# A LIMITED-MEMORY ALGORITHM FOR BOUND-CONSTRAINED OPTIMIZATION

by

*Richard H. Byrd,*[*] *Peihuang Lu,*[†] *and Jorge Nocedal*[†‡]

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. We show 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.

## 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 $f : \Re^n \rightarrow \Re$ is a nonlinear function whose gradient $g$ is available, the vectors $l$ and $u$ represent lower and upper bounds on the variables, respectively, 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 $n$.

[*]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.

[†]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.

[‡]This author was also supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, while at the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439.

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 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], [18], [3] to determine the active set, because recent studies [7], [5] indicate that it possesses 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 piecewise 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, piecewise 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. 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 backtracking line search along $d_k = \bar{x}_{k+1} - x_k$ that ensures that

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

where $\lambda_k$ is the steplength and $\alpha$ is a parameter that has the value $10^{-4}$ in our code. 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.

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 computational demands similar to those of the limited-memory algorithm (L-BFGS) for unconstrained problems described by Liu and Nocedal [19] 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 $\theta$ 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 $\theta I$ $m$-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

$$\bar{W}_k \equiv \begin{bmatrix} \frac{1}{\theta} Y_k & S_k \end{bmatrix},$$

$$\bar{M}_k \equiv \begin{bmatrix} 0 & -R_k^{-1} \\ -R_k^{-T} & R_k^{-T}(D_k + \frac{1}{\theta} Y_k^T Y_k) R_k^{-1} \end{bmatrix},$$

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].)

Since the algorithm performs a backtracking line search, 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 piecewise linear path obtained by projecting points along the steepest descent direction, $x_k - tg_k$, 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 piecewise 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 $\{t^j : t^j \leq t^{j+1}, j = 1, ..., n\}$. We then search along $P(x^0 - tg, l, u)$, a piecewise linear path that 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 $\Delta t$,

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

where the parameters of this one-dimensional quadratic are

$$\begin{aligned} f_{j-1} &= f + g^T z^{j-1} + \tfrac{1}{2} z^{j-1^T} B z^{j-1}, \\ f'_{j-1} &= g^T d^{j-1} + d^{j-1^T} B z^{j-1}, && (4.4) \\ f''_{j-1} &= d^{j-1^T} B d^{j-1}. && (4.5) \end{aligned}$$

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 $f'_{j-1} > 0$, 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

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

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

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

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 computation of the generalized Cauchy point could require $O(n^2)$ operations, since in the worst case $n$ segments of the piecewise 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 $i = 1, \ldots, n$ compute

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

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

• Initialize
    $\mathcal{F} := \{i : t_i > 0\}$
    $p := W^T d$    ($2mn$ operations )
    $c := 0$
    $f' := g^T d = -d^T d$    ($n$ operations)
    $f'' := \theta d^T d - d^T WMW^T d = -\theta f' - p^T Mp$    ($O(m^2)$ operations)
    $\Delta t_{min} := -\frac{f'}{f''}$
    $t_{old} := 0$
    $t := \min\{t_i : i \in \mathcal{F}\}$    (using the heapsort algorithm)
    $b := i$ such that $t_i = t$    (remove $b$ from $\mathcal{F}$).
    $\Delta t := t - 0$

• Examination of subsequent segments
  While $\Delta t_{min} > \Delta t$ do

    $x_b^{cp} := \begin{cases} u_b & \text{if } d_b > 0 \\ l_b & \text{if } d_b < 0 \end{cases}$
    $z_b := x_b^{cp} - x_b$
    $c := c + \Delta t p$    ($O(m)$ operations)
    $f' := f' + \Delta t f'' + g_b^2 + \theta g_b z_b - g_b w_b^T Mc$    ($O(m^2)$ operations)
    $f'' := f'' - \theta g_b^2 - 2g_b w_b^T Mp - g_b^2 w_b^T Mw_b$    ($O(m^2)$ operations)
    $p := p + g_b w_b$    ($O(m)$ operations)
    $d_b := 0$
    $\Delta t_{min} := -\frac{f'}{f''}$
    $t_{old} := t$
```

```
        $t := \min\{t_i : i \in \mathcal{F}\}$    (using the heapsort algorithm)
        $b := i$ such that $t_i = t$    (Remove $b$ from $\mathcal{F}$)
        $\Delta t := t - t_{old}$
    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 is used, as will be discussed in the next section.

Our operation counts take into account only 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 preceding 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 $\gamma$ 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

$$\begin{aligned} \min \quad & \hat{m}_k(\hat{d}) \equiv \hat{d}^T \hat{r}^c + \tfrac{1}{2}\hat{d}^T \hat{B}_k \hat{d} + \gamma & (5.5) \\ \text{subject to} \quad & l_i - x_i^c \le \hat{d}_i \le u_i - x_i^c \qquad i \in \mathcal{F}, & (5.6) \end{aligned}$$

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 $\alpha^*$ 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 remains only 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, [17]), 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 $\theta, W$ 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 $N \equiv (I - \frac{1}{\theta} M W^T Z Z^T W)$
   - $N := \frac{1}{\theta} W^T Z Z^T W$ $\quad (2m^2 t + mt$ 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 $\alpha^*$ satisfying (5.8) ($t$ operations)

8. Compute $\bar{x}_i$ as in (5.9) ($t$ operations)

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

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

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

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

in Step 4, which requires $2m^2t$ 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 $Y^TZZ^TY$, $S^TZZ^TY$ and $S^TZZ^TS$. 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 addition, when $t$ is much larger then $n - t$, it seems more efficient to use the relationship $Y^TZZ^TY = Y^TY - Y^TAA^TY$, which follows from (5.1), to compute $Y^TZZ^TY$. Similar relationships can be used for the matrices $S^TZZ^TY$ and $S^TZZ^TS$. These devices can potentially result in significant savings, but they have not been implemented in the code experimented with in Section 6.

## 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)    ($(2m+1)t + O(m^2)$ operations)
2. $p := -\hat{r}$, $\hat{d} := 0$, and $\rho_2 := \hat{r}^T\hat{r}$    ($t$ operations)
3. Stop if $\|\hat{r}\| < \min(0.1, \sqrt{\|\hat{r}^c\|})\|\hat{r}^c\|$
4. $\alpha_1 := \max\{\alpha : \hat{l} \leq \hat{x}^c + \hat{d} + \alpha p \leq \hat{u}\}$    ($t$ operations)
5. $q := \hat{B}_k p$    ($4mt$ operations)
6. $\alpha_2 := \rho_2 / p^T q$    ($t$ operations)
7. If $\alpha_2 > \alpha_1$ set $\hat{d} := \hat{d} + \alpha_1 p$ and stop;
   otherwise compute:
      • $\hat{d} := \hat{d} + \alpha_2 p$    ($t$ operations)
      • $\hat{r} := \hat{r} + \alpha_2 q$    ($t$ operations)
      • $\rho_1 := \rho_2$; $\rho_2 = \hat{r}^T\hat{r}$; $\beta := \rho_2/\rho_1$    ($t$ operations)
      • $p := -\hat{r} + \beta p$    ($t$ 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. If we compare this with the cost of the primal direct method (5.12), for $t >> m$, the direct method seems 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 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

$$\begin{array}{rll} \min & g_k{}^T d + \frac{1}{2} d^T B_k d & (5.15) \\ \text{subject to} & A_k^T d = b_k & (5.16) \\ & l \leq x_k + d \leq u. & (5.17) \end{array}$$

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

$$g_k + B_k d^* + A_k \lambda^* = 0, \qquad (5.18)$$
$$A_k^T d^* = b_k. \qquad (5.19)$$

Premultiplying (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

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

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 $\lambda^*$, and hence $d^*$ is given by

$$B_k d^* = -A_k \lambda^* - g_k. \qquad (5.21)$$

(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 $\theta$ 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)

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

Backtrack if necessary:

- Compute $\alpha^* = \max\ \{\alpha : \ l_i \le x_c + \alpha(x_k + d^* - x^c) \le u_i, i \in \mathcal{F}\}$ $\quad$ ($t$ operations)
- Set $\bar{x} = x_c + \alpha^*(x_k + d^* - x^c)$. $\quad$ ($t$ 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. By comparison, if implemented as described above, the direct primal method requires $2m^2t + 6mt + 4t + O(m^3)$ operations for the subspace minimization. These figures indicate 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 Section 5.1. Similarly, in the dual case, the cost of computing $\bar{W}^T A_k A_k^T \bar{W}$ could be reduced by updating this matrix from one iteration to the next and, if $t_a > n - t_a$, by computing the matrix $\bar{W}^T Z_k Z_k^T \bar{W}$ first, and subtracting from $\bar{W}^T \bar{W}$. Such devices would require a more complex implementation, but might reduce the difference in cost between the primal and dual approaches. In fact, the primal and dual approaches have more in common than appears here, in 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).

## 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 SUBMIN of LANCELOT [10] using partitioned BFGS updating. 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.

**Bound L-BFGS Algorithm**
Choose a starting point $x_0$, and an 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 the direct primal method, the conjugate gradient method, or the dual method.
4. Using a backtracking line search, starting from the unit steplength, compute a steplength $\lambda_k$ such that $x_{k+1} = x_k + \lambda_k d_k$ satisfies (2.5) with $\alpha = 10^{-4}$.
5. Compute $\nabla f(x_{k+1})$.
6. If $y_k$ satisfies (3.9) with eps$= 10^{-8}$, 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.

Our code is written in double-precision Fortran 77. For the heapsort, during the generalized Cauchy point computation, we use the Harwell routine KB12AD written by Gould [13]. The backtracking line search was performed by the routine LNSRCH of Dennis and Schnabel [12]. For more details on how to update the limited-memory matrices in Step 7, see [6]. When testing the routine SUBMIN of LANCELOT [10] we used the default options and BFGS updating.

We selected seven problems, two bound-constrained quadratic optimization problems from the MINPACK-2 collection [2], and five nonlinear problems from the CUTE collection [4], to test the algorithms. To study a variety of cases, we tightened the bounds on several problems, resulting in more active bounds at the solutions of these problems. Table 1 lists the test problems and the bounds added to those already given in the specification of the problem. The number of variables is denoted by $n$, and the number of bounds active at the solution by $n_a$. We note that in those problems without active bounds at the solution ($n_a = 0$), some bounds may become active during the iteration.

Table 1: Test Problems

| Problem | Variant | n | $n_a$ | Reference | Additional Bounds |
|---|---|---|---|---|---|
| EDENSCH | 1 | 2000 | 0 | CUTE [4] | none |
| EDENSCH | 2 | 2000 | 1 | " | $[0, 1.5] \forall$ odd $i$ |
| EDENSCH | 3 | 2000 | 667 | " | $[-1, 0.5]$ $i = 4, 7, 10, \ldots$ |
| EDENSCH | 4 | 2000 | 999 | " | $[0, 0.99] \forall$ odd $i$ |
| EDENSCH | 5 | 2000 | 100 | " | $[0, 0.5] \forall$ odd $i$ |
| LMINSURF | 1 | 1024 | 124 | " | none |
| LMINSURF | 2 | 1024 | 147 | " | $[2, 10] \forall$ odd $i$ |
| LMINSURF | 3 | 1024 | 172 | " | $[5, 10] \forall$ odd $i$ |
| LMINSURF | 4 | 1024 | 227 | " | $[5.5, 6] \forall i$ |
| PENALTY 1 | 1 | 1000 | 0 | " | none |
| PENALTY 1 | 2 | 1000 | 0 | " | $[0, 1] \forall$ odd $i$ |
| PENALTY 1 | 3 | 1000 | 334 | " | $[0.1, 1]$ $i = 4, 7, 10, \ldots$ |
| PENALTY 1 | 4 | 1000 | 500 | " | $[0.1, 1] \forall$ odd $i$ |
| RAYBENDL | 1 | 44 | 4 | " | none |
| RAYBENDL | 2 | 44 | 6 | " | $[2, 95] \forall i$ |
| ORTHREG | 1 | 133 | 0 | " | none |
| TORSION | 1 | 1024 | 320 | MINPACK-2 [2] | none |
| JOURNAL | 1 | 1024 | 330 | MINPACK-2 [2] | none |

The results of our numerical tests are given in Table 2. All computations were performed on a Sun SPARCstation 2 with a 40-MHz CPU and 32-MB memory. In every run all methods converged to the same solution point; in fact, this was one requirement in the selection of the test problems. The number of corrections stored in the limited-memory method was $m = 4$. We record the number of iterations (iter), the number of inner conjugate gradient iterations (cg) for those methods that use them, and the total CPU time (time). A $*$ indicates that the convergence tolerance (6.1) was not met.

Table 2: Test Results of 3 Versions of the New Limited-Memory Method with $m = 4$ and LANCELOT's Subroutine SUBMIN using BFGS Updating

| Problem | Variant | Primal | Dual | CG | LANCELOT/BFGS |
|---|---|---|---|---|---|
| | | iter/time | iter/time | iter/cg/time | iter/cg/time |
| EDENSCH | 1 | 31/37.1 | 26/12.1 | 30/115/35.2 | 41/41/28.9 |
| EDENSCH | 2 | 17/15.4 | 17/10.9 | 20/64/20.7 | 62/105/241.8 |
| EDENSCH | 3 | 16/11.2 | 16/11.5 | 15/40/10.2 | 53/19/33.8* |
| EDENSCH | 4 | 15/9.21 | 15/12.4 | 16/42/10.2 | 32/16/16.9 |
| EDENSCH | 5 | 12/7.18 | 12/9.6 | 12/24/6.22 | 15/3/7.5 |
| LMINSURF | 1 | 166/76.4 | 166/62.3 | 168/1219/149 | 295/617/177.2 |
| LMINSURF | 2 | 420/213 | 403/141 | 430/2312/309 | 112/499/85.6 |
| LMINSURF | 3 | 474/239 | 462/165 | 542/2908/381 | 92/444/75.5 |
| LMINSURF | 4 | 107/52.8 | 107/45.4 | 126/577/75.2 | 43/297/34.9 |
| PENALTY 1 | 1 | 96/41.1 | 97/21.5 | 98/295/43.8 | 47/554/73.2 |
| PENALTY 1 | 2 | 66/30.4 | 61/15 | 59/158/25.8 | 55/513/97.1 |
| PENALTY 1 | 3 | 30/11.3 | 30/10.1 | 30/57/9.2 | 33/0/41.5 |
| PENALTY 1 | 4 | 30/10.9 | 30/12 | 30/58/8.9 | 32/0/30.3 |
| RAYBENDL | 1 | 1179/40.9 | 976/21.1 | 1733/13755/92.8 | 895/22645/65 |
| RAYBENDL | 2 | 1425/53.1 | 998/24.5 | 1737/13137/93.2 | 894/20468/57 |
| ORTHREG | 1 | 266/19.3 | 442/14.5 | 262/1504/27.8 | 114/164/8.4 |
| TORSION | 1 | 57/27.7 | 55/27.1 | 59/329/39.9 | 11/94/15.1 |
| JOURNAL | 1 | 132/75.5 | 120/54.9 | 155/660/91.7 | 8/116/16.54 |

The test results indicate that the dual method was the fastest of the three subspace minimization approaches in most cases. This is to be expected for this implementation, given that the number of active bounds at the solution was always less than $n/2$. The differences in the number of iterations required by the direct primal and dual methods are due to rounding errors. We note also that the direct primal method usually had a running time either close to or faster than that for the conjugate gradient method. In view of the discussion in Section 5.2 and the fact that there were usually more than $m/2 = 2$ conjugate gradient iterations per outer iteration on the average, this is not surprising.

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. Also, we used BFGS updating in LANCELOT (as opposed to SR1 updating) to minimize the differences with our limited-memory code. However, a few observations on the two methods can be made.

The limited-memory method tends to spend less time per iteration than LANCELOT, but in many cases requires more iterations. We also observed that LANCELOT is able to identify the correct active set sooner. This is likely to be because LANCELOT tends to form a more accurate Hessian approximation than the limited-memory matrix. It should be noted that the objective function in all these problems has a significant degree of the kind of partial separability that LANCELOT is designed to exploit. However, problem PENALTY 1 was coded so as to prevent LANCELOT from exploiting partial separability, which may explain its poor performance on this problem.

For comparison, we also tried the option in LANCELOT using exact Hessians. The results, shown in Table 3, indicate that in most cases the number of iterations and the computational time decreased significantly.

Table 3: Results of LANCELOT's subroutine SUBMIN using exact Hessian

| Problem | variant | LANCELOT/Exact Hessian |
|---|---|---|
| | | iter/cg/time |
| EDENSCH | 1 | 13/13/11.8 |
| EDENSCH | 2 | 11/364/109.4 |
| EDENSCH | 3 | 13/14/9.1 |
| EDENSCH | 4 | 12/18/817.2 |
| EDENSCH | 5 | 9/0/5.0 |
| LMINSURF | 1 | 272/531/138.2 |
| LMINSURF | 2 | 123/512/86.7 |
| LMINSURF | 3 | 99/660/98.1 |
| LMINSURF | 4 | 25/301/27.7 |
| PENALTY 1 | 1 | 62/776/99.1 |
| PENALTY 1 | 2 | 54/524/92.4 |
| PENALTY 1 | 3 | 33/0/40.9 |
| PENALTY 1 | 4 | 32/0/30.3 |
| RAYBENDL | 1 | 108/109/2.1 |
| RAYBENDL | 2 | 80/95/1.5 |
| ORTHREG | 1 | 27/67/2.2 |
| TORSION | 1 | 11/80/12 |
| JOURNAL | 1 | 7/177/14.8 |

Taking everything together, the new algorithm has most of the efficiency of the unconstrained limited-memory algorithm (L-BFGS) [19] 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 when an exact Hessian is available, or when significant advantage can be taken of sparsity. A well-documented and carefully coded implementation of the algorithm described in this paper will soon be available and can be obtained by contacting the authors at nocedal@eecs.nwu.edu.

## 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, Ill., 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, and Ph. L. Toint, "CUTE: Constrained and unconstrained testing environment," Research Report, IBM T.J. Watson Research Center, Yorktown, New York, 1993.

[5] J. V. Burke, and J. J. Moré, "On the identification of active constraints," *SIAM J. Numer. Anal.* 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," Technical report, EECS Department, Northwestern University, 1991, to appear in *Mathematical Programming.*

[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* 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.* 25 (1988): 433–460.

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

[11] A. R. Conn and J. Moré, Private communication, 1993.

[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*, Advanced Computing Department, AEA Industrial Technology, Harwell Laboratory, Oxfordshire, United Kingdom, 1990.

[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] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, New York, Academic Press, 1970.

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

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

[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.