AIAA Meeting Papers on Disc, 1996, pp. 226-236
A9638725, DAAE07-94-C-R094, AIAA Paper 96-4008

Sequentially decomposed programming

Sigurd A. Nelson, II
Michigan Univ., Ann Arbor

Panos Y. Papalambros
Michigan Univ., Ann Arbor

AIAA, NASA, and ISSMO, Symposium on Multidisciplinary Analysis and Optimization, 6th, Bellevue, WA, Sept. 4-6, 1996, Technical Papers. Pt. 1 (A96-38701 10-31), Reston, VA, American Institute of Aeronautics and Astronautics, 1996, p. 226-236

Model-based decomposition is a powerful tool for breaking design problems into smaller subproblems, establishing hierarchical structure, and analyzing the interrelations in engineering design problems. However, the theoretical foundation for solving decomposed problems is not yet well established. We show that the formulation of the coordination problem is critical in quickly identifying the correct active constraints, and that solving subproblems independently may hinder the local convergence of algorithms tailored to hierarchical coordination. Conversely, it is believed that hierarchical decomposition algorithms have excellent global convergence properties and usually exhibit superior improvement in the first few iterations when compared to the undecomposed case. Based on insights given in the paper, we outline a Sequentially Decomposed Programming (SDP) algorithm. SDP has two phases: when far from the solution, the first phase is enacted and decomposition is used; when close to the solution, the second phase is underway and decomposition is not used. The principles defining SDP are applied to Sequential Quadratic Programming (SQP) to define an SDP-SQP implementation. A global convergence proof and a simple example are given. (Author)

96-4008
A96-38725
AIAA-96-4008-CP

Sequentially Decomposed Programming

Sigurd A. Nelson II
Graduate Student
and
Panos Y. Papalambros
Professor
Design Laboratory
Department of Mechanical Engineering and Applied Mechanics
University of Michigan
Ann Arbor, Michigan, 48109, U.S.A.

Model-based decomposition is a powerful tool for breaking design problems into smaller subproblems, establishing hierarchical structure, and analyzing the interrelations in engineering design problems. However, the theoretical foundation for solving decomposed problems is not yet well established. We show that the formulation of the coordination problem is critical in quickly identifying the correct active constraints, and that solving subproblems independently may hinder the local convergence of algorithms tailored to hierarchical coordination. Conversely, it is believed that hierarchical decomposition algorithms have excellent global convergence properties and usually exhibit superior improvement in the first few iterations when compared to the undecomposed case. Based on insights given in the paper, we outline a Sequentially Decomposed Programming (SDP) algorithm. SDP has two phases: when far from the solution, the first phase is enacted and decomposition is used; when close to the solution, the second phase is underway and decomposition is not used. The principles defining SDP are applied to Sequential Quadratic Programming (SQP) to define an SDP-SQP implementation. A global convergence proof and a simple example are given.

1 Introduction

1.1 Overview of Work and Related Research

Optimization methods have been applied with practical success to individual components of a system using well developed models and simulations. Arguably, this is because the physics, design goals, and other modeling issues are such that computer automation is a relatively straightforward task. At the component level, simulations are developed by the same people, with the same interests and therefore often have a consistency allowing the simulations to work well together.

However, difficulties arise when design must be performed at the system level, where a system is a collection of connected components or processes. Different computational models are coupled via common design quantities, and the utility of the design must reflect how the system performs as a whole.

If an optimal design problem is stated as a nonlinear program (NLP)

$$\begin{array}{lll} \min & f(\boldsymbol{x}) & \\ & g_i(\boldsymbol{x}) \le 0 & i = 1 \ldots m_{\text{ineq}} \\ \text{subject to} & & \\ & h_i(\boldsymbol{x}) = 0 & i = 1 \ldots m_{\text{eq}} \end{array} \quad (1)$$

where n, m_{ineq}, m_{eq}, are "large" and f, g_i, h_i, are nonlinear and computationally expensive to evaluate, then finding a solution $\boldsymbol{x}^*$ is inherently difficult. Many researchers are currently of the opinion that the inherent structure of the problem must be exploited in order to reliably solve large NLP, for example Conn et al.[1] and Papalambros.[2]

In order to exploit the structure of a design problem, the concept of "structure" must first be defined. Several such concepts exist, for example, Conn et al. (op cit.), Lootsma,[3] Dantzig and Wolfe,[4] Azarm and Li[5,6] and Sobiesky et al.[7] all work with different concepts of structure, each leading to a suitable optimization method. It is therefore necessary to use the concept of structure which most adequately describes the system.

This paper represents an ongoing investigation into the use of a particular type of structure (presented in Section 1.2) coupled with a particular class of algorithms (presented in Section 1.3).

1.2 Structure of NLP

The work presented here assumes structure to be based on optimal model-based partitioning as concieved by Wagner and Papalambros[8] and refined by Michelena and Papalambros.[9] When applied to large scale NLP (Eq. 1) functions are assigned vertices, and variables are assigned hyperedges in a hypergraph. A hyperedge is an entity that connects two or more vertices. Thus if two or more functions are dependent on the same variable, the corresponding vertices are connected with a hyperedge. A hypergraph is said to be *disjoint* when at least one vertex cannot be reached from at least one other vertex by following a path of hyperedges. A hypergraph is optimally partitioned when a predetermined number of

evenly sized, disjoint hypergraphs remain after the removal of a minimal number of hyperlinks.

If a suitable partition for an NLP is found, the functions and variables in Eq. 1 can be reordered and grouped, Eq. 2.

$$\begin{aligned} \min_{x \in \Re^n} \quad & f_0(x_0) + \sum_{j=1}^{p} f_j(x_0, x_j) \\ \text{s. t.} \quad & g_{0i_0}(x_0) \le 0 \\ & h_{0i_0}(x_0) = 0 \qquad j = 1 \ldots p \\ & g_{ji_j}(x_0, x_j) \le 0 \qquad i_j = 1 \ldots m_{j,\text{ineq}} \\ & h_{ji_j}(x_0, x_j) = 0 \qquad i_j = 1 \ldots m_{j,\text{eq}} \end{aligned} \tag{2}$$

The vector of variables $x_0 \in \Re^{n_0}$ which are common to most functions, corresponds to the removed hyperlinks, and is termed the vector of *linking variables*. If the linking variables are held constant, then Eq. 2 can be rewritten as the sum of p different independent subproblems, Eq. 3.

$$\begin{aligned} \min_{x_j \in \Re^{n_j}} \quad & f_j(x_0, x_j) \\ \text{s. t.} \quad & g_{ji_j}(x_0, x_j) \le 0 \qquad i_j = 1 \ldots m_{j,\text{ineq}} \\ & h_{ji_j}(x_0, x_j) = 0 \qquad i_j = 1 \ldots m_{j,\text{eq}} \end{aligned} \tag{3}$$

As an example, consider a problem which first appeared in Beightler and Phillips[10] and was used as a decomposition example in Wagner.[11]

$$\begin{aligned} \min_{R, L, t_s, t_h \ge 0.1} \quad & 0.662RLt_s + 1.777R^2t_h \\ & + 1.58Lt_s^2 + 19.48Rt_s^2 \\ \text{s. t.} \quad & g_1 = 0.131R - t_h \le 0 \\ & g_2 = 0.0193R - t_s \le 0 \\ & g_3 = 413000 - R^2L \le 0 \\ & g_4 = 0.00417L - 1.0 \le 0 \end{aligned} \tag{4}$$

Figure 1 displays the NLP in Eq. 4 as a hypergraph. The design quantities $\{R,\ L,\ t_h,\ t_s\}$ are drawn as amorphous shaded objects (the hyperlinks) which link every function that depends on the particular design variable. In other words, the hyperlink L connects the vertices $\{f_1,\ f_3,\ g_3,$ and $g_4\}$ because $\{f_1,\ f_3,\ g_3,$ and $g_4\}$ are all functions of L.

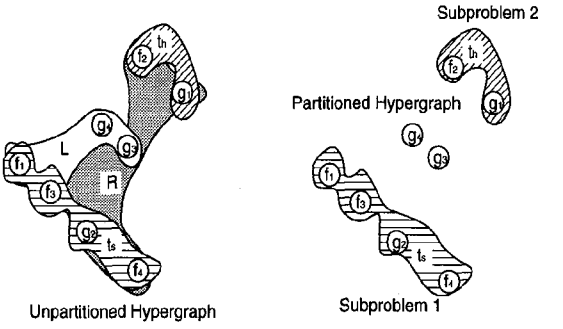

FIGURE 1: Hypergraph representation of Eq. 4 before (shown on the left) and after (shown on the right) partitioning

Decomposing the NLP is analogous to removing hyperlinks and finding disjoint subgraphs; namely, there is no path between certain sets of vertices. In Figure 1, the hyperlinks corresponding to R and L have been removed, so that no paths exist between the subgraphs corresponding to Subproblem 1 and Subproblem 2. Thus the two subsystems can be stated as two smaller NLP's.

$$\begin{aligned} \min_{t_s \ge 0.1} \quad & f_1 = 0.662RLt_s + 1.58Lt_s^2 \\ & 19.84Rt_s^2 \\ \text{s. t.} \quad & g_{1,1} = 0.0193R - t_s \le 0 \end{aligned} \tag{5}$$

$$\begin{aligned} \min_{t_h \ge 0.1} \quad & f_2 = 1.777R^2t_h \\ \text{s. t.} \quad & g_{2,1} = 0.131R - t_h \le 0 \end{aligned} \tag{6}$$

1.3 Structure of Algorithms

The appeal of the structure defined in Section 1.2 is that each subproblem is smaller (in dimension, etc.) and therefore easier to solve – provided that the optimal value of the linking variables x_0^* is known. In fact, this structure naturally leads to a hierarchical approach to solving the original NLP. Loosely speaking, a hierarchical approach is outlined by Model Algorithm 1.

Model Algorithm 1: Hierarchical Coordination

1. (Subproblems) Holding the linking variables constant at the value $x_0 = x_0^k$, for each subproblem $j = 1 \ldots p$, find the minimizer of the jth subproblem, $x_j^\dagger(x_0)$ (the solution to Eq. 3).
2. (Coordination) Find a value for x_0^{k+1} which will improve the (overall) objective and/or maintain feasibility.

3. Repeat until some convergence criteria are met.

Hierarchical coordination algorithms are often portrayed as a top-down authority structure as in the diagram in Figure 2. Many researchers have suggested approaches that fit within the framework of Model Algorithm 1 for engineering design problems (Sobiesky, (op cit.) Thareja and Haftka,[12] Vanderplaats et al.[13]).

However, Vanderplaats and Yoshida[14] have shown that $\partial x_j^{\dagger}(x_0)/\partial x_0$ need not be a continuous function, which can create difficulties in formulating a coordination problem. The discontinuities in $\partial x_j^{\dagger}(x_0)/\partial x_0$ occur because of a change in the active set of constraints, and Vanderplaats et al. (op cit.) reformulated the coordination problem to include all variables and constraints in a sequential linear programming approach in order to overcome this difficulty. One conclusion of Vanderplaats et al. was that decomposition will probably not lead to advances in computational efficiency.

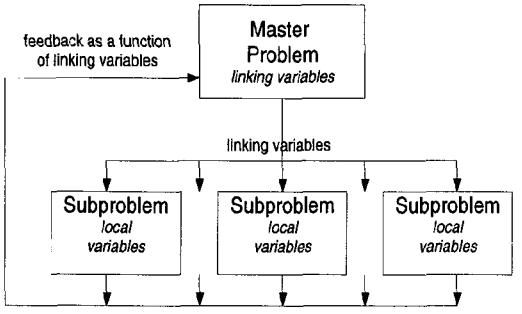

FIGURE 2: The top down authoritative structure of a hierarchical coordination strategy.

The work here stems from optimal model-based partitioning and has some similarities to the algorithms of Vanderplaats, et al. (op cit.), but is specifically tailored to address convergence and efficiency issues.

In Section 2, two additional theoretical concerns with coordination of hierarchical decomposition problems are presented. In Section 3, Sequentially Decomposed Programming (SDP) is outlined as a generic method that will overcome the trouble with discontinuities and the problems presented in Section 2. In Section 3.1 SDP is applied to Sequential Quadratic Programming to define SDP-SQP. In Section 4, a proof of global convergence is given. Section 5 gives an example of the performance of SDP, and Section 6 discusses open issues and the current state of ongoing work.

2 Difficulties with Decomposition: Hierarchical Coordination Strategies

Within the work presented here, the focus is on computational methods. If superscript k represents the iteration counter, then x^{k+1} depends on the values of $f(x^k)$, $g_i(x^k)$, $i = 1 \ldots m_{\text{ineq}}$, $h_i(x^k)$, $i = 1 \ldots m_{\text{eq}}$ and their respective gradients. It is important to remember that coordination methods for hierarchical strategies deal only with the direct consequences of changing the linking variables x_0. In other words, if $x_j^{\dagger}$ is the solution to the jth subproblem, it is necessary to estimate how the subproblems will behave, or how $x_j^{\dagger}$, f_j, g_{ji}, h_{ji}, will change with respect to x_0. For infinitesimal changes in the linking variable, the rate of change of $x_j^{\dagger}$, f_j, g_{ji} and h_{ji} are known as sensitivities. Sensitivities can be discontinuous when the active set of constraints changes (see Vanderplaats op cit.), but the following two subsections present other possible problems with hierarchically decomposed methods. The first argument focuses on the effectiveness of hierarchical algorithms with respect to global convergence issues, and the second argument focuses on local convergence issues.

2.1 Difficulties in Determining the Correct Active Constraints

Consider the linearly constrained nonlinear program in Eq. 7.

$$\begin{aligned} \min_{x \in \Re^2} \quad & 5x_0^2 - 2x_1x_0 + 4x_1^2 - 16x_0 \\ & - 12x_1 + 0.1e^{x_0 - 2} + 0.2e^{x_1 - 1} \\ & g_1 = 3 - x_1 \le 0 \\ \text{s. t.} \quad & g_2 = -3x_0/4 + x_1 - 1 \le 0 \\ & g_3 = -x_0 + 4x_1 + 2.5 \le 0 \end{aligned} \tag{7}$$

Eq. 7 is hierarchically decomposed into a master problem (Eq. 8) and a single subproblem (Eq. 9).

$$\min_{x_0 \in \Re} \quad 5x_0^2 - 16x_0 + 0.1e^{x_0 - 2} + f_1(x_0) \tag{8}$$

$$\begin{aligned} \min_{x_1 \in \Re} \quad & f_1 = -2x_1x_0 + 4x_1^2 - 12x_1 + 0.2e^{x_1 - 1} \\ & g_{1,1} = 3 - x_1 \le 0 \\ \text{s. t.} \quad & g_{1,2} = \frac{-3x_0}{4} + x_1 - 1 \le 0 \quad , p \\ & g_{1,3} = -x_0 + 4x_1 + 2.5 \le 0 \end{aligned} \tag{9}$$

At the point $x^1 = [\ 0\ 1\ ]$ (the linking variable $x_0 = 0$) we would like to know how f_1 changes as the linking variable x_0^1 is changed. The constraint $g_{1,2}$ is active at x^1, so by substitution

$$\left.\frac{dx_1}{dx_0}\right|_{\substack{g_{1,2}=0 \\ x_0=0}} = \frac{3}{4}$$

$$\left.\frac{df_1}{dx_0}\right|_{\substack{g_{1,2}=0 \\ x_0=0}} = \begin{pmatrix} 11.5x_0 - 21 + 0.1e^{x_0-2} \\ -0.15e^{0.75x_0} \end{pmatrix} = -20.86 \quad (10)$$

Thus at x^1 it is possible to decrease the value of f_1 by increasing the value of x_0. A hierarchical coordination scheme using sensitivities would be similar in form to Model Algorithm 2, shown below.

Model Algorithm 2: Coordination with Sensitivities

1. For ${x_0}^k$ find the minimizer ${x^\dagger}_j{}^k$ for each subproblem $j = 1 \ldots n_p$.
2. At $x^{\dagger k} = \left[{x_0}^k \; {x^\dagger}_1{}^k \; \ldots \; {x^\dagger}_p{}^k \right]$, calculate $df^\dagger{}_j({x_0}^k)/dx_0$.
3. Set the value of ${x_0}^{k+1}$ (and changing ${x_j}^{k+1}$, $j = 1 \ldots p$ as predicted through the use of sensitivities) within acceptable move limits in order to decrease the value of the objective function $f = f_0 + \sum f_j$ and/or maintain/attain feasibility.
4. Repeat until some convergence criteria are met.

Figure 3 displays the contours of the objective function and the lines along which the constraints are satisfied for the NLP in Eq. 7

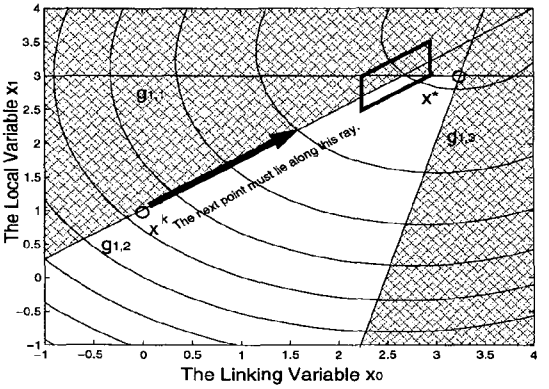

FIGURE 3: The contour lines and constraints of the NLP stated in Eq. 7.

Starting from x^k, the next point produced by Model Algorithm 2 must be along the ray (outlined in heavy black), which coincides with constraint $g_{1,2}$. Depending on the type of minimization used, the next point would probably be somewhere in the region marked by a parallelepiped. However, because of the definition of sensitivities, and the way sensitivities are incorporated into the coordination problem, the point (in this case, the solution x^* to the NLP) that identifies the correct set of constraints does not lie along constraint $g_{1,2}$.

Put in more formal terms, suppose that the active inequality constraints at the solution are indexed $i = 1 \ldots m^*_{\text{ineq}}$. Given a point x^k (not the solution), let χ^k represent the set of points which satisfies all of the equality constraints and all of the inequality constraints with indices $i = 1 \ldots m^*_{\text{ineq}}$ when linearized at the point x^k. Let X^k represent the set of points which can be reached using a particular coordination strategy, so that X^k is the feasible space of the coordination problem. If coordination is formulated as a decision making process with dimension n_0, then X^k is an affine subspace with dimension n_0 that includes the current point x^k.

In Figure 3, $\chi^1 \cap X^1 = \varnothing$, so it is not possible to correctly identify the active set of constraints without extra iterations. Thus, if $\chi^k \neq \varnothing$ then the coordination problem should be formulated such that $\chi^k \cap X^k \neq \varnothing$. (Note that this does not necessarily require the solution to the coordination problem identify the active set of constraints, only that the possibility should exist.)

It is possible to define sensitivities in terms of second order derivatives, but the underlying problem illustrated here still stands: If coordination is formulated as a decision making process with dimension n_0, then there are instances in which the coordination problem can not correctly identify the active set of constraints without extra iterations.

This is important because identifying the correct active set of constraints is both a condition eventually achieved by most algorithms and usually a sufficient condition for whatever local convergence behavior is appropriate (see, for example Schittkowski,[15] Powell,[16] Han[17]). In other words, in this simple example, coordinating with any approximate problem that used all of the variables and all of the constraints would have correctly identified the active constraints, whereas the coordination problem based on sensitivities could not. There are instances when using a larger-dimensional problem for coordination is prohibitive, but one must consider the number of iterations in a large NLP, especially if func-

tion calls are relatively expensive relative to solving the coordination problem. If function calls are expensive, then using a coordination method that takes more computing power to solve may be beneficial if fewer iterations are needed.

2.2 Using Decomposition in the Neighborhood of the Solution

In this section, it is assumed that the active constraint set has been correctly determined and that the current iterate x^k is in the neighborhood of the solution x^*. By neighborhood, it is meant that if some traditional optimization algorithm were used, then the observed local convergence rate would be in accord with the theoretical analysis. When iterations are close to the solution, the focus of analysis is on local convergence, so the distance between the current iteration x^k and the solution x^* as measured by the norm $\|x^k - x^*\|$ is of particular interest.

It is also assumed that whatever hierarchical algorithm is used, the algorithm fits the outline of Model Algorithm 1, so that given a starting point x_2^1 the sequence of points produced is $\{x^1, x^{\dagger 1}, x^2, x^{\dagger 2}, x^3, x^{\dagger 3}, \ldots\}$.

Eq. 11 is an unconstrained quadratic objective function, which has been decomposed into a master problem (Eq. 12) and a single subproblem (Eq. 13).

$$\min_{x \in \Re^2} \quad f = 1.5x_0^2 + 2x_0x_1 + 1.5x_1^2 \tag{11}$$

$$\min_{x_0} \quad f = 1.5x_0^2 + f_1(x_0) \tag{12}$$

$$\min_{x_1} \quad f_1 = 2x_0x_1 + 1.5x_1^2 \tag{13}$$

Figure 4 is a picture of the contour lines of Eq. 11. However, Figure 4 could be the contour lines of any convex function as seen in the plane uniquely determined by the solution x^*, the current iterate x^k and the iterate after solving all of the subproblems $x^{\dagger k}$.

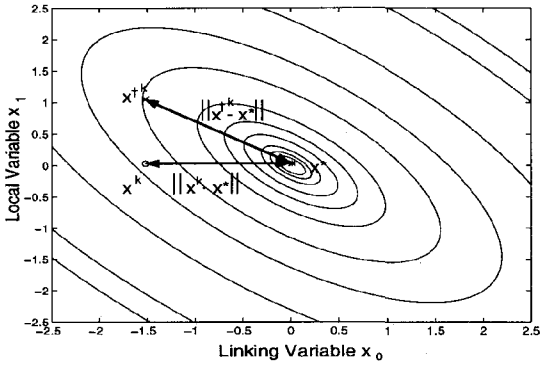

FIGURE 4: Contour lines for the quadratic objective function Eq. 11.

Starting from $x^k = [\ -1.5\ 0\ ]$, we see that $x^{\dagger k} = [\ -1.5\ 1\ ]$ is farther from the solution as measured by the norm $\|x - x^*\|$, despite the fact that $f(x^k) > f(x^{\dagger k})$. Figure 4 highlights the fact that solving the subproblems while keeping linking variables constant may actually increase the distance from the current iterate to the solution, even though the objective function has decreased. In general, this non-intuitive fact makes the construction of an algorithm which guarantees $\|x^k - x^*\| > \|x^{k+1} - x^*\|$ theoretically very difficult. Even worse, an acceptable convergence rate may be impossible to prove. Regardless of the algorithm, when close to the solution it would be more prudent to devote computational power to reduce $\|x - x^*\|$, as opposed to $f(x)$.

Both situations presented in Sections 2.1 and 2.2 are problems with solving a nonlinear program using only a specific subset of variables at a time.

3 Sequentially Decomposed Programming

Two theoretical difficulties with traditional hierarchical decomposed methods were presented: Expressing $x_j^{\dagger}(x_0)$ can make it difficult to identify the correct set of active constraints and that in the neighborhood of the solution, hierarchical decomposition methods may have problematic convergence rates; furthermore, as discussed earlier $\partial x_j^{\dagger}(x_0)/\partial x_0$ is not always a continuous function. Sequentially Decomposed Programming (SDP) is a method designed to overcome these difficulties using two distinct ideas.

First, in the coordination problem, sensitivities are not used. Instead, coordination is performed using an approximate problem (for instance a quadratic program) *which uses all of the constraints and all of the variables in the original problem.* This technique was originally used by Vanderplaats, Yang and Kim (op cit.) so that changes in constraint activity in the subproblems are accounted for, and the feasible space of the model prob-

lem is more likely to contain points which will correctly identify the active constraint set.

Second, when near a solution, decomposition (i.e., solving subproblems separately) is not performed. A specific test for determining when x^k is "near the solution", is discussed in Section 3.1. Sequentially Decomposed Programming is presented as Model Algorithm 3.

Model Algorithm 3: SDP

1. (Use of Decomposition to Solve Subproblems) If not near the solution, hold the linking variables constant at $x_0 = x_0^k$ and find a minimizer $x_j^{\dagger k}$ for each subproblem $j = 1 \ldots p$.
2. (Coordination) Using an approximate problem of the nonlinear program which includes all of the constraints and all of the variables in the original problem, find a satisfactory next point.
3. Continue until some convergence criteria are met.

If these two principles are followed by any optimization algorithm, we call this SDP. In Section 3.1 these two principles are applied to Sequential Quadratic Programming.

3.1 SDP using Sequentially Quadratic Programming

In order to apply SDP to SQP, a few notes concerning notation, structure, and sparsity must be made. First, the Lagrangian (Eq. 14) is expressed as a sum of terms (Eq. 15).

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m_i} \lambda_i g_i(x) + \sum_{i=1}^{m_e} \mu_i h_i(x) = \sum_{j=0}^{p} L_j(x_0, x_j, \lambda_j, \mu_j) \tag{14}$$

$$L_j(x_0, x_j, \lambda_j, \mu_j) = f_j(x_0, x_j) + \sum_{i=1}^{m_{j,\text{ineq}}} \lambda_{ji} g_{ji}(x) + \sum_{i=1}^{m_{j,\text{eq}}} \mu_{ji} h_{ji}(x) \tag{15}$$

Each of the j terms represented in Eq. 15 corresponds to the jth subproblem of the form in Eq. 3. The Hessian of the Lagrangian $\mathbf{H}$ can also be represented as a sum of sparse Hessians $\mathbf{H}_j$ as shown in Eq. 16, each corresponding to a subproblem.

$$\mathbf{H}_j = \nabla^2 L_j(x_0, x_j, \lambda_j, \mu_j) = \begin{bmatrix} \nabla_{x_0 x_0}{}^2 L_j & 0 & \nabla_{x_0 x_j}{}^2 L_j & 0 \\ 0 & 0 & 0 & 0 \\ \nabla_{x_j x_0}{}^2 L_j & 0 & \nabla_{x_j x_j}{}^2 L_j & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{j00} & 0 & \mathbf{H}_{j0j} & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{H}_{jj0} & 0 & \mathbf{H}_{jjj} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

The first subscripted index refers to the subproblem, and the second and third subscript refer to derivatives with respect to the corresponding set of design problems. $\mathbf{B}_j^k$ is then the approximation of $\mathbf{H}_j$ during the kth iteration, $\mathbf{B}_{jjj}^k$ is the approximation of $\mathbf{H}_{jjj}$, and $\mathbf{B}^k = \sum \mathbf{B}_j^k$ is the approximation of $\mathbf{H}$. SDP-SQP is defined as Model Algorithm 4. Step 1.3 is the classic update for the QP subproblems in SQP[16], but it is applied to the approximate Hessian as a whole, so that elements corresponding to x_0 are also approximated according to the quasi-Newton condition. Using the sparse structure outlined by Eq. 14 Model Algorithm 4 is an application of SDP to the classic SQP algorithm of Wilson,[19] Han (op cit.) and Powell. (op cit.)

Model Algorithm 4: SDP-SQP

1. (Using Decomposition to Solve the Subproblems) If during the past few outer iterations, the set of active constraints predicted by Eq. 19 has remained the same, then holding the linking variables constant, $x_0 = x_0^k$, find a minimizer $x_j^{\dagger k}$ for each subproblem $j = 1 \ldots p$ by repeating Step 1.1 through Step 1.4. Start with $k_j = 1$, $x_j^{1_j} = x_j^k$ and $\mathbf{B}_j^{1_j} = \mathbf{B}_j^k$.
   1.1 At $x_j^{k_j}$ solve for a search direction d_j using the quadratic program.

$$\min_{d_j \in \Re^{n_j}} \frac{1}{2} d_j{}^T \mathbf{B}_{jjj}^{k_j} d_j + \nabla f_j(x_0^k, x_j^{k_j}) d_j$$
$$\text{s. t.} \quad \nabla g_{ji}(x_0^k, x_j^{k_j}) d_j + g_{ji}(x_0^k, x_j^{k_j}) \le 0 \quad \nabla h_{ji}(x_0^k, x_j^{k_j}) d_j + h_{ji}(x_0^k, x_j^{k_j}) \le 0 \tag{17}$$

   1.2 Set $x_j^{k_j+1} = x_j^{k_j} + \alpha d_j$ where α is chosen to sufficiently decrease some appropriate merit function.
   1.3 Update $\mathbf{B}_j^{k_j+1}$ using Eq. 18.

$$\mathbf{B}_j^{k_j+1} = \mathbf{B}_j^{k_j} + \frac{zz^T}{z\delta} + \frac{(\mathbf{B}_j^{k_j}\delta)(\mathbf{B}_j^{k_j}\delta)^T}{\delta^T\mathbf{B}_j^{k_j}\delta}$$

$$\delta = \left[0 \ldots x_j^{k_j+1} \ldots 0\right] - \left[0 \ldots x_j^{k_j} \ldots 0\right]$$

$$y = \nabla L_j(x_0^k, x_j^{k_j+1}, \lambda^k, \mu^k) - \nabla L_j(x_0^k, x_j^{k_j}, \lambda^k, \mu^k)$$

$$z = \theta y + (1-\theta)(\mathbf{B}_j^{k_j}\delta) \tag{18}$$

$$\theta = \begin{cases} 1 & \text{if } \delta^T z \geq 0.2\delta^T\mathbf{B}_j^{k_j}\delta \\ \dfrac{0.8\delta^T\mathbf{B}_j^{k_j}\delta}{\delta^T\mathbf{B}_j^{k_j}\delta - \delta^T z} & \text{if } \delta^T z < 0.2\delta^T\mathbf{B}_j^{k_j}\delta \end{cases}$$

1.4 If convergence criteria are satisfied, set $x^{\dagger}{}_j^k = x_j^{k_j}$ and $\mathbf{B}^{\dagger}{}_j^k = \mathbf{B}_j^{k_j}$, otherwise repeat Step 1.1 through Step 1.3.

2. At $x^{\dagger k} = \left[\, x_0^k \; x^{\dagger}{}_1^k \; \ldots \; x^{\dagger}{}_{n_p}^k \,\right]$ and $\mathbf{B}^{\dagger k} = \sum \mathbf{B}^{\dagger}{}_j^k$, solve for a search direction $\boldsymbol{d}$ using the quadratic program.

$$\begin{aligned} \min_{\boldsymbol{d} \in \Re^n} \quad & \frac{1}{2}\boldsymbol{d}^T\mathbf{B}^{\dagger k}\boldsymbol{d} + \nabla f(x^{\dagger k})\boldsymbol{d} \\ \text{s. t.} \quad & \nabla g_i(x^{\dagger k})\boldsymbol{d} + g_i(x^{\dagger k}) \leq 0 \\ & \nabla h_i(x^{\dagger k})\boldsymbol{d} + h_i(x^{\dagger k}) \leq 0 \end{aligned} \tag{19}$$

3. Set $x^{k+1} = x^{\dagger k} + \alpha\boldsymbol{d}$ where α is chosen to sufficiently decrease some appropriate merit function.
4. For $j = 1 \ldots p$ update $\mathbf{B}_j^{k+1}$ using Eq. 18.
5. Repeat Steps 1 to 4 until convergence

Model Algorithm 4 incorporates both elements required in the definition of an SDP algorithm. First, in Step 2, coordination is performed using all variables and all constraints. Second, in Step 1, there is a test to ensure that the subproblems are not solved independently when x^k is near the solution. The test used in Model Algorithm 4 requires the active constraint set to remain the same. It is worthwhile to note that other possibilities exist (for example, that unit step-lengths are accepted, or that successive coordination steps are progressively shorter). It is expected that the test will be modified according to the application, as in the case of unconstrained minimization. The only requirement of the test is that as the algorithm converges, eventually the test should always be true.

Additionally, Model Algorithm 4 assembles the coordination problem using estimates obtained during the solution process of the subproblems. (This is not a requirement of SDP, but it is expected to help.) This is accomplished by expressing the Lagrangian as a sum of terms (as in Eq. 14) and updating a sparse approximation for each term. The idea stems from the work of Griewank and Toint,[18] who have shown that for the unconstrained case, the update defined by Eq. 18 retains the null space of $\mathbf{B}_j^k$. For the case at hand, retaining the null space essentially retains the sparsity pattern shown in Eq. 16. To the authors' knowledge, this has not been applied to the constrained case, but it is hoped that the use of Eq. 18 will provide for better Hessian estimates in fewer iterations.

4 A Proof of Global Convergence

Luenberger[20] has proven that SQP or Modified quasi-Newton methods converge via a descent sequence to a point x^* which satisfies the first order KKT conditions. For simplicity, the proof assumes only inequality constraints, but the extension to equality constraints is straightforward. The proof consists of two theorems. The first theorem establishes the limit of any convergent subsequence of points generated by SQP is a solution, provided that a suitable merit function exists. The second theorem shows that the absolute value penalty function is a suitable merit function.

The new third theorem presented here establishes that points generated during the course of the subproblems also represent a descent sequence of the same merit function, so that $\{x^1, x^{\dagger 1}, x^2, x^{\dagger 2}, x^3, x^{\dagger 3}, \ldots\}$ is a descent sequence, thus proving that SDP-SQP converges to a point satisfying first order KKT optimality conditions through a descent sequence.

Theorem 1: Global Convergence Theorem (Luenberger)

Let $A(x)$ be an algorithm on some space χ, and suppose that, given x^1, the sequence $\{x^k\}_{k=1}^{\infty}$ is generated satisfying $x^{k+1} \in A(x^k)$.

Let a solution set $\Gamma \subset \chi$ be given, and suppose

1. All points x^k are contained in a compact set $S \subset \chi$
2. there is a continuous function P on χ such that if $x \notin \Gamma$ then $P(y) < P(x)$ for all $y \in A(x)$, and if $x \in \Gamma$ then $P(y) \leq P(x)$ for all $y \in A(x)$.
3. the mapping A is closed at points outside Γ.

Then the limit of any convergent subsequence of $\{x^k\}$ is a solution.

Theorem 2: Use of the Absolute Value Penalty Function (Luenberger)

Let $\boldsymbol{d}$, μ, (with $\boldsymbol{d} \neq \boldsymbol{0}$) be a solution of the quadratic program (Eq. 17). Then if $c > \max\{\mu_j\}$ the vector $\boldsymbol{d}$ is a descent direction for the penalty function.

$$P(\boldsymbol{x}) = f(\boldsymbol{x}) + c \sum_{i=1}^{m_{\text{ineq}}} g_i^{+}(\boldsymbol{x}) \tag{20}$$

where $g^{+} \equiv \max\{0, g\}$.

Theorem 3: Solutions to Subproblems are also descent.

Let $\boldsymbol{d}_j$, μ, with $\boldsymbol{d}_j \neq \boldsymbol{0}$ be a solution of the quadratic program used in the solution process of the subproblems (Eq. 19). As a convention, let $\boldsymbol{d} = \left[0 \ \ldots \ \boldsymbol{d}_j \ \ldots \ 0\right]$. Then if $c > \max\{\mu_{ji}\}$, the vector $\boldsymbol{d}$ is a descent direction for the same absolute value penalty function stated in Eq. 20.

Proof of Theorem 3

Let $I_{j,\text{ineq}}(\boldsymbol{x})$ denote the inequality constraints for the jth subproblem which are violated at the point $\boldsymbol{x}$. The penalty function P is written as a function of the scalar α.

$$P(\boldsymbol{x} + \alpha\boldsymbol{d}) \equiv f(\boldsymbol{x} + \alpha\boldsymbol{d}) + c \sum_{i=1}^{m_{\text{ineq}}} g_i^{+}(\boldsymbol{x} + \alpha\boldsymbol{d}) \tag{21}$$

Note that all of the constraints are used in Eq. 21 and that $f = \sum f_j$. Note also that all functions (objective and constraint) which are not in the jth subproblem will not change in value because they do not depend on $\boldsymbol{x}_j$.

$$f(\boldsymbol{x} + \alpha\boldsymbol{d}) = f(\boldsymbol{x}) - f_j(\boldsymbol{x}) + f_j(\boldsymbol{x} + \alpha\boldsymbol{d}) \tag{22}$$

$$\text{if } J \neq j \text{ then } g_{Ji}(\boldsymbol{x} + \alpha\boldsymbol{d}) = g_{Ji}(\boldsymbol{x}) \tag{23}$$

$$P(\boldsymbol{x} + \alpha\boldsymbol{d}) = f(\boldsymbol{x}) - \alpha\nabla f_j(\boldsymbol{x})\boldsymbol{d} + \tag{24}$$

$$c\left(\sum_{i=1}^{m_{j,\text{ineq}}} \left(g_{ji}(\boldsymbol{x}) + \alpha\nabla g_{ji}(\boldsymbol{x})\boldsymbol{d}\right)^{+} + \sum_{\substack{J=1 \\ J\neq j}}^{p} \sum_{i=1}^{m_{J,\text{ineq}}} g_{Ji}^{+}(\boldsymbol{x})\right) + o(\alpha)$$

Two of the first order KKT conditions for the solution of the quadatic approximation for the subproblem (Eq. 17) are

$$\nabla f_j(\boldsymbol{x}) = -\mathbf{B}_{jjj}\boldsymbol{d}_j + \sum_{i=1}^{m_{j,\text{ineq}}} \mu_{ji}\nabla g_{ji}(\boldsymbol{x}) \tag{25}$$

$$\nabla g_{ji}(\boldsymbol{x})\boldsymbol{d}_j \leq -g_{ji}(\boldsymbol{x}) \tag{26}$$

allowing the relations shown in Eq. 27 and Eq. 28.

$$\sum_{i \in I_{j,\text{ineq}}} \nabla g_{ji}(\boldsymbol{x})\boldsymbol{d} \leq \sum_{i=1}^{m_{j,\text{ineq}}} -g_{ji}^{+}(\boldsymbol{x}) \tag{27}$$

$$\nabla f_j(\boldsymbol{x})\boldsymbol{d} \leq -\boldsymbol{d}_j^{T}\mathbf{B}_{jjj}\boldsymbol{d}_j + \max\{\mu_{ji}\} \sum_{i=1}^{m_{j,\text{ineq}}} g_{ji}^{+}(\boldsymbol{x}) \tag{28}$$

Substituting Eq. 27 and Eq. 28 into Eq. 24 gives Eq. 29.

$$P(\boldsymbol{x} + \alpha\boldsymbol{d}) \leq P(\boldsymbol{x}) + \tag{29}$$

$$\alpha\left(-\boldsymbol{d}_j^{T}\mathbf{B}_{jjj}\boldsymbol{d}_j + (c - \max\{\mu_{ji}\}) \sum_{i=1}^{m_{j,\text{ineq}}} g_{ji}^{+}(\boldsymbol{x})\right) + o(\alpha)$$

Because $\mathbf{B}_{jjj}$ is constructed to be positive definite and $c > \max\{\mu_j\}$ it follows that for a sufficiently small α,

$$P(\boldsymbol{x} + \alpha\boldsymbol{d}) \leq P(\boldsymbol{x}) \tag{30}$$

Thus each subproblem reduces the penalty function.

5 Comparison by Example

As a small example, both SQP and SDP-SQP are applied to a scaled version of the NLP in Eq. 4. The variable transformations are given in Eq. 31.

$$\begin{aligned} [x_{01}, x_{02}] &= [R, 0.1L] \\ [x_{11}] &= [10t_s] \\ [x_{21}] &= [t_h] \end{aligned} \tag{31}$$

The SQP algorithm used is the routine "constr" in the MATLAB optimization toolbox[21], and the SDP-SQP routine was also written in MATLAB. The collection of objective functions and constraints for $j = 0$ is given in Eq. 32.

$$f_0 = 0$$

$$g_{00} = 1 - \frac{x_{01}^2 x_{02}}{413000} \le 0 \qquad (32)$$

$$g_{01} = x_{02} - \frac{1.0}{0.0417} \le 0$$

The two subproblems appear as Eq. 33 (subproblem 1) and Eq. 34 (subproblem 2).

$$\min_{x_{11} \ge 1} \quad f_1 = \begin{pmatrix} 0.662 x_{01} x_{02} x_{11} + \\ 0.0158 x_{02} x_{11}^2 + \\ 0.1984 x_{01} x_{11}^2 \end{pmatrix} \qquad (33)$$

$$\text{s. t.} \quad g_{11} = 0.0193 x_{01} - t_s \le 0$$

$$\min_{x_{21} \ge 0.1} \quad f_2 = 1.777 x_{01}^2 x_{21} \times 10^{-4} \qquad (34)$$

$$\text{s. t.} \quad g_{21} = 0.131 x_{01} - x_{21} \le 0$$

Both algorithms use the starting point $\boldsymbol{x} = [60, 20, 40, 15]$. The iteration history is shown as a function of the outer iteration in Figure 5.

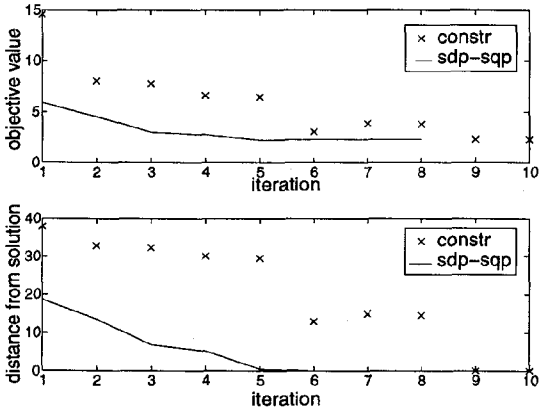

FIGURE 5: Iteration history of the SDP-SQP and plain SQP algorithm. The distance from the solution is measured by the euclidian norm in the x-space.

SDP-SQP used decomposition for the first 2 iterations, and converged upon the solution in a total of 8 iterations, compared to SQP which performed 11 iterations. Additionally, SDP-SQP evaluated subproblem 0 eight times, subproblem 1 ten times and subproblem 2 twelve times, whereas SQP made 11 evaluations of all three subproblems.

The example problem is not of the scale intended for SDP-SQP. However, the results shown here are promising, and larger applications may show significant benefit.

6 Open Issues and Conclusions

This presentation has intentionally divided the application of Sequentially Decomposed Programming into two areas. First the general definition of SDP was given in Section 3 as Model Algorithm 3. Second, the principles of SDP were then applied to Sequential Quadratic Programming in Section 3.1 as Model Algorithm 4. As this investigation is ongoing, the open issues concerning both SDP and SDP-SQP are addressed separately.

6.1 SDP

Upon reviewing Section 2, it is pertinent to wonder if decomposition is computationally useful. To this end, a few comments should be made.

First, SDP is defined as a two phase process, and the number of iterations in the initial phase (which uses decomposition to independently solve the subproblems) is not known a priori, so further computational experience with SDP is necessary. If measured in computation time, the length of time spent in the first phase determines the benefit of SDP. However, the transition to the second phase (where decomposition is not directly used) can be based on any test that will eventually become true during the course of the algorithm.

Second, even if the first phase does not last long enough to warrant the effort of implementing SDP when measured in terms of function evaluation and computational time, it is possible to tailor the algorithm such that the coordination problem uses approximations compiled during the solution process of the subproblems. In fact, using such approximations may actually increase the robustness because the approximations used during coordination would be more accurate.

Third, optimization at the component level is already widely used in industry. Through years of experience, these optimization routines have been finely tuned and their performance is well understood. A hierarchically decomposed method such as SDP can integrate already-used optimization routines without significantly reworking and recoding the system model or the optimization algorithm.

Fourth, the human aspect of engineering requires understanding the solution of any engineering design problem. If the problem has been decomposed, then the interaction between subproblems may be understood better by examining the behavior of the individual subproblems. Indeed this may be the only way for an indi-

vidual to understand the behavior of a very large engineering design problem.

6.2 SDP - SQP

As defined by Model Algorithm 4, SDP-SQP provides a new class of optimization algorithms, and there are both theoretical and practical issues that still need to be addressed.

Defining the approximation of the Hessian as a sum of sparse matrices is not new. The original definition used in EQ, stems from a combination of work by Broyden,[22] Fletcher,[23] Goldfarb,[24] and Shanno,[25] Powell (op cit.), and Greiwank and Toint (op cit.). However, theoretical works by Greiwank and Toint establishing local convergence rate for unconstrained optimization problems assume that the underlying functions are convex on $\Re^n$. Hopefully this can be relaxed so the point where L_j need only be convex on $\Re^{n_j}$, and $L = \sum L_j$ is strictly locally convex near the solution on $\Re^n$. It is hoped that understanding the convexity requirements of L_j will lead to a local convergence analysis theorem.

Furthermore, there is very little experience with applying Model Algorithm 4 to practical engineering problems where issues such as parallelism, computation time, and the ratio of functions call time to QP solution time and problem size become more concrete issues.

6.3 Conclusion

The graph partitioning methods developed for model-based decomposition[8,9,11] of optimal designs have removed much of the ad hoc process that characterized earlier decomposition approaches. Many different decomposition forms can be discovered formally based on rigorous methodologies. However, the actual solution of decomposed nonlinear programming problems has been shown to present important convergence difficulties. This article attempted to place these difficulties in a formal context and to define generic algorithms that may posses desirable convergence properties. In particular SDP is any hierarchical coordination algorithm with two defining properties: First, coordination of subproblems is performed using all variables and all constraints. Second, at some time before convergence, the subproblems are not solved indepently.

The SDP algorithm properties are currently under further investigation

7 Acknowledgement

This research has been partially supported by the Automotive Research Center at the University of Michigan, a US Army Center of Excellence in Modeling and Simulation of Ground Vehicles, under Contract No. DAAE07-94-C-R094. This support is gratefully acknowledged. Additionally, the authors are also grateful to the Professor Spillios Fassois, Antony Kontos, and the Univerisity of Patras for providing an excellent research atmosphere for the first author and supporting the sabbatical leave of the second author.

8 References

1. Conn, A. R., Gould, N. I. M. and Toint, Ph. L., *LANCELOT, A Fortran Package for Large-Scale Nonlinear Optimization*. Springer-Verlag, 1992.
2. Papalambros, P. Y., "Optimal design design of mechanical engineering systems," *Transactions of the ASME*, Vol. 117, 1995, pp. 55-62.
3. Lootsma, F. A., "Exploitation of structure in nonlinear optimization," *Parallel Computing*, 1990, pages 31-45.
4. Dantzig, G. B., and Wolfe, P., "Decomposition Principle for Linear Programs," *Operations Research*, Vol. 8, 1960, pages 101-111.
5. Azarm, S., and Li, W. "A two level decomposition method for design optimization," *Engineering Optimization*, Vol. 13, 1988, pp. 211-224.
6. Azarm, W., and Li, W. "Optimality and constrained derivatives in two-level design optimization," *Transactions of the ASME: Journal of Mechanical Design*, Vol. 112, 1990, pp. 563-568.
7. Sobieszczanski-Sobieski, J., James B. B., and Riley, M. F., "Structural sizing by generalized multilevel optimization," *AIAA Journal*, Vol. 25, No. 1, 1987, pp 139-145.
8. Wagner, T. C., and Papalambros, P. Y., "A general framework for decomposition analysis in optimal design," *Advances in Design Automation*, Vol. 2, 1993, pp. 315-25.
9. Michelena N., and Papalambros, P. Y., "Optimal model-based decomposition of powertrain system design," *Transactions of ASME, Journal of Mechanical Design*, Vol. 117, No. 4, 1995, pp. 499-505.
10. Beightler, C., and Phillips, D. T.. *Applied Geometric Programming*. Wiley, 1976.
11. Wagner, T. C., "A general decomposition methodology for optimal system design," PhD thesis, Univerisity of Michigan, 1993.
12. Thareja, R. R., and Haftka, R. T., "Efficient single-level solution of hierarchical problems in structural optimization," *AIAA Journal*, Vol. 28, No. 3, 1990, pp. 506-514.

13. Vanderplaats, G. N., Yang, Y. J., and Kim, D. S., "Sequential linearization method for multilevel optimization," *AIAA Journal*, Vol. 28, No. 2, 1990, pp. 290-295.
14. Vanderplaats, G. N., and Yoshida, N., "Efficient calculation of optimum design sensitivity," *AIAA Journal*, Vol. 23, No. 11, 1985, pp. 1798-1803.
15. Schittkowski, K., "The Nonlinear Programming Method of Wilson, Han, and Powell with an Augmented Lagrantian Type Line Search Function," *Numerische Mathematik*, Vol. 38, 1981, 83-114.
16. Powell, M. J. D., "The convergence of variable metric methods for nonlinear constrained optimization calculations," *Nonlinear Programming 3*, Academic Press, 1978, pp. 27-64.
17. Han, S. P., "Superlinearly convergent variable metric algorithms for general nonlinear programming problems," *Mathematical Programming*, Vol. 11, 1976, pp.263-282.
18. Griewank, A., and Toint, Ph. L., "Partitioned variable metric updates for large structured optimization problems," *Numerische Mathematik*, Vol. 39, 1982, pp. 119-137.
19. Wilson, R. B., "A simplical algorithm for concave programming," PhD thesis, Graduate School of Business Administration, Harvard University, 1963.
20. Luenberger, D. G., *Linear and Nonlinear Programming*, Addison-Wesley, 1984.
21. Grace, A., *MATLAB Optimization Toolbox Manual*, The Math Works Inc., 1994.
22. Broyden, C. G., "The convergence of a class of double-rank minimization algorithms," *Journal for the Institute of Mathematics and it's Applications*, Vol. 6, 1970, pp. 76-90.
23. Fletcher, R., "A new approach to variable metric algorithms," *Computer Journal*, Vol. 13, 1970, pp. 317-322.
24. Goldfarb, D., "A family of variable metric methods derived by variational means," *Mathematics of Computation* Vol. 24, 1970, pp. 23-26.
25. Shanno, D. F., "Conditioning of quasi-Newton methods for function minimization" *Mathematics of Computation*, Vol. 24, 1970, pp. 647-657.