Limited Memory Bundle Method for Large Bound Constrained Nonsmooth Optimization

Napsu Karmitsa

Department of Mathematics, University of Turku, FI-20014 Turku, Finland. E-mail: napsu@karmitsa.fi

1. Abstract

Practical optimization problems often involve nonsmooth functions of hundreds or thousands of variables. As a rule, the variables in such large problems are restricted to certain meaningful intervals. In the report [Haarala, Mäkelä, 2006] we have described an efficient adaptive limited memory bundle method for large-scale nonsmooth, possibly nonconvex, bound constrained optimization. Although it works very well in numerical experiments it suffers from one theoretical drawback. Namely, it is not necessarily globally convergent. In this paper, a globally convergent variant of this method is proposed. In addition, some results from numerical experiments are given.

2. Keywords: Nondifferentiable programming, large-scale optimization, bundle methods, limited memory methods, box constraints.

3. Introduction

In this paper, we describe an adaptive limited memory bundle algorithm (LMBM-B) for solving large, possibly nonconvex, nonsmooth (nondifferentiable) bound constrained optimization problems. We write this problem as

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad x^l \leq x \leq x^u,
\end{align*}
\]

where the objective function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is supposed to be locally Lipschitz continuous and the number of variables \( n \) is supposed to be large (say 1000 or more). Moreover, the vectors \( x^l \) and \( x^u \) representing the lower and the upper bounds on the variables are fixed and the inequalities in Eq.(1) are taken component-wise. A point \( x \in \mathbb{R}^n \) satisfying the bounds is called feasible and a set \( \mathcal{F} \) of all feasible points is called the feasible region for problem (1). That is, \( \mathcal{F} = \{ x \in \mathbb{R}^n \mid x^l \leq x \leq x^u \} \).

In [1, 2, 3] we have proposed a limited memory bundle method (LMBM) for general, possibly nonconvex, nonsmooth large-scale unconstrained optimization. LMBM is a hybrid of the variable metric bundle methods [4, 5] and the limited memory variable metric methods (see e.g. [6, 7]), where the first ones have been developed for small- and medium-scale nonsmooth optimization and the latter ones, on the contrary, for smooth large-scale optimization. LMBM exploits the ideas of the variable metric bundle methods, namely the utilization of null steps and simple aggregation of subgradients (generalized gradients [8]), but the search direction is calculated using a limited memory approach. Therefore, the time-consuming quadratic direction finding problem appearing in standard bundle methods (see e.g. [9, 10, 11]) need not to be solved and the number of stored subgradients (i.e. the size of the bundle) is independent of the dimension of the problem. Furthermore, LMBM uses only few vectors to represent the variable metric updates and, thus, it avoids storing and manipulating large matrices as is the case in variable metric bundle methods [4, 5].

In [12], a new variant of the method, LMBM-B, suitable for solving bound constrained problems was introduced. The constraint handling in LMBM-B is based on gradient projection (naturally, we use subgradients instead of gradients) and dual subspace minimization and it is adopted from the smooth limited memory BFGS method for bound constrained optimization [13]. Although numerically very efficient, the method described in [12] is not necessary globally convergent in nonsmooth case.

In order to prove the global convergence of LMBM-B some modifications had to be made. Namely, we included stark projections in aggregation procedure to guarantee the convergence of aggregate subgradients to zero, some corrections to limited memory matrices to preserve sufficient positive definiteness and boundedness of these matrices whenever necessary, and a slightly modified line search procedure. Although this may sound like an easy task several open question had to be answered and many implementational challenges had to be solved before preserving even a hint of the efficiency of the previous version together with the theoretical convergence properties.
In what follows, we assume that at every feasible point \(x \in \mathcal{F}\) we can evaluate the value of the objective function \(f(x)\) and an arbitrary subgradient \(\xi \in \mathbb{R}^n\) from the subdifferential [8]

\[
\partial f(x) = \text{conv}\{ \lim_{i \to \infty} \nabla f(x_i) \mid x_i \to x \text{ and } \nabla f(x_i) \text{ exists} \},
\]

where "conv" denotes the convex hull of a set.

4. Method

The globally convergent version of LMBM-B (see Figure 1) is characterized by the usage of null steps together with the aggregation and projection of subgradients. Moreover, the limited memory approach is utilized in the calculation of the search direction and the aggregate values. The usage of null steps gives further information about the nonsmooth objective function in the case the search direction is not “good enough”. On the other hand, simple aggregation and stark projection of subgradients guarantees the convergence of projected aggregate subgradients to zero and make it possible to evaluate a termination criterion.

![Figure 1: Adaptive limited memory bundle method with bounds.](image-url)
The search direction is calculated using a two-stage approach. First, we define the quadratic model function $q_k$ that approximates the objective function at the iteration point $x_k$ by

$$q_k(x) = f(x_k) + x^T_k (x - x_k) + \frac{1}{2} (x - x_k)^T B_k (x - x_k).$$

(3)

Here $\tilde{\xi}_k$ is the aggregate subgradient of the objective function and $B_k$ is a positive definite limited memory variable metric update that, in smooth case, represents the approximation of the Hessian matrix. Now, the generalized gradient projection method is used to find the generalized Cauchy point $x^*_k$ [14] and, at the same time, to identify the active set $F^*_k$ of the problem. The procedure used in LMBM-B is rather similar to that in [13]. We only use here the aggregate subgradient of the objective function instead of gradient and, in addition to the limited memory BFGS update formula, we utilize the limited memory SR1 update whenever necessary. That is, after a null step (see Figure 1).

The generalized Cauchy point at iteration $k$ is defined as the first local minimizer (starting from $x_k$) of the univariate piecewise quadratic function

$$\tilde{q}_k(t) = q_k(\mathcal{P}_c[x_k - t\tilde{\xi}_k, x^i, x^u]),$$

(4)

along the projected gradient direction $\mathcal{P}_c[x_k - \tilde{\xi}_k, x^i, x^u] - x_k$ (see e.g. [14]). Here we have defined the projection operator $\mathcal{P}_c[\cdot]$ (component-wise) by

$$\mathcal{P}_c[x, x^i, x^u] = \begin{cases} x^i, & \text{if } x_i < x^i_k \\ x_i, & \text{if } x_i \in [x^i_k, x^u_k] \\ x^u, & \text{if } x_i > x^u_k. \end{cases}$$

(5)

This operator projects the point $x$ into the feasible region $F$ defined by bounds $x^i$ and $x^u$. Now, if we denote by $t_k^*$ the value of $t$ corresponding to the first local minimum of $\tilde{q}_k(t)$, the generalized Cauchy point is given by

$$x^*_k = \mathcal{P}_c[x_k - t_k^*\tilde{\xi}_k, x^i, x^u].$$

(6)

The variables whose values at $x^*_k$ are at lower or upper bound, comprise the active set $F^*_k = \{i \mid x^*_k = x^i_k \text{ or } x^*_k = x^u_k\}$. Here, we have denoted by $x^*_k[i]$ the $i$th component of the vector $x^*_k$. The calculation of this generalized Cauchy point makes it possible to add and delete several bounds from the active set during a single iteration, which may be an important feature for both nonsmooth [15] and large-scale [16] problems. For the practical computation of generalized Cauchy point see [12, 13, 17].

When the generalized Cauchy point has been found, we approximately minimize the quadratic model function (3) over the space of free variables, in other words, the variables in the active set are treated as equality constraints. The subspace minimization procedure used is in principal the same as the dual space method in [13] but, as before, we use the aggregate subgradient of the objective function and we utilize the limited memory SR1 update if the previous step taken was a null step.

We solve $d$ from the smooth quadratic problem

$$\begin{cases} \text{minimize} & \tilde{\xi}_k^T d + \frac{1}{2} d^T B_k d \\ \text{such that} & A_k^T d = b_k \quad \text{and} \\ & x^i \leq x_k + d \leq x^u, \end{cases}$$

(7)

where $A_k$ is the matrix of active constraints gradients at $x^*_k$ and $b_k = A_k^T (x^*_k - x_k)$. Note that $A_k$ consists of $n_A$ unit vectors (here $n_A$ is the number of elements in the active set $F^*_k$) and $A_k^T A_k$ is equal to identity.

We first ignore the bound constraints. The first order sufficient optimality conditions for problem (7) without bounds are

$$\begin{align} \tilde{\xi}_k + B_k d_k^* + A_k \mu_k^* &= 0 \\
A_k^T d_k^* &= b_k. \end{align}$$

(8)

(9)
Now, by multiplying Eq. (8) by $A_k^T D_k$ (we denote by $D_k$ the update formula that is inverse of $B_k$) and by using Eq. (9), we obtain
\[ (A_k^T D_k A_k) \mu_k^* = -A_k^T D_k \xi_k^* - b_k, \]
which determines Lagrange multipliers $\mu_k^* \in \mathbb{R}^n$. The linear system (10) can be solved by utilizing the Sherman-Morrison-Woodbury formula and the compact representation of limited memory matrices (see [13]). Thus, $d_k^*$ can be given by
\[ B_k d_k^* = -A_k \mu_k^* - \xi_k^*. \]
(11)
If there are no active variables, we simply obtain $d_k^* = -D_k \xi_k^*$, which is the formula used also in the original unconstrained version of the limited memory bundle method [1, 2, 3]. In the case the vector $x_k + d_k^*$ violates the bounds in Eq. (7), we, similarly to [13], backtrack along the line joining the infeasible point $x_k + d_k^*$ and the generalized Cauchy point $x_k^*$ to regain the feasible region. That is, we first compute
\[ \alpha_k^* = \min \{ 1, \max \{ \alpha | x_k^i - \bar{x}_k^i + \alpha (x_k^i + d_k^i) - x_k^0, \ i \in I_T^k \} \}, \]
where $I_T^k = \{ j | j = \{ 1, \ldots, n \} \setminus I_k^k \}$ is the set of free variables, and then we set $\bar{x} = x_k^* + \alpha_k^*(x_k + d_k^* - x_k^0)$ and $d_k = \bar{x} - x_k$. Again, see [12, 13, 17] for details of the practical computations.
LMBM-B generates a sequence of basic points $(x_k) \subset F$ together with a sequence of auxiliary points $(y_k) \subset F$. A new iteration point $x_{k+1}$ and a new auxiliary point $y_{k+1}$ are produced using a special line search procedure [17] such that
\[ x_{k+1} = x_k + t_{L}^k d_k \quad \text{and} \quad y_{k+1} = x_k + t_{R}^k d_k, \]
with $y_1 = x_1$, where $t_{L}^k \in (0, t_{max}^k]$ and $t_{R}^k \in [0, t_{R}^k]$ are step sizes, $t_{max}^k \geq 1$ is the upper bound for the step size that assures the feasibility of produced points.
A necessary condition for a serious step is to have
\[ t_{R}^k = t_{L}^k > 0 \quad \text{and} \quad f(y_{k+1}) \leq f(x_k) - \varepsilon_L t_{R}^k w_k, \]
where $\varepsilon_L \in (0, 1/2)$ is a line search parameter and $w_k > 0$ represents the desirable amount of descent of $f$ at $x_k$. If condition (14) is satisfied, we set $x_{k+1} = y_{k+1}$ and a serious step is taken.
Otherwise, we take a null step. In this case, the usage of special line search procedure guarantees that we have
\[ t_{R}^k > t_{L}^k = 0 \quad \text{and} \quad -\beta_{k+1} \geq D_k \xi_k^* \geq -\varepsilon_R w_k, \]
where $\varepsilon_R \in (\varepsilon_L, 1/2)$ is a line search parameter, $\xi_k^* = \partial f(y_{k+1})$, $D_k \xi_k^*$ denotes a stark projection of $\xi_k^*$ at $x$ (to be described short after), and $\beta_{k+1}$ is the subgradient locality measure [18, 19] similar to bundle methods. In the case of a null step, we set $x_{k+1} = x_k$ but information about the objective function is increased because we store the auxiliary point $y_{k+1}$ and the corresponding auxiliary subgradient $\xi_{k+1}$.
LMBM-B uses the original subgradient $\xi_k$ after the serious step and the aggregate subgradient $\xi_k^*$ after the null step for direction finding (i.e. we set $\xi_k = \xi_k^*$ if the previous step was a serious step). The aggregation procedure used in the previous versions of LMBM [1, 2, 3, 12] is similar to that of the original variable metric bundle methods [4, 5] except that the variable metric updates are calculated using limited memory approach. However, in order to guarantee the global convergence of the bound constrained version, we need to consider projections of subgradients instead of original subgradients in the aggregation procedure. It may seem that utilization of active set $I_A^k$ in the projection procedure would be a natural choice. However, active set $I_A^k$ is calculated at the generalized Cauchy point $x_k^*$ and it may change in consecutive null steps, which is highly undesirable from the view point of global convergence. Therefore, we instead calculate a very simple projection at point $x_k$ that is not changing ($x_{k+1} = x_k$ in null steps). We define this projection operator $P_x[\cdot]$ at point $x$ (component-wise) by
\[ P_x[\xi] = \begin{cases} 0, & \text{if } x_i - x_i^0 \geq 0 \\
\xi_i, & \text{if } x_i \in (x_i^0, x_i^0) \\
0, & \text{if } x_i^0 - x_i \leq 0. \end{cases} \]
(16)
In what follows we call this stark projection the $P_x$-projection (at point $x$).
Now, the aggregation procedure is made by determining multipliers $\lambda_i^k$ satisfying $\lambda_i^k \geq 0$ for all $i \in \{1, 2, 3\}$, and $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function
\[
\varphi(\lambda_1, \lambda_2, \lambda_3) = \mathcal{P}_{x_k}[\lambda_1 \xi_{m} + \lambda_2 \xi_{k+1} + \lambda_3 \tilde{\xi}_k]^T D_k \mathcal{P}_{x_k}[\lambda_1 \xi_{m} + \lambda_2 \xi_{k+1} + \lambda_3 \tilde{\xi}_k] + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k).
\]
Here $\xi_{m} \in \partial f(x_k)$ is the current subgradient ($m$ denotes the index of the iteration after the latest serious step, i.e. $x_k = x_m$, $\xi_{k+1} \in \partial f(y_{k+1})$ is the auxiliary subgradient, and $\xi_k$ is the current aggregate subgradient from the previous iteration ($\tilde{\xi}_1 = \xi_1$). In addition, $\beta_{k+1}$ is the current locality measure and $\tilde{\beta}_k$ is the current aggregate locality measure ($\tilde{\beta}_1 = 0$).

The next $\xi_{k+1}$ is defined as a convex combination of the subgradients mentioned above:
\[
\xi_{k+1} = \lambda_1^k \xi_{m} + \lambda_2^k \xi_{k+1} + \lambda_3^k \tilde{\xi}_k
\]
and the next $\tilde{\beta}_{k+1}$ as a convex combination of the locality measures:
\[
\tilde{\beta}_{k+1} = \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.
\]
\(\mathcal{P}_{x^*}\)-projection depends only on point $x$ and not on the subgradient $\xi$ on focus. Thus, $\mathcal{P}_{x_k}[\lambda_1 \xi_{m} + \lambda_2 \xi_{k+1} + \lambda_3 \tilde{\xi}_k] = \lambda_1 \mathcal{P}_{x_k}[\xi_{m}] + \lambda_2 \mathcal{P}_{x_k}[\xi_{k+1}] + \lambda_3 \mathcal{P}_{x_k}[\tilde{\xi}_k]$ and this makes the minimization of function (17) rather an easy task.

We utilize the limited memory approach (see e.g. [6, 7] and Appendix) in the calculation of the generalized Cauchy point, search direction, and aggregate values. The idea of limited memory matrix updating is, instead of storing the large $n \times n$-matrices $B_k$ and $D_k$, to store a certain (usually small constant) number $\hat{n}_c$ of vectors, so-called correction pairs obtained at the previous iterations of the algorithm, and to use these correction pairs to implicitly define the variable metric matrices. When the storage space available is used up, the oldest correction pair is deleted to make room for new one; thus, except for the first few iterations, we always have the $\hat{n}_c$ most recent correction pairs available.

The utilization of limited memory approach naturally means that the variable metric updates are not as accurate as if we used standard variable metric updates (see e.g. [20]). However, both the storage space required and the number of operations needed in the calculations are significantly smaller. Namely, the number of operations needed is $O(\hat{n})$ while with standard variable metric updates used in original variable metric bundle methods [4, 5], it is $O(n^2)$.

In the adaptive LMBM, first introduced in [1] and then used in [12] and here, the number of stored correction pairs $\hat{n}_c$ may change during the computation. This means that we can start the optimization with a small $\hat{n}_c$ and when we are closer to the optimal point, $\hat{n}_c$ may be increased until some predefined upper limit $\hat{m}_c$ is achieved. The aim of this adaptability is to improve the accuracy of the basic method without losing much from efficiency, that is, without increasing computational costs too much.

The limited memory variable metric matrices used in our algorithm are represented in the compact matrix form originally described in [6]. We use both the limited memory BFGS and the limited memory SR1 update formulae in the calculations of the search direction and the aggregate values. If the previous step was a null step, the matrices $D_k$ and $B_k$ are formed using the limited memory SR1 updates (see Appendix, Eq.(26) and Eq.(27)). The SR1 update formulae give us a possibility to preserve the boundedness and some other properties of generated matrices that guarantee the global convergence of the method. Otherwise, since these properties are not required after a serious step, the more efficient limited memory BFGS updates (see Appendix, Eq.(21) and Eq.(24)) are employed.

The SR1-update formulae do not in general preserve the positive definiteness. Moreover, both the nonconvexity and bounds may prevent the condition $d_i^T (\xi_{i+1} - \xi_m) > 0$ for all $i = 1, \ldots, k-1$, that is the classical condition for the positive definiteness of the BFGS update, from satisfying. Thus, to maintain the positive definiteness of the generated matrices, we simply skip the individual updates (discard the newest correction pair not the oldest one) if they would violate positive definiteness.

The basic assumption for bundle methods to converge is that after a null step we have $z^T D_{k+1} z \leq z^T D_k z$ for all $z \in \mathbb{R}^n$. In LMBM-B this is guaranteed by the special limited memory SR1 update [1, 3]. In addition, to ensure the global convergence of the method, the matrices $D_k$ are assumed to be
uniformly positive definite and uniformly bounded (we say that a matrix is bounded if its eigenvalues lie in the compact interval that does not contain zero). This is guaranteed by adding some positive definite correction matrices to matrices $D_k$ when necessary (i.e. we set $D_k = D_k + \rho I$ with $\rho > 0$). The detailed algorithm of globally convergent LMBM-B is given in [17].

**Remark.** Under mild assumptions LMBM-B is proved to be globally convergent for locally Lipschitz continuous objective functions, which are not necessarily differentiable or convex [17].

5. Numerical Experiments

We now compare the proposed globally convergent limited memory bundle method LMBM-B to the proximal bundle method PBNCGC (version 2.0, [10, 21]) in a limited number of nonsmooth large-scale test problems. We used the solver PBNCGC as a benchmark since the proximal bundle method is the most frequently used bundle method in nonsmooth optimization. In addition, we compared the new version LMBM-B to the older, non-globally convergent, version LMBM-B-OLD [12]. The experiments were performed in a Intel® Core™ 2 CPU 1.80GHz and all the algorithms were implemented in Fortran77 with double-precision arithmetic.

The solvers were tested with 10 nonsmooth academic minimization problems described in [22]*. The number of variables used in our experiments were 1000, 2000, and 4000. The solvers were tested with relatively small amount of stored subgradient information. That is, the size of the bundle $m_\xi$ was set to 10 for LMBM-B and LMBM-B-OLD and to 100 for PBNCGC (since the previous experiments [1, 3] have shown that a larger bundle usually works better with PBNCGC). With both LMBM-B and LMBM-B-OLD, we used the values $\hat{m}_\mu = 15$ and $\hat{m}_\varepsilon = 7$ due to good results of previous experiments with the older version [12]. The final accuracy parameter $\varepsilon$ was set to $10^{-5}$ in all the cases and, otherwise, the default parameters of the solvers were used. In addition to the usual stopping criteria of the solvers, we terminated the experiments if the CPU time elapsed exceeded half an hour.

The results of experiments are summarized in Tables 1, 2, and 3, where $N_i$ and $N_f$ denote the numbers of iterations and function evaluations used, respectively, $f$ denotes the value of the objective function at termination, and the time is an average CPU time elapsed per problem and it is given in seconds (only the accurately and successfully terminated problems were included).

In Tables 1, 2, and 3 we see the superiority of the different variants of LMBM-B when comparing the computational times; the computation times elapsed with LMBM-B and LMBM-B-OLD were usually hundreds of times shorter than those of PBNCGC. On the other hand, there was not a very big difference in the computational times between the different variants of LMBM-B. Although, the globally convergent version LMBM-B usually needed more computation time than the older one. This is due to fewer function evaluations required with LMBM-B-OLD (see Tables 1, 2, and 3). The increased number of function evaluations needed with LMBM-B is probably due to less accurate search direction caused by the stark projection of subgradients. Thus, different projection possibilities need to be studied.

The proximal bundle solver PBNCGC always needed less function evaluations than the different variants of LMBM-B. However, as can be seen when comparing the computational times, each individual iteration with PBNCGC was much more costly than that with LMBM-B or LMBM-B-OLD. Indeed, with PBNCGC all the problems with 4000 variables but two (problems 2 and 6) were terminated because the time limit exceeded. This also explains the inaccurate results obtained with PBNCGC (see Table 3).

The new variant LMBM-B failed to solve two of the problems (problems 1 and 2) with any number of variables tested. These failures were quite predictable, since both of these problems are reported to be difficult to solve with limited memory bundle method even without the bound constraints [2].

To sum up, the new solver LMBM-B did not beat up LMBM-B-OLD due to larger number of function evaluations needed. Although in theory, the new version is globally convergent and the older version is not, there was no significant improvement in the accuracy or robustness of the method. However, when comparing to PBNCGC, the new solver LMBM-B was substantially faster.

---

*All the problems can be downloaded from the website http://napsu.karmitsa.fi/lmbm
Table 1: Results for bound constrained problems with 1000 variables.

<table>
<thead>
<tr>
<th>Solver</th>
<th>LMBM-B</th>
<th>LMBM-B-OLD</th>
<th>PBNCGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Ni/Nf</td>
<td>f</td>
<td>Ni/Nf</td>
</tr>
<tr>
<td>1</td>
<td>-/-</td>
<td>fail</td>
<td>11346/11385</td>
</tr>
<tr>
<td>2</td>
<td>-/-</td>
<td>fail</td>
<td>676/825</td>
</tr>
<tr>
<td>3</td>
<td>286/2029</td>
<td>-1396.08</td>
<td>35/62</td>
</tr>
<tr>
<td>4</td>
<td>157/921</td>
<td>2334.75</td>
<td>64/86</td>
</tr>
<tr>
<td>5</td>
<td>95/632</td>
<td>2042.62</td>
<td>63/252</td>
</tr>
<tr>
<td>6</td>
<td>278/291</td>
<td>0.09547</td>
<td>526/526</td>
</tr>
<tr>
<td>8</td>
<td>120/648</td>
<td>-698.121</td>
<td>74/310</td>
</tr>
<tr>
<td>9</td>
<td>129/895</td>
<td>8.45046</td>
<td>55/123</td>
</tr>
<tr>
<td>10</td>
<td>142/812</td>
<td>147.301</td>
<td>63/137</td>
</tr>
</tbody>
</table>

Time 1.07  0.21*  67.48

* Problem 1 that took 71.91 sec to compute is not included in average CPU time of LMBM-B-OLD.

Table 2: Results for bound constrained problems with 2000 variables.

<table>
<thead>
<tr>
<th>Solver</th>
<th>LMBM-B</th>
<th>LMBM-B-OLD</th>
<th>PBNCGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Ni/Nf</td>
<td>f</td>
<td>Ni/Nf</td>
</tr>
<tr>
<td>1</td>
<td>-/-</td>
<td>fail</td>
<td>-/-</td>
</tr>
<tr>
<td>2</td>
<td>-/-</td>
<td>fail</td>
<td>3375/3453</td>
</tr>
<tr>
<td>3</td>
<td>112/644</td>
<td>-2793.50</td>
<td>36/71</td>
</tr>
<tr>
<td>4</td>
<td>150/749</td>
<td>4671.97</td>
<td>74/139</td>
</tr>
<tr>
<td>5</td>
<td>101/421</td>
<td>4087.25</td>
<td>74/276</td>
</tr>
<tr>
<td>6</td>
<td>517/522</td>
<td>0.09531</td>
<td>-/-</td>
</tr>
<tr>
<td>7</td>
<td>63/427</td>
<td>200.717</td>
<td>53/91</td>
</tr>
<tr>
<td>8</td>
<td>138/686</td>
<td>-1396.05</td>
<td>68/164</td>
</tr>
<tr>
<td>9</td>
<td>86/500</td>
<td>16.9159</td>
<td>63/177</td>
</tr>
<tr>
<td>10</td>
<td>146/812</td>
<td>294.911</td>
<td>108/110</td>
</tr>
</tbody>
</table>

Time 1.11  0.68  540.14

Table 3: Results for bound constrained problems with 4000 variables.

<table>
<thead>
<tr>
<th>Solver</th>
<th>LMBM-B</th>
<th>LMBM-B-OLD</th>
<th>PBNCGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Ni/Nf</td>
<td>f</td>
<td>Ni/Nf</td>
</tr>
<tr>
<td>1</td>
<td>-/-</td>
<td>fail</td>
<td>-/-</td>
</tr>
<tr>
<td>2</td>
<td>-/-</td>
<td>fail</td>
<td>24/28</td>
</tr>
<tr>
<td>3</td>
<td>66/403</td>
<td>-5555.69</td>
<td>42/71</td>
</tr>
<tr>
<td>4</td>
<td>172/946</td>
<td>9346.40</td>
<td>88/126</td>
</tr>
<tr>
<td>5</td>
<td>149/686</td>
<td>8176.57</td>
<td>100/152</td>
</tr>
<tr>
<td>6</td>
<td>-/-</td>
<td>fail</td>
<td>19/38</td>
</tr>
<tr>
<td>7</td>
<td>281/2369</td>
<td>399.900</td>
<td>-/-</td>
</tr>
<tr>
<td>8</td>
<td>120/521</td>
<td>-2794.24</td>
<td>123/329</td>
</tr>
<tr>
<td>9</td>
<td>129/639</td>
<td>33.8113</td>
<td>76/128</td>
</tr>
<tr>
<td>10</td>
<td>204/1117</td>
<td>589.780</td>
<td>95/230</td>
</tr>
</tbody>
</table>

Time 5.57  1.90  1562.28
6. Conclusions
In this paper, we have described a new variant LMBM-B of the limited memory bundle method for bound constrained nonsmooth large-scale optimization. The preliminary numerical experiments confirm that LMBM-B is efficient for both convex and nonconvex large-scale nonsmooth optimization problems. With large numbers of variables it used significantly less CPU time than the proximal bundle method tested. Moreover, the difference between the computational times of this globally convergent variant and the previous (non convergent) version of the method was not as large as we foreboded.

A drawback of the new variant is clearly the increased number of function evaluations needed. This is probably due to less accurate search direction caused by the stark projection of subgradients. Thus, different projection possibilities need to be studied.

The fact that LMBM-B only generates feasible points may be essential in the case the objective function or the subgradient values are undefined or difficult to compute if some of the constraints are violated. Furthermore, it can be an advantage in many industrial applications, where function evaluation may be very expensive. Since any intermediate solution can be employed, the iterations can be stopped whenever the result is satisfactory. Due to this feasibility, the efficiency of the method, and the fact that the objective function need not to be differentiable or convex, we expect LMBM-B to be very useful in solving optimization problems arising in real world modeling.

Acknowledgements
I would like to thank Prof. Marko M. Mäkelä for his valuable comments. This work was financially supported by Magnus Ehrnrooth Foundation and University of Turku (Finland).

Appendix
The limited memory variable metric matrices used in our algorithm are represented in the compact matrix form originally described in [6].

Let us denote by \( \hat{m}_c \) the user-specified maximum number of stored correction pairs \( (3 \leq \hat{m}_c) \) and by \( \tilde{m}_k = \min \{ k - 1, \hat{m}_c \} \) the current number of stored correction pairs. Then the \( n \times \tilde{m}_k \) dimensional correction matrices \( S_k \) and \( U_k \) are defined by

\[
S_k = [s_{k-\hat{m}_k} \ldots s_{k-1}] \quad \text{and} \quad U_k = [u_{k-\hat{m}_k} \ldots u_{k-1}],
\]

where the correction pairs \( s_i = y_{i+1} - x_i \) and \( u_i = \xi_{i+1} - \xi_i \), \( i < k \) and \( m \) denotes the index of the iteration after the latest serious step) are obtained at the previous iterations.

The inverse limited memory BFGS update is defined by the formula

\[
D_k = \partial_k I + [S_k \quad \partial_k U_k] \begin{bmatrix} (R_k^{-1})^T (C_k + \partial_k U_k^T U_k) R_k^{-1} & -(R_k^{-1})^T \\ 0 & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \partial_k U_k^T \end{bmatrix},
\]

where \( R_k \) is an upper triangular matrix of order \( \tilde{m}_k \) given by the form

\[
(R_k)_{ij} = \begin{cases} (s_{k-\hat{m}_k-1+i})^T (u_{k-\hat{m}_k-1+j}), & \text{if } i \leq j \\ 0, & \text{otherwise}, \end{cases}
\]

and \( \partial_k \) is a positive scaling parameter.

\( C_k \) is a diagonal matrix of order \( \tilde{m}_k \) such that

\[
C_k = \text{diag} [s_{k-\hat{m}_k}^T u_{k-\hat{m}_k}, \ldots, s_{k-1}^T u_{k-1}],
\]

and the correction matrices \( S_k \) and \( U_k \) are defined by

\[
S_k = [s_{k-\hat{m}_k} \ldots s_{k-1}] \quad \text{and} \quad U_k = [u_{k-\hat{m}_k} \ldots u_{k-1}],
\]

The similar representation for the direct limited memory BFGS update can be written by

\[
B_k = \frac{1}{\partial_k} I - \left[ \frac{1}{\partial_k} S_k \quad U_k \right] \begin{bmatrix} \frac{1}{\partial_k} S_k^T S_k & L_k \\ L_k^T & -C_k \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{\partial_k} S_k^T \\ U_k^T \end{bmatrix},
\]

where

\[
L_k = S_k^T U_k - R_k.
\]
In addition, the inverse limited memory SR1 update is defined by

\[ D_k = \vartheta_k I - (\vartheta_k U_k - S_k)(\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k - S_k)^T. \]  

(26)

and, correspondingly, the direct SR1 update is defined by

\[ B_k = \frac{1}{\vartheta_k} I + (U_k - \frac{1}{\vartheta_k} S_k)(L_k + L_k^T + C_k - \frac{1}{\vartheta_k} S_k^T S_k)^{-1}(U_k - \frac{1}{\vartheta_k} S_k)^T \]  

(27)

With SR1 updates we use the value \( \vartheta_k = 1 \) for all \( k \).

In our proposal, the individual updates that would violate positive definiteness are skipped.

7. References


