NORTHWESTERN UNIVERSITY
Department of Electrical Engineering
and Computer Science

A LIMITED MEMORY ALGORITHM FOR BOUND CONSTRAINED OPTIMIZATION

by

Richard H. Byrd[1], *Peihuang Lu*[2], *Jorge Nocedal*[2] *and Ciyou Zhu*[2]

Technical Report NAM-08

Revised: May 1994

[1] Computer Science Department, University of Colorado at Boulder, Boulder Colorado 80309. This author was supported by NSF grant CCR-9101795, ARO grant DAAL 03-91-G-0151, and AFOSR grant AFOSR-90-0109.

[2] Department of Electrical Engineering and Computer Science, Northwestern University, Evanston Il 60208; nocedal@eecs.nwu.edu. These authors were supported by National Science Foundation Grants CCR-9101359 and ASC-9213149, and by Department of Energy Grant DE-FG02-87ER25047-A004.

A LIMITED MEMORY ALGORITHM FOR BOUND CONSTRAINED OPTIMIZATION

by

Richard H. Byrd, Peihuang Lu, Jorge Nocedal and Ciyou Zhu

ABSTRACT

An algorithm for solving large nonlinear optimization problems with simple bounds is described. It is based on the gradient projection method and uses a limited memory BFGS matrix to approximate the Hessian of the objective function. It is shown how to take advantage of the form of the limited memory approximation to implement the algorithm efficiently. The results of numerical tests on a set of large problems are reported.

Key words: bound constrained optimization, limited memory method, nonlinear optimization, quasi-Newton method, large-scale optimization.

Abbreviated title: A Limited Memory Method

1. Introduction.

In this paper we describe a limited memory quasi-Newton algorithm for solving large nonlinear optimization problems with simple bounds on the variables. We write this problem as

$$\min f(x) \tag{1.1}$$

$$\text{subject to} \quad l \leq x \leq u, \tag{1.2}$$

where $f : \Re^n \to \Re$ is a nonlinear function whose gradient g is available, the vectors l and u represent lower and upper bounds on the variables, and the number of variables n is assumed to be large. The algorithm does not require second derivatives or knowledge of the structure of the objective function, and can therefore be applied when the Hessian matrix is not practical to compute. A limited memory quasi-Newton update is used to approximate the Hessian matrix in such a way that the storage required is linear in n.

The algorithm described in this paper is similar to the algorithms proposed by Conn, Gould and Toint [9] and Moré and Toraldo [20], in that the gradient projection method is used to determine a set of active constraints at each iteration. Our algorithm is distinguished from the these methods by our use of line searches (as opposed to trust regions) but mainly by our use of limited memory BFGS matrices to approximate the Hessian of the objective function. The properties of these limited memory matrices have far reaching consequences in the implementation of the method, as will be discussed later on. We find that by making use of the compact representations of limited memory matrices described by Byrd, Nocedal and Schnabel [6], the computational cost of one iteration of the algorithm can be kept to be of order n.

We used the gradient projection approach [16], [17], [3] to determine the active set, because recent studies [7], [5] indicate that it possess good theoretical properties, and because it also appears to be efficient on many large problems [8], [20]. However some of the main components of our algorithm could be useful in other frameworks, as long as limited memory matrices are used to approximate the Hessian of the objective function.

2. Outline of the algorithm.

At the beginning of each iteration, the current iterate x_k, the function value f_k, the gradient g_k, and a positive definite limited memory approximation B_k are given. This allows us to form a quadratic model of f at x_k,

$$m_k(x) = f(x_k) + g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T B_k (x - x_k). \tag{2.1}$$

Just as in the method studied by Conn, Gould and Toint [9] the algorithm approximately minimizes $m_k(x)$ subject to the bounds given by (1.2). This is done by first using the gradient projection method to find a set of active bounds, followed by a minimization of m_k treating those bounds as equality constraints.

To do this, we first consider the piece-wise linear path

$$x(t) = P(x_k - tg_k, l, u),$$

obtained by projecting the steepest descent direction onto the feasible region, where

$$P(x, l, u)_i = \begin{cases} l_i & \text{if } x_i < l_i \\ x_i & \text{if } x_i \in [l_i, u_i] \\ u_i & \text{if } x_i > u_i. \end{cases} \tag{2.2}$$

We then compute the generalized Cauchy point x^c, which is defined as the first local minimizer of the univariate, piece-wise quadratic

$$q_k(t) = m_k(x(t)).$$

The variables whose value at x^c is at lower or upper bound, comprising the active set $\mathcal{A}(x^c)$, are held fixed. We then consider the following quadratic problem over the subspace of free variables,

$$\min \{m_k(x):\ x_i = x_i^c,\ \ \forall i \in \mathcal{A}(x^c)\} \tag{2.3}$$

$$\text{subject to}\ \ l_i \le x_i \le u_i \quad \forall i \notin \mathcal{A}(x^c). \tag{2.4}$$

We first solve or approximately solve (2.3), ignoring the bounds on the free variables, which can be accomplished either by direct or iterative methods on the subspace of free variables, or by a dual approach, handling the active bounds in (2.3) by Lagrange multipliers. When an iterative method is employed we use x^c as the starting point for this iteration. We then truncate the path toward the solution so as to satisfy the bounds (2.4).

After an approximate solution $\bar{x}_{k+1}$ of this problem has been obtained, we compute the new iterate x_{k+1} by a line search along $d_k = \bar{x}_{k+1} - x_k$ that satisfies the sufficient decrease condition

$$f(x_{k+1}) \le f(x_k) + \alpha \lambda_k g_k^T d_k, \tag{2.5}$$

and that also attempts to enforce the curvature condition

$$|g_{k+1}^T d_k| \le \beta |g_k^T d_k|, \tag{2.6}$$

where λ_k is the steplength and α, β are parameters that have the values 10^{-4} and 0.9, respectively, in our code. The line search, which ensures that the iterates remain in the feasible region, is described in §6. We then evaluate the gradient at x_{k+1}, compute a new limited memory Hessian approximation B_{k+1} and begin a new iteration.

Because in our algorithm every Hessian approximation B_k is positive definite, the approximate solution $\bar{x}_{k+1}$ of the quadratic problem (2.3)-(2.4) defines a descent direction $d_k = \bar{x}_{k+1} - x_k$ for the objective function f. To see this, first note that the generalized Cauchy point x^c, which is a minimizer of $m_k(x)$ on the projected steepest descent direction, satisfies $m_k(x_k) > m_k(x^c)$ if the projected gradient is nonzero. Since the point $\bar{x}_{k+1}$ is on a path from x^c to the minimizer of (2.3), along which m_k decreases, the value of m_k at $\bar{x}_{k+1}$ is no larger than its value at x^c. Therefore we have

$$f(x_k) = m_k(x_k) > m_k(x^c) \ge m_k(\bar{x}_{k+1}) = f(x_k) + g_k^T d_k + \tfrac{1}{2} d_k^T B_k d_k.$$

This inequality implies that $g_k^T d_k < 0$ if B_k is positive definite and d_k is not zero. In this paper we do not present any convergence analysis or study the possibility of zigzagging. However, given the use of gradient projection in the step computation we believe analyses similar to those in [7] and [9] should be possible, and that zigzagging should only be a problem in the degenerate case.

The Hessian approximations B_k used in our algorithm are limited memory BFGS matrices (Nocedal [21] and Byrd, Nocedal and Schnabel [6]). Even though these matrices do not take advantage of the structure of the problem, they require only a small amount of storage and, as we will show, allow the computation of the generalized Cauchy point and the subspace minimization to be performed in $O(n)$ operations. The new algorithm therefore has similar computational demands as the limited memory algorithm (L-BFGS) for unconstrained problems described by Liu and Nocedal [18] and Gilbert and Lemaréchal [14].

In the next three sections we describe in detail the limited memory matrices, the computation of the Cauchy point, and the minimization of the quadratic problem on a subspace.

3. Limited Memory BFGS Matrices.

In our algorithm, the limited memory BFGS matrices are represented in the compact form described by Byrd, Nocedal and Schnabel [6]. At every iterate x_k the algorithm stores a small number, say m, of correction pairs $\{s_i, y_i\}$, $i = k-1, \ldots, k-m$, where

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

These correction pairs contain information about the curvature of the function, and in conjunction with the BFGS formula, define the limited memory iteration matrix B_k. The question is how to best represent these matrices without explicitly forming them.

In [6] it is proposed to use a compact (or outer product) form to define the limited memory matrix B_k in terms of the $n \times m$ correction matrices

$$Y_k = [y_{k-m}, \ldots, y_{k-1}], \qquad S_k = [s_{k-m}, \ldots, s_{k-1}]. \tag{3.1}$$

More specifically, it is shown in [6] that if θ is a positive scaling parameter, and if the m correction pairs $\{s_i, y_i\}_{i=k-1}^{k-m}$, satisfy $s_i^T y_i > 0$, then the matrix obtained by updating θI m-times, using the BFGS formula and the pairs $\{s_i, y_i\}_{i=k-1}^{k-m}$ can be written as

$$B_k = \theta I - W_k M_k W_k^T \tag{3.2}$$

where

$$W_k = \begin{bmatrix} Y_k & \theta S_k \end{bmatrix}, \tag{3.3}$$

$$M_k = \begin{bmatrix} -D_k & L_k^T \\ L_k & \theta S_k^T S_k \end{bmatrix}^{-1}, \tag{3.4}$$

and where L_k and D_k are the $m \times m$ matrices

$$(L_k)_{i,j} = \begin{cases} (s_{k-m-1+i})^T (y_{k-m-1+j}) & \text{if } i > j \\ 0 & \text{otherwise,} \end{cases} \tag{3.5}$$

$$D_k = diag\left[s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1}\right]. \tag{3.6}$$

(We should point out that (3.2) is a slight rearrangement of equation (3.5) in [6]). Note that since M_k is a $2m \times 2m$ matrix, and since m is chosen to be a small integer, the cost of computing the inverse in (3.4) is negligible. It is shown in [6] that by using the compact representation (3.2) various computations involving B_k become inexpensive. In particular, the product of B_k times a vector, which occurs often in the algorithm of this paper, can be performed efficiently.

There is a similar representation of the inverse limited memory BFGS matrix H_k that approximates the inverse of the Hessian matrix:

$$H_k \equiv \frac{1}{\theta} I + \bar{W}_k \bar{M}_k \bar{W}_k^T, \tag{3.7}$$

where

$$\bar{W}_k \equiv \left[\begin{array}{cc} \frac{1}{\theta} Y_k & S_k \ , \end{array} \right]$$

$$\bar{M}_k \equiv \left[\begin{array}{cc} 0 & -R_k^{-1} \\ & \\ -R_k^{-T} & R_k^{-T}(D_k + \frac{1}{\theta} Y_k^T Y_k) R_k^{-1} \end{array} \right],$$

and

$$(R_k)_{i,j} = \begin{cases} (s_{k-m-1+i})^T (y_{k-m-1+j}) & \text{if } i \leq j \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

(We note that (3.7) is a slight rearrangement of equation (3.1) in [6]).

Representations of limited memory matrices analogous to these also exist for other quasi-Newton update formulae, such as SR1 or DFP (see [6]), and in principle could be used for B_k in our algorithm. We consider here only BFGS since we have considerable computational experience in the unconstrained case indicating the limited memory BFGS performs well.

Since the bounds on the problem may prevent the line search from satisfying (2.6) (see §6), we cannot guarantee that the condition $s_k^T y_k > 0$ always holds (cf. Dennis and Schnabel [12]). Therefore to maintain the positive definiteness of the limited memory BFGS matrix, we discard a correction pair $\{s_k, y_k\}$ if the curvature condition

$$s_k^T y_k > \text{eps } \|y\|^2 \tag{3.9}$$

is not satisfied for a small positive constant eps. If this happens we do not delete the oldest correction pair, as is normally done in limited memory updating. This means that the m directions in S_k and Y_k may actually include some with indices less than $k - m$.

4. The generalized Cauchy point.

The objective of the procedure described in this section is to find the first local minimizer of the quadratic model along the piece-wise linear path obtained by projecting points along the steepest descent direction, $x_k - t g_k$, onto the feasible region. We define $x^0 = x_k$ and, throughout this section, drop the index k of the outer iteration, so that g, x and B stand for g_k, x_k and B_k. We use subscripts to denote the components of a vector; for example g_i denotes the i-th component of g. Superscripts will be used to represent iterates during the piece-wise search for the Cauchy point.

To define the breakpoints in each coordinate direction we compute

$$t_i = \begin{cases} (x_i^0 - u_i)/g_i & \text{if } g_i < 0 \\ (x_i^0 - l_i)/g_i & \text{if } g_i > 0 \\ \infty & \text{otherwise,} \end{cases} \tag{4.1}$$

and sort $\{t_i, i = 1, \ldots, n\}$ in increasing order to obtain the ordered set $\{t^j : t^j \leq t^{j+1}, j = 1, ..., n\}$. This is done by means of a heapsort algorithm [1]. We then search along $P(x^0 - tg, l, u)$, a piece-wise linear path which can be expressed as

$$x_i(t) = \begin{cases} x_i^0 - tg_i & \text{if } t \leq t_i \\ x_i^0 - t_i g_i & \text{otherwise.} \end{cases}$$

Suppose that we are examining the interval $[t^{j-1}, t^j]$. Let us define the $(j-1)$-th breakpoint as

$$x^{j-1} = x(t^{j-1})$$

so that on $[t^{j-1}, t^j]$

$$x(t) = x^{j-1} + \Delta t d^{j-1},$$

where

$$\Delta t = t - t^{j-1}$$

and

$$d_i^{j-1} = \begin{cases} -g_i & \text{if } t^{j-1} < t_i \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Using this notation we write the quadratic (2.1), on the line segment $[x(t^{j-1}), x(t^j)]$, as

$$\begin{aligned} m(x) &= f + g^T(x - x^0) + \tfrac{1}{2}(x - x^0)^T B(x - x^0) \\ &= f + g^T(z^{j-1} + \Delta t d^{j-1}) + \tfrac{1}{2}(z^{j-1} + \Delta t d^{j-1})^T B(z^{j-1} + \Delta t d^{j-1}), \end{aligned}$$

where

$$z^{j-1} = x^{j-1} - x^0. \tag{4.3}$$

Therefore on the line segment $[x(t^{j-1}), x(t^j)]$, $m(x)$ can be written as a quadratic in Δt,

$$\begin{aligned} \hat{m}(\Delta t) &= (f + g^T z^{j-1} + \tfrac{1}{2} {z^{j-1}}^T B z^{j-1}) + (g^T d^{j-1} + {d^{j-1}}^T B z^{j-1})\Delta t \\ &\quad + \tfrac{1}{2}({d^{j-1}}^T B d^{j-1})\Delta t^2 \\ &\equiv f_{j-1} + f'_{j-1}\Delta t + \tfrac{1}{2} f''_{j-1}\Delta t^2, \end{aligned}$$

where the parameters of this one-dimensional quadratic are

$$f_{j-1} = f + g^T z^{j-1} + \tfrac{1}{2} {z^{j-1}}^T B z^{j-1},$$

$$f'_{j-1} = g^T d^{j-1} + {d^{j-1}}^T B z^{j-1}, \tag{4.4}$$

$$f''_{j-1} = {d^{j-1}}^T B d^{j-1}. \tag{4.5}$$

Differentiating $\hat{m}(\Delta t)$ and equating to zero, we obtain $\Delta t^* = -f'_{j-1}/f''_{j-1}$. Since B is positive definite, this defines a minimizer provided $t^{j-1} + \Delta t^*$ lies on $[t^{j-1}, t^j)$. Otherwise the generalized Cauchy point lies at $x(t^{j-1})$ if $f'_{j-1} \geq 0$, and beyond or at $x(t^j)$ if $f'_{j-1} < 0$.

If the generalized Cauchy point has not been found after exploring the interval $[t^{j-1}, t^j]$, we set

$$x^j = x^{j-1} + \Delta t^{j-1} d^{j-1}, \qquad \Delta t^{j-1} = t^j - t^{j-1}, \tag{4.6}$$

and update the directional derivatives f'_j and f''_j as the search moves to the next interval. Let us assume for the moment that only one variable becomes active at t^j, and let us denote its index by b. Then $t_b = t^j$, and we zero out the corresponding component of the search direction,

$$d^j = d^{j-1} + g_b e_b, \tag{4.7}$$

where e_b is the b-th unit vector. From the definitions (4.3) and (4.6) we have

$$z^j = z^{j-1} + \Delta t^{j-1} d^{j-1}. \tag{4.8}$$

Therefore using (4.4), (4.5), (4.7) and (4.8) we obtain

$$\begin{aligned} f'_j &= g^T d^j + {d^j}^T B z^j \\ &= g^T d^{j-1} + g_b^2 + {d^{j-1}}^T B z^{j-1} + \Delta t^{j-1} {d^{j-1}}^T B d^{j-1} + g_b e_b^T B z^j \\ &= f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + g_b e_b^T B z^j \end{aligned} \tag{4.9}$$

and

$$\begin{aligned} f''_j &= {d^j}^T B d^j \\ &= {d^{j-1}}^T B d^{j-1} + 2 g_b e_b^T B d^{j-1} + g_b^2 {e_b}^T B e_b \\ &= f''_{j-1} + 2 g_b e_b^T B d^{j-1} + g_b^2 {e_b}^T B e_b. \end{aligned} \tag{4.10}$$

The only expensive computations in (4.9) and (4.10) are

$$ {e_b}^T B z^j, \quad {e_b}^T B d^{j-1}, \quad {e_b}^T B e_b, $$

which can require $O(n)$ operations since B is a dense limited memory matrix. Therefore it would appear that the the computation of the generalized Cauchy point could require $O(n^2)$ operations, since in the worst case n segments of the piece-wise linear path can be examined. This cost would be prohibitive for large problems. However, using the limited memory BFGS formula (3.2) and the definition (4.2), the updating formulae (4.9)-(4.10) become

$$f'_j = f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + \theta g_b z_b^j - g_b w_b^T M W^T z^j, \tag{4.11}$$

$$f''_j = f''_{j-1} - 2\theta g_b^2 - 2 g_b w_b^T M W^T d^{j-1} + \theta g_b^2 - g_b^2 w_b^T M w_b, \tag{4.12}$$

where w_b^T stands for the b-th row of the matrix W. The only $O(n)$ operations remaining in (4.11) and (4.12) are $W^T z^j$ and $W^T d^{j-1}$. We note, however, from (4.7) and (4.8) that z^j and d^j are updated at every iteration by a simple computation. Therefore if we maintain the two $2m$-vectors

$$p^j \equiv W^T d^j = W^T (d^{j-1} + g_b e_b) = p^{j-1} + g_b w_b,$$

$$c^j \equiv W^T z^j = W^T(z^{j-1} + \Delta t^{j-1} d^{j-1}) = c^{j-1} + \Delta t^{j-1} p^{j-1},$$

then updating f'_j and f''_j using the expressions

$$f'_j = f'_{j-1} + \Delta t^{j-1} f''_{j-1} + g_b^2 + \theta g_b z_b^j - g_b w_b^T M c^j,$$

$$f''_j = f''_{j-1} - \theta g_b^2 - 2 g_b w_b^T M p^{j-1} - g_b^2 w_b^T M w_b,$$

will require only $O(m^2)$ operations. If more than one variable becomes active at t^j – an atypical situation – we repeat the updating process just described, before examining the new interval $[t^j, t^{j+1}]$. We have thus been able to achieve a significant reduction in the cost of computing the generalized Cauchy point.

Remark. *The examination of the first segment of the projected steepest descent path, during the computation of the generalized Cauchy point, requires $O(n)$ operations. However all subsequent segments require only $O(m^2)$ operations, where m is the number of correction vectors stored in the limited memory matrix.*

Since m is usually small, say less than 10, the cost of examining all segments after the first one is negligible. The following algorithm describes in more detail how to achieve these savings in computation. Note that it is not necessary to keep track of the n-vector z^j since only the component z_b^j corresponding to the bound that has become active is needed to update $f_j{}'$ and $f_j{}''$.

Algorithm CP: Computation of the generalized Cauchy point.
Given x, l, u, g, and $B = \theta I - WMW^T$

For $\quad i = 1, \ldots, n$ compute

$$t_i := \begin{cases} (x_i - u_i)/g_i & \text{if } g_i < 0 \\ (x_i - l_i)/g_i & \text{if } g_i > 0 \\ \infty & \text{otherwise} \end{cases} \qquad (n \text{ operations})$$

$$d_i := \begin{cases} 0 & \text{if } t_i = 0 \\ -g_i & \text{otherwise} \end{cases}$$

- Initialize

$$\begin{aligned}
\mathcal{F} &:= \{ i : t_i > 0 \} \\
p &:= W^T d \qquad (2mn \text{ operations }) \\
c &:= 0 \\
f' &:= g^T d = -d^T d \qquad (n \text{ operations}) \\
f'' &: = \theta d^T d - d^T W M W^T d = -\theta f' - p^T M p \qquad (O(m^2) \text{ operations}) \\
\Delta t_{min} &:= -\tfrac{f'}{f''} \\
t_{old} &:= 0 \\
t &:= \min\{t_i : i \in \mathcal{F}\} \quad (\text{using the heapsort algorithm}) \\
b &:= i \text{ such that } t_i = t \qquad (\text{remove } b \text{ from } \mathcal{F}). \\
\Delta t &:= t - 0
\end{aligned}$$

- Examination of subsequent segments
  While $\Delta t_{min} \geq \Delta t$ do

$$
\begin{aligned}
x_b^{cp} &:= \begin{cases} u_b & \text{if } d_b > 0 \\ l_b & \text{if } d_b < 0 \end{cases} \\
z_b &:= x_b^{cp} - x_b \\
c &:= c + \Delta t p \qquad (O(m) \text{ operations}) \\
f' &:= f' + \Delta t f'' + g_b^2 + \theta g_b z_b - g_b w_b^T M c \qquad (O(m^2) \text{ operations}) \\
f'' &:= f'' - \theta g_b^2 - 2 g_b w_b^T M p - g_b^2 w_b^T M w_b \qquad (O(m^2) \text{ operations}) \\
p &:= p + g_b w_b \qquad (O(m) \text{ operations}) \\
d_b &:= 0 \\
\Delta t_{min} &:= -\frac{f'}{f''} \\
t_{old} &:= t \\
t &:= \min\{t_i : i \in \mathcal{F}\} \qquad (\text{using the heapsort algorithm}) \\
b &:= i \text{ such that } t_i = t \qquad (\text{Remove } b \text{ from } \mathcal{F}) \\
\Delta t &:= t - t_{old}
\end{aligned}
$$

  end while

- $\Delta t_{min} := max\{\Delta t_{min}, 0\}$
- $t_{old} := t_{old} + \Delta t_{min}$
- $x_i^{cp} := x_i + t_{old} d_i$, $\forall\, i$ such that $t_i \geq t$
- For all $i \in \mathcal{F}$ with $t_i = t$, remove i from $\mathcal{F}$.
- $c := c + \Delta t_{min} p$

The last step of this algorithm updates the $2m$-vector c so that upon termination

$$c = W^T(x^c - x_k). \tag{4.13}$$

This vector will be used to initialize the subspace minimization when the primal direct method or the conjugate gradient method are used, as will be discussed in the next section.

Our operation counts only take into account multiplications and divisions. Note that there are no $O(n)$ computations inside the loop. If n_{int} denotes the total number of segments explored, then the total cost of Algorithm CP is $(2m+2)n + O(m^2) \times n_{int}$ operations plus $n \log n$ operations which is the approximate cost of the heapsort algorithm [1].

5. Methods for subspace minimization.

Once the Cauchy point x^c has been found, we proceed to approximately minimize the quadratic model m_k over the space of free variables, and impose the bounds on the problem. We consider three approaches to minimize the model, a direct primal method based on the Sherman-Morrison-Woodbury formula, a primal iterative method using the conjugate gradient method, and a direct dual method using Lagrange multipiers. Which of these is most appropriate seems problem dependent, and we have experimented numerically with all three. In all these approaches we first work on minimizing m_k ignoring the bounds, and at an appropriate point truncate the move so as to satisfy the bound constraints.

The following notation will be used throughout this section. The integer t denotes the number of free variables at the Cauchy point x^c; in other words there are $n-t$ variables at bound at x^c. As in the previous section $\mathcal{F}$ denotes the set of indices corresponding to the free variables, and we note that this set is defined upon completion of the Cauchy point computation. We define Z_k to be the $n \times t$ matrix whose columns are unit vectors (i.e. columns of the identity matrix) that span the subspace of the free variables at x^c. Similarly A_k denotes the $n \times (n-t)$ matrix of active constraint gradients at x^c, which consists of $n-t$ unit vectors. Note that $A_k^T Z_k = 0$ and that

$$A_k A_k^T + Z_k Z_k^T = I. \tag{5.1}$$

5.1. A Direct Primal Method.

In a primal approach, we fix the $n-t$ variables at bound at the generalized Cauchy point x^c, and solve the quadratic problem (2.3) over the subspace of the remaining t free variables, starting from x^c and imposing the free variable bounds (2.4). Thus we consider only the points $x \in \Re^n$ of the form

$$x = x^c + Z_k \hat{d}, \tag{5.2}$$

where $\hat{d}$ is a vector of dimension t. Using this notation, for points of the form (5.2) we can write the quadratic (2.1) as

$$\begin{aligned} m_k(x) &= f_k + g_k^T(x - x^c + x^c - x_k) + \frac{1}{2}(x - x^c + x^c - x_k)^T B_k (x - x^c + x^c - x_k) \\ &= (g_k + B_k(x^c - x_k))^T (x - x^c) + \frac{1}{2}(x - x^c)^T B_k (x - x^c) + \gamma \\ &\equiv \hat{d}^T \hat{r}^c + \frac{1}{2}\hat{d}^T \hat{B}_k \hat{d} + \gamma, \end{aligned} \tag{5.3}$$

where γ is a constant,

$$\hat{B}_k = Z_k^T B_k Z_k$$

is the reduced Hessian of m_k, and

$$\hat{r}^c = Z_k^T (g_k + B_k(x^c - x_k))$$

is the reduced gradient of m_k at x^c. Using (3.2) and (4.13), we can express this reduced gradient as

$$\hat{r}^c = Z_k^T (g_k + \theta(x^c - x_k) - W_k M_k c), \tag{5.4}$$

which, given that the vector c was saved from the Cauchy point computation, costs $(2m+1)t + O(m^2)$ extra operations. Then the subspace problem (2.3) can be formulated as

$$\min \qquad \hat{m}_k(\hat{d}) \equiv \hat{d}^T \hat{r}^c + \tfrac{1}{2}\hat{d}^T \hat{B}_k \hat{d} + \gamma \tag{5.5}$$

$$\text{subject to} \quad l_i - x_i^c \le \hat{d}_i \le u_i - x_i^c \qquad i \in \mathcal{F}, \tag{5.6}$$

where the subscript i denotes the i-th component of a vector. The minimization (5.5) can be solved either by a direct method, as we discuss here, or by an iterative method as discussed in the next subsection, and the constraints (5.6) can be imposed by backtracking.

Since the reduced limited memory matrix $\hat{B}_k$ is a small-rank correction of a diagonal matrix, we can formally compute its inverse by means of the Sherman-Morrison-Woodbury formula, and obtain the unconstrained solution of the subspace problem (5.5),

$$\hat{d}^u = -\hat{B}_k^{-1} \hat{r}^c. \tag{5.7}$$

We can then backtrack towards the feasible region, if necessary, to obtain

$$\hat{d}^* = \alpha^* \hat{d}^u,$$

where the positive scalar α^* is defined by

$$\alpha^* = \max\{\alpha : \ \alpha \le 1, \ l_i - x_i^c \le \alpha \hat{d}_i^u \le u_i - x_i^c, \quad i \in \mathcal{F}\}. \tag{5.8}$$

Therefore the approximate solution $\bar{x}$ of the subproblem (2.3)-(2.4) is given by

$$\bar{x}_i = \begin{cases} x_i^c & \text{if } i \notin \mathcal{F} \\ x_i^c + (Z_k \hat{d}^*)_i & \text{if } i \in \mathcal{F}. \end{cases} \tag{5.9}$$

It only remains to consider how to perform the computation in (5.7). Since B_k is given by (3.2) and $Z_k^T Z_k = I$, the reduced matrix $\hat{B}$ is given by

$$\hat{B} = \theta I - (Z^T W)(M W^T Z),$$

where we have dropped the subscripts for simplicity. Applying the Sherman-Morrison-Woodbury formula (see for example [22]), we obtain

$$\hat{B}^{-1} = \frac{1}{\theta} I + \frac{1}{\theta} Z^T W \left(I - \frac{1}{\theta} M W^T Z Z^T W\right)^{-1} M W^T Z \frac{1}{\theta}, \tag{5.10}$$

so that the unconstrained subspace Newton direction $\hat{d}^u$ is given by

$$\hat{d}^u = \frac{1}{\theta} \hat{r}^c + \frac{1}{\theta^2} Z^T W \left(I - \frac{1}{\theta} M W^T Z Z^T W\right)^{-1} M W^T Z \hat{r}^c. \tag{5.11}$$

Given a set of free variables at x^c that determines the matrix Z, and a limited memory BFGS matrix B defined in terms of θ, W and M, the following procedure implements the approach just described. Note that since the columns of Z are unit vectors, the operation Zv, amounts to selecting appropriate elements from v. Here and throughout the paper our operation counts include only multiplications and divisions. Recall that t denotes the number of free variables and that m is the number of corrections stored in the limited memory matrix.

Direct Primal Method.

1. Compute $Z\hat{r}^c$ by (5.4) $\quad$ ($(2m+1)t + O(m^2)$ operations)
2. $v := W^T Z\hat{r}^c$ $\quad$ ($2mt$ operations)
3. $v := Mv$ $\quad$ ($O(m^2)$ operations)
4. Form $N \equiv (I - \frac{1}{\theta} MW^T ZZ^T W)$
   - $N := \frac{1}{\theta} W^T ZZ^T W$ $\quad$ ($2m^2 t + mt$ operations)
   - $N := I - MN$ $\quad$ ($O(m^3)$ operations)
5. $v := N^{-1} v$ $\quad$ ($O(m^3)$ operations)
6. $\hat{d}^u := \frac{1}{\theta}\hat{r}^c + \frac{1}{\theta^2} Z^T W v$ $\quad$ ($2mt + t$ operations)
7. Find α^* satisfying (5.8) $\quad$ (t operations)
8. Compute $\bar{x}_i$ as in (5.9) $\quad$ (t operations)

The total cost of this subspace minimization step based on the Sherman-Morrison-Woodbury formula is

$$2m^2 t + 6mt + 4t + O(m^3) \tag{5.12}$$

operations. This is quite acceptable when t is small, i.e. when there are few free variables. However in many problems the opposite is true: few constraints are active and t is large. In this case the cost of the direct primal method can be quite large, but the following mechanism can provide significant savings.

Note that when t is large, it is the computation of the matrix

$$W^T ZZ^T W = \begin{bmatrix} Y^T \\ \theta S^T \end{bmatrix} ZZ^T \begin{bmatrix} Y & \theta S \end{bmatrix} = \begin{bmatrix} Y^T ZZ^T Y & \theta Y^T ZZ^T S \\ \theta S^T ZZ^T Y & \theta^2 S^T ZZ^T S \end{bmatrix}$$

in step 4, which requires $2m^2 t$ operations, that drives up the cost. Fortunately we can reduce the cost when only a few variables enter or leave the active set from one iteration to the next by saving the matrices $Y^T ZZ^T Y$, $S^T ZZ^T Y$ and $S^T ZZ^T S$. These matrices can be updated to account for the parts of the inner products corresponding to variables that have changed status, and to add rows and columns corresponding to the new step. In our computational experiments in §6 we have implemented this device. In addition, when t is much larger then $n - t$, it seems more efficient to use the relationship $Y^T ZZ^T Y = Y^T Y - Y^T AA^T Y$, which follows from (5.1), to compute $Y^T ZZ^T Y$. Similar relationships can be used for the matrices $S^T ZZ^T Y$ and $S^T ZZ^T S$. An expression using these relationships is described at the end of §5.3.

5.2. A primal conjugate gradient method.

Another approach for approximately solving the subspace problem (5.5) is to apply the conjugate gradient method to the positive definite linear system

$$\hat{B}_k \hat{d}^u = -\hat{r}^c, \tag{5.13}$$

and stop the iteration when a boundary is encountered or when the residual is small enough. Note that the accuracy of the solution controls the rate of convergence of the algorithm, once the correct active set is identified, and should therefore be chosen with care. We follow Conn, Gould and Toint [8] and stop the conjugate gradient iteration when the residual $\hat{r}$ of (5.13) satisfies

$$\|\hat{r}\| < \min(0.1, \sqrt{\|\hat{r}^c\|})\|\hat{r}^c\|.$$

We also stop the iteration at a bound when a conjugate gradient step is about to violate a bound, thus guaranteeing that (5.6) is satisfied. The conjugate gradient method is appropriate here since almost all of the eigenvalues of $\hat{B}_k$ are identical.

We now describe the conjugate gradient method and give its operation counts. Note that the effective number of variables is t, the number of free variables. Given B_k, the following procedure computes an approximate solution of (5.5).

The conjugate gradient method.

1. $\hat{r} := \hat{r}^c$ computed by (5.4) $\quad$ ($(2m+1)t + O(m^2)$ operations)
2. $p := -\hat{r}$, $\hat{d} := 0$, and $\rho_2 := \hat{r}^T\hat{r}$ $\quad$ (t operations)
3. Stop if $\|\hat{r}\| < \min(0.1, \sqrt{\|\hat{r}^c\|})\|\hat{r}^c\|$
4. $\alpha_1 := \max\{\alpha : \hat{l} \leq \hat{x}^c + \hat{d} + \alpha p \leq \hat{u}\}$ $\quad$ (t operations)
5. $q := \hat{B}_k p$ $\quad$ ($4mt$ operations)
6. $\alpha_2 := \rho_2 / p^T q$ $\quad$ (t operations)
7. If $\alpha_2 > \alpha_1$ set $\hat{d} := \hat{d} + \alpha_1 p$ and stop;
   otherwise compute:
   - $\hat{d} := \hat{d} + \alpha_2 p$ $\quad$ (t operations)
   - $\hat{r} := \hat{r} + \alpha_2 q$ $\quad$ (t operations)
   - $\rho_1 := \rho_2$; $\rho_2 = \hat{r}^T\hat{r}$; $\beta := \rho_2/\rho_1$ $\quad$ (t operations)
   - $p := -\hat{r} + \beta p$ $\quad$ (t operations)
   - go to 3

The matrix-vector multiplication of step 5 should be performed as described in [6]. The total operation count of this conjugate gradient procedure is approximately

$$(2m+2)t + (4m+6)t \times citer + O(m^2), \tag{5.14}$$

where *citer* is the number of conjugate gradient iterations. Comparing this to the cost of the primal direct method (5.12) it seems that, for $t >> m$, the direct method is more efficient unless $citer \leq m/2$. Note that the costs of both methods increase as the number of free variables t becomes larger. Since the limited memory matrix B_k is a rank $2m$ correction of the identity matrix, the termination properties of the conjugate gradient method guarantee that the subspace problem will be solved in at most $2m$ conjugate gradient iterations.

We should point out that the conjugate gradient iteration could stop at a boundary even when the unconstrained solution of the subspace problem is inside the box. Consider for example the case when the unconstrained solution lies near a corner and the starting point of the conjugate gradient iteration lies near another corner along the same edge of the box. Then the iterates could soon fall outside of the feasible region. This example also illustrates the difficulties that the conjugate gradient approach can have on nearly degenerate problems [11].

5.3. A dual method for subspace minimization.

Since it often happens that the number of active bounds is small relative to the size of the problem it should be efficient to handle these bounds explicitly with Lagrange multipliers. Such an approach is often referred to as a dual or a range space method (see [15]).

We will write

$$x \equiv x_k + d,$$

and restrict $x_k + d$ to lie on the subspace of free variables at x^c by imposing the condition

$$\begin{aligned} A_k^T d &= A_k^T (x^c - x_k) \\ &\equiv b_k. \end{aligned}$$

(Recall that A_k is the matrix of constraint gradients.) Using this notation we formulate the subspace problem as

$$\min \quad {g_k}^T d + \tfrac{1}{2} d^T B_k d \tag{5.15}$$

$$\text{subject to} \quad A_k^T d = b_k \tag{5.16}$$

$$l \leq x_k + d \leq u. \tag{5.17}$$

We first solve this problem without the bound constraint (5.17). The optimality conditions for (5.15)-(5.16) are

$$g_k + B_k d^* + A_k \lambda^* = 0, \tag{5.18}$$

$$A_k^T d^* = b_k. \tag{5.19}$$

Pre-multiplying (5.18) by $A_k^T H_k$, where H_k is the inverse of B_k, we obtain

$$A_k^T H_k g_k + A_k^T d^* + A_k^T H_k A_k \lambda^* = 0,$$

and using (5.19) we obtain

$$(A_k^T H_k A_k)\lambda^* = -A_k^T H_k g_k - b_k. \tag{5.20}$$

Since the columns of A_k are unit vectors and A_k has full column rank, we see $A_k^T H_k A_k$ is a principal submatrix of H_k. Thus (5.20) determines λ^*, and hence d^* is given by

$$B_k d^* = -A_k \lambda^* - g_k. \tag{5.21}$$

(In the special case where there are no active constraints, we simply obtain $B_k d^* = -g_k$.) If the vector $x_k + d^*$ violates the bounds (5.17), we backtrack to the feasible region along the line joining this infeasible point and the generalized Cauchy point x^c.

The linear system (5.20) can be solved by the Sherman-Morrison-Woodbury formula. Using the inverse limited memory BFGS matrix (3.7), and recalling the identity $A_k^T A_k = I$, we obtain

$$A_k^T H_k A_k = \frac{1}{\theta} I + (A^T \bar{W})(\bar{M} \bar{W}^T A).$$

(We have again omitted the subscripts of M, W and θ for simplicity.) Applying the Sherman-Morrison-Woodbury formula we obtain

$$(A_k^T H_k A_k)^{-1} = \theta I - \theta A_k^T \bar{W}(I + \theta \bar{M} \bar{W}^T A_k A_k^T \bar{W})^{-1} \bar{M} \bar{W}^T A_k \theta. \tag{5.22}$$

Given g_k, a set of active variables at x^c that determines the matrix of constraint gradients A_k, and an inverse limited memory BFGS matrix H_k, the following procedure implements the dual approach just described. Let us recall that t denotes the number of free variables and let us define $t_a = n - t$, so that t_a denotes the number of active constraints at x^c. As before, the operation counts given below include only multiplications and divisions, and m denotes the number of corrections stored in the limited memory matrix.

Dual method

If $t_a = 0$, compute $d^* = -H_k g_k = -\frac{1}{\theta} g_k - \bar{W}\bar{M}\bar{W}^T g_k$ as follows

- $w := \bar{W}^T g_k$ (0 operations)
- $w := \bar{M} w$ ($O(m^2)$ operations)
- $d^* := -\frac{1}{\theta} g_k - \bar{W} w$ ($(2m+1)n$ operations)

If $t_a > 0$, compute

1. $u = -A_k^T H_k g_k - b = -\frac{1}{\theta} A_k^T g_k - A^T \bar{W}\bar{M}\bar{W}^T g_k - b$
   - $b := A_k^T (x^c - x_k)$ (0 operations)

- $v := \bar{W}^T g_k$ (0 operations)
- $v := \bar{M} w$ ($O(m^2)$ operations)
- $u := A_k^T \bar{W} v$ ($2mt_a$ operations)
- $u := -\frac{1}{\theta} A_k^T g_k - u - b$ (t_a operations)

2. $\lambda^* = -(A_k^T H_k A_k)^{-1} u = -\theta u + \theta^2 A_k^T \bar{W}(I + \theta \bar{M} \bar{W}^T A_k A_k^T \bar{W})^{-1} \bar{M} \bar{W}^T A_k u$

- $w := \bar{W}^T A_k u$ ($2mt_a$ operations)
- Form $\bar{N} = (I + \theta \bar{M} \bar{W}^T A_k A_k^T \bar{W})$
  - $\bar{N} := \theta W^T A_k A_k^T W$ ($(2m^2 + m)t_a$ operations)
  - $\bar{N} := I + \bar{M} \bar{N}$ ($O(m^3)$ operations)
- $w := \bar{N}^{-1} \bar{M} w$ ($O(m^3)$ operations)
- $\lambda^* := \theta^2 A_k^T \bar{W} w$ ($2mt_a$ operations)
- $\lambda^* := -\theta u + \lambda^*$ (t_a operations)

3. $d^* = -H_k(A_k \lambda^* + g_k) = -\frac{1}{\theta}(A_k \lambda^* + g_k) - \bar{W}(\bar{M} \bar{W}^T A_k \lambda^* + v)$

- $w := \bar{W}^T A_k \lambda^*$ ($2mt_a$ operations)
- $w := \bar{M} w + v$ ($2m$ operations)
- $d^* := -\frac{1}{\theta}(A_k \lambda^* + g_k) + \bar{W} w$ ($(2m+1)n$ operations)

Backtrack if necessary:

- Compute α^* =max $\{\alpha : \, l_i \leq x_c + \alpha(x_k + d^* - x^c) \leq u_i, i \in \mathcal{F}\}$ (t operations)
- Set $\bar{x} = x_c + \alpha^*(x_k + d^* - x^c)$. ($t$ operations)

Since the vectors $S^T g_k$ and $Y^T g_k$ have been computed while updating H_k [6], they can be saved so that the product $\bar{W}^T g_k$ requires no further computation.

The total number of operations of this procedure, when no bounds are active ($t_a = 0$), is $(2m+1)n + O(m^2)$. If t_a bounds are active,

$$(2m+3)n + 9mt_a + 2m^2 t_a + O(m^3)$$

operations are required to compute the unconstrained subspace solution. A comparison with the cost of the primal computation implemented as described above, given in (5.12), indicates that the dual method would be less expensive when the number of bound variables is much less than the number of free variables.

However, this comparison does not take into account the devices for saving costs in the computation of inner products discussed at the end of §5.1. In fact, the primal and dual approaches can be brought closer by noting that the matrix $(I - \frac{1}{\theta} M W^T Z Z^T W)^{-1} M$ appearing in (5.10) can be shown to be identical to the matrix $(I + \theta \bar{M} \bar{W}^T A_k A_k^T \bar{W})^{-1} \bar{M}$ in (5.22). If the second expression is used in the primal approach, the cost for the primal computation becomes $2m^2t_a + 6mt + 4t + O(m^3)$, making it more competitive with the dual approach. We have used this expression for the primal method in our computational experiments. Also, in the dual approach, as with the primal, we save and reuse the inner products in $\bar{W}^T A_k A_k^T \bar{W}$ that are relevant for the next iteration.

6. Numerical Experiments

We have tested our limited memory algorithm using the three options for subspace minimization (the direct primal, primal conjugate gradient and dual methods), and compared the results with those obtained with the subroutine SBMIN of LANCELOT [10]. Both our code and LANCELOT were terminated when

$$\|P(x_k - g_k, l, u) - x_k\|_\infty < 10^{-5}. \tag{6.1}$$

(Note from (2.2) that $P(x_k - g_k, l, u) - x_k$ is the projected gradient.) The algorithm we tested is given as follows.

L-BFGS-B Algorithm
Choose a starting point x_0, and a integer m that determines the number of limited memory corrections stored. Define the initial limited memory matrix to be the identity and set $k := 0$.

1. If the convergence test (6.1) is satisfied stop.
2. Compute the Cauchy point by Algorithm CP.
3. Compute a search direction d_k by either the direct primal method, the conjugate gradient method or the dual method.
4. Perform a line search along d_k, subject to the bounds on the problem, to compute a steplength λ_k, and set $x_{k+1} = x_k + \lambda_k d_k$. The line search starts with the unit steplength, satisfies (2.5) with $\alpha = 10^{-4}$, and attempts to satisfy (2.6) with $\beta = 0.9$.
5. Compute $\nabla f(x_{k+1})$.
6. If y_k satisfies (3.9) with eps$= 2.2 \times 10^{-16}$, add s_k and y_k to S_k and Y_k. If more than m updates are stored, delete the oldest column from S_k and Y_k.
7. Update $S_k^T S_k$, $Y_k^T Y_k$, L_k and R_k, and set $\theta = y_k^T y_k / y_k^T s_k$.
8. Set $k := k + 1$ and go to 1.

The line search was performed by means of the routine of Moré and Thuente [19] which attempts to enforce the Wolfe conditions (2.5) and (2.6) by a sequence of polynomial interpolations. Since steplengths greater than one may be tried, we prevent the routine from generating infeasible points by defining the maximum steplength for the routine as the step to the closest bound along the current search direction. This approach implies that, if the objective function is bounded below, the line search will generate a point x_{k+1} that satisfies (2.5) and either satisfies (2.6) or hits a bound that was not active at x_k. We have observed that this line search performs well in practice. The first implementation of our algorithm used a backtracking line search, but we found that this approach has a drawback. On several problems the backtracking line search generated steplengths for which the updating condition (3.9) did not hold, and the BFGS update had to be skipped. This resulted in very poor performance in some problems. In contrast, when using the new line search, the update is very rarely skipped in our tests, and the performance of the code was markedly better. To explore this further we compared the early version of our code, using the backtracking line search, with the L-BFGS code for unconstrained optimization [18] (which uses the routine of Moré and Thuente) on *unconstrained* problems, and found that L-BFGS was superior in a significant number of problems. Our results suggest that backtracking line searches can significantly degrade the performance of BFGS, and that satisfying the Wolfe conditions as often as possible is important in practice.

Our code is written in double precision FORTRAN 77. For more details on how to update the limited memory matrices in step 7 see [6]. When testing the routine SBMIN of LANCELOT [10] we tried three options: BFGS, SR1 and exact Hessians. In all these cases we used the default settings of LANCELOT.

The test problems were selected from the CUTE collection [4] (which contains some problems from the MINPACK-2 collection [2]). All bound constrained problems in CUTE were tested, but some problems were discarded for one of the following reasons: (a) the number of variables was less than 4; (b) the various algorithms converged to a different solution point; (c) too many instances of essentially the same problem are given in CUTE; in this case we selected a representative sample.

The results of our numerical tests are given in Tables 1, 2 and 3. All computations were performed on a Sun SPARCstation 2 with a 40-MHz CPU and 32-MB memory. We record the number of variables n the number of function evaluations and the total run time. The limited memory code always computes the function and gradient together, so that nfg denotes the total number of function and gradient evaluations. However for LANCELOT the number of function evaluations is not necessarily the same as the number of gradient evaluations, and in the tables we only record the number (nf) of function evaluations. The notation F1 indicates that the solution was not obtained after 999 function evaluations, and F2 indicates that the run was terminated because the next iterate generated from the line search was too close to the current iterate to be recognized as distinct. This occurred only in some cases when the limited memory method was quite near the solution but was unable to meet the stopping criterion. In Table 1 nact denotes the number of active bounds at the solution; if this number is zero it does not mean that bounds were not encountered during the solution process.

Problem	n	nact	L-BFGS-B m=5		L-BFGS-B m=17		LANCELOT BFGS		LANCELOT SR1	
			nfg	time	nfg	time	nf	time	nf	time
ALLINIT	4	1	19	0.13	19	0.16	18	0.19	12	0.14
BDEXP	100	0	15	0.28	16	0.40	18	1.31	24	1.59
BIGGS5	6	1	117	1.08	69	1.80	47	0.70	41	0.58
BQPGASIM	50	7	25	0.36	23	0.54	322	9.07	8	0.59
BQPGAUSS	2003	27	F1	349.75	F1	631.31	F1	1722.40	20	2149.33
HATFLDA	4	0	39	0.21	36	0.32	39	0.32	36	0.31
HATFLDB	4	1	34	0.16	32	0.26	31	0.25	33	0.31
HATFLDC	25	0	23	0.26	23	0.44	5	0.14	5	0.11
HS110	50	50	2	0.03	2	0.06	2	0.17	2	0.13
HS25	3	1	2	0.06	2	0.05	1	0.06	1	0.05
HS45	5	5	11	0.04	11	0.02	3	0.05	3	0.04
JNLBRNGA	15625	5657	313	843.59	280	1222.33	24	1478.94	24	1474.09
JNLBRNGB	100	68	61	1.23	55	2.33	6	0.45	6	0.45
LINVERSE	19	6	63	0.71	107	3.88	F1	19.72	18	0.51
MAXLIKA	8	1	F1	59.09	157	12.31	F1	208.60	98	24.65
MCCORMCK	10	0	11	0.10	11	0.13	7	0.13	7	0.10
NONSCOMP	25	0	31	0.34	30	0.72	6	0.15	9	0.19
OBSTCLAE	5625	2724	282	277.71	308	495.42	7	1765.61	7	1763.42
OBSTCLAL	100	63	15	0.30	14	0.32	8	0.61	8	0.50
OBSTCLBL	100	84	11	0.24	11	0.25	5	0.44	5	0.45
OBSTCLBM	15625	4309	133	383.61	138	641.45	7	1275.77	7	1275.72
OBSTCLBU	100	84	12	0.25	12	0.26	4	0.39	4	0.43
PALMER1	4	0	34	0.25	24	0.20	37	0.61	32	0.52
PALMER2	4	0	F1	2.14	30	0.33	88	1.23	27	0.36
PALMER3	4	1	F2	0.59	F2	0.19	71	0.92	49	0.61
PALMER4	4	0	25	0.15	25	0.16	F1	12.13	44	0.57
PROBPENL	500	0	3	0.37	3	0.38	4	2.51	3	1.98
PSPDOC	4	0	11	0.07	11	0.10	3	0.06	3	0.06
S368	8	2	11	0.12	12	0.16	F1	24.58	33	0.77
TORSION1	100	68	12	0.22	10	0.16	7	0.59	7	0.53
TORSION2	100	68	11	0.21	10	0.19	6	0.37	6	0.37
TORSION3	100	88	5	0.10	5	0.11	5	0.38	5	0.27
TORSION4	100	88	7	0.15	7	0.15	4	0.37	4	0.37
TORSION6	14884	12316	301	835.80	400	1582.75	10	171.64	10	168.67

Table 1. Test results of new limited memory method (L-BFGS-B) using primal method for subspace minimization, and results of LANCELOT's BFGS and SR1 options.

Problem	n	L-BFGS-B m=3		L-BFGS-B m=5		L-BFGS-B m=17		L-BFGS-B m=29	
		nfg	time	nfg	time	nfg	time	nfg	time
ALLINIT	4	22	0.13	19	0.13	19	0.16	19	0.17
BDEXP	100	15	0.29	15	0.28	16	0.40	16	0.44
BIGGS5	6	127	0.74	117	1.08	69	1.80	71	3.20
BQPGASIM	50	28	0.34	25	0.36	23	0.54	23	0.56
BQPGAUSS	2003	F1	352.09	F1	349.75	F1	631.31	F1	932.04
HATFLDA	4	47	0.17	39	0.21	36	0.32	37	0.50
HATFLDB	4	37	0.15	34	0.16	32	0.26	32	0.27
HATFLDC	25	25	0.23	23	0.26	23	0.44	23	0.50
HS110	50	2	0.05	2	0.03	2	0.06	2	0.04
HS25	3	2	0.07	2	0.06	2	0.05	2	0.07
HS45	5	11	0.04	11	0.04	11	0.02	11	0.04
JNLBRNGA	15625	382	891.11	313	843.59	280	1222.33	322	1916.31
JNLBRNGB	100	84	1.32	61	1.23	55	2.33	56	3.55
LINVERSE	19	96	0.91	63	0.71	107	3.88	99	4.68
MAXLIKA	8	F1	57.76	F1	59.09	157	12.31	118	13.32
MCCORMCK	10	12	0.10	11	0.10	11	0.13	11	0.11
NONSCOMP	25	32	0.29	31	0.34	30	0.72	30	0.88
OBSTCLAE	5625	266	253.63	282	277.71	308	495.42	284	641.66
OBSTCLAL	100	16	0.32	15	0.30	14	0.32	14	0.24
OBSTCLBL	100	12	0.22	11	0.24	11	0.25	11	0.22
OBSTCLBM	15625	140	349.89	133	383.61	138	641.45	153	948.07
OBSTCLBU	100	11	0.22	12	0.25	12	0.26	12	0.25
PALMER1	4	34	0.19	34	0.25	24	0.20	24	0.20
PALMER2	4	37	0.21	F1	2.14	30	0.33	30	0.31
PALMER3	4	93	0.50	F2	0.59	F2	0.19	F2	0.20
PALMER4	4	F2	0.63	25	0.15	25	0.16	25	0.16
PROBPENL	500	3	0.36	3	0.37	3	0.38	3	0.36
PSPDOC	4	15	0.10	11	0.07	11	0.10	11	0.09
S368	8	12	0.16	11	0.12	12	0.16	12	0.15
TORSION1	100	12	0.17	12	0.22	10	0.16	10	0.17
TORSION2	100	14	0.23	11	0.21	10	0.19	10	0.19
TORSION3	100	5	0.10	5	0.10	5	0.11	5	0.11
TORSION4	100	8	0.17	7	0.15	7	0.15	7	0.16
TORSION6	14884	298	646.21	301	835.80	400	1582.75	385	2442.59

Table 2. Results of new limited memory method, using the primal method for subspace minimization, for various values of the memory parameter m.

Problem	n	L-BFGS-B primal		L-BFGS-B dual		L-BFGS-B cg		LANCELOT Hessian	
		nfg	time	nfg	time	nfg	time	nf	time
ALLINIT	4	19	0.13	19	0.13	19	0.13	7	0.09
BDEXP	100	15	0.28	15	0.28	15	0.34	11	0.80
BIGGS5	6	117	1.08	130	0.97	101	0.99	19	0.29
BQPGASIM	50	25	0.36	25	0.34	26	0.45	4	0.40
BQPGAUSS	2003	F1	349.75	F1	293.81	F1	471.29	9	1908.49
HATFLDA	4	39	0.21	39	0.18	43	0.22	24	0.22
HATFLDB	4	34	0.16	34	0.15	35	0.15	21	0.20
HATFLDC	25	23	0.26	23	0.20	24	0.29	5	0.14
HS110	50	2	0.03	2	0.05	2	0.05	2	0.17
HS25	3	2	0.06	2	0.07	2	0.07	1	0.06
HS45	5	11	0.04	11	0.04	11	0.02	3	0.04
JNLBRNGA	15625	313	843.59	347	988.31	552	1629.03	22	1775.51
JNLBRNGB	100	61	1.23	61	1.16	62	1.35	5	0.40
LINVERSE	19	63	0.71	63	0.64	192	1.79	19	0.50
MAXLIKA	8	F1	59.09	F1	59.90	801	46.88	9	2.09
MCCORMCK	10	11	0.10	11	0.08	11	0.07	5	0.08
NONSCOMP	25	31	0.34	31	0.30	34	0.37	9	0.20
OBSTCLAE	5625	282	277.71	292	292.78	301	302.50	6	1696.45
OBSTCLAL	100	15	0.30	15	0.30	15	0.35	5	0.50
OBSTCLBL	100	11	0.24	11	0.27	11	0.21	3	0.36
OBSTCLBM	15625	133	383.61	151	431.99	230	650.09	6	2300.28
OBSTCLBU	100	12	0.25	12	0.32	11	0.22	2	0.31
PALMER1	4	34	0.25	34	0.23	32	0.21	30	0.44
PALMER2	4	F1	2.14	F1	1.77	31	0.19	21	0.28
PALMER3	4	F2	0.59	F2	0.66	34	0.25	38	0.48
PALMER4	4	25	0.15	25	0.14	30	0.19	41	0.55
PROBPENL	500	3	0.37	3	0.39	3	0.39	2	1.59
PSPDOC	4	11	0.07	11	0.09	11	0.08	7	0.10
S368	8	11	0.12	11	0.11	12	0.16	5	0.13
TORSION1	100	12	0.22	12	0.21	12	0.20	4	0.31
TORSION2	100	11	0.21	11	0.26	11	0.21	4	0.35
TORSION3	100	5	0.10	5	0.11	5	0.11	2	0.29
TORSION4	100	7	0.15	7	0.19	7	0.15	3	0.31
TORSION6	14884	301	835.80	318	991.12	364	841.01	9	150.15

Table 3. Results of new limited memory method using three methods for subspace minimization (primal, dual and cg), for $m = 5$, and results of LANCELOT using exact Hessian.

The differences in the number of function evaluations required by the direct primal and dual methods are due to rounding errors, and are relatively small. Their computing time is also quite similar due to use of the form described at the end of §5.3. Our computational experience suggests to us that the conjugate gradient method for subspace minimization is the least effective approach; it tends to take more time and function evaluations. Even though Table 3 appears to indicate that the cg option results in fewer failures, tests with different values of m resulted, overall, in three more failures for the cg method than for the primal method. The limited memory method is sometimes unable to locate the solution accurately, and this can result in an excessive number of function evaluations (F1) or failure to make further progress (F2). The reasons for this are not clear to us, but are being investigated.

The tests described here are not intended to establish the superiority of LANCELOT or of the new limited memory algorithm, since these methods are designed for solving different types of problems. LANCELOT is tailored for sparse or partially separable problems whereas the limited memory method is well suited for unstructured or dense problems. We use LANCELOT simply as a benchmark, and for this reason ran it only with its default settings and did not experiments with its various options to find the one that would give the best results on these problems. However, a few observations on the two methods can be made.

The results in Table 1 indicate that the limited memory method usually requires more function evaluations than the SR1 option of Lancelot. (The BFGS option of Lancelot is clearly inferior to the SR1 option). However in terms of computing time the limited memory method is often the most efficient; this can be seen by examining the problems with at least 50 or 100 variables (which are the only ones for which times are meaningful). To our surprise we observe in Table 3 that even when using exact Hessians, LANCELOT requires more time than the limited memory method on many of the large problems. This is in spite of the fact that the objective function in all these problems has a significant degree of the kind of partial separability that LANCELOT is designed to exploit. On the other hand, LANCELOT using exact Hessians requires a much smaller number of function evaluations than the limited memory method.

Taking everything together, the new algorithm (L-BFGS-B) has most of the efficiency of the unconstrained limited memory algorithm (L-BFGS) [18] together with the capability of handling bounds, at the cost of a significantly more complex code. Like the unconstrained method, the bound limited memory algorithm's main advantages are its low computational cost per iteration, its modest storage requirements, and its ability to solve problems in which the Hessian matrices are large, unstructured, dense, and unavailable. It is less likely to be competitive, in terms of function evaluations, when an exact Hessian is available, or when significant advantage can be taken of structure. However this study appears to indicate that, in terms of computing time, the new limited memory code is very competitive with all versions of LANCELOT on many problems.

A code implementing the new algorithm is described in [23], and can be obtained by contacting the authors at nocedal@eecs.nwu.edu.

7. *

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, Mass: Addison-Wesley Pub. Co., 1974.

[2] B. M. Averick and J. J. Moré, "User guide for the MINPACK-2 test problem collection," Argonne National Laboratory, Mathematics and Computer Science Division Report ANL/MCS-TM-157, (Argonne, IL, 1991).

[3] D.P. Bertsekas, "Projected Newton methods for optimization problems with simple constraints", *SIAM J. Control and Optimization* 20 (1982): 221-246.

[4] I. Bongartz, A.R. Conn, N.I.M. Gould, Ph.L. Toint (1993). "CUTE: constrained and unconstrained testing environment", Research Report, IBM T.J. Watson Research Center, Yorktown, USA.

[5] J. V. Burke, and J. J. Moré, "On the identification of active constraints", *SIAM J. Numer. Anal.* Vol. 25, No 5 (1988): 1197-1211.

[6] R. H. Byrd, J. Nocedal and R. B. Schnabel, "Representation of quasi-Newton matrices and their use in limited memory methods", *Mathematical Programming* 63, 4, 1994, pp. 129-156

[7] P. H. Calamai, and J. J. Moré, "Projected gradient methods for linearly constrained problems" *Mathematical Programming* 39 (1987): 93-116

[8] A. R. Conn, N. I. M. Gould, and PH. L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables", *Mathematics of Computation.* Vol. 50, No 182 (1988): 399-430.

[9] A. R. Conn, N. I. M. Gould, and PH. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds", *SIAM J. Numer. Anal.*, vol. 25 (1988): 433-460.

[10] A.R. Conn, N.I.M. Gould, Ph.L. Toint (1992). "LANCELOT: a FORTRAN package for large-scale nonlinear optimization (Release A)", Number 17 in Springer Series in Computational Mathematics, Springer-Verlag, New York.

[11] A.R. Conn and J. Moré, Private communication.

[12] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, N.J.: Prentice-Hall, 1983.

[13] *Harwell Subroutine Library, Release 10* (1990). Advanced Computing Department, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, United Kingdom.

[14] J.C. Gilbert and C. Lemaréchal, "Some numerical experiments with variable storage quasi-Newton algorithms," *Mathematical Programming* 45 (1989) 407–436.

[15] P. E. Gill, W Murray and M. H. Wright, *Practical Optimization*, London: Academic Press, 1981.

[16] A. A. Goldstein, "Convex programming in Hilbert space", *Bull. Amer. Math. Soc.* 70 (1964): 709-710.

[17] E. S. Levitin and B. T. Polyak, "Constrained minimization problems", *USSR Comput. Math. and Math. Phys.* 6 (1966): 1-50.

[18] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization methods", *Mathematical Programming* 45 (1989): 503-528.

[19] J. J. Moré and D.J. Thuente (1990), "On line search algorithms with guaranteed sufficient decrease", Mathematics and Computer Science Division Preprint MCS-P153-0590, Argonne National Laboratory (Argonne, IL).

[20] J. J. Moré and G. Toraldo, "Algorithms for bound constrained quadratic programming problems", *Numer. Math.* 55 (1989): 377-400.

[21] J. Nocedal, "Updating quasi-Newton matrices with limited storage", *Mathematics of Computation* 35 (1980): 773-782.

[22] J.M. Ortega and W.C. Rheinboldt, "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press, (1970)

[23] C. Zhu, R.H. Byrd, P. Lu and J. Nocedal, "LBFGS-B: Fortran subroutines for large-scale bound constrained optimization", Report NAM-11, EECS Department, Northwestern University, 1994.