Hindawi Publishing Corporation
Abstract and Applied Analysis
Volume 2013, Article ID 478407, 6 pages
http://dx.doi.org/10.1155/2013/478407

Research Article

A Newton-Like Trust Region Method for Large-Scale Unconstrained Nonconvex Minimization

Yang Weiwei, Yang Yueting, Zhang Chenhui, and Cao Mingyuan

School of Mathematics and Statistics, Beihua University, Jilin 132013, China

Correspondence should be addressed to Yang Yueting; yyt2858@163.com

Received 8 June 2013; Accepted 4 September 2013

Academic Editor: Bo-Qing Dong

We present a new Newton-like method for large-scale unconstrained nonconvex minimization. And a new straightforward limited memory quasi-Newton updating based on the modified quasi-Newton equation is deduced to construct the trust region subproblem, in which the information of both the function value and gradient is used to construct approximate Hessian. The global convergence of the algorithm is proved. Numerical results indicate that the proposed method is competitive and efficient on some classical large-scale nonconvex test problems.

1. Introduction

We consider the following unconstrained optimization:

$$\min_{x\in R^n} f(x), \tag{1}$$

where $f : R^n \to R$ is continuously differentiable.

Trust region methods [1–14] are robust, can be applied to ill-conditioned problems, and have strong global convergence properties. Another advantage of trust region methods is that there is no need to require the approximate Hessian of the trust region subproblem to be positive definite. So, trust region methods are important and efficient for nonconvex optimization problems [6–8, 10, 12, 14]. For a given iterate $x_k \in R^n$, the main computation of trust region algorithms is solving the following quadratic subproblem:

$$\begin{aligned} \min_{s\in R^n} \quad & \phi_k(s) = g_k^T s + \frac{1}{2} s^T B_k s, \\ \text{s.t.} \quad & \|s\| \le \Delta_k, \end{aligned} \tag{2}$$

where $g_k = \nabla f(x_k)$ is the gradient of $f(x)$ at x_k, B_k is the true Hessian $\nabla^2 f(x_k)$ or its approximation, $\Delta_k > 0$ is a trust region radius, and $\|\cdot\|$ refers to the Euclidean norm on R^n. For a trial step s_k, which is generated by solving the subproblem (2), adequacy of the predicted reduction and true variation of the objective function is measured by means of the ratio

$$r_k = \frac{f(x_k) - f(x_k + s_k)}{\phi_k(0) - \phi_k(s_k)}. \tag{3}$$

Then the trust region radius Δ_k is updated according to the value of r_k. Trust region methods ensure that at least a Cauchy (steepest descent-like) decrease on each iteration satisfies an evaluation complexity bound of the same order under identical conditions [11]. It follows that Newton's method globalized by trust region regularization satisfies the same $O(\varepsilon^{-2})$ evaluation upper bound; such a bound can also be shown to be tight [12] provided additionally that the Hessian on the path of the iterates for which pure Newton steps are taken is Lipschitz continuous.

Newton's method has been efficiently safeguarded to ensure its global convergence to first- and even second-order critical points, in the presence of local nonconvexity of the objective using line search [3], trust region [4], or other regularization techniques [9, 13]. Many variants of these globalization techniques have been proposed. These generally retain fast local convergence under some nondegeneracy assumptions, are often suitable when solving large-scale problems, and sometimes allow approximate rather than true Hessians to be employed. Solving-large scale problems needs expensive computation and storage. So many researchers have studied the limited memory techniques [15–24]. The limited memory techniques are firstly applied to line search method. Liu and Nocedal [15, 16] proposed a limited memory BFGS method (L-BFGS) for solving unconstrained optimization and proved its global convergence. Byrd et al. [17] gave the compact representations of the limited memory BFGS and SR1 formula, which made it possible for combining limited memory techniques with trust region method. Considering that the L-BFGS updating formula used the gradient information merely and ignored the available function value information, Yang and Xu [19] deduced modified quasi-Newton formula with limited memory compact representation based on the modified quasi-Newton equation with a vector parameter [18]. Recently, some researchers combined the limited memory techniques with trust region method for solving large-scale unconstrained and constrained optimizations [20–24].

In this paper, we deduce a new straightforward limited memory quasi-Newton updating based on the modified quasi-Newton equation, which uses both available gradient and function value information, to construct the trust region subproblem. Then the corresponding trust region method is proposed for large-scale unconstrained nonconvex minimization. The global convergence of the new algorithm is proved under some appropriate conditions.

The rest of the paper is organized as follows. In the next section, we deduce a new straightforward limited memory quasi-Newton updating. In Section 3, a Newton-like trust region method for large-scale unconstrained nonconvex minimization is proposed and the convergence property is proved under some reasonable assumptions. Some numerical results are given in Section 4.

2. The Modified Limited Memory Quasi-Newton Formula

In this section, we deduce a straightforward limited memory quasi-Newton updating based on the modified quasi-Newton equation, which employs both the gradients and function values to construct the approximate Hessian and is a compensation for the missing data in limited memory techniques. And then we apply the derived formula in trust region method.

Consider the following modified quasi-Newton equation [18]:

$$B_{k+1}s_k = \widehat{y}_k, \tag{4}$$

where $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, $\widehat{y}_k = (1 + (\theta_k / s_k^T y_k)) y_k = \lambda_k y_k$, and $\theta_k = 6(f(x_k) - f(x_{k+1})) + 3(g_k + g_{k+1})^T s_k$. The quasi-Newton updating matrix constructed by (4) achieves a higher order accuracy in approximating Hessian. Based on (4), the modified BFGS (MBFGS) updating is as follows:

$$\begin{aligned} B_{k+1} &= B_k + \frac{\widehat{y}_k \widehat{y}_k^T}{\widehat{y}_k^T s_k} - \frac{(B_k s_k)(B_k s_k)^T}{s_k^T B_k s_k} \\ &= B_k + \lambda_k \frac{y_k y_k^T}{y_k^T s_k} - \frac{(B_k s_k)(B_k s_k)^T}{s_k^T B_k s_k}. \end{aligned} \tag{5}$$

For twice continuously differentiable function, if x_k converges to a point x^* at which $g(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite, then $\lim_{k\to\infty}\theta_k = 0$, and then $\lim_{k\to\infty}\lambda_k = 1$. Moreover, if k is sufficiently large, the MBFGS updating approaches to the BFGS updating.

Then formula (5) can be rewritten into the straightforward formula

$$B_{k+1} = B_k - a_k a_k^T + b_k b_k^T, \tag{6}$$

where $a_k = B_k s_k / (s_k^T B_k s_k)^{1/2}$ and $b_k = (\lambda_k / y_k^T s_k)^{1/2} y_k$. Thus, B_k can be recursively expressed as

$$\begin{aligned} B_k &= B_0 + \sum_{i=0}^{k-1} \left(b_i b_i^T - a_i a_i^T \right) \\ &= B_0 + [b_0, b_1, \ldots, b_{k-1}] \begin{bmatrix} b_0^T \\ b_1^T \\ \vdots \\ b_{k-1}^T \end{bmatrix} \\ &\quad - [a_0, a_1, \ldots, a_{k-1}] \begin{bmatrix} a_0^T \\ a_1^T \\ \vdots \\ a_{k-1}^T \end{bmatrix}. \end{aligned} \tag{7}$$

Let $Y_k = [b_0, b_1, \ldots, b_{k-1}]$, and let $S_k = [a_0, a_1, \ldots, a_{k-1}]$. Then the above formula can be simply written as

$$B_k = B_0 + Y_k Y_k^T - S_k S_k^T. \tag{8}$$

Formula (8) is called the whole memory quasi-Newton formula. For a given positive integer m (m usually is taken for $3, 5, 7, \ldots$), if we use the last m pairs $(s_{k-m}, y_{k-m}), \ldots, (s_{k-1}, y_{k-1})$ at the kth ($k \geq m$) iteration to update the starting matrix $B_k^{(0)}$ m times, according to (8), we get the following limited memory MBFGS (L-MBFGS) formula:

$$B_k = B_k^{(m)} = B_k^{(0)} + Y_k Y_k^T - S_k S_k^T, \tag{9}$$

where $Y_k = [b_{k-m}, \ldots, b_{k-1}]$, $S_k = [a_{k-m}, \ldots, a_{k-1}]$; then

$$\begin{aligned} B_k &= B_k^{(0)} + [b_{k-m}, \ldots, b_{k-1}] \begin{bmatrix} b_{k-m}^T \\ \vdots \\ b_{k-1}^T \end{bmatrix} \\ &\quad - [a_{k-m}, \ldots, a_{k-1}] \begin{bmatrix} a_{k-m}^T \\ \vdots \\ a_{k-1}^T \end{bmatrix}, \end{aligned} \tag{10}$$

where $a_{k-m+j} = B_{k-m+j}s_{k-m+j}/(s_{k-m+j}^T B_{k-m+j} s_{k-m+j})^{1/2}$ and $b_{k-m+j} = (\lambda_{k-m+j}/y_{k-m+j}^T s_{k-m+j})^{1/2} y_{k-m+j}$ $(j = 0, 1, \ldots, m-1)$.

Since the vectors b_i and a_i $(i = k-m, \ldots, k-2)$ can be obtained and saved from the previous iterations, we only need to compute the vectors b_{k-1} and a_{k-1} to achieve the limited memory quasi-Newton updating matrix. Suppose $B_k^{(0)} = I$, the computation of b_{k-1} needs $3n+3$ multiplications. Then we consider the computation of a_{k-1}. If B_{k-1} can be saved and multiplies by s_{k-1} directly, the process needs n^2 multiplications. In this paper, we compute the product $B_{k-1}s_{k-1}$ by (9). Consider

$$\begin{aligned} B_{k-1}s_{k-1} &= B_{k-1}^{(0)} s_{k-1} + Y_{k-1}Y_{k-1}^T s_{k-1} - S_{k-1}S_{k-1}^T s_{k-1} \\ &= B_{k-1}^{(0)} s_{k-1} + [b_{k-m-1}, \ldots, b_{k-2}] \begin{bmatrix} b_{k-m-1}^T s_{k-1} \\ \vdots \\ b_{k-2}^T s_{k-1} \end{bmatrix} \\ &\quad - [a_{k-m-1}, \ldots, a_{k-2}] \begin{bmatrix} a_{k-m-1}^T s_{k-1} \\ \vdots \\ a_{k-2}^T s_{k-1} \end{bmatrix}. \end{aligned} \tag{11}$$

So we need $4mn$ multiplications to achieve $B_{k-1}s_{k-1}$. Let $\overline{a}_{k-1} = B_{k-1}s_{k-1}$; then $a_{k-1} = (s_{k-1}^T \overline{a}_{k-1})^{-1/2} \overline{a}_{k-1}$. It takes $2n+1$ multiplications to compute a_{k-1}. Ignoring lower order terms, it is a total of $(4m+5)n$ multiplications to obtain B_k.

It is noticed that the only difference between the limited memory quasi-Newton method and the standard quasi-Newton method is in the matrix updating. Instead of storing the matrices B_k, we need to store m pairs vectors $\{a_i, b_i\}$ to define B_k implicitly. The product $B_k v$ or $v^T B_k v$ is obtained by performing a sequence of inner products involving v and the m most recent vectors pairs $\{a_i, b_i\}$.

In the following, we discuss the computation of the products $B_k v$ and $v^T B_k v$, $v \in R^n$. As the situation of (11), we need $4mn$ multiplications to obtain $B_k v$. If $B_k v$ has been computed, we only need to solve a vector product to obtain $v^T B_k v$ which needs n multiplications. If $B_k v$ has not been computed, we compute $v^T B_k v$ directly by using (9). Consider

$$\begin{aligned} v^T B_k v &= v^T B_k^{(0)} v + v^T Y_k Y_k^T v - v^T S_k S_k^T v \\ &= v^T B_k^{(0)} v + \left(Y_k^T v\right)^T \left(Y_k^T v\right) - \left(S_k^T v\right)^T \left(S_k^T v\right). \end{aligned} \tag{12}$$

The whole computation only requires $(2m+1)n + 4m$ multiplications. Thus, $2mn$ multiplications are saved in contrast to the previous method.

If we take $B_k^{(0)} = \gamma_k I$, $v^T v$ and $Y_k^T v$, $S_k^T v$ have been obtained and saved from the previous iteration, from (11), there are $2m+1$ multiplications to compute $v^T B_k v$; it is a considerable improvement on computation comparing with $(2m+1)n$.

Algorithm 1. Compute and save S_k, Y_k.

For $j = 0, 1, \ldots, m-1$,

Step 1. Compute $b_{k-m+j} = (\lambda_{k-m+j}/y_{k-m+j}^T s_{k-m+j})^{1/2} \times y_{k-m+j}$.

Step 2. Compute $a_{k-m+j} = B_{k-m+j}s_{k-m+j}$.

Step 3. Compute $(s_{k-m+j}^T a_{k-m+i})^{-1/2} a_{k-m+i}$.

Algorithm 2. Compute $B_k v$, $v^T B_k v$.

Let x_k be the current iteration point, the vectors a_{k-1}, b_{k-1}, g_k and matrixes S_{k-1}, Y_{k-1} have been obtained by the previous iteration.

Step 1. Update S_k, Y_k.

Step 2. Compute $S_k^T v$, $Y_k^T v$.

Step 3. Compute $B_k v$ by (11); compute $v^T B_k v$ by (12).

We use the form of (9) to store B_k. Instead of updating B_k into B_{k+1}, we update S_k, Y_k into S_{k+1}, Y_{k+1}.

3. Newton-Like Trust Region Method

In this section, we present a Newton-like trust region method for large-scale unconstrained nonconvex minimization.

Algorithm 3.

Step 0. Given $x_0 \in R^n$, $\varepsilon > 0$, $\widehat{\Delta} > 0$, $\Delta_0 \in (0, \widehat{\Delta})$, $\eta \in [0, 1/4)$, $B_0 \in R^{n \times n}$ is a given matrix. Compute $g_0 = \nabla f(x_0)$; set $k := 0$.

Step 1. If $\|g_k\| < \varepsilon$, then stop.

Step 2. Solve the subproblem (2) to obtain s_k.

Step 3. Compute

$$r_k = \frac{f(x_k) - f(x_k + s_k)}{\phi_k(0) - \phi_k(s_k)}. \tag{13}$$

Step 4. Compute

$$x_{k+1} = \begin{cases} x_k + s_k, & \text{if } r_k > \eta, \\ x_k, & \text{otherwise.} \end{cases} \tag{14}$$

Step 5. Update the trust region radius as the following:

$$\Delta_{k+1} = \begin{cases} \dfrac{1}{4}\Delta_k, & \text{if } r_k < \dfrac{1}{4}, \\ \min\left\{2\Delta_k, \widehat{\Delta}\right\}, & \text{if } r_k > \dfrac{3}{4}, \\ \Delta_k, & \text{otherwise.} \end{cases} \tag{15}$$

Step 6. By implementing Algorithm 1 to update S_k, Y_k into S_{k+1}, Y_{k+1} in order to update B_k into B_{k+1}, set $k := k+1$; go to Step 1.

In Step 2, using CG-Steihaug algorithm in [3] to solve the subproblem (2), the algorithm is suitable for solving large-scale unconstrained optimization. In the solving process, the products $B_k v$ and $v^T B_k v$ are computed by Algorithm 2. Then the whole computation of solving subproblem only requires $O(n)$ multiplications.

To give the convergence result, we need the following assumptions.

Assumption 4.

(H1) The level set $\Omega = \{x \mid f(x) \le f(x_0)\}$ is contained in a bounded convex set.

(H2) The gradient of the objective function $f(x)$ is Lipschitz continuous in the neighborhood of x^*; that is, there is a constant $L > 0$ such that

$$\|g(x) - g(y)\| \le L\|x - y\|, \quad \forall x, y \in R^n. \tag{16}$$

(H3) The solution s_k of the subproblem (2) satisfies

$$\phi_k(0) - \phi_k(s_k) \ge \sigma\|g_k\| \min\left\{\Delta_k, \frac{\|g_k\|}{\|B_k\|}\right\}, \tag{17}$$

where $\sigma \in (0, 1]$.

(H4) The solution s_k of subproblem (2) satisfies

$$\|s_k\| \le \gamma\Delta_k, \tag{18}$$

for $\gamma \ge 1$.

Lemma 5. *Suppose that (H1) holds and B_k is positive definite; there exist constants $M_2 > M_1 \ge 0$ such that*

$$\begin{aligned} M_1 &\le \frac{(g_{k+1} - g_k)^T (x_{k+1} - x_k)}{\|x_{k+1} - x_k\|^2} \le M_2, \\ M_1 &\le \frac{\|g_{k+1} - g_k\|^2}{(g_{k+1} - g_k)^T (x_{k+1} - x_k)} \le M_2, \end{aligned} \tag{19}$$

for any $x_{k+1}, x_k \in \Omega$ with $x_{k+1} \ne x_k$. Then matrices $\{B_k\}$ are uniformly bounded.

Proof. From Taylor expansion

$$f(x_{k+1}) = f(x_k) + g_k^T s_k + \frac{1}{2} s_k^T \nabla^2 f(x_k + t s_k) s_k, \quad t \in (0, 1), \tag{20}$$

we have

$$\left|2\left(f(x_k) - f(x_{k+1}) + g_k^T s_k\right)\right| = s_k^T \nabla^2 f(x_k + t s_k) s_k, \quad t \in (0, 1). \tag{21}$$

Then

$$\begin{aligned} |\theta_k| &= \left|6(f(x_k) - f(x_{k+1})) + 3(g_k + g_{k+1})^T s_k\right| \\ &\le 3\left|s_k^T \nabla^2 f(x_k + t s_k) s_k - (g_{k+1} - g_k)^T s_k\right|. \end{aligned} \tag{22}$$

Table 1

Problem	Objective function	Problem	Objective function
1	Gaussian function	2	Powell badly scaled function
3	Gulf function	4	Chebyquad function
5	Boundary value function	6	Broyden tridiagonal function
7	Separable cubic function	8	Arwhead function
9	Extended denschnb function	10	Extended denschnf function

From (19), we obtain that

$$|\theta_k| \le 6M_2\|s_k\|^2. \tag{23}$$

It is obvious that

$$|\lambda_k| = \left|1 + \frac{\theta_k}{s_k^T y_k}\right| \le 1 + \frac{6M_2}{M_1}. \tag{24}$$

Thus,

$$b_k^T b_k \le |\lambda_k| \frac{\|y_k^2\|}{y_k^T s_k} \le M_2\left(1 + \frac{6M_2}{M_1}\right). \tag{25}$$

Since $\mathrm{Tr}(xy^T) = x^T y (x, y \in R^n)$, $\mathrm{Tr}(A + B) = \mathrm{Tr}(A) + \mathrm{Tr}(B)$ $(A, B \in R^{n\times n})$ and from (9) (in which $B_k^{(0)} = I$), we have

$$\begin{aligned} B_k = B_k^{(0)} &+ [b_{k-m}, \dots, b_{k-1}] \begin{bmatrix} b_{k-m}^T \\ \vdots \\ b_{k-1}^T \end{bmatrix} \\ &- [a_{k-m}, \dots, a_{k-1}] \begin{bmatrix} a_{k-m}^T \\ \vdots \\ a_{k-1}^T \end{bmatrix}; \end{aligned} \tag{26}$$

then by (25) and B_k being positive definite, we have

$$\begin{aligned} \mathrm{Tr}(B_k) &= \mathrm{Tr}\left(B_k^{(0)}\right) + \sum_{j=0}^{m-1} \left(b_{k-m+j}^T b_{k-m+j} - a_{k-m+j}^T a_{k-m+j}\right) \\ &\le \mathrm{Tr}\left(B_k^{(0)}\right) + \sum_{j=0}^{m-1} b_{k-m+j}^T b_{k-m+j} \\ &\le n + m\left(M_2 + \frac{6M_2^2}{M_1}\right). \end{aligned} \tag{27}$$

By the definition of Euclidean norm: $\|A\| = \sqrt{\rho(A^T A)}$ $(A \in R^{m\times n})$, when $A \in R^{n\times n}$ is a symmetric matrix, $\|A\| = \rho(A)$. Obviously, B_k is a symmetric matrix.

Table 2: Numerical results for NLMTR and NTR.

Prob/Dim	NLMTR	NTR
	iter/nf/$\|g_k\|/f^*$/cpu	iter/nf/$\|g_k\|/f^*$/cpu
1/3	$4/8/2.7405e-009/1.1279e-008/0.00$	$28/64/3.0591e-007/6.7392e-015/0.02$
2/2	$33/79/2.3479e+003/4.3276e-004/0.01$	$36/78/0.0028/0.0014/0.00$
3/3	$39/83/0.0014/9.5599e-005/0.01$	$76/170/9.8496e-011/3.9977e-012/0.02$
4/5	$33/77/4.5033e-006/1.5576e-012/0.01$	$9/20/3.6793e-011/8.3131e-023/0.00$
5/10	$47/98/0.0095/5.5443e-004/0.01$	**
5/50	$51/107/3.6715e-004/8.5719e-006/0.01$	**
6/10	$/0.1274/4.7049e-004/0.01$	$36/92/3.9484e-007/2.5660e-015/0.01$
7/10	$18/40/7.2477e-009/1.3136e-017/0.00$	$10/20/3.5034e-009/3.7240e-018/0.00$
7/50	$22/44/9.0075e-009/1.9726e-017/0.01$	$11/28/2.7054e-009/2.1585e-018/0.01$
5/100	$45/95/1.0124e-004/1.1843e-006/0.01$	**
5/500	$36/78/4.3127e-006/1.0217e-008/0.21$	**
7/100	$23/46/5.2647e-009/6.6238e-018/0.02$	$12/31/3.0016e-011/2.8353e-022/0.77$
7/500	$25/50/3.9054e-009/3.8097e-018/0.48$	$12/28/4.3635e-009/5.4178e-018/2.64$
8/100	$39/96/0.0210/1.8391e-005/0.02$	$13/33/1.3995e-011/-1.4211e-014/1.14$
9/100	$41/91/6.3678e-004/6.7291e-008/0.03$	$9/19/1.0376e-010/2.1414e-021/0.72$
9/500	$41/91/0.0014/3.3646e-007/0.29$	$11/24/6.1012e-010/4.7336e-020/12.09$
10/100	$40/90/0.0111/4.1508e-007/0.02$	$26/68/2.4382e-011/7.7829e-025/2.15$
10/500	$42/94/0.0247/2.0754e-006/0.33$	$18/49/1.2142e-007/2.1634e-017/14.44$
5/1000	$34/74/1.0801e-006/1.2890e-009/0.72$	**
5/2000	$32/70/2.7030e-007/1.6186e-010/2.65$	**
5/5000	$29/64/4.3275e-008/1.0388e-011/15.08$	**
7/1000	$25/50/5.7571e-009/8.2784e-018/1.82$	$10/21/3.6235e-009/3.9031e-018/15.85$
7/2000	$25/50/8.3098e-009/1.7247e-017/7.13$	$11/23/2.5187e-010/1.9410e-020/130.06$
7/5000	$26/52/9.1295e-009/2.0187e-017/80.52$	**
9/1000	$41/91/0.0020/6.7291e-007/1.10$	$11/24/1.3827e-009/2.5035e-019/43.52$
9/2000	$44/97/0.0028/1.3458e-006/3.63$	$8/23/5.3761e-010/3.6664e-020/112.23$
9/5000	$44/97/0.0045/3.3646e-006/22.28$	**
10/1000	$42/94/0.0350/4.1508e-006/1.19$	$17/49/1.6202e-007/3.8522e-017/44.47$
10/2000	$42/94/0.0494/8.3015e-006/4.49$	$14/45/3.6692e-007/1.9770e-016/197.92$
10/5000	$45/100/0.0782/2.0754e-005/23.59$	**

**The algorithm fails.

Suppose the eigenvalues of B_k are $0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$; then

$$\begin{aligned}\|B_k\| = \lambda_n &\le \sum_{i=1}^{n} \lambda_i = \operatorname{Tr}(B_k) \\ &\le n + m\left(M_2 + \frac{6M_2^2}{M_1}\right).\end{aligned} \tag{28}$$

So, B_k is uniformly bounded. □

Theorem 6. *Let $\eta = 0$ in Algorithm 3. Suppose that Assumption 4 holds and $\|B_k\| \le \beta$ for some constant β. Let the sequence $\{x_k\}$ be generated by Algorithm 3. Then one has*

$$\lim_{k\to\infty} \inf \|g_k\| = 0. \tag{29}$$

The proof is similar to Theorem 4.7 in [3] and is omitted.

4. Numerical Results

In this section, we apply Algorithm 3 to solve nonconvex programming problems. Preliminary numerical results to illustrate the performance of Algorithm 3 are denoted by NLMTR. The contrast tests are called NTR, which is the same as NLMTR except that B_k is updated by BFGS formula. All tests are implemented by using Matlab R2008a on a PC with CPU 2.00 GHz and 2.00 GB RAM. The test problem collections for nonconvex unconstrained minimization are taken from Moré et al. in [25], the CUTEr collection [26, 27]. These problems are listed in Table 1.

All numerical results are listed in Table 2, in which iter stands for the number of iterations, which equals the number of gradient evaluations; nf stands for the number of objective function evaluations; Prob stands for the problem label; Dim stands for the number of variables of the tested problem; cpu denotes the CPU time for solving the problems; $\|g_k\|$ is the terminated gradient; and f^* denotes the optimal value.

We compare NLMTR with NTR. The trial step s_k is computed by CG-steihaug algorithm [3]. The matrix B_k of NLMTR is updated by the straightforward modified L-MBFGS formula (9). Choosing $\eta = 0.1$, $m = 3$. The matrices B_k of NTR is updated by BFGS formula in [3]. The iteration is terminated by $\|g_k\| \le \varepsilon$ or $\|s_k\| \le \varepsilon$, where $\varepsilon = 10^{-8}$. The related figures are listed in Table 2.

From Table 2, we can see that for small-scale problems, the optimal values and the gradient norms of NTR are more accurate than NLMTR. For middle-scale problems, the accuracy of NTR is higher, but the cpu time of NLMTR is shorter. For large-scale problems, the cpu time of NTR is much more than NLMTR, and for some problems NTR fails, especially when $n = 5000$. So NLMTR is suitable for solving large-scale nonconvex problems.

Acknowledgments

This work is supported in part by the NNSF (11171003) of China, the Key Project of Chinese Ministry of Education (no. 211039), and Natural Science Foundation of Jilin Province of China (no. 201215102).

References

[1] M. J. D. Powell, "A new algorithm for unconstrained optimization," in *Nonlinear Programming*, J. B. Rosen, O. L. Mangasarian, and K. Ritter, Eds., pp. 31–65, Academic Press, New York, NY, USA, 1970.

[2] J. Nocedal and Y.-X. Yuan, "Combining trust region and line search techniques," in *Advances in Nonlinear Programming*, Y. Yuan, Ed., vol. 14, pp. 153–175, Kluwer Academic, Dordrecht, The Netherlands, 1998.

[3] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 1999.

[4] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa, USA, 2000.

[5] H. P. Wu and Q. Ni, "A new trust region algorithm with a conic model," *Numerical Mathematics*, vol. 30, no. 1, pp. 57–67, 2008.

[6] Ph. L. Toint, "Global convergence of a class of trust-region methods for nonconvex minimization in Hilbert space," *IMA Journal of Numerical Analysis*, vol. 8, no. 2, pp. 231–252, 1988.

[7] M. J. D. Powell and Y. Yuan, "A trust region algorithm for equality constrained optimization," *Mathematical Programming*, vol. 49, no. 2, pp. 189–211, 1990.

[8] D.-H. Li and M. Fukushima, "A modified BFGS method and its global convergence in nonconvex minimization," *Journal of Computational and Applied Mathematics*, vol. 129, no. 1-2, pp. 15–35, 2001.

[9] Y. Nesterov and B. T. Polyak, "Cubic regularization of Newton method and its global performance," *Mathematical Programming*, vol. 108, no. 1, pp. 177–205, 2006.

[10] Q. Guo and J.-G. Liu, "Global convergence of a modified BFGS-type method for unconstrained non-convex minimization," *Journal of Applied Mathematics & Computing*, vol. 24, no. 1-2, pp. 325–331, 2007.

[11] S. Gratton, A. Sartenaer, and P. L. Toint, "Recursive trust-region methods for multiscale nonlinear optimization," *SIAM Journal on Optimization*, vol. 19, no. 1, pp. 414–444, 2008.

[12] C. Cartis, N. I. M. Gould, and P. L. Toint, "On the complexity of steepest descent, Newton's and regularized Newton's methods for nonconvex unconstrained optimization problems," *SIAM Journal on Optimization*, vol. 20, no. 6, pp. 2833–2852, 2010.

[13] C. Cartis, N. I. M. Gould, and P. L. Toint, "Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results," *Mathematical Programming*, vol. 127, no. 2, pp. 245–295, 2011.

[14] D. Xue, W. Sun, and H. He, "A structured trust region method for nonconvex programming with separable structure," *Numerical Algebra, Control and Optimization*, vol. 3, no. 2, pp. 283–293, 2013.

[15] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Mathematics of Computation*, vol. 35, no. 151, pp. 773–782, 1980.

[16] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 3, pp. 503–528, 1989.

[17] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-Newton matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 2, pp. 129–156, 1994.

[18] C. Xu and J. Zhang, "A survey of quasi-Newton equations and quasi-Newton methods for optimization," *Annals of Operations Research*, vol. 103, pp. 213–234, 2001.

[19] Y. T. Yang and C. X. Xu, "A compact limited memory method for large scale unconstrained optimization," *European Journal of Operational Research*, vol. 180, no. 1, pp. 48–56, 2007.

[20] Q. Ni and Y. Yuan, "A subspace limited memory quasi-Newton algorithm for large-scale nonlinear bound constrained optimization," *Mathematics of Computation*, vol. 66, no. 220, pp. 1509–1520, 1997.

[21] O. P. Burdakov, J. M. Martínez, and E. A. Pilotta, "A limited-memory multipoint symmetric secant method for bound constrained optimization," *Annals of Operations Research*, vol. 117, no. 1–4, pp. 51–70, 2002.

[22] Z. H. Wang, "A limited memory trust region method for unconstrained optimization and its implementation," *Mathematica Numerica Sinica*, vol. 27, no. 4, pp. 395–404, 2005.

[23] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: an SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[24] H. Liu and Q. Ni, "New limited-memory symmetric secant rank one algorithm for large-scale unconstrained optimization," *Transactions of Naniing University of Aeronautics and Astronautics*, vol. 25, no. 3, pp. 235–239, 2008.

[25] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software," *Association for Computing Machinery*, vol. 7, no. 1, pp. 17–41, 1981.

[26] N. I. M. Gould, D. Orban, and P. L. Toint, "GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization," *Association for Computing Machinery*, vol. 29, no. 4, pp. 353–372, 2003.

[27] H. Y. Benson, "Cute models," http://orfe.princeton.edu/~rvdb/ampl/nlmodels/cute/.