A Limited Memory Variable Metric Method in Subspaces and Bound Constrained Optimization Problems*

Steven J. Benson and Jorge J. Moré

Abstract

We describe an algorithm for solving nonlinear optimization problems with lower and upper bounds that constrain the variables. The algorithm uses projected gradients to construct a limited memory BFGS matrix and determine a step direction. The algorithm has been implemented and distributed as part of the Toolkit for Advanced Optimization (TAO). We include numerical results demonstrate is effectiveness on a set of large test problems and its scalability to multiple processors.

1 Introduction

In this paper, we describe how variable metric methods can be applied in a subspace defined by a set of linear equalities. Then we descibe an algorithm that applies these methods to solve large nonlinear optimization problems with simple bounds on the variables. We write this problem as

$$\min\{f(x) : l \leq x \leq u\}, \tag{1.1}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a nonlinear function whose gradient ∇f is available, the vectors l and u are fixed, and the inequalities are taken componentwise. Since the algorithm does not require second derivatives, the method can be applied when the Hessian is not available or not practical to compute.

The method in this paper is similar to the ones in Byrd, Lu, Nocedal, and Zhu[7] in that we create a quadratic model function using gradient information in such a way that the storage required in linear in n. The main difference, however, is that while they use gradients to construct the model function and then minimize the model over a sequence of subspaces, we use projected gradients to generate the model function and compute a search direction in the full space of variables.

2 Bound Constrained Optimization Problem

A classical result shows that the bound-constrained optimization problem (1.1) has a unique solution on the feasible region

$$\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\} \tag{2.1}$$

when the function $f : \mathbb{R}^n \mapsto R$ is strictly convex. This result holds for unbounded Ω, and we thus allow the components of l and u to be infinite. Solutions to problem (1.1) satisfy

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38.

the Kuhn-Tucker conditions

$$\begin{array}{rcll} \partial_i f(x) & = & 0 & \text{if} \quad x_i \in (l_i, u_i) \\ \partial_i f(x) & \geq & 0 & \text{if} \quad x_i = l_i \\ \partial_i f(x) & \leq & 0 & \text{if} \quad x_i = u_i, \end{array}$$

where $\partial_i f(x)$ is the partial derivative of f with respect to the ith variable.

The projection operator

$$[T_\Omega d]_i = \begin{cases} d_i & \text{if} \quad x_i \in (l_i, u_i) \\ \min\{d_i, 0\} & \text{if} \quad x_i = l_i \\ \max\{d_i, 0\} & \text{if} \quad x_i = u_i \end{cases} \tag{2.2}$$

can be helpful in our analysis because x^* is a solution of (1.1) if and only if the projected gradient $T_\Omega \nabla f(x^*) = 0$. Given a tolerance τ, an approximate solutions to the bound constrained problem (1.1) is any vector $x \in \Omega$ such that

$$\|T_\Omega \nabla f(x)\| \leq \tau. \tag{2.3}$$

Note that (2.3) holds whenever x is sufficiently close to x^* and in the face of Ω that contains x^*.

The concept of a face is standard in convex analysis; for the convex set (2.1), the face of Ω that contains x is

$$\Big\{ y \in \Omega : y_i = x_i \text{ if } x_i \in \{l_i, u_i\} \Big\}.$$

Thus, the face of the feasible set that contains x can be described in terms of the set of active constraints

$$\mathcal{A}(x) = \{ i : x_i = l_i \text{ or } x_i = u_i \}.$$

Variables with indices in $\mathcal{A}(x)$ are the active variables, and those with indices outside $\mathcal{A}(x)$ are the free variables. Similarly, the binding variables are those with indices in

$$\mathcal{B}(x) = \{ i : x_i = l_i \text{ and } \partial_i f(x) \geq 0, \text{ or } x_i = u_i \text{ and } \partial_i f(x) \leq 0 \}.$$

The Kuhn-Tucker conditions show that $\mathcal{B}(x) = \mathcal{A}(x)$ at a solution.

3 Reduced Space and Subspace Variable Metric Methods

A standard transformation of variables can reduce the problem

$$\min\{ f(x) : Ax = b \}, \tag{3.1}$$

where $A \in \mathbb{R}^{(n-p)\times n}$, $b \in \mathbb{R}^{n-p}$ into an unconstrained optimization problem in p variables. Given a point x_0 such that $Ax_0 = b$ and a matrix $Z \in \mathbb{R}^{n \times p}$ whose columns are orthonormal ($Z^T Z = I$) and span the nullspace of A ($AZ = 0$), all points x satisfying $Ax = b$ can be written as $x = x_0 + Z\overline{x}$ for some $\overline{x} \in \mathbb{R}^p$. The vector $\overline{x}$ lies in a *reduced space* of $\mathbb{R}^n$ and the vector $Z\overline{x}$ lies in a *subspace* of $\mathbb{R}^n$. An objective function $\overline{f}:\mathbb{R}^p \mapsto \mathbb{R}$ in the reduced space can be the written as

$$\overline{f}(\overline{x}) = f(\hat{x} + Z\overline{x})$$

The gradient vector and Hessian matrix of this function can be written as

$$\begin{aligned} \nabla\overline{f}(\overline{x}) &= Z^T\nabla f(\hat{x}+Z\overline{x}) \\ \nabla^2\overline{f}(\overline{x}) &= Z^T\nabla f(\hat{x}+Z\overline{x})Z \end{aligned}$$

There are many algorithms for unconstrained optimization that have proven effective in theory and practice, and the BFGS algorithm, in particular, has been a popular choice. The BFGS method approximates the Hessian with the correction pairs $\{s_i, y_i\}$ such that

$$s_i = x_{i+1} - x_i \qquad \text{and} \qquad y_i = \nabla f(x_{i+1}) - \nabla f(x_i). \tag{3.2}$$

Given an initial matrix H_k that approximates the inverse Hessian matrix, a new approximation H_{k+1} is defined by the recursive formula

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \tag{3.3}$$

where

$$\rho_k = \frac{1}{\langle y_k, s_k\rangle} \qquad \text{and} \qquad V_k = I - \rho_k y_k s_k^T. \tag{3.4}$$

This matrix can easily be seen to be positive definite if H_k is positive definite and $\langle y_k, s_k\rangle > 0$. The matrix H_{k+1} minimizes $\|H - H_k\|_W$ subject to constaints that $H = H^T$ and $Hy_k = s_k$. The matrix norm $\|A\|_W = \|W^{1/2}AW^{1/2}\|_F$ where W is any matrix satisfying the secant equation $Wy_k = s_k$ and $\|A\|_F$ is the Frobenius norm. We will refer the matrix H_k defined by (3.2), (3.3) and (3.4) as the **full BFGS matrix**.

With the updated matrix H_k, the BFGS method defines $x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)$ for some positive step length α_k. Any line search that enforces the Wolfe conditions will generate a step length that produces a new point and gradient such that $\rho_{k+1} > 0$. A more complete discussion of the method and its properties can be found in [10].

In a affine subspace $Ax = b$, the BFGS step direction can be generalized.

Definition 3.1 *Let $Z \in \mathbb{R}^{n\times p}$ be a matrix with orthonormal columns. Given a symmetric positive definite matrix $\overline{H}_k \in \mathbb{R}^{p\times p}$ and a correction pair $\{\overline{s}_k, \overline{y}_k\}$ such that $\overline{s}_k = Z^T x_{k+1} - Z^T x_k$, $\overline{y}_k = Z^T\nabla f(x_{k+1}) - Z^T\nabla f(x_k)$, and $\overline{\rho}_i = 1/\langle\overline{y}_i, \overline{s}_i\rangle > 0$, define the* **reduced BFGS matrix** *with respect to Z, $\overline{H}_{k+1}$, to be*

$$\overline{H}_{k+1} = \overline{V}_k^T \overline{H}_k \overline{V}_k + \overline{\rho}_k \overline{s}_k \overline{s}_k^T \tag{3.5}$$

where

$$\overline{\rho}_k = \frac{1}{\langle\overline{y}_k, \overline{s}_k\rangle} \qquad \textit{and} \qquad \overline{V}_k = \overline{I} - \overline{\rho}_k \overline{y}_k \overline{s}_k^T. \tag{3.6}$$

The matrix $\overline{H}_{k+1}$ minimizes $\|\overline{H} - \overline{H}_k\|_W$ subject to the constraints that $\overline{H}_{k+1}$ is symmetric and $\overline{s}_k = \overline{H}\overline{y}_k$. The matrix norm used here is analagous the one used is the unconstrained BFGS method except that that matrices must satisfy the reduced secant equation $\overline{s}_k = W\overline{y}_k$. When there are no linear constraints, $p = n$, $Z = I$, and $\overline{H}_{k+1} = H_{k+1}$.

Example 1 *Consider the following bound constrained problem*

$$\min\{f(\chi_1, \chi_2, \chi_3) = \chi_1^2 - \chi_1\chi_2 + 0.75\chi_2^2 + 0.5\chi_3^2 + 4\chi_1 - 4\chi_2 - 2\chi_3 : \chi_3 \geq 0\}, \tag{3.7}$$

which has a unique minimum at $(0, 8/3, 2)$. The optimal face is

$$\{(\chi_1, \chi_2, \chi_3) : \chi_1 = 0;\},$$

so fix $\chi_1 = 0$ and consider the unconstrained problem in variables χ_2 and χ_3. Here $Z \in \mathbb{R}^{3\times 2}$ and its columns are the second and third columns of the identity matrix.

The gradient of f is defined by

$$\nabla f = [2\chi_1 - \chi_2 + 4, -\chi_1 + 1.5\chi_2 - 4, \chi_3 - 2]^T$$

and

$$\nabla f(0,0,0) = [4, -4, -2]^T, \qquad \nabla f(0,2,1) = [2, -1, -1]^T$$

Start the algorithm at the origin, step in the direction of the gradient to where $\overline{x}_1 = [2, 1]^T$. This step sets $\overline{y}_0 = [3, 1]^T$ and $\overline{s}_0 = [2, 1]^T$. If $\overline{H}_0 = \overline{I}$, then

$$\overline{H}_1 = \begin{bmatrix} 33/49 & -1/49 \\ -1/49 & 52/49 \end{bmatrix},$$

the secant equation holds, and

$$\overline{H}_1 Z^T \nabla f(0,2,1) = \overline{H}_1[-1, -1]^T = [-32/49, -51/49]^T.$$

The reduced BFGS matrix is not, however, the same as the matrix $Z^T H_k Z$. More importantly, the step direction generated by the reduced BFGS matrix is not the same as the step directions $\left[Z^T H_k Z\right]\left[Z^T \nabla f(x_k)\right]$ or $\left[Z^T H_k \nabla f(x_k)\right]$

Example 2 *Consider the bound constrained problem in Example 1. Let $H_0 = I$ and H_1 be the full BFGS matrix. Start at the origin at step in the direction of the projected gradient to $x_1 = [0, 2, 1]^T$. The correction pairs*

$$y_0 = [-2, 3, 1]^T; \qquad s_0 = [0, 2, 1]^T$$

produce

$$H_1 = \begin{bmatrix} 1 & 4/7 & 2/7 \\ 4/7 & 1 & 1/7 \\ 2/7 & 1/7 & 8/7 \end{bmatrix},$$

The matrix $Z^T H_1 Z$ clearly does not equal $\overline{H}_1$. The step directions $Z^T H_1 \nabla f(x_1)$ and $Z^T H_1 Z Z^T \nabla f(x_1)$ in the reduced space do not equal the step direction generated from the reduced BFGS matrix. That is,

$$Z^T H_1 \nabla f(0,2,1) = Z^T[8/7, 0, -5/7]^T = [0, -5/7]^T \neq \overline{H}_1 Z^T \nabla f(0,2,1)$$

and

$$Z^T H_1 Z Z^T \nabla f(0,2,1) = [-8/7, -9/7]^T \neq \overline{H}_1 Z^T \nabla f(0,2,1).$$

Furthermore, the reduced secant equation does not hold,

$$Z^T H_1 Z \overline{y}_0 = [22/7, 11/7]^T \neq \overline{s}_0.$$

Any step direction, $\overline{\Delta x}$, in the reduced space can be mapped into a step direction in the full space, Δx, by $\Delta x = Z\overline{\Delta x}$. In Example 1, the full step direction $Z\overline{H}_1 Z^T \nabla f(x_1) = [0, -32/49, -51/49]^T$. This direction is different from the full space directions generated in Example 2.

In some instances, it may be inconvenient to work explicitly in the reduced space. In optimization problems with linear inequalities, for instance, the subspace may change from one iteration to the next. The cost of creating reduced space data structures, mapping data into them, and computing the reduced BFGS matrix at each iteration could be prohibitably expensive. In these contexts, it is convenient and more efficient to update the BFGS matrix in the full space and then reduce these matrices in the appropriate space. Even better, we would like to develop methods that build a matrix in the full space and generate directions in a subspace that coincide with the directions generated by the reduced space BFGS method.

Specifically, these methods should generated matrices H_k such that

$$Z^T H_k d = \overline{H}_k Z^T d, \forall d \in \mathbb{R}^n. \tag{3.8}$$

If (3.8) is true, the step direction, $Z\overline{H}_k Z^T \nabla f(x_k)$, generated in the reduced space and mapped to the full space equals the direction, $ZZ^T H_k \nabla f(x_k)$, generated in the full space and projected into the subspace.

Example 3 *Multiples of the identity matrices cI and $c\overline{I}$ satisfy (3.8). for all real c.*

Example 4 *Some implementations of the BFGS algorithm for unconstrained optimization select H_0 after the first iterate using a correction pair $\{s, y\}$. In particular, suppose $\overline{H}_0 = \langle \overline{s}, \overline{y}\rangle / \langle \overline{y}, \overline{y}\rangle \overline{I}$ and $H_0 = \langle s, y\rangle / \langle y, y\rangle I$. $\overline{H}_0$ and H_0 satisfy (3.8) if and only if $y = ZZ^T y$.*

Theorem 3.2 *Assume a matrix Z such that $Z^T Z = I$ and matrices $H_k \in \mathbb{R}^{n\times n}$ and $\overline{H}_k^{p\times p}$ that satisfy*

$$Z^T H_k d = \overline{H}_k Z^T d, \qquad \forall d \in \mathbb{R}^n.$$

Furthermore, assume correction pairs $s_k, y_k \in \mathbb{R}^n$ such that $\langle s_k, y_k\rangle > 0$ and $\langle Z^T s_k, Z^T y_k\rangle > 0$, define $\overline{s}_k = Z^T s_k$, $\overline{y}_k = Z^T y_k$. If H_{k+1} is the matrix defined by (3.3) and (3.4), and $\overline{H}_{k+1}$ is the reduced BFGS matrix defined by (3.5) and (3.6), then

$$Z^T H_{k+1} d = \overline{H}_{k+1} Z^T d, \qquad \forall d \in \mathbb{R}^n.$$

if and only if $s_k = Z\overline{s}_k$ and $y_k = Z\overline{y}_k$.

Proof. First assume that $s_k = Z\overline{s}_k$ and $y_k = Z\overline{y}_k$. This implies that $\overline{\rho}_k = \rho_k$ and

$$\begin{aligned}
Z^T H_{k+1} d &= Z^T (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) d + \rho_k Z^T s_k s_k^T d \\
&= Z^T (I - \overline{\rho}_k Z\overline{s}_k \overline{y}_k^T Z^T) H_k (I - \overline{\rho}_k Z\overline{y}_k \overline{s}_k^T Z^T) d + +\overline{\rho}_k Z^T Z\overline{s}_k \overline{s}_k Z^T d \\
&= (I - \overline{\rho}_k Z^T Z\overline{s}_k \overline{y}_k^T) Z^T H_k (I - \overline{\rho}_k Z\overline{y}_k \overline{s}_k^T Z^T) d + \overline{\rho}_k \overline{s}_k \overline{s}_k^T Z^T d \\
&= (I - \overline{\rho}_k Z^T Z\overline{s}_k \overline{y}_k^T) \overline{H}_k Z^T (I - \overline{\rho}_k Z\overline{y}_k \overline{s}_k^T Z^T) d + \overline{\rho}_k \overline{s}_k \overline{s}_k^T Z^T d \\
&= (I - \overline{\rho}_k Z^T Z\overline{s}_k \overline{y}_k^T) \overline{H}_k (I - \overline{\rho}_k Z^T Z\overline{y}_k \overline{s}_k^T) Z^T d + \overline{\rho}_k \overline{s}_k \overline{s}_k^T Z^T d \\
&= (I - \overline{\rho}_k \overline{s}_k \overline{y}_k^T) \overline{H}_k (I - \overline{\rho}_k \overline{y}_k \overline{s}_k^T) Z^T d + \overline{\rho}_k \overline{s}_k \overline{s}_k^T Z^T d \\
&= \overline{H}_{k+1} Z^T d
\end{aligned}$$

Conversely, assume that

$$Z^T H_{k+1} d = \overline{H}_{k+1} Z^T d, \forall d \in \mathbb{R}^n.$$

Then

$$\begin{aligned}
Z^T H_{k+1}(s - Z\overline{s}) &= \overline{H}_{k+1} Z^T (s - Z\overline{s}) \\
&= \overline{H}_{k+1} (Z^T s - Z^T Z\overline{s}) \\
&= \overline{H}_{k+1} (\overline{s} - \overline{s}) \\
&= 0.
\end{aligned} \tag{3.9}$$

By similar arguments,

$$Z^T H_k (s_k - Z\overline{s}_k) = 0, \tag{3.10}$$

$$Z^T H_{k+1} (y_k - Z\overline{y}_k) = 0, \tag{3.11}$$

and

$$Z^T H_k (y_k - Z\overline{y}_k) = 0. \tag{3.12}$$

For any vector $e \in \mathbb{R}^n$,

$$\begin{aligned}
Z^T H_{k+1} e &= Z^T (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) e + \rho_k Z^T s_k s_k^T e \\
&= Z^T H_k e - \rho_k Z^T s_k y_k^T H_k e - Z^T H_k \rho_k y_k s_k^T e + \rho_k^2 Z^T s_k y_k^T H_k y_k s_k^T e + \rho_k Z^T s_k s_k^T e \\
&= Z^T H_k e - \rho_k \overline{s}_k y_k^T H_k e - \rho_k \overline{H}_k Z^T y_k s_k^T e + \rho_k^2 \overline{s}_k y_k^T H_k y_k s_k^T e + \rho_k \overline{s}_k s_k^T e \\
&= Z^T H_k e + (-\rho_k y_k^T H_k e + \rho_k^2 y_k^T H_k y_k s_k^T e + \rho_k s_k^T e)\overline{s}_k - (\rho_k s_k^T e)\overline{H}_k \overline{y}_k
\end{aligned} \tag{3.13}$$

If $e = s - Z\overline{s}$, then by (3.9),(3.10), and (3.13),

$$(-\rho_k y_k^T H_k e + \rho_k^2 y_k^T H_k y_k s_k^T e + \rho_k s_k^T e)\overline{s}_k - (\rho_k s_k^T e)\overline{H}_k \overline{y}_k = 0.$$

Since $\overline{s}_k$ and $\overline{H}_k\overline{y}_k$ may be linearly independent and $\rho_k > 0$, the coefficients $-y_k^T H_k e + \rho_k y_k^T H_k y_k s_k^T e + s_k^T e = 0$ and $s_k^T e = 0$. The equations

$$s_k^T e = s_k^T (s_k - Z\overline{s}_k) = s_k^T s_k - s^T Z\overline{s}_k = s_k^T s_k - \overline{s}_k^T \overline{s}$$

and

$$\|e\|^2 = s_k^T s_k - \overline{s}_k^T \overline{s}_k - \overline{s}_k^T \overline{s}_k + \overline{s}_k^T \overline{s}_k = s_k^T s_k - \overline{s}_k^T \overline{s}_k,$$

imply that $s_k = Z\overline{s}_k$.

To prove that $y_k = Z\overline{y}_k$, let $e = y_k - Z\overline{y}_k$. By (3.11), (3.12), (3.13), and the same argument as above, $s_k^T e = 0$ and $y_k^T H_k e = 0$.

Since

$$\begin{aligned} y_k^T H_k e &= y_k^T H_k (y_k - Z\overline{y}) \\ &= (y_k - Z\overline{y}_k)^T H_k (y_k - Z\overline{y}_k) + (Z\overline{y}_k)^T H_k (y_k - Z\overline{y}_k) \\ &= (y_k - Z\overline{y}_k)^T H_k (y_k - Z\overline{y}_k) + \overline{y}_k^T Z^T H_k (y_k - Z\overline{y}_k) \\ &= (y_k - Z\overline{y}_k)^T H_k (y_k - Z\overline{y}_k) \end{aligned}$$

and H_k is positive definite, the equation $y_k^T H_k e = 0$ implies that $y_k - Z\overline{y}_k$. ∎

Corollary 3.3 *Given the assumptions of Theorem 3.2, define* $\overline{s}_k = Z^T (x_{k+1} - x_k)$ *and* $\overline{y}_k = Z^T \nabla f(x_{k+1}) - Z^T \nabla f(x_k)$. *If* H_k *and* $\overline{H}_k$ *are also defined in Theorem 3.2, then* $Z^T H_{k+1} d = \overline{H}_{k+1} Z^T d$ *if and only if* $y_k = ZZ^T \nabla f(x_{k+1}) - ZZ^T \nabla f(x_k)$ *and* $s_k = ZZ^T (x_{k+1} - x_k)$.

This corollary states that $x_{k+1} - x_k$ must lie in the subspace and that y_k is the difference of gradients that are projected into the subspace.

Corollary 3.4 *If* $y_k = ZZ^T \nabla f(x_{k+1}) - ZZ^T \nabla f(x_k)$, $s_k = ZZ^T (x_{k+1} - x_k)$, *and* $Z^T H_k d = \overline{H}_k Z^T d$ *for all* $d \in \mathbb{R}^n$, *then the step direction* $Z^T H_{k+1} \nabla f(x_{k+1})$ *generated in the full space equals the direction* $\overline{H}_{k+1} Z^T \nabla f(x_{k+1})$ *generated in the reduced space, and* $ZZ^T H_{k+1} \nabla f(x_{k+1}) = Z\overline{H}_{k+1} Z^T \nabla f(x_{k+1})$.

Other reduced space methods use the reduced Hessian matrix $Z^T \nabla^2 f(x) Z$ and its inverse. It is very tempting generate a step direction in the reduced space using the reduced matrix $Z^T H Z$. However, the matrix H approximates the inverse Hessian matrix, not the Hessian matrix, and the property

$$\left[Z^T B Z\right]^{-1} = Z^T B^{-1} Z$$

is generally not true for B equal to an exact Hessian or approximations of it. Therefore the matrix this matrix $Z^T H Z$ is not same as the matrix generated using this matrix may not generate the reduced BFGS step direction. If the matrix B and its inverse H are constructed according to Theorem 3.2, then this equality holds, the step generated in the reduced space will be the true BFGS step direction, and the full matrix H can be used to generate BFGS step directions.

Example 5 *Consider the bound constrained problem in Example 1. Start at the origin at step in the direction of the projected gradient to* $x_1 = [0, 2, 1]$*. Define* $s_i = x_{i+1} - x_i$ *and* $y_i = T_\Omega \nabla f(x_{i+1}) - T_\Omega \nabla f(x_i)$ *so that*

$$y_0 = [0, 3, 1]; \qquad\qquad s_0 = [0, 2, 1].$$

Then

$$H_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 33/49 & -1/49 \\ 0 & -1/49 & 52/49 \end{bmatrix},$$

In the reduced space, the step direction $Z^T H_1 \nabla f(0, 2, 1) = Z[-32/49, -51/49]^T$ *equals the step direction produced by the reduced BFGS matrix, and in the full space* $ZZ^T H_1 \nabla f(x) = Z\overline{H}_1 Z^T \nabla f(x)$*. The reduced secant equation* $Z^T H_1 Z\overline{y} = \overline{s}^T$ *also holds.*

In this example, the conditions $y = Z\overline{y}$, and $s = Z\overline{s}$ are true in addition to the standard conditions that $\overline{y} = Z^T y$ and $\overline{s} = Z^T s$,

4 BLMVM

Like the unconstrained BFGS method, BLMVM creates a convex quadratic model function

$$m_k(d) = f(x_k) + \nabla f(x_k)^T d + \tfrac{1}{2} d^T B_k d. \tag{4.1}$$

The matrix B_k will be updated at each iteration using correction pairs s_k and y_k. Letting $H_k = B_k^{-1}$, the minimizer d_k of this of this quadratic can be written explicitly as $d_k = -H_k \nabla f(x_k)$ and used as a step direction in the algorithm

Unlike the unconstrained BFGS method, however, BLMVM defines the correction pairs

$$s_k = x_{k+1} - x_k, \qquad\qquad y_k = T_\Omega \nabla f(x_{k+1}) - T_\Omega \nabla f(x_k) \tag{4.2}$$

to update H_k. This choice of correction pairs is motivated two observations. The first observation is that just as the norm of the gradient will converge to zero as x_k converges to an unconstrained minimum, the norm of the projected gradient should converge to zero as x_k converges to a bound constrained minimum. The second observation pertains the theory provided in the previous section. This choice of y_i is like updating the BFGS matrix in the space of nonbinding variables. For points sufficiently near the solution, the set of binding variables correctly identifies the face with the optimimal solution.

Since the inverse Hessian approximation is usually dense, the cost of storing H_k and manipulating it is prohibitive for large problems. To reduce these costs, the L-BFGS method stores a limited number of correction pairs $\{s_i, y_i\}$, $i = k-m, \ldots, k-1$. Once the number of correction vectors exceeds the number of vectors available to store them, a new correction pair is added by removing the oldest one. Given an initial inverse Hessian approximation

H_k^0, the L-BFGS algorithm defines

$$
\begin{aligned}
H_k = \;& \left(V_{k-1}^T \dots V_{k-m}^T\right) H_k^0 \left(V_{k-m} \dots V_{k-1}\right) + \\
& \rho_{k-m} \left(V_{k-1}^T \dots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T \left(V_{k-m+1} \dots V_{k-1}\right) + \\
& \dots + \\
& \rho_{k-2} \left(V_{k-1}^T\right) s_{k-2} s_{k-2}^T \left(V_{k-1}\right) + \\
& \rho_{k-1} s_{k-1} s_{k-1}^T .
\end{aligned}
\tag{4.3}
$$

This representation of the matrix provides an efficient solution to the quadratic problem (4.1). Given any m correction pairs $\{s_i, y_i\}$ such that $\{s_{k-m}, y_{k-m}\}$ is the least recent pair and $\{s_{k-1}, y_{k-1}\}$ is the most recent pair, the operation action $\hat{d} := H_k v$ can be computed using the pseudo code[10]:

L-BFGS two-loop recursion:

$\hat{d} \leftarrow v$

for $i = k-1, \dots, k-m$

- $\beta_i \leftarrow \rho_i \langle \hat{d}, s_i \rangle$;
- $\hat{d} \leftarrow \hat{d} - \beta_i y_i$;

$\hat{d} \leftarrow H_k^0 \hat{d}$;

for $i = k-m, \dots, k-1$

- $\hat{d} \leftarrow \hat{d} + \left(\beta_i - \rho_i \langle \hat{d}, y_i \rangle\right) s_i$

to generate a step direction

$$\hat{d} = -H_k \nabla f(x_k).$$

The matrix H_k^0 can change from one iteration to the next; BLMVM used the limited memory BFGS matrix and selects $H_k^0 = \langle s_k, y_k \rangle / \langle y_k, y_k \rangle I$.

The computational cost of computing s_i, y_i, ρ_i, $\|y_i\|$, and this pseudo code is $(8m+9)n$ flops, which is dominated by the cost of evaluating $f(x_k)$ and $\nabla f(x_k)$ in many applications.

Our algorithm uses a projected line search to enforce the bounds of the variables. We select a step length α_k and define $x_{k+1} = P[x_k + \alpha_k d_k]$. The operator P is a projection onto Ω and can be computed in n operations by

$$P[x] = \operatorname{mid}(l, u, x),$$

where $\operatorname{mid}(l, u, x)$ is the vector whose ith component is the median of the set $\{l_i, u_i, x_i\}$.

Since there may not exist a step length that satisfies the Wolfe conditions, we apply a simple backtracking method to select a step size such that

$$f(x_{k+1}) \le f(x_k) + \mu \langle \nabla f(x_k), x_{k+1} - x_k \rangle \tag{4.4}$$

for some $\mu \in (0, 1/2)$. The step size is computed by setting α_k to the first member of the sequence $\gamma_0 (\frac{1}{2})^j$ for $j = 0, 1, \dots$ such that x_{j+1} satisfies the sufficient decrease condition

(4.4). The choice of γ_0 is the infimum of 1.0 and all $\gamma > 0$ such that $P[x_k+\gamma d_k] = P[x_k+\hat{\gamma} d_k]$ for all $\hat{\gamma} \geq \gamma$. This line search allows the set of active and binding variables to change by more than one variable at each iteration. This choice of line search also means that we that we do not have to explicitly project the direction $H\nabla f(x)$. More sophisticated projected searches are possible, but this simple search has proved to be sufficient in all cases tried.

Condition (4.4) is true for sufficiently small $alpha_k$ if $\langle -T_\Omega H_k \nabla f(x_k), \nabla f(x)\rangle \leq 0$. When there are finite bounds on the variables, the set of binding variables may change significantly in the early iterates. Although $-H_k\nabla f(x_k)$ will always be a descent direction since H_k is positive definite, the direction $-T_\Omega H_k \nabla f(x_k)$ may not be a descent direction. In the event that $\langle T_\Omega H_k \nabla f(x_k), \nabla f(x)\rangle \leq 0$, we let $d = -T_\Omega \nabla f(x_k)$. In practice, we have found that projected gradients rarely need to be used as a step direction.

The algorithm can be summarized as follows:

Algorithm BLMVM: Choose $x_0 \in \Omega$. Let $d = -T_\Omega \nabla f(x_0)$. For $k = 0, \ldots,$

1. Compute α_k using a projected line search and let $x_{k+1} = P[x_k + \alpha_k d_k]$.
2. Compute $\nabla f(x_{k+1})$ and its projection $T_\Omega \nabla f(x_{k+1})$.
3. If $\|T_\Omega \nabla f(x_{k+1})\| < \epsilon$, stop.
4. Compute y_k and s_k according to (4.2). If $y_k^T s_k > 0$, update H_k.
5. Compute $H_{k+1}\nabla f(x_{k+1})$ using the L-BFGS two-loop recursion.
6. If $\langle -T_\Omega(H_{k+1}\nabla f(x_{k+1})), \nabla f(x_{k+1})\rangle > 0$, let $d = -H_{k+1}\nabla f(x_{k+1})$; otherwise let $d = -\nabla f(x_{k+1})$.

For points sufficiently close to the solution, not only does $\overline{y}_k = Z^T y_k$ and $\overline{s}_k = Z^T s_k$, but $y_k = Z\overline{y}_k$ and $s_k = Z\overline{s}_k$. If the m correction pairs stored by the algorithm possess this property, the algorithm will generate the same direction in the reduced space as applying the L-BFGS algorithm in the reduced space.

Theorem 4.1 *Let $Z \in \mathbb{R}^{n\times p}$ be a matrix whose columns are the columns of the identity matrix corresponing to the nonbinding variables. Furthermore, et $\overline{H}_k$ be the reduced L-BFGS matrix on the set of nonbinding variables at x_k and let H_k be the quasi Newton matrix generated by (4.2), (3.4), and (3.3). If*

(i) $x_{k-m}, \ldots, x_k$ lie on the same face, and

(ii) $\mathcal{B}(x_{k-m}) = \ldots = \mathcal{B}(x_k) = \mathcal{B}(x^*)$,

then

$$T_\Omega(H_k \nabla f(x_k)) = Z\overline{H}_k Z^T \nabla f(x_k). \tag{4.5}$$

Proof. The definition of T_Ω and Z is such that $T_\Omega \nabla f(x_k) = ZZ^T\nabla f(x_k)$. As in Example 4, The matrix H_k^0 satisfies (4.5). By the first condition, $s_k = ZZ^T(x_{k+1} - x_k)$ for $i = k-m, \ldots, k-1$. By the second condition, $y_k = ZZ^T\nabla f(x_{k+1}) - ZZ^T\nabla f(x_k)$ for $i = k-m, \ldots, k-1$. Therefore, by Theorem 3.2 and Corollary 3.4, the statment holds. ■

Note: For what problems will this condition hold close to the solution.?

This theorem obviously applies when the lower bounds equal $-\infty$ and the upper bounds equal ∞ because no variable is ever binding. The theorem also applies when $x_{k-m}, \ldots, x_k$ lie sufficiently close to the solution x^* and on the same face as the solution.

Even when the iterates are on a face other than the optimal one, the step directions are very good. The reason is that the gradient with respect to variables that are likely to equal one of the bounds will have little effect upon the step direction.

5 Numerical Experiments

We implemented the algorithm in C++ language and used the PETSc [2, 3, 4] package for linear algebra support. Using a set a benchmark problems we compared its performance to the limited memory variable metric method L-BFGS-B[7]. The calculations were performed using a Pentium II processor with 512 KB cache and a clock speed of 400MHz and the Linux operating system.

The L-BFGS-B solver also uses a limited memory BFGS matrix to compute step directions. However, it does not use projected gradients to update the matrix. Instead it updates the full matrix using gradients and works with reduced matrices Z^THZ and reduced gradients $Z^T\nabla f$ to compute a step direction. There process is similar, but not identical to the procedure used in Example 2. While BLMVM always works with the full space, L-BFGS-B explicitly minimizes the function over a sequence of subspaces. The L-BFGS-B method also uses the compact form of the L-BFGS matrix instead of the two-loop recursion.

Table 5.1 shows the performance of each solver on bound constrained optimization problems in the MINPACK-2[1] test suite. Table 5.2 shows the performance solving systems of nonlinear equations in the MINPACK-2 test suite. We minimized the square of the Euclidean norm of the residual. The variables in these problems are not constrained by bounds. Table 5.3 shows the peformance of the solvers for bound constrained problems in the CUTE test suite. On the optimization problems, both codes were terminated when

$$\|T_\Omega \nabla f(x_k)\| \leq \min\{10^{-5}, 10^{-10}|f(x_k)|\}.$$

For the nonlinear equations, the solvers were terminated when the objective function was less the 10^{-4}. These tables show the time and the number of function evaluations required to solve each problem. With each function evaluation, the solvers also received the gradient. We limited the solvers to $10,000$ function and gradient evaluations per problem. Problems requiring less than 0.03 seconds to solve were not included in the test suite.

We applied the performance profiling techniques introduced by Dolan and Moré[8] to interpret the results. Their profiles require a single statistic that measures the performance of a solver on a problem. We computed one profile using execution time and another using the number of function and gradient evaluations.

To create these performance profiles, we first identify the best solver for each problem in the test suite. Then we compute the ratio of each solver's performance versus the per-

Problem		n	BLMVM nfg	BLMVM s	L-BFGS-B nfg	L-BFGS-B s		n	BLMVM nfg	BLMVM s	L-BFGS-B nfg	L-BFGS-B s
DEPT_1		2500	236	1.60	235	3.02		10000	501	14.98	502	31.61
DEPT_2		2500	187	1.21	179	1.93		10000	324	9.29	420	22.49
DEPT_3		2500	92	0.56	93	0.78		10000	184	5.05	233	9.49
DPJB_1		2500	373	2.48	10000	39.34		10000	690	19.76	1924	107.68
DPJB_2		2500	505	3.14	520	5.21		10000	1085	29.70	1104	52.83
DPJB_3		2500	2342	14.01	2607	24.68		10000	5680	155.11	6743	307.47
DMSA_A		2500	307	2.99	313	5.05		10000	680	27.40	656	49.30
DMSA_B		2500	378	3.69	552	8.84		10000	770	31.38	1769	131.72
DMSA_C		2500	399	3.87	942	14.71		10000	854	34.60	2742	199.53
DMSO_A		2500	388	3.50	390	5.99		10000	808	31.70	749	55.25
DMSO_B		2500	256	2.28	262	4.02		10000	517	20.01	493	36.18
DMSO_C		2500	340	2.69	358	4.51		10000	777	30.31	1400	102.19
DODC_A		2500	277	2.92	260	4.38		10000	495	22.08	499	40.18
DODC_B		2500	218	2.33	227	3.92		10000	471	21.53	453	36.72
DODC_C		2500	253	2.54	224	3.72		10000	493	21.29	496	38.87
DSSC_A		2500	172	2.95	180	4.22		10000	345	24.48	415	43.69
DSSC_B		2500	122	2.04	139	2.97		10000	269	18.61	275	26.44
DSSC_C		2500	90	1.45	92	1.81		10000	196	13.15	157	13.45
DGL1_A		2500	10000	45.81	10000	111.05		10000	10000	212.65	10000	564.22
DGL1_B		2500	10000	45.99	10000	111.02		10000	10000	212.89	10000	565.45
DGL2_A		2500	1853	23.47	1643	31.52		10000	3172	168.77	3431	301.45
DGL2_B		2500	3147	39.39	2969	56.70		10000	6490	348.66	6109	542.75
DGL2_C		2500	5342	67.40	5561	106.08		10000	10000	537.10	10000	883.52

Table 5.1: Performance on MINPACK-2 Problems.

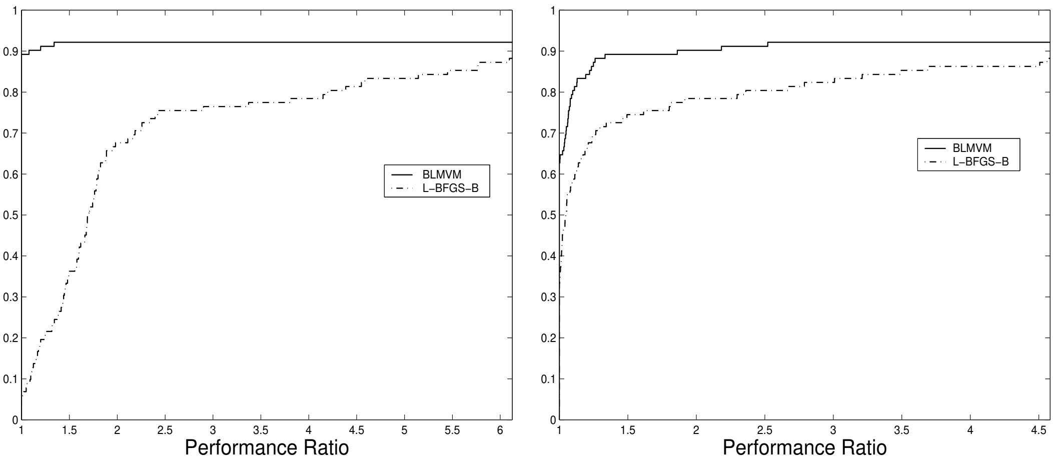
Problem	n	BLMVM nfg	BLMVM seconds	L-BFGS-B nfg	L-BFGS-B seconds
DSFD	140	251	1.24	234	1.43
DSFD	350	393	3.86	358	4.63
DSFD	700	592	10.71	645	15.86
DSFD	1400	1189	41.69	1206	57.24
DFIC	80	835	2.15	941	3.04
DFIC	200	1890	8.89	2289	14.23
DFIC	400	5444	45.33	4411	49.71
DFIC	800	9970	155.20	8168	174.55
DSFI	100	49	0.04	45	0.07
DSFI	625	121	0.57	128	1.24
DSFI	2500	437	8.45	416	16.52
DSFI	10000	1451	125.04	1527	281.68
DFDC	100	502	5.53	621	7.28
DFDC	625	586	44.83	618	49.73
DFDC	2500	587	195.95	595	206.08
DFDC	10000	355	502.35	353	526.91

Table 5.2: Performance Solving Nonlinear Equations.

		BLMVM		L-BFGS-B	
Problem	n	nfg	seconds	nfg	seconds
BDEXP	5000	20	0.42	20	0.68
BIGGSB1	1000	1574	2.77	2847	12.15
CHENHARK	1000	10000	19.04	9686	36.82
EXPLIN	1000	67	0.06	36	0.05
EXPLIN2	1000	29	0.03	23	0.03
EXPQUAD	1000	179	0.26	71	0.29
GRIDGENA	578	61	0.13	10000	24.57
HADAMALS	1024	32	0.63	24	0.47
HARKERP2	1000	75	6.43	10000	855.74
JNLBRNG1	15625	466	48.43	10000	929.87
JNLBRNG2	15625	714	74.78	752	108.28
JNLBRNGA	15625	435	40.24	403	53.13
JNLBRNGB	15625	2377	221.85	2748	355.67
LINVERSE	1999	824	10.76	742	12.56
MCCORMCK	10000	16	0.79	15	1.18
NCVXBQP1	10000	2	0.06	2	0.07
NCVXBQP2	10000	132	3.09	10000	174.06
NCVXBQP3	10000	124	3.41	10000	171.98
NOBNDTOR	14884	310	28.10	343	46.70
NONSCOMP	10000	51	1.67	55	3.65
OBSTCLAE	15625	233	21.84	701	100.17
OBSTCLAL	15625	168	15.76	197	24.88
OBSTCLBL	15625	143	13.35	382	55.39
OBSTCLBM	15625	166	15.66	185	26.43
OBSTCLBU	15625	152	14.23	227	32.12
QR3DLS	610	10000	75.73	10000	91.11
QRTQUAD	1000	286	0.42	131	0.50
SINEALI	1000	65	0.16	31	0.14
TORSION1	14884	222	20.10	298	37.93
TORSION2	14884	272	24.63	521	71.46
TORSION3	14884	123	10.98	125	13.80
TORSION4	14884	122	10.77	450	55.41
TORSION5	14884	63	5.65	61	6.15
TORSION6	14884	75	6.60	342	40.23
TORSIONA	14884	241	24.88	299	41.82
TORSIONB	14884	278	28.70	449	67.39
TORSIONC	14884	133	13.78	130	16.22
TORSIOND	14884	129	13.34	450	60.70
TORSIONE	14884	62	6.34	52	5.88
TORSIONF	14884	75	7.58	338	43.71

Table 5.3: Performance on CUTE Problems.

Figure 5.1: Comparson of BLMVM and L-BFGS-B.



(a) Time

(b) Function Evaluations

formance of the best solver. A ratio of 1.0 indicates that the solver was at least as good as the other solvers on that problem while a ratio greater than one indicates how much worse the solver performed compared to the best solver. When a solver failed to find a solution satisfying the specified criteria, we set the ratio to $+\infty$.

Given these ratios, we define a function for each solver. For every number greater than or equal to one, this function is defined to be the percentage of instances where the solver's performance ratio is less than this number. This cumulative distribution function can be plotted for each solver. The left side of the graph shows how often each solver was the best solver and the right side of the graph show how often each solver successfully found a solution.

Figures 5.1 compare the performance of BLMVM to L-BFGS-B using execution time and number of function evaluations. In both algorithms, the function and gradient vector were evaluated in the same routine. The right side of each graph indicates the robustness of each algorithm by the percentage of problems that each algorithm successfully solved. Both algorithms fared well by solving about the problems to the prescribed accuracy. BLMVM failed on 8 problems while L-BFGS-B failed on 12 problems.

The left side of the graphs show the percentage of instances in which each algorithm performed better than the other. In terms of function evaluations, BLMVM used fewer evaluations on about 60% of the problems while L-BFGS-B used fewer function evaluations on about 30% of the problems. In terms of execution time, BLMVM performed better on 90% of the test suite, while L-BFGS-B used less time on only a few problems.

In addition, we found that when function evaluations were used a measure, BLMVM performed L-BFGS-B by a factor of two on 15% of the problems and a factor of three on

10% of the problems. When execution time was used as the measure, BLMVM performed L-BFGS-B by a factor of two on 25% of the problems and a factor of three on 15% of the problems. On average, BLMVM was about 40% faster than the competition.

Since BLMVM uses fewer function evaluations on most problems, we feel that the step direction used in BLMVM is better than the one used by L-BFGS-B. In terms of execution time, BLMVM shows a distinct advantage. In all but a few cases, it found a solution in less time. These facts indicate that one iteration of the BLMVM algorithm is usually cheaper that one iteration of L-BFGS-B. Whereas L-BFGS-B first identifies a face in the feasible region and then minimizes over that face, BLMVM always computes a step direction in the full space. By not explicitly working in a subspace, BLMVM saves the time that other algorithms spend identifying it. These features, and the use of the two-loop recursion instead of the compact form, make the BLMVM algorithm efficient and relatively simple to implement.

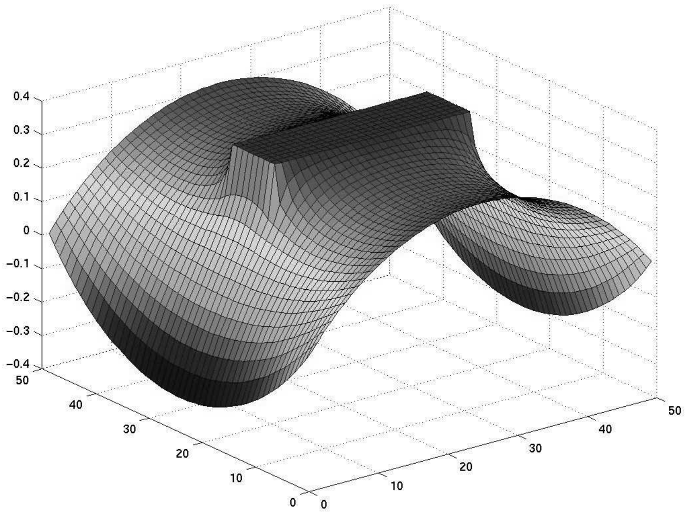
6 Scalability Using Multiple Processors

To demonstrate the parallel efficiency of the algorithm, we solved the obstacle problem from the MINPACK-2 test suite using 1 - 512 processors. This benchmark is a variational problem over a two dimensional region. It asks for the surface with the least area that satisfies a set of boundary conditions and lies above an obstacle within the region. Given a two dimensional region $\mathcal{D}$, an obstacle $v_L(x)$ defined on that region, and boundary conditions $v_D(x)$, the infinite dimensional version of this problem can be stated as

$$\min \left\{ f(v) : v \in H^1, \ v(x) = v_D(x) \ \ \forall x \in \partial\mathcal{D}, \ v(x) \geq v_L(x) \ \ \forall x \in \mathcal{D} \right\}$$

where

$$f(v) = \int_{\mathcal{D}} \sqrt{1 + \|\nabla v(x)\|^2} \, dx$$



We prepared this problem for parallel computing using the grid management facilities of PETSc, which relies on MPI [9] for all communication between processors. PETSc provides support for discretizing the rectangular region $\mathcal{D}$, partitioning the surface into multiple regions and assigning each processor to one these regions. Each processor computes the surface area of its region and the gradient of the objective value with respect to the variables in its region.

These computations were performed on the Cray T3E supercomputer at the National Energy Research Scientific Computing Center (NERSC). Each processor on this computer has 256 MB of RAM, and clock speed of 450 MHz, and a peak performance of 900 Mflops.

Our first tests discretized the region into a 1600×1600 mesh, which creates $2,560,000$ variables. The obstacle was a rectangular plate covered by the 640×960 mesh points in the middle of the mesh. The height of the plate was set to 0.2. The memory requirements of a problem this large are so high that it exhausted the memory of the computer when only four processors were used. We solved the problem using as few as 8 processors and as many as 512 processors. Table 6.1 shows the number of iterations, and seconds used to solve the problem in each test. The table also indicates the percentage of time spent in various parts of the algorithm. The time spent adding vectors is shown under the label `AXPY`. This category also includes time spent copying vectors, projecting vectors, and performing other vector operations that do not require communication between processors. The `Dot` column indicates the percentage of time spent performing vector inner products and vector norms. The column `FG` shows the percentage of time spent computing the function and gradient.

Since more than 60% of the execution time was spent evaluating the function and gradient, an efficient implementation of this routine is mandatory for good parallel performance. Our function has a structure that allows an efficient parallel evaluation, and its scalability is demonstrated by the fact that the percentage of time spent in this routine did not increase with the number of processors.

Processors Used	BLMVM Iterations	Execution Time	Percentage of Time AXPY	Dot	FG
8	996	1083.8	31	9	60
16	991	538.2	30	10	60
32	966	267.7	29	11	60
64	993	139.5	27	13	60
128	987	72.4	25	15	60
256	996	39.2	26	18	56
512	1000	21.6	23	22	53

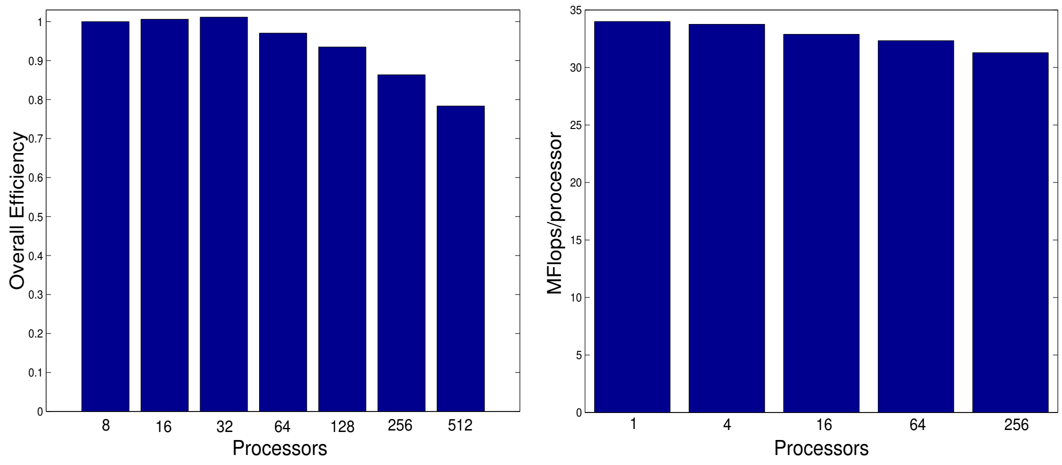
Table 6.1: Performance of BLMVM on Obstacle Problem.

The rest of the time was spent computing the BLMVM step direction and projecting the new step into the feasible region. Operations such as scaling vectors, adding vectors, copying the elements of one vector to another, and applying the projections P and G do not require any communication between processors. These operations scale very well to more processors, and the percentage of time spent in these operations declined as the number of processors increased. The bottleneck in numerical computations involve vector inner products and norms. These operations require communication between processors, whose overhead degrades performance. In this problem, the percentage of time spent in these operations more than doubled from 9% to 22%.

The overall efficiency of our implementation is shown in Figure 2(a). Each bar indicates the performance of BLMVM of each test relative to the performance of BLMVM on eight processors. This number is the ratio of eight times the execution time using eight processors and the number of processors multiplied by the time need to solve the problem using those processors. Relative to the performance of BLMVM on this problem using eight processors, the overall parallel efficiency using 256 processors was over 86% and the overall parallel efficiency using 256 processors was over 78%.

In a second set of tests we solved obstacle problem using a mesh that was refined according the number of processors used to solve the problem. In these tests each processor owned $10,000$ variables. The mesh was 100×100 when one processor was used and 1600×1600 when 256 processors were used. The length and width of the obstacle were adjusted proportionately. Since BLMVM uses more iterations to solve problems with finer meshes, we used the rate of floating point operations as the measure of efficiency. Figure 2(b) shows the average number of floating point operations performed per second on each processor. The problem used over 34 MFlops when one processor was used and that rate only decreased to about 31 MFlops when when 256 processors were used. Our parallel efficiency by this measure was over 90%.

Figure 6.1: Parallel Efficiency of BLMVM on the Obstacle Problem.



(a) Overall Implementation Efficiency

(b) Floating Point Efficiency

7 Conclusion

We have described a robust and efficient method bound constrained optimization that requires only function and gradient evaluations from the objective function. Its limited storage requirements make it well suited for large optimization problems. On some large problems, the algorithm can also compute a solution in parallel with high level of efficiency. Against other solvers in its class, numerical experiments showed that it performed very well.

Our implementation of this algorithm is available in the Toolkit for Advanced Optimization (TAO). The TAO project [5, 6] focuses on the design and implementation of component-based optimization software for the solution of large-scale optimization applications. Its design enables connection to lower-level support (parallel vectors, sparse matrix data structures, preconditioners, solvers) provided in toolkits such as PETSc, and thus we are able to build on top of these toolkits instead of having to redevelop code. Initial work in TAO has centered on the development of solvers for unconstrained and bound-constrained minimization and nonlinear least squares.

References

[1] B. M. Averick, R. G. Carter, and J. J. Moré, *The MINPACK-2 test problem collection*, Tech. Rep. ANL/MCS-TM-150, Argonne National Laboratory, 1991.

[2] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software

Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.

[3] ——, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.

[4] ——, *PETSc 2.0 users manual*, Tech. Rep. ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, Apr 2001.

[5] S. Benson, L. C. McInnes, and J. Moré, *Toolkit for Advanced Optimization (TAO) Web page*. See `http://www.mcs.anl.gov/tao`.

[6] ——, *TAO users manual*, Tech. Rep. ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2000.

[7] R. H. Byrd, P. Lu, J. Nocedal, and C. Y. Zhu, *A limited memory algorithm for bound constrained optimization*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1190–1208.

[8] E. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Tech. Rep. ANL/MCS-P861-1200, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, Jan. 2001.

[9] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1994.

[10] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer-Verlag, New York, 1999.