement on the displacement of the vertex (nos. 22 and 23).

9) 47-Bar Truss: Three problems are constructed for the structure with different constraints and one loading: problem A—stress, displacement, and frequency constraints (no. 24); problem B—stress constraints only (no. 25); problem C—stress and displacement constraints (no. 26).

10) 49-Bar Truss: Two design problems are constructed for the structure with different loading conditions and design variable linking is used to define 25 design variables: problem A—two loading conditions (no. 27), and problem B—three loading conditions (no. 28).

11) 72-Bar Tower: Three problems are constructed with different constraints, and members are grouped into 16 design variables: problem A—stress, displacement, and frequency constraints (no. 29); problem B—stress constraints only (no. 30); and problem C—stress and displacement constraints (no. 31).

12) 10-Bar Truss: Load case 1 of the 10-member truss is used, and the structure is modeled with a frame program that has the capability to use different types of cross-sectional shapes. Four problems are constructed: problem E—load case 1, stress and dis-

placement constraints, *I*-section with one design variable for each member (no. 32); problem F—same as problem E with a different initial design point (no. 33); problem G—load case 1, stress and displacement constraints, *I*-section with two design variables for each member (no. 34); and problem H—same as problem G with an additional frequency constraint (no. 35).

13) 1-Bay 2-Story Frame[15]: It uses *I*-section with four variables for each section. There are two design problems for the structure due to different constraints: problem A—stress and displacement constraints are imposed (no. 36); problem B—stress, displacement, and frequency constraints are imposed (no. 37).

14) 47-Bar Plane Truss: The structure has the same loading as before except it is solved with a program that has the capability to use different cross-sectional shapes. Stress and displacement constraints are imposed and *I*-section with one design variable for each member is used (no. 38).

15) 2-Bay 6-Story Frame[15]: This structure is made of two types of materials and has hollow tubular sections. One design problem is studied with stress and displacement constraints (no. 39).

16) 200-Bar Truss: The 200-bar truss is solved using the frame program with one stress constraints imposed. The *I*-Section with one design variable for each member is used (no. 40).

B. Large-Scale Problem

The large-scale structural design problem considered here is a mast flight system developed by Lenzi and Shipley[23] and optimized previously by Canfield et al.[24] The primary structure is a cantilever truss column that is called the control of flexible structures (COFS) model. The cross section of the column is an equilateral triangle with the longerons located at its vertices. The truss structure repeats itself in 2-bay segments with a total of 54 bays. it consists of 489 truss elements, and the cross-sectional area of each element is considered as a design variable. All of the elements are made up of graphite epoxy. Nonstructural masses are attached at the nodes. Both the stress and natural frequency constraints are imposed, giving a total of 490 constraints besides the simple bounds. The problem is to minimize weight of the structure (problem no. 41 in Table 1).

VI. Numerical Tests

The constrained conjugate directions method is implemented by extending the optimization software IDESIGN.[20] The tests are carried out on an Apollo DN10000 workstation. A computer program TRUSSOPT[25] is employed for the finite element analysis and gradient calculations in solving 31 truss design problems (set 1: problems 1–31). Cross-sectional areas are treated as design variables for all of these problems. Another computer program FROPT[26] is used for the same purpose in solving nine frame design problems (set 2: problems 32–40). This program can use different cross-sectional shapes and can treat truss and frame models. To solve the large-scale truss design problem, computer program ANALYZE[27] is combined with IDESIGN for the finite element analysis and gradient calculations.

For the first 40 test problems, the following stopping criterion is used: maximum constraint violation should be less than 0.001%, and $c_5 = 0.001$. The corresponding numbers for the large-scale problem are 0.01% and 0.01. These criteria are quite severe and guarantee a local minimum point.

Numerical tests are carried out in two stages: 1) variations of various parameters of the CCD algorithm are investigated, and 2) the CCD method is compared with the CSD and SQP methods.

A. Variations of the CCD Method

Several variations of the CCD algorithm can be defined. For example, different formulas for β can give different search directions, different values for the line search termination parameter c_2 in Eq. (14) can give different paths to the solution point, different restart procedures can give different behavior to the algorithm, etc. Based on a previous study,[10] it has been determined the $c_1 = 0.1$ in Eq. (12) gives reasonable performance for the algorithm, so this value is used throughout the present study. Various line search termination criteria and restart procedures are further evaluated. Also, the large-scale problem is used separately to evaluate various variations.

As a first study, nine problems in set 2 are used to test the following three values for the line search termination parameter c_2 in Eqs. (14): 0.01, 0.05, and 0.2. The following restart criterion is used: if $k \geq t$, the algorithm is restarted, where $t = \min\{(n - m' - 1), 11\}$ and $t \geq 5$. The results are summarized in Table 2. Note that the three numbers shown in each column in Table 2 and the following Table 3 correspond to the number of problems solved, average number of iterations, and average CPU time; the averages are taken only for the problems solved by all of the algorithms. It can be seen that the smallest value for the parameter c_2 (i.e., more accurate line search) gives good performance in terms of robustness, especially for $\beta_2 - \beta_4$; β_3 gives the best performance in terms of efficiency.

Next, using the first 40 problems, the following two restart procedures are tested with $c_1 = 0.1$ and $c_2 = 0.01$: case A, $t = \min\{(n - m' - 1), 11\}$ and $t \geq 5$; case B, $t = 4$. The results are summarized in Table 3. Both the cases give almost similar performance, with the case A having a slight edge, and β_3 giving the best performance in terms of efficiency.

For the large-scale problem (no. 41), four values for the line search termination parameter c_2 are tested: 0.01, 0.10, 0.20, and 0.25. The algorithm is restarted every sixth iteration. The results are summarized in Table 4. The problem is solved for all of the β formulas and all of the c_2 values. The β_1 formula gives the best results, and $c_2 < 0.2$ shows reasonable performance.

Restart of the algorithm at every third, fourth, and sixth iteration is also tested for the large-scale problem. The results are summarized in Table 5. The problem is solved successfully for all of the cases. Restart at every fourth iteration gives very good efficiency, and the β_1 formula performs the best.

It is interesting to note that, in all of the tests, very similar results are obtained with β_4 and β_5, so only one of them needs to be used.

B. Comparative Evaluation of the CCD Method

All of the test problems are also solved with the CSD and SQP methods. Acceptable violation of constraints, tolerance for the convergence parameter (norm of the search direction), and the maximum number of iterations are the same for all of the methods. The same starting point is used for all of the tests for each problem.

With the SQP method, all of the 40 problems in sets 1 and 2 can be solved. Under the same conditions, however, only 16 of the 40 problems can be solved with the CSD method. The CCD method can solve 34 problems with different β values and step size determination schemes. The results are summarized in Table 6.

Table 2 Test of line search termination parameter for nine problems in set 2

β	$c_2 = 0.01$	$c_2 = 0.05$	$c_2 = 0.20$
1	5/219/133[a]	5/210/122	6/199/101
2	8/119/105	6/137/103	6/235/120
3	8/106/86	6/130/97	8/107/69
4	9/135/108	8/157/109	6/125/83
5	9/136/108	5/157/111	6/125/82

[a]No. of problems solved/average no. of iterations/average CPU, s.

Table 3 Test of restart procedure for first 40 test problems

β	Case A[c]	Case B[b]
1	24/102/157[a]	24/85/96
2	27/51/75	34/67/78
3	30/45/61	28/49/67
4	28/53/75	31/56/101
5	29/54/86	28/56/97

[a]No. of problems solved/average no. of iterations/average CPU, s.
[b]Case B; $t = 4$.
[c]Case A; $t = \min\{(n - m' - 1), 11\}$ and $t \leq 5$.

Table 4 Test of line search termination parameter for the large-scale problem

β	$c_2 = 0.01$	$c_2 = 0.10$	$c_2 = 0.20$	$c_2 = 0.25$
1	40/34,698[a]	40/30,597	42/28,634	46/31,114
2	52/47,890	59/44,236	68/46,343	79/54,633
3	56/53,174	52/42,061	51/36,757	69/46,201
4	52/49,902	59/45,138	68/46,785	79/54,574
5	52/50,380	59/45,478	68/47,134	79/54,672

[a]No. of problems solved/average CPU, s.

Table 5 Test of restart procedure for the large-scale problem

β	Third iteration	Fourth iteration	Sixth iteration
1	55/38,597[a]	37/22,984	42/28,634
2	65/44,938	50/36,196	68/46,343
3	51/36,723	65/45,660	51/36,757
4	59/40,505	50/36,561	68/46,785
5	59/40,532	50/36,626	68/47,134

[a]No of iterations/CPU, s.

Table 6 Results for the 40 problems with the CSD, CCD and SQP methods

	CSD		CCD		SQP	
Prob.	NIT[a]	CPU	NIT	CPU	NIT	CPU
1	F[b]		18	1	17	0[c]
2	123	7	53	6	18	1
3	11	0	10	0	10	0
4	F		396	19	20	0
5	F		216	633	54	55
6	34	300	52	495	32	254
7	F		F		50	1239
8	F		F		152	2704
9	F		99	14	18	1
10	F		35	5	31	3
11	F		65	53	15	3
12	9	0	9	1	9	0
13	F		F		16	1
14	49	3	40	4	17	1
15	F		882	100	15	0
16	8	0	10	0	8	0
17	8	0	10	0	8	0
18	35	0	12	0	9	0
19	27	0	13	0	9	0
20	9	0	10	0	9	0
21	23	0	14	0	10	0
22	6	0	12	0	4	0
23	F		F		17	0
24	F		73	71	29	41
25	7	8	10	6	5	8
26	F		73	34	29	38
27	20	10	22	15	19	10
28	16	12	26	26	19	17
29	F		F		25	14
30	10	3	10	9	11	3
31	F		F		38	18
32	F		808	103	177	9
33	F		763	99	116	5
34	F		165	27	87	8
35	F		116	58	67	15
36	F		79	29	124	36
37	F		74	80	151	91
38	F		913	976	125	120
39	F		242	348	29	41
40	F		896	8422	359	2693

[a]NIT: number of iterations.
[b]F: failed.
[c]Zero implies that no CPU time was recorded on Apollo DN10000 workstation.

The large-scale problem (no. 41) is also tried with all of the methods. With the SQP method, the problem can be solved in 62 iterations with CPU time of 62,367 s. The CSD method fails to solve the problem in 200 iterations. The CCD method can also solve the problem with all of the variations of various parameters, with the best CPU time being 22,984 s for 37 iterations. Thus the CCD method is more efficient than the SQP method by a factor of approximately three. The CCD algorithm was also tried from two better starting points, one feasible (maximum constraint violation 0.07%) and the other infeasible (maximum constraint violation 94.2%), and relaxed stopping criteria were used. From the feasible point, the algorithm converged in five iterations with CPU of 2868 s. From the infeasible point, the algorithm converged in 10 iterations with CPU of 6069 s. These tests show that the method can find the optimum point in 5-10 iterations if a good starting point is available, which is the case in many practical applications.

CPU time consists of two parts. The first part is the time needed in the optimization process in which the main calculation is in solving subproblem QP1 for the CCD method and using and updating the approximate Hessian matrix for the SQP method. The second part consists of time for the finite element analysis, gradient calculations, and function evaluations. The results for the large-scale problem imply that the SQP method spends a lot of CPU time in updating and using the Hessian matrix (while solving the QP subproblem) because it takes the smaller number of function evaluations (139 for SQP vs 146 for CCD) but the CPU time is much larger.

VII. Discussion and Conclusions

In this paper, the concept of conjugate directions is extended to the constrained optimization problems. The basic idea is to use the CSD directions, obtained as the solution of a QP subproblem, to generate conjugate directions in the constrained space. The resulting CCD method is a very simple modification of the CSD method; however, it performs much better than the original method.

The new method also avoids the major drawback of the SQP method for large-scale problems; i.e., it avoids generation of a large Hessian matrix containing approximate second-order information and its use in definition and solution of the QP subproblem. Numerical experiments show that the SQP method needs a large number of operations with the Hessian matrix, which is the primary cause of its inefficiency for application to large-scale problems.

Several computational aspects of the new method are discussed, such as starting point selection, search direction calculation, step size determination, restart strategy, and stopping criterion. A set of 41 structural design problems consisting of small-, medium-, and large-scale problems is used to evaluate the method and its variations. Based on the study, the following conclusions are stated: 1) the proposed CCD method is more robust and efficient than the CSD method; 2) the method is less robust and less efficient than the SQP method for moderate size problems; 3) the method is more efficient than the SQP method for a large-scale problem; and 4) the performance of the method is affected by the starting point, restart procedures, step size determination, and the stopping criterion.

Based on the foregoing conclusions, it can be stated that the concept of constrained conjugate directions is a viable approach for optimization of large-scale systems. However, more work needs to be done on the restart strategies, scaling and preconditioning procedures, use of different QP solution procedures, effect of active set changes, and step size determination procedures to fully develop the method and realize its potential for large-scale dense problems.

Acknowledgment

This paper is based on a part of the research sponsored by the National Science Foundation under the project "Design Sensitivity Analysis and Optimization of Nonlinear Structural Systems," Grant MSM 89-13218.

References

[1]Schittkowski, K., "A Unified Outline of Nonlinear Programming Algorithms," *Journal of Mechanisms, Transmissions and Automation in Design,* Vol. 107, Dec. 1985, pp. 499–453.

[2]Arora, J. S., *Introduction to Optimum Design,* McGraw-Hill, New York, 1989.

[3]Arora, J. S., "Computational Design Optimization: A Review and Future Directions," *Structural Safety,* Vol. 7, 1990, pp. 131–148.

[4]Fletcher, R., *Practical Methods of Optimization,* Wiley, New York, 1987.

[5]Polak, E., and Ribière, G., "Note sur la Convergence de Methods de Di-

rections Conjuguees," *Revue Francaise d'Informatique de Recherche Operationnelle,* Vol. 16, 1969, pp. 35–43.

[6]Gabriele, G. A., and Ragsdell, K. M., "Large Scale Nonlinear Programming Using the Generalized Reduced Gradient Method," *Journal of Mechanical Design, Transactions of ASME,* Vol. 102, July 1980, pp. 566–573.

[7]Coleman, T., and Li, Y. (eds.), *Large-Scale Numerical Optimization,* Society for Industrial and Applied Mathematics, 3600 University City Center, Philadelphia, PA 19104-2688, 1990.

[8]Gill, P. E. (organizer), *Large-Scale Nonlinear Optimization,* Session MS19, Fourth SIAM Conference on Optimization, Society for Industrial and Applied Mathematics, Philadelphia, PA 19104-2688, May 11–14, 1992.

[9]Betts, J. T., and Huffman, W. P., "Application of Sparse Nonlinear Programming to Trajectory Optimization," *Journal of Guidance, Control, and Dynamics,* Vol. 15, No. 1, 1992, pp. 198–207.

[10]Li, G., and Arora, J. S., "Constrained Conjugate Directions Methods for Optimum Design of Large Scale Engineering Systems," Optimal Design Lab., College of Engineering, Univ. of Iowa, TR ODL-91.15, Iowa City, IA, July 1991.

[11]Dennis, J. E., Jr., and Turner, K., "Generalized Conjugate Gradients," *Linear Algebra and Its Applications,* Vol. 88/89, 1987, pp. 187–209.

[12]Gilbert, J. C., and Nocedal., J., "Global Convergence Properties of Conjugate Gradient Methods for Optimization," *SIAM Journal of Optimization,* Vol. 2, No. 1, 1992, pp. 21–42.

[13]Luenberger, D. G., *Linear and Nonlinear Programming,* Addison-Wesley, Reading, MA, 1984.

[14]Pshenichny, B. N., and Denilin, Y. M., *Numerical Methods in Extremal Problems,* English translation by V. Zhitomirsky, Mir Publishers, Moscow, 1978.

[15]Haug, E. J., and Arora, J. S., *Applied Optimal Design,* Wiley-Interscience, New York, 1979.

[16]Lim, O. K., and Arora, J. S., "An Active Set RQP Algorithm for Engineering Design Optimization," *Computational Methods in Applied Mechanics and Engineering,* Vol. 57, No. 1, 1986, pp. 51–65.

[17]Lim, O. K., and Arora, J. S., "Dynamic Response Optimization Using an Active Set RQP Algorithm," *International Journal for Numerical Methods in Engineering,* Vol. 24, No. 10, 1987, pp. 1827–1840.

[18]Tseng, C. H., and Arora, J. S., "On Implementation of Computational Algorithms for Optimal Design," *International Journal for Numerical Methods in Engineering,* Vol. 26, No. 6, 1988, pp. 1365–1402.

[19]Thanedar, P. B., Arora, J. S., Li, G. Y., and Lin, T. C., "Robustness, Generality and Efficiency of Optimization Algorithms for Practical Applications," *Structural Optimization,* Vol. 2, No. 4, 1990, pp. 202–212.

[20]Arora, J. S., and Tseng, C. H., "IDESIGN User's Manual," Version 3.5, Optimal Design Lab., College of Engineering, Univ. of Iowa, TR ODL-87.1, Iowa City, IA, July 1987.

[21]Powell, M. J. D., "Some Properties of the Conjugate Gradient Method," *Mathematical Programming,* Vol. 11, 1976, pp. 42–49.

[22]Belegundu, A. D., and Arora, J. S., "A Recursive Quadratic Programming Method with Active Set Strategy for Optimal Design," *International Journal for Numerical Methods in Engineering,* Vol. 21, 1984, pp. 1583–1599.

[23]Lenzi, D. C., and Shipley, J. W., "MAST Flight System Beam Structure and Beam Structural Performance," *Proceedings of the NASA/DOD Control/Structures Interaction Technology,* NASA-CP 2447, Pt. 1, 1986, pp. 265–279.

[24]Canfield, R. A., Grandhi, R. V., and Venkayya, V. B., "Structural Optimization with Stiffness and Frequency Constraints," *Mechanics of Structures and Machines,* Vol. 17, 1989, pp. 95–110.

[25]Arora, J. S., and Thanedar, P. B., "Optimal Design of Trusses with the Program IDESIGN," Optimal Design Lab., College of Engineering, Univ. of Iowa, TR ODL-85.6, Iowa City, IA, July 1985.

[26]Lim, O. K., Arora, J. S., and Thanedar, P. B., *Optimal Design of Plane Frames with the Program RQP,* Optimal Design Lab., College of Engineering, Univ. of Iowa, TR ODL-85.25, Iowa City, IA, July 1985.

[27]Venkayya, V. B., and Tischler, V. A., "ANALYZE: Analysis of Aerospace Structures with Membrane Elements," Air Force Flight Dynamics Lab., AFFDL TR-78-170, Wright-Patterson AFB, OH, 1978.