JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 82, No. 3, SEPTEMBER 1994

A Sparse Nonlinear Optimization Algorithm[1]

J. T. BETTS[2] AND P. D. FRANK[3]

Communicated by H. Y. Huang

Abstract. One of the most effective numerical techniques for solving nonlinear programming problems is the sequential quadratic programming approach. Many large nonlinear programming problems arise naturally in data fitting and when discretization techniques are applied to systems described by ordinary or partial differential equations. Problems of this type are characterized by matrices which are large and sparse. This paper describes a nonlinear programming algorithm which exploits the matrix sparsity produced by these applications. Numerical experience is reported for a collection of trajectory optimization problems with nonlinear equality and inequality constraints.

Key Words. Sparse nonlinear programming, sequential quadratic programming, trajectory optimization.

1. Introduction

Nonlinear programming problems arise naturally in data fitting applications and when discretization techniques are applied to systems described by ordinary or partial differential equations. For applications of this type, the number of variables and constraints may be large (i.e., $100 < N < 100000$), and the corresponding Jacobian and Hessian matrices are very sparse (i.e., less than 1% of the elements are nonzero). For small problems with dense matrices, one of the most successful numerical techniques is the sequential quadratic programming approach. However, when algorithms appropriate for dense applications are applied to large sparse problems, the computational expense is dominated by the solution of the quadratic

[1]The authors wish to acknowledge the insightful contributions of Dr. William Huffman.

[2]Senior Principal Scientist, Mathematics and Engineering Analysis Department, Research and Technology Division, Boeing Computer Services, Seattle, Washington.

[3]Senior Principal Scientist, Mathematics and Engineering Analysis Department, Research and Technology Division, Boeing Computer Services, Seattle, Washington.

0022-3239/94/0900-0519$07.00/0

programming subproblem and the evaluation of the Hessian matrices. This paper describes an algorithm for solving large sparse nonlinear programming problems, which addresses the deficiencies of methods suitable for small dense problems. The method is designed to solve general nonlinear programming problems. In particular it is not necessary to assume that the number of degrees of freedom is small or that the constraints have a special structure. When used in conjunction with a sparse finite difference technique for computing the Hessian and Jacobian matrices, the overall approach is an especially efficient method for solving the discretized optimal control problem. Numerical tests also suggest that a limited memory secant method can provide significant gains if sparse finite differencing is not beneficial.

After defining the nonlinear programming problem in Section 2, the sparse nonlinear programming method is stated in Section 3. Section 4 describes in more detail how the quadratic programming subproblem is solved, and Section 5 presents a method for constructing a Hessian approximation. Extensive numerical results are presented in Section 6, and concluding remarks are found in Section 7.

2. Nonlinear Programming Problem

The nonlinear programming problem can be stated as follows: Find the N-vector x which minimizes the objective function

$$f = f(x), \tag{1}$$

subject to the constraints

$$c_L \leq c(x) \leq c_U, \tag{2}$$

where $c(x)$ is an m-vector of constraint functions, and the simple bounds

$$x_L \leq x \leq x_U. \tag{3}$$

Equality constraints are imposed by setting $c_L = c_U$, and variables can be fixed by setting $x_L = x_U$. It will be assumed that the objective and constraint functions are twice continuously differentiable, although the derivatives may be difficult to compute.

The solution point x^* must satisfy the Kuhn–Tucker necessary conditions for a local minimum:

(i) x^* is feasible, i.e., (2) and (3) are satisfied;

(ii) there exist Lagrange multipliers λ and v such that

$$g = G^T\lambda + v, \tag{4}$$

where $\nabla_x f(x) = g(x) = g$ is the N-dimensional gradient vector and G is the $m \times N$ Jacobian matrix of constraint gradients;

(iii) the Lagrange multiplier for a constraint or variable active at its lower bound must be nonnegative;
(iv) the Lagrange multiplier for a constraint or variable active at its upper bound must be nonpositive;
(v) the Lagrange multiplier for a strictly feasible constraint or free variable must be zero.

3. Sparse Nonlinear Programming Algorithm

The solution of a nonlinear program can be accomplished in a wide variety of ways. The basic approach utilized by the algorithm is to solve a sequence of quadratic programming subproblems. The fundamental premise of the approach is to approximate the nonlinear constraint functions by a linear model, and the general objective function by a quadratic model. First, background on the quadratic programming subproblem, and the associated definition of a merit function are presented. Then, a description of three distinct optimization strategies will be given.

3.1. QP Subproblem. A primary feature of the nonlinear programming algorithm to be described is the ability to solve a quadratic programming (QP) subproblem. Solution of the QP subproblem is used to define new estimates for the variables according to the formula

$$\bar{x} = x + \alpha p, \tag{5}$$

where the vector p is referred to as the search direction. The scalar α determines the step length and is initialized to one. The search direction p is found by minimizing the quadratic

$$g^T p + (1/2) p^T H p, \tag{6}$$

subject to the linear constraints

$$b_l \leq \begin{bmatrix} Gp \\ p \end{bmatrix} \leq b_u, \tag{7}$$

where H is a symmetric $N \times N$ positive-definite approximation to the Hessian matrix. The upper-bound vector is defined by

$$b_u = \begin{bmatrix} c_U - c \\ x_U - x \end{bmatrix}, \tag{8}$$

with a similar definition for the lower-bound vector b_l. The technique for solving this quadratic program when the relevant matrices are large and sparse will be described in the next section.

3.2. Merit Function. When a quadratic program is used to approximate a general nonlinearly constrained problem, it may be necessary to adjust the steplength α in order to achieve sufficient reduction in a merit function that in some way combines the objective function and constraint violations.

The merit function which we use is similar to that proposed by Gill, Murray, Saunders, and Wright in Ref. 1 and is related to the function given by Rockafellar in Ref. 2,

$$M(x, \lambda, v, s, t) = f - \lambda^T(c - s) - v^T(x - t) + (1/2)(c - s)^T Q(c - s) + (1/2)(x - t)^T R(x - t). \tag{9}$$

The diagonal penalty matrices are defined by $Q_{ii} = \rho_i$ and $R_{ii} = \gamma_i$. Observe that the merit function differs from that given in Ref. 1 by inclusion of terms for the bounds and linear constraints. One of the algorithm options described below maintains feasibility for the equality constraints during the iterative process and thus does not rely on the quadratic program to maintain feasibility of the linear constraints. For this merit function, the slack variables s at the beginning of a step are defined by

$$s_i = \begin{cases} c_{Li}, & \text{if } c_{Li} > c_i - \lambda_i/\rho_i, \\ c_i - \lambda_i/\rho_i, & \text{if } c_{Li} \le c_i - \lambda_i/\rho_i \le c_{Ui}, \\ c_{Ui}, & \text{if } c_i - \lambda_i/\rho_i > c_{Ui}, \end{cases} \tag{10}$$

$$t_i = \begin{cases} x_{Li}, & \text{if } x_{Li} > x_i - v_i/\gamma_i, \\ x_i - v_i/\gamma_i, & \text{if } x_{Li} \le x_i - v_i/\gamma_i \le x_{Ui}, \\ x_{Ui}, & \text{if } x_i - v_i/\gamma_i > x_{Ui}. \end{cases} \tag{11}$$

These expressions for the slack variables yield a minimum value for the merit function M, for given values of the variables x, λ, v and penalty weights, subject to the bounds on the slacks. The search direction in the real varaibles x as given by (5) is augmented to permit the multipliers and the slack variables to vary according to

$$\begin{bmatrix} \bar{x} \\ \bar{\lambda} \\ \bar{v} \\ \bar{s} \\ \bar{t} \end{bmatrix} = \begin{bmatrix} x \\ \lambda \\ v \\ s \\ t \end{bmatrix} + \alpha \begin{bmatrix} p \\ \xi \\ \eta \\ q \\ \delta \end{bmatrix}. \tag{12}$$

The multiplier search directions ξ and η are defined using the QP multipliers μ and ω according to

$$\xi \equiv \mu - \lambda, \tag{13}$$

$$\eta \equiv \omega - v. \tag{14}$$

From the QP (6)–(8), the predicted slack variables are just

$$\bar{s} = Gp + c = s + q. \tag{15}$$

Using this expression, define the slack vector step by

$$q = Gp + (c - s). \tag{16}$$

A similar technique defines the bound slack vector search direction

$$\delta = p + (x - t). \tag{17}$$

Note that, when a full step is taken $\alpha = 1$, the updated estimate for the Lagrange multipliers $\bar{\lambda}$ and $\bar{v}$ are just the QP estimates μ and ω. The slack variables s and t are just the linear estimates of the constraints, and the terms $c - s$ and $x - t$ in the merit function are measures of the deviation from linearity.

3.3. Parameter Definitions. In Ref. 1, it is shown that the penalty weights ρ_i and γ_i are finite provided the Hessian matrix H used in the QP subproblem is positive definite. For nonlinear programming applications, the Hessian of the Lagrangian,

$$H_L = \nabla_x^2 f - \sum_{i=1}^{m_z} \lambda_i \nabla_x^2 c_i, \tag{18}$$

can be constructed; however, in general it is not positive definite. In fact, it is only necessary that the projected Hessian of the Lagrangian be positive definite at the solution with the correct active set of constraints. Consequently, for the QP subproblem, we use the modified matrix

$$H = H_L + \tau(|\sigma| + 1)I. \tag{19}$$

The Levenberg parameter τ is chosen such that $0 \leq \tau \leq 1$ and is normalized using the Gerschgorin bound for the most negative eigenvalue of H_L, i.e.,

$$\sigma = \min_{1 \leq i \leq N} \left\{ h_{ii} - \sum_{i \neq j}^{N} |h_{ij}| \right\}, \tag{20}$$

and h_{ij} is used to denote the nonzero elements of H_L.

The proper choice for the Levenberg parameter τ can greatly affect the performance of the nonlinear programming algorithm. Quadratic convergence can only be obtained when $\tau = 0$ and the correct active set has been identified. On the other hand, if $\tau = 1$, in order to guarantee a positive-definite Hessian, the search direction p is significantly biased toward a gradient direction and convergence is degraded. A strategy similar to that used for adjusting a trust region (cf. Ref. 3) is employed by the algorithm to maintain a current value for the Levenberg parameter τ and adjust it

from iteration to iteration. The inertia (i.e., the number of positive, negative, and zero eigenvalues) of the related KT matrix described in the next section is used to infer that the projected Hessian is positive definite. Basically, the philosophy is to reduce the Levenberg parameter when the predicted reduction in the merit function agrees with the actual reduction, and increase it when the agreement is poor. The process is accelerated by making the change in τ proportional to the observed rate of change in the projected gradient.

Although the Levenberg parameter is used to ensure that the projected Hessian approximation is positive definite, it is still necessary to define the penalty weights Q and R. In Ref. 1, it is shown that convergence of the method requires choosing the weights such that

$$M_0' \leq -(1/2)p^T H p, \tag{21}$$

where M_0' denotes the directional derivative of the merit function (9) with respect to the steplength α evaluated at $\alpha = 0$. To achieve this, let us define

$$r_i = \begin{cases} \rho_i - \rho_0, & \text{if } 1 \leq i \leq m, \\ \gamma_{i-m} - \rho_0, & \text{if } m < i \leq m + N, \end{cases} \tag{22}$$

where $\rho_0 > 0$ is a strictly positive threshold. Since (21) provides a single condition for the $m + N$ penalty prameters, we make the choice unique by minimizing the norm $\|r\|_2$. After some lengthy algebra, we find that

$$r = a(a^T a)^{-1}\varsigma, \tag{23}$$

where

$$a_i = \begin{cases} (c_i - s)^2, & \text{if } 1 \leq i \leq m, \\ (x_{i-m} - t_{i-m})^2, & \text{if } m < i \leq m + N, \end{cases} \tag{24}$$

and

$$\begin{aligned} \varsigma = &-(1/2)p^T H p + \mu^T q + \omega^T \delta - 2\xi^T (c - s) - 2\eta^T (x - t) \\ &- \rho_0 (c - s)^T (c - s) - \rho_0 (x - t)^T (x - t). \end{aligned} \tag{25}$$

Typically, the threshold parameter ρ_0 is set to machine precision and only increased if the minimum norm solution is zero. In essence then, the penalty weights are chosen to be as small as possible consistent with the descent condition (21).

3.4. Algorithm Strategy. The design of an efficient, yet robust nonlinear programming algorithm is affected by a number of (possibly conflicting) factors. For example, the algorithm described in Ref. 4 is a feasible region method, since successive iterates maintain constraint feasibility. This

philosophy is motivated by a number of considerations. The trajectory optimization applications of interest are characterized by a relatively large number of equality constraints, derived from the dynamics. When the constraints are satisfied, the variables describe a valid trajectory for a fixed discretization history. Mesh refinement to improve the accuracy of the discretization is meaningful when performed about a valid trajectory and may not be elsewhere. Secondly, vehicle characteristics (e.g., aerodynamic and propulsion data) are usually only valid in regions bout real trajectories. Finally in practice, many problems are poorly posed, and this situation is readily detected when attempting to locate a feasible point.

In contrast, the algorithm described in Ref. 5 does not produce a feasible point until the solution is obtained. For a well-posed problem, a strategy which takes direct steps toward the solution without recourse to intermediate constraint satisfaction may be more efficient, especially for very nonlinear constraints. Furthermore, maintaining strict feasibility with respect to inequality constraints may be prohibitively slow when the number of inequalities is large and the active set is wrong. In order to efficiently deal with the combinatorial nature of the problem, it is desirable to correctly identify the active set at points well removed from the solution. However, for very nonlinear constraints, it may be necessary to use very large penalty weights to achieve feasibility, thus introducing the possibility for numerical instability.

In order to explore the conflicting benefits of these alternate strategies, three different approaches will be investigated.

(M) Minimize. Beginning at x^0, solve a sequence of quadratic programs until the solution x^* is found.

(FM) Find a feasible point, then minimize. Beginning at x^0, solve a sequence of quadratic programs to locate a feasible point x^f, and then beginning from x^f solve a sequence of quadratic programs until the solution x^* is found.

(FME) Find a feasible point, then minimize subject to equalities. Beginning at x^0, solve a sequence of quadratic programs to locate a feasible point x^f, and then beginning from x^f solve a sequence of quadratic programs while maintaining feasible equalities until the solution x^* is found.

Philosophically, the first strategy is probably the most aggressive, while the last strategy is probably the most conservative.

3.5. Finding a Feasible Point. The first step in either the FM or FME strategy is to determine a point which is feasible with respect to the

constraints. A fourth strategy F, to locate a feasible point only, is also available in the software. The approach employed is to take a series of steps of the form given by (5) with the search direction computed to solve a least distance program. This can be accomplished if we impose the requirement that the search direction have minimum norm, i.e., $\|p\|_2$. However, it is also possible to encounter a locally infeasible subproblem, simply because a linear approximation is not sufficiently accurate to model the nonlinear constraints. Therefore, the search direction is computed to minimize

$$(1/2)p^{\mathrm{T}}p, \tag{26}$$

subject to the linear constraints

$$\hat{b}_l \leq \begin{bmatrix} Gp \\ p \end{bmatrix} \leq \hat{b}_u. \tag{27}$$

The bound vectors $\hat{b}_l$ and $\hat{b}_u$ are defined by

$$\hat{b}_{l,i} = \begin{cases} \beta(c_{Li} - c_i), & \text{if } c_i < c_{Li}, \\ (c_{Li} - c_i), & \text{if } c_i \geq c_{Li}, \end{cases} \tag{28}$$

$$\hat{b}_{u,i} = \begin{cases} \beta(c_{Ui} - c_i), & \text{if } c_i > c_{Ui}, \\ (c_{Ui} - c_i), & \text{if } c_i \leq c_{Ui}, \end{cases} \tag{29}$$

for $i = 1, \ldots, m$, with a similar definition for the simple bounds x_L and x_U. When the scalar relaxation parameter $\beta = 0$, the subproblem is feasible; however, when $\beta = 1$, the full nonlinear constraint violation is treated and the resulting subproblem may or may not be feasible. In general, a feasible subproblem can be created by relaxing the constraints, that is, for some value $0 \leq \beta < 1$.

Since the solution of this subproblem is based on a linear model of the constraint functions, it may be necessary to adjust the steplength α in (5) to produce a reduction in the constraint error. Specifically, a line search is used to adjust α so that

$$\Upsilon(\alpha) - \Upsilon(0) < \kappa_1 \alpha \Upsilon'(0),$$

$$\Upsilon'(\alpha) < \kappa_2 \Upsilon'(0),$$

for $0 < \kappa_1 < \kappa_2 < 1$, where the constraint violation is defined by

$$\Upsilon(x) = \sum_{i=1}^{m} [2c_i - \min(c_{Ui}, c_i) - \max(c_{Li}, c_i)]^2 + \sum_{i=1}^{N} [2x_i - \min(x_{Ui}, x_i) - \max(x_{Li}, x_i)]^2. \tag{30}$$

The overall strategy for locating a feasible point can now be described. Beginning at the point x, with the primary strategy, the procedure is as follows:

Step 1. Evaluate the constraints and Jacobian and terminate if constraints are feasible.

Step 2. Compute the search direction.

(i) Primary strategy. Solve the QP subproblem with $\beta = 1$ and

(a) if solution is feasible, then go to Step 3; otherwise,
(b) change to equality relaxation strategy, and go to Step 2(ii).

(ii) Equality relaxation strategy. Ignoring inequality constraints, solve the QP subproblem and

(a) if solution is feasible, then go to Step 3; otherwise,
(b) reduce the relaxation parameter, and repeat Step 2(ii).

(iii) Inequality relaxation strategy. Solve the QP subproblem and

(a) if solution is feasible, then go to Step 3; otherwise,
(b) reduce relaxation parameter and repeat Step 2(iii).

Step 3. Line search:

(i) if the primary or inequality relaxation strategy is being used, then choose the steplength to reduce the constraint violation $\Upsilon(\bar{x})$; otherwise,
(ii) if the equality relaxation strategy is being used, then choose the steplength to reduce the equality constraint violation $\Upsilon_e(\bar{x})$.

Step 4. Relaxation adjustment.

(i) If the primary strategy is being used, then return to Step 1.
(ii) If the equality relaxation strategy is being used and if $\beta < 1$, then increase β, and if equality constraints are

feasible, then change to inequality relaxation strategy, and return to Step 1.

(iii) If the inequality relaxation strategy is being used and if $\beta < 1$, then increase β, and return to Step 1.

The essential feature of this strategy is to give priority to satisfying equality constraints. At each step, an attempt is made to either solve the full (unrelaxed) problem and/or increase the relaxation parameter.

3.6. Minimization Process. The three strategies M, FM, and FME execute a series of steps to minimize the merit function (9). In the case of strategy M, the iteration begins from the arbitrary and possibly infeasible point x^0. On the other hand, strategy FM begins the minimization of the merit function from a feasible point x^f. Finally, the FME strategy not only begins the minimization at a feasible point, but maintains feasibility with respect to the equality constraints. Let us denote the equalities by e, with Jacobian E.

The iteration begins at the point x, and proceeds as follows:

Step 1. Evaluate gradient information g and G and then:

(i) evaluate

$$\vartheta = g - G^T\lambda - v; \tag{31}$$

(ii) terminate if the Kuhn–Tucker conditions are satisfied;

(iii) if this is the first iteration, go to Step (vi); otherwise, compute the rate of change in the projected gradient norm,

$$\rho_3 = \|\vartheta^{(k)}\|_\infty / \|\vartheta^{(k-1)}\|_\infty, \tag{32}$$

and

(iv) if $\rho_1 \leq 0.25\rho_2$, then set $\tau^{(k+1)} = \min(2\tau^{(k)}, 1)$; otherwise,

(v) if $\rho_1 \geq 0.75\rho_2$, then set $\tau^{(k+1)} = \tau^{(k)} \min(0.5, \rho_3)$;

(vi) compute H_L from (18).

Step 2. Construct the optimization search direction:

(i) compute H from (19);

(ii) compute p by solving the QP subproblem (6)–(7);

(iii) if the inertia of K is incorrect and

(a) if $\tau < 1$, then increase τ and return to Step (i);

(b) if $\tau = 1$ and $H \neq I$, then set $\tau = 0$, $H = I$, and return to Step (i);

(c) if $H = I$, the constraints are locally inconsistent; terminate the algorithm;

(iv) compute ξ and η from (13) and (14);
(v) compute q and δ from (16) and (17);
(vi) compute the penalty parameters to satisfy (21) using (22)–(25); and
(vii) initialize $p^{(2)} = p^{(3)} = 0$, $\alpha = 1$, $\tilde{\alpha} = 0$.

Step 3. Compute the predicted point for

(i) the variables from

$$\bar{x} = x + \alpha p + \alpha^2 p^{(2)} + \alpha^3 p^{(3)}; \tag{33}$$

(ii) the multipliers and slacks from (12);
(iii) then evaluate the constraints $\bar{c} = c(\bar{x})$ at the predicted point; and then
(iv) if $\|\bar{e}\|_\infty \leq \epsilon$ or if strategy FME is not used, set $\hat{x} \Leftarrow \bar{x}$ and go to Step 7.

Step 4. Solve the underdetermined system

$$Ed = \bar{e} \tag{34}$$

for the direction d and initialize $v = 1$.

Step 5. Compute the corrected point

$$\hat{x} = \bar{x} - vd. \tag{35}$$

Step 6. Evaluate the constraints $\hat{e}$ at the corrected point $\hat{x}$; then:

(i) if $\|\hat{e}\|_\infty \leq \epsilon$ and $\tilde{\alpha} = 0$, compute

$$p^{(2)} = (1/\alpha^2)[\hat{x} - x - \alpha p]; \tag{36}$$

save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$ and $\tilde{\alpha} \Leftarrow \alpha$), and then go to Step 7;
(ii) else, if $\|\hat{e}\|_\infty \leq \epsilon$ and $\tilde{\alpha} \neq 0$, compute the elements of $p_i^{(2)}$ and $p_i^{(3)}$, for $i = 1, \ldots, N$, from the system

$$\begin{bmatrix} \alpha^2 & \alpha^3 \\ \tilde{\alpha}^2 & \tilde{\alpha}^3 \end{bmatrix} \begin{bmatrix} p_i^{(2)} \\ p_i^{(3)} \end{bmatrix} = \begin{bmatrix} \hat{x}_i - x_i - \alpha p_i \\ \tilde{x}_i - x_i - \tilde{\alpha} p_i \end{bmatrix}; \tag{37}$$

save the corrected point (set $\tilde{x} \Leftarrow \hat{x}$ and $\tilde{\alpha} \Leftarrow \alpha$), and then go to Step 7;
(iii) else, if $\|\hat{e}\|_\infty \leq \|\bar{e}\|_\infty$, update the corrected point (set $\bar{x} \Leftarrow \hat{x}$ and $\bar{e} \Leftarrow \hat{e}$), and return to Step 4;
(iv) else, reduce the steplength v to achieve constraint reduction, and return to Step 5.

Step 7. Evaluate the merit function $M(\hat{x}, \bar{\lambda}, \bar{v}, \bar{s}, \bar{t}) = \hat{M}$ and

(i) if the merit function $\hat{M}$ is sufficiently less than M, then $\hat{x}$ is an improved point; terminate the line search and go to Step 8;

(ii) else, change the steplength α to reduce M, and return to Step 3.

Step 8. Update all quantities:

(i) compute the actual reduction

$$\rho_1 = M^{(k)} - M^{(k-1)}; \tag{38}$$

(ii) compute the predicted reduction

$$\rho_2 = \tilde{M}^{(k)} - M^{(k-1)}, \tag{39}$$

where $\tilde{M}^{(k)}$ is the predicted value of the merit function; and

(iii) return to Step 1.

The steps outlined describe the fundamental elements of the optimization process; however, a number of points deserve additional clarification. First, note that the algorithm consists of an outer loop (Steps 1–7) to minimize the merit function, and an inner loop (Steps 3–6) to eliminate the error in the equality constraints. The outer loop can be viewed as a univariate line search in the direction p, with the steplength α adjusted to minimize the merit function. The inner loop can be viewed as a nonlinear root-solving process designed to eliminate the error in the equality constraints for the specified value of α. Note that Steps 4–6 are only executed for the FME strategy. The inner constraint elimination process must be initiated with an estimate of the variables, and this estimate is given by the expression (33). Notice that the first prediction is based on a linear model for the constraints, since $p^{(2)} = p^{(3)} = 0$ in Step 2. However, after the constraint error has been eliminated, the value of $p^{(2)}$ is updated by (36) and the second prediction is based on a quadratic model of the constraints. After the second corrected point is obtained, subsequent predictions utilize the cubic model defined by solving (37). Adjusting the value of the steplength α as required in Step 7(ii) is accomplished using a univariate search procedure similar to that described in Ref. 6, which constructs a quadratic and cubic model of the merit function. The reduction is considered sufficient when

$$M(\alpha) - M(0) < \kappa_1 \alpha M'(0),$$

$$M'(\alpha) < \kappa_2 M'(0),$$

for $0 < \kappa_1 < \kappa_2 < 1$.

Because the constraint elimination process requires the solution of the underdetermined system (34), there is some ambiguity in the algorithm. This ambiguity is eliminated by choosing the minimum norm direction which can be obtained by solving the augmented system

$$\begin{bmatrix} I & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} d \\ -\bar{\omega} \end{bmatrix} = \begin{bmatrix} 0 \\ \bar{e} \end{bmatrix}. \tag{40}$$

Notice that the Jacobian E is evaluated at the reference point x, and not reevaluated during the inner loop iteration, even though the right-hand side $\bar{e}$ does change. Because the Jacobian is not reevaluated, the inner loop will have a linear convergence rate. Nevertheless, this approach has been found attractive, because:

(i) the coefficient matrix can be factored only once per outer optimization iteration, thereby significantly reducing the linear algebra expense; and

(ii) the corrections defined by d are orthogonal to the constraint tangent space at the reference point x, and hence tend to produce a well-conditioned constraint iteration process.

In order to evaluate the Hessian matrix (18), an estimate of the Lagrange multipliers is needed. The values obtained by solving the QP with $H = I$ are used for the first iteration; thereafter, the values $\bar{\lambda}$ from (12) are used. Furthermore, for the very first iteration, the multiplier search directions $\xi = 0$ and $\eta = 0$, so that the multipliers will be initialized to the QP estimates μ and ω. The multipliers are reset in a similar fashion, after a defective QP subproblem is encountered, in Step 2(iii)(b). The Levenberg parameter τ in (19) and the penalty weights r_i in (22) are initialized to zero, and consequently the merit function is initially just the Lagrangian.

Gradient and Hessian information can be computed (a) analytically, (b) using finite difference estimates, or (c) recursively. Many of the numerical results construct this information using sparse finite differencing as described in Refs. 4, 7, and 8. Although central difference estimates must be used during optimization, forward difference estimates are used when finding a feasible point. Numerical experience also suggests that it is not necessary to compute a Hessian matrix for every iteration, when the algorithm is progressing well. A recursive limited-memory secant approximation technique for the Hessian, suitable for large problems, is described in a later section.

4. Sparse Quadratic Programming Algorithm

4.1. Sparse Linear Algebra. Development of efficient, robust software for the solution of sparse linear systems is a field of active research.

The nonlinear programming algorithm described employs a state-of-the-art linear algebra package. The package solves $Ax = b$ for x, where A is an $n \times n$ real symmetric indefinite sparse matrix. Since A is symmetric, it can be factored using a square-root-free Cholesky factorization $A = LDL^T$, where L is a unit lower triangular matrix and D is a diagonal matrix. Since A is not necessarily positive definite, pivoting to preserve stability is required. The package uses the threshold pivoting generalization of Bunch and Kaufman 2×2 block pivoting for sparse symmetric indefinite matrices proposed by Duff and refined by Liu. The software requires storage for the nonzero elements in the lower triangular portion of the matrix and a work array. A complete description of the multifrontal factorization algorithm is found in Refs. 9 and 10.

4.2. Schur-Complement Method. The quadratic programming algorithm used is based on a method proposed by Gill, Murray, Saunders, and Wright in Ref. 11, and the implementation exploits the multifrontal algorithm for the solution of sparse linear systems. The QP (6)–(8) is first stated in the following standard form: minimize

$$c^T x + (1/2)x^T H x, \tag{41}$$

subject to the linear constraints

$$Ax = b, \tag{42}$$

and the simple bounds

$$x_L \leq x \leq x_U. \tag{43}$$

Note that the variables x include slack variables to replace the general inequality constraints and do not correspond to the notation used elsewhere in the paper. Similar modifications are necessary to define the other quantities in the standard form QP. When written in this form, variables are either fixed at their bounds or free to move within the bounds. In keeping with the notation of Ref. 11, we denote the n_{FR} free variables by FR and the fixed variables by FX. For a specific estimate of the active set of constraints, the step to the constrained minimum is obtained by solving the Kuhn–Tucker or KT system (cf. Ref. 4)

$$\begin{bmatrix} H_{FR} & A_{FR}^T \\ A_{FR} & 0 \end{bmatrix} \begin{bmatrix} -p_{FR} \\ \pi \end{bmatrix} \equiv K_0 \begin{bmatrix} -p_{FR} \\ \pi \end{bmatrix} = \begin{bmatrix} g_{FR} \\ 0 \end{bmatrix}. \tag{44}$$

If the estimate of the active set is correct, the solution of the KT system defines the solution of the QP. However, in general, it will be necessary to change the active set and solve a series of equality constrained problems. In Ref. 11, it is demonstrated that the solution to a problem with a new active

set can be obtained by the symmetric addition of a row and column to the original K_0 with a corresponding augmentation of the right-hand side. In fact, after k iterations, the KT system is of dimension $n_0 + k$ and has the form

$$\begin{bmatrix} K_0 & U \\ U^T & V \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} f_0 \\ w \end{bmatrix}, \tag{45}$$

where U is $n_0 \times k$ and V is $k \times k$. The initial right-hand side of (44) is denoted by the n_0-vector f_0, and the k-vector w defines the additions to the right-hand side to reflect changes in the active set.

The fundamental feature of the method is that the system (45) can be solved using factorization of K_0 and C, the $k \times k$ Schur complement of K_0,

$$C \equiv V - U^T K_0^{-1} U. \tag{46}$$

Using the Schur complement, the values for y and z are computed by solving in turn

$$K_0 v_0 = f_0, \tag{47}$$

$$Cz = w - U^T v_0, \tag{48}$$

$$K_0 y = f - Uz. \tag{49}$$

Thus each iteration of the QP requires one solution with the factorization of K_0 and one solution with the factorization of C. The solution for v_0 only needs to be done once at the first iteration. Each change in the active set adds a new row and column to C, and it is relatively straightforward to update both C and its factorization to accommodate the change. It is important to keep C small enough to maintain a stable, dense factorization, and this is achieved by refactoring the entire KT matrix whenever $k > 100$. In general, the penalty for refactoring may be substantial. However, when the QP algorithm is used within the general NLP algorithm, it is possible to exploit previous estimates of the active set to give the QP a warm start. In fact, as the NLP algorithm approaches a solution, it is expected that the active set will be correctly identified—and the resulting number of iterations k for the QP subproblem will become small. Limited computational experience with the Schur complement method indicates a speedup of nearly 200 to 1 over the dense QP algorithm (LSSOL), which is used in the nonlinear program NPSOL in Ref. 5.

5. Computing the Hessian Using a Limited-Memory Secant Update

Computing the Hessian of the Lagrangian by sparse differences can be very expensive for some problems. An alternative is to use a limited-

memory version of a secant update method, such as the BFGS update (see, e.g., Ref. 12).

The use of a limited-memory Hessian avoids the problem of storing and factoring a dense secant Hessian matrix. At each iteration, this is accomplished by constructing an approximation to the Hessian using information from a limited number L of previous iterations to modify a sparse positive-definite matrix, usually the identity. The sequence of L rank-two updates that would transform this sparse matrix into the Hessian approximation at the current iteration is then computed and stored as $2L$ vectors. The values of L typically range from two to ten.

Linear system solutions with the limited-memory Hessian involve solutions with a factorization of the original sparse matrix. The updates are then accounted for by using the Sherman–Morrison–Woodbury formula (see, e.g., Ref. 13) for updating the inverse of a matrix. The above process is discussed in more detail below.

The hope is that the limited-memory BFGS update will retain the good local convergence of the full BFGS update, while saving greatly on linear algebra costs. One reason for which this should be true is that the limited-memory update is based on information obtained from the most recent iterates. It only ignores information obtained from iterates in the distant past. The computational results presented in the next section provide some validation to the above premise.

The inspiration for this work is the limited-memory algorithm described by Nash and Nocedal in Ref. 14. However, their algorithm directly updates an approximation to the inverse of the Hessian. To solve linear systems involving the Hessian, the inverse Hessian updates are accounted for by successive dot products with the right-hand side. However, the inverse Hessian does not appear directly in the linear system solutions required for the Schur-complement QP. The Schur-complement QP method requires solutions with a factorization of the KT linear system (44). Since the Hessian is only a block of the KT system matrix, it was considered easier to work directly with the secant Hessian than with the secant inverse Hessian. Thus, the method described below is based on updating the Hessian approximation.

Using subscripts to denote iteration numbers, the BFGS Hessian update is given by

$$H_k = H_{k-1} + y_k y_k^T / y_k^T s_k - (H_{k-1} s_k)(H_{k-1} s_k)^T / s_k^T H_{k-1} s_k, \quad (50)$$

where

$$y_k = g_k - g_{k-1} \quad \text{and} \quad s_k = x_k - x_{k-1}.$$

For the nonlinear programming algorithm, H and g in Eq. (50) denote the Hessian and the gradient of the Lagrangian, respectively.

The limited-memory version of the BFGS is considered next. Let H_0 denote the Hessian L iterations prior to iteration k. The Hessian at iteration k is formed by L BFGS updates to H_0. This Hessian can be represented by

$$H_k = H_0 + \sum_{i=1}^{L} u_i u_i^T - \sum_{i=1}^{L} v_i v_i^T, \tag{51}$$

where

$$u_i = y_i/[y_i^T s_i]^{1/2} \quad \text{and} \quad v_i = H_{i-1}s_i/[s_i^T(H_{i-1}s_i)]^{1/2}.$$

The underlying assumption for the above approach is that H_0 is a sparse positive-definite matrix, the identity matrix in our implementation, and the summation terms in Eq. (51) represent a small number of dense updates.

The vectors v_i in (51) are defined in terms of the dot products $H_{i-1}s_i$. These dot products can be computed by successive use of Eq. (51) with upper limits increasing from 1 to $L-1$.

The limited-memory secant Hessian yields an estimate which can be incorporated within the Schur-complement QP algorithm described in the preceding section. However, to fully exploit sparsity, some care must be exercised when it is used. The main idea is to perform factorizations on a matrix formed by modifying K_0 such that only the sparse portion of the Hessian (i.e., H_0) appears. Thus, from the definition of the KT matrix (44), we have

$$K_0 = \begin{bmatrix} (H_k)_{FR} & A_{FR}^T \\ A_{FR} & 0 \end{bmatrix}. \tag{52}$$

Now, let us define the $(n_{FR}+m) \times 2L$ matrix

$$W = \begin{bmatrix} (u_1)_{FR} \cdots (u_L)_{FR} & (v_1)_{FR} \cdots (v_L)_{FR} \\ 0 & 0 \end{bmatrix}, \tag{53}$$

the $2L \times 2L$ diagonal matrix

$$D = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}, \tag{54}$$

and the beginning KT matrix in the limited-memory sequence

$$K_b = \begin{bmatrix} (H_0)_{FR} & A_{FR}^T \\ A_{FR} & 0 \end{bmatrix}. \tag{55}$$

Then, if we substitute the definition of H_k from (51) into (52), we obtain the expression

$$K_0 = K_b + WDW^T. \tag{56}$$

It is important to emphasize that, while K_0 is now dense, the matrix K_b is still sparse.

Now, in the Schur-complement QP method, Eqs. (46), (47), and (49) require solving linear systems of the form $K_0 q = b$, where q and b are the appropriate left and right-hand side vectors. Using the Sherman–Morrison–Woodbury formula, the solution to $K_0 q = b$ is given by

$$q = K_b^{-1} b - K_b^{-1} W (W^T K_b^{-1} W + D^{-1})^{-1} (K_b^{-1} W)^T b. \tag{57}$$

Notice that the solution is obtained by solving large sparse linear systems involving K_b, not large dense linear systems involving K_0.

Efficient implementation of the computations indicated in Eq. (57) can be done as follows:

Step 1. Preprocessing, applicable to any right-hand side.

- (i) Factor K_b.
- (ii) Compute $K_b^{-1} W$.
- (iii) Compute the $2L \times 2L$ matrix M, where $M = W^T K_b^{-1} W + D^{-1}$.
- (iv) Factor M.

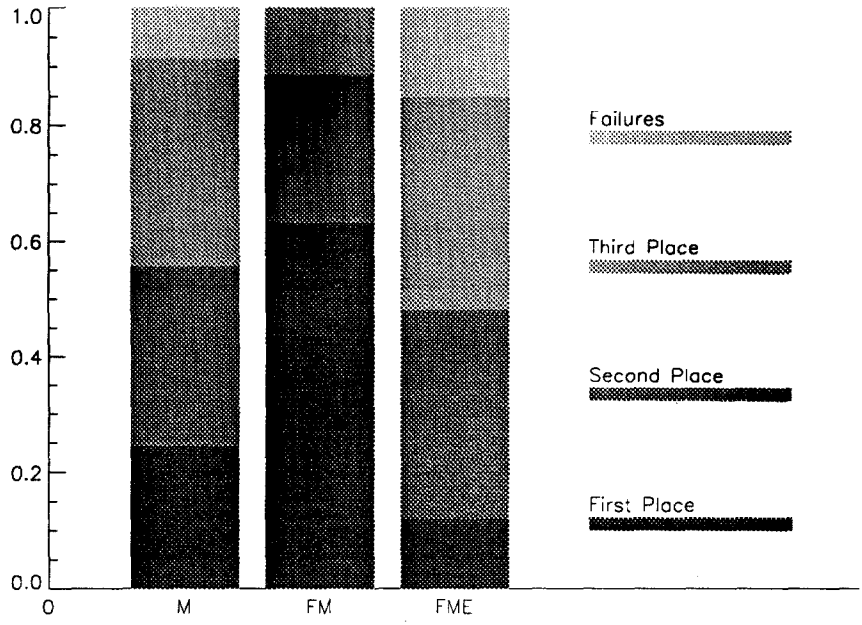
Step 2. Solve for the given right-hand side b.

- (i) Solve for $K_b^{-1} b$.
- (ii) Compute $W^T K_b^{-1}$.
- (iii) Solve $Mz = W^T K_b^{-1} b$ for z.
- (iv) Compute the solution $q = K_b^{-1} b - (K_b^{-1} W) z$.

The linear algebra costs for the above algorithm are reasonable on large problems, because K_b is formed using a very sparse representation of the free portion of the Hessian, usually the identity. In addition, the square matrix M is only of dimension $2L$ and the matrix W has only $2L$ columns. Test results for an implementation of the limited-memory update method are presented in the next section.

6. Computational Results

Computational results on a series of trajectory optimization problems are summarized in this section. The test set consists of 109 problems constructed using different types of discretization methods and numbers of gridpoints. The trajectory test set consists of the following class of problems: (a) quadratic-linear; (b) linear tangent steering; (c) spherical, nonrotating Earth trajectories including Shuttle reentry with heating constraints;



(d) Goddard rocket problem; (e) NLQR guidance problem: (f) minimum time to climb; (g) commercial aircraft trajectory; (h) minimum lateral acceleration guidance; (i) brachistochrone; (j) wind shear; and (k) low-thrust orbit transfer using equinoctial elements. Space does not permit a complete presentation of results for all problems, although more details are described in Refs. 15 and 16. Instead, we will describe the problem set and present a summary of significant algorithm performance parameters. All solutions were obtained using a Sun Sparcstation IPX. Except for the subsection on limited-memory update testing, the test results all reflect use of a Hessian computed by sparse differencing.

A summary of the results for the test problem set is given in Fig. 1. All 109 problems were run using the three optimization strategies. The algorithm performance was measured in terms of the number of function evaluations [the number of times $f(x)$ and $c(x)$ are evaluated] and the solution time. For each test problem, a first, second, and third-place strategy was selected, where the first-place strategy required the smallest number of function evaluations. If a particular strategy failed to solve the problem, this was counted as a failure. It is clear from Fig. 1, that strategy FM was in first place over 63% of the time. Furthermore, FM was either the best or second best strategy nearly 89% of the time. Finally, notice that

strategy FM solved all 109 problems (no failures). For all but seven cases, the least number of function evaluations corresponds to the shortest solution time, and consequently comparing strategies based on run time leads to the same conclusions. These results clearly indicate why strategy FM has been selected as the default. We note that, for three problems, there are no degrees of freedom, in which case F is the only possible strategy, and these cases were eliminated from the comparison.

All results were obtained using Sun compiler options ($-$C, $-$g). The total time required to solve all 109 problems using the best strategy was 23447.04 sec. When the fast compiler option was used, the same results were obtained in 16520.26 sec. Some insight into the solution time can be gained by noting that the largest problem in the set ($n = 2406$) was solved in 1031.9 sec, and the longest solution time (3590.5 sec) corresponded to an equinoctial orbit transfer problem with $n = 1502$. In contrast to these extreme cases, more typical performance is characterized by the median case, which took 92.3 sec for a problem with $n = 601$. In general, time comparisons with other methods have been very encouraging, and a detailed study will be reported in the near future.

6.1. Tests Using the Limited-Memory Update Hessian. The limited-memory update method for approximating the Hessian of the Lagrangian was incorporated into the sparse nonlinear programming code. This method was tested on nine instances of the trajectory optimization problems. The results for the limited-memory Hessian method are compared with the results for the same nonlinear programming code using a Hessian computed by sparse finite differences and using a full dense BFGS Hessian. In each case, the FM nonlinear programming option was used.

The full BFGS Hessian is impractical for large problems, due to high linear algebra costs and storage requirements. However, it is included to help determine how well the limited-memory method is working. That is, if the iteration counts for the limited-memory method are competitive with those for the full BFGS update, then its limitations can be attributed to the fact that it is a secant method, rather than it being a restricted version of a secant method. Due to the high cost of linear algebra on the full BFGS Hessians, the full BFGS method was not expected to be competitive in terms of CPU time; and it is not.

Of the nine test problems, one problem could not be solved using the BFGS method or the limited-memory method. The results, averaged over the other eight problems, are summarized in Table 1. The numbers in the columns of Table 1 are the respective ratios of the numbers of iterations, function evaluations, and CPU time required by the competing methods. In Table 1, fd Hessian denotes the Hessian computed using sparse differences.

Table 1. Limited-memory versus BFGS and sparse differences.

Quantity	Iterations	Function evaluations	Time
Ratio vs fd Hessian	4.63	0.93	1.95
Ratio vs BFGS Hessian	1.07	1.01	0.06

Not surprisingly, the sparse difference Hessian requires many fewer iterations than the limited-memory update Hessian. However, the reduction in the number of function evaluations using the limited-memory method is disappointingly small. In addition, the limited-memory method requires more CPU time than the sparse differencing method.

In assessing the above results, one must account for the fact that the currently available test problem set is biased in favor of the sparse difference method. This is because the number of function evaluations required to compute a Hessian by sparse differences is roughly half of the square of the number of index sets (see Refs. 4, 7, and 8). An index set is a set of variables whose influence on a given function is independent of the members of the set. Although the test problems had hundreds of variables, they only had on the order of eight index sets. In addition, the function evaluations require relatively little CPU time. For problems with more index sets and expensive function evaluations, the advantage should shift considerably toward the limited-memory update method. It is anticipated that future testing will validate this hypothesis.

The results in Table 1 indicate that the limited-memory update method performs nearly as well as the full BFGS method in terms of iterations and function evaluations. The limited-memory update method has the expected huge advantage in CPU time due to its economical representation of the secant updates.

7. Summary and Conclusions

This paper presents a method for solving the space nonlinear programming problem. A comparison of three different strategies for using a sparse quadratic programming algorithm suggest that an approach which first locates a feasible point and then stays near the constraints produces a reasonable compromise between speed and robustness. Computational experience with the algorithm when applied to a collection of discretized trajectory optimization problems has demonstrated a significant speed enhancement when compared to standard dense optimization methods.

The primary deficiencies with the current algorithm involve the Hessian matrix and robustness issues. Inertia-controlling methods as described in Ref. 17 may prove useful for improving robustness. Investigation of trust-region strategies may be attractive for dealing with indefinite Hessian approximations. Application of a limited-memory update for the Jacobian (instead of the Hessian) may be attractive to improve the efficiency of the constraint satisfaction portions of the algorithm [e.g., (40)]. Finally, it may be possible to utilize recursive Hessian approximations which exploit the structure of the problem such as proposed in Ref. 18.

References

1. GILL, P. E., MURRAY, W., SAUNDERS, M. A., and WRIGHT, M. H., *Some Theoretical Properties of an Augmented Lagrangian Merit Function*, Report SOL 86-6, Department of Operations Research, Stanford University, 1986.
2. ROCKAFELLAR, R. T., *The Multiplier Method of Hestenes and Powell Applied to Convex Programming*, Journal of Optimization Theory and Applications, Vol. 12, pp. 555–562, 1973.
3. FLETCHER, R., *Practical Methods of Optimization, Vol. 2: Constrained Optimization*, John Wiley and Sons, New York, New York, 1985.
4. BETTS, J. T., and HUFFMAN, W. P., *Application of Sparse Nonlinear Programming to Trajectory Optimization*, Journal of Guidance, Control, and Dynamics, Vol. 15, 1992.
5. GILL, P. E., MURRAY, W., SAUNDERS, M. A., and WRIGHT, M. H., *User's Guide for NPSOL (Version 4.0): A Fortran Package for Nonlinear Programming*, Report SOL 86-2, Department of Operations Research, Stanford University, 1986.
6. BETTS, J. T., and HALLMAN, W. P., *NLP2 Optimization Algorithm Documentation*, Report TOR-0089(4464-06)-1, The Aerospace Corporation, 1989.
7. BETTS, J. T., and HUFFMAN, W. P., *Trajectory Optimization on a Parallel Processor*, Journal of Guidance, Control, and Dynamics, Vol. 14, 1991.
8. HUFFMAN, W., and CARTER, M., *Software for Sparse Finite Difference Derivatives*, Report ECA-LR-71, Boeing Computer Services, 1991.
9. ASHCRAFT, C. C., *A Vector Implementation of the Multifrontal Method for Large Sparse, Symmetric Positive-Definite Linear Systems*, Technical Report ETA-TR-51, Boeing Computer Services, 1987.
10. ASHCRAFT, C. C., and GRIMES, R. G., *The Influence of Relaxed Supernode Partitions on the Multifrontal Method*, Technical Report ETA-TR-60, Boeing Computer Services, 1988.
11. GILL, P. E., MURRAY, W., SAUNDERS, M. A., and WRIGHT, M. H., *A Schur-Complement Method for Sparse Quadratic Programming*, Report SOL 87-12, Department of Operations Research, Stanford University, 1987.

12. DENNIS, J. E., and SCHNABEL, R. B., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.
13. STEWART, G. W., *Introduction to Matrix Computations*, Academic Press, New York, New York, 1973.
14. NASH, S. G., and NOCEDAL, J., *A Numerical Study of the Limited Memory BFGS Method and the Truncated-Newton Method for Large-Scale Optimization*, SIAM Journal on Optimization, Vol. 1, pp. 358–372, 1991.
15. BETTS, J. T., and HUFFMAN, W. P., *Path Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming*, Journal of Guidance, Control, and Dynamics, Vol. 16, 1993.
16. BETTS, J. T., and HUFFMAN, W. P., *Sparse Nonlinear Programming Test Problems (Release 1.0)*, Report BCSTECH-93-016, Boeing Computer Services, 1993.
17. GILL, P. E., MURRAY, W., SAUNDERS, M. A., and WRIGHT, M. H., *Inertia-Controlling Methods for General Sparse Quadratic Programming*, SIAM Review, Vol. 33, pp. 1–36, 1991.
18. KELLEY, C. T., and SACHS, E. W., *A Pointwise Quasi-Newton Method for Unconstrained Optimal Control Problems*, Numerische Mathematik, Vol. 55, pp. 159–176, 1989.