Reports of the Department of Mathematical Information Technology
Series B. Scientific Computing
No. B. 1/2006

Limited Memory Bundle Algorithm for Large Bound Constrained Nonsmooth Minimization Problems

Marjo S. Haarala Marko M. Mäkelä

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
FI–40014 University of Jyväskylä
FINLAND
fax +358 14 260 2731
http://www.mit.jyu.fi/

ISBN 951-39-2418-1
ISSN 1456-436X

Limited Memory Bundle Algorithm for Large Bound Constrained Nonsmooth Minimization Problems[*]

Marjo S. Haarala[†] Marko M. Mäkelä[‡]

Abstract

Typically, practical optimization problems involve nonsmooth functions of hundreds or thousands of variables. As a rule, the variables in such problems are restricted to certain meaningful intervals. In this paper, we propose an efficient adaptive limited memory bundle method for large-scale nonsmooth, possibly nonconvex, bound constrained optimization. The method combines the nonsmooth variable metric bundle method and the smooth limited memory variable metric method, while the constraint handling is based on the projected gradient method and the dual subspace minimization. The preliminary numerical experiments to be presented confirm the usability of the method.

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

1 Introduction

In this paper, we propose an adaptive limited memory bundle algorithm for solving large-scale nonsmooth (nondifferentiable) bound constrained optimization problems

$$\begin{cases} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u, \end{cases} \tag{1}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is supposed to be locally Lipschitz continuous and the number of variables n is supposed to be large. Moreover, the vectors $\boldsymbol{x}^l$

[*]This research was supported by the Academy of Finland, grant #104641 and the University of the Witwatersrand, Johannesburg, South Africa.

[†]School of Computational & Applied Mathematics, University of the Witwatersrand, Private Bag 3, Wits 2050, Johannesburg, South Africa, mshaarala@cam.wits.ac.za

[‡]Department of Mathematical Information Technology, University of Jyväskylä, PO Box 35 (Agora), FI-40014 University of Jyväskylä, Finland, makela@mit.jyu.fi

and $\boldsymbol{x}^u$ representing the lower and the upper bounds on the variables, respectively, are fixed and the inequalities in (1) are taken componentwise.

Nonsmooth optimization problems are in general difficult to solve, even when they are unconstrained. The direct application of smooth gradient-based methods to nonsmooth problems may lead to a failure in convergence, in optimality conditions, or in gradient approximation (see, e.g., [20]). Furthermore, derivative free methods, like genetic algorithms (see, e.g., [11]) or Powell's method (see, e.g., [9]) may be unreliable and become inefficient whenever the dimension of the problem increases. Thus, special tools for solving nonsmooth optimization problems are needed.

Nowadays different variants of bundle methods (see, e.g., [15, 17, 26, 32]) are recognized the most effective and reliable methods for nonsmooth optimization problems. Their basic assumption is that at every point $\boldsymbol{x} \in \mathbb{R}^n$, we can evaluate the value of the objective function $f(\boldsymbol{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \mathbb{R}^n$ from the subdifferential [5]

$$\partial f(\boldsymbol{x}) = \operatorname{conv}\{ \lim_{i\to\infty} \nabla f(\boldsymbol{x}_i) \mid \boldsymbol{x}_i \to \boldsymbol{x} \text{ and } \nabla f(\boldsymbol{x}_i) \text{ exists} \},$$

where "conv" denotes the convex hull of a set. The idea behind bundle methods is to approximate $\partial f(\boldsymbol{x})$ by gathering subgradient information from previous iterations into a bundle. A search direction can then be found as a solution a quadratic subproblem (see, e.g., [17, 26, 32]). By utilizing so called subgradient aggregation strategy [17], it is possible to prove the global convergence of bundle methods with a bounded number of stored subgradients.

While standard bundle methods are very efficient for small- and medium-scale problems, they are not, in general, competent in large-scale settings (see, e.g., [2, 14, 16]). In [12, 13, 14] we have proposed a limited memory bundle method for general, possibly nonconvex, nonsmooth large-scale optimization. The method combines the variable metric bundle methods [23, 33] for small- and medium-scale nonsmooth optimization with the limited memory variable metric methods (see, e.g., [4, 10, 22, 29]) for smooth large-scale optimization.

The basic limited memory bundle method [13, 14] as well as its adaptive version [12] are only suitable for unconstrained problems. However, besides nonsmoothness, real world optimization problems often involve some kind of constraints. Indeed, some problems are not even defined if their variables are not restricted into certain meaningful intervals. Typically, when using bundle methods, the problems with simple constraints (such as bound or linear constraints) are solved by including the constraints directly to the quadratic subproblem (see, e.g., [18, 19]). However, we do not solve any quadratic subproblems in the limited memory bundle method (at least not in the sense of bundle methods) and, thus, we had to sought for alternative ways for constraint handling. Our approach 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 [3]. The method to be described here differs from the original limited memory bundle method [12, 13, 14] mainly in the calculation of the search direction. Furthermore, the line search is now enforced to produce feasible points (that is, the point $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u$). On the other hand, the new method differs from the limited memory BFGS method [3] in the fact that, indeed, it is capable of handling nonsmooth objectives by utilizing null steps and aggregation of subgradients. Moreover, we wish to point out that limited memory bundle method to be presented only generates feasible points. This can be an advantage in the case the objective function or the subgradient values are undefined or difficult to compute if some of the constraints are violated.

The rest of this paper is organized as follows. In Section 2, we describe the adaptive limited memory bundle method for bound constrained optimization. We start by giving a brief introduction to the method. After that, we describe in detail the adaptive limited memory bundle algorithm, identification of the active set, and the subspace minimization procedure used. In Section 3, some preliminary results of numerical experiments are presented. The numerical results demonstrate the usability of the new method with both convex and nonconvex large-scale nonsmooth bound constrained minimization problems. Finally, in Section 4, we conclude and give some ideas of further development. A detailed description of limited memory matrix updating is given in Appendix.

2 Method

In this section, we describe the adaptive limited memory bundle method for bound constrained large-scale nonsmooth optimization. We start by giving a simple flowchart (in Figure 1) to point out the basic ideas of the algorithm. The limited memory bundle method is characterized by the usage of null steps together with the aggregation 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, a simple aggregation of subgradients guarantees the convergence of the aggregate subgradients to zero and makes it possible to evaluate a termination criterion [12, 13].

The search direction is calculated using two-stage approach. First, we define the quadratic model function q_k that approximates the objective function at the iteration point $\boldsymbol{x}_k$ by

$$q_k(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \tilde{\boldsymbol{\xi}}_k^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T B_k(\boldsymbol{x} - \boldsymbol{x}_k), \tag{2}$$

where $\tilde{\boldsymbol{\xi}}_k$ is the aggregate subgradient of the objective function and B_k is the 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 [6] and, at the same time, to identify the active set $\mathcal{I}_A = \{i \mid x_i = x_i^l \text{ or } x_i = x_i^u\}$ of the problem. The calculation of the 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 [31] and large-scale [7] problems. After the active set has been identified,

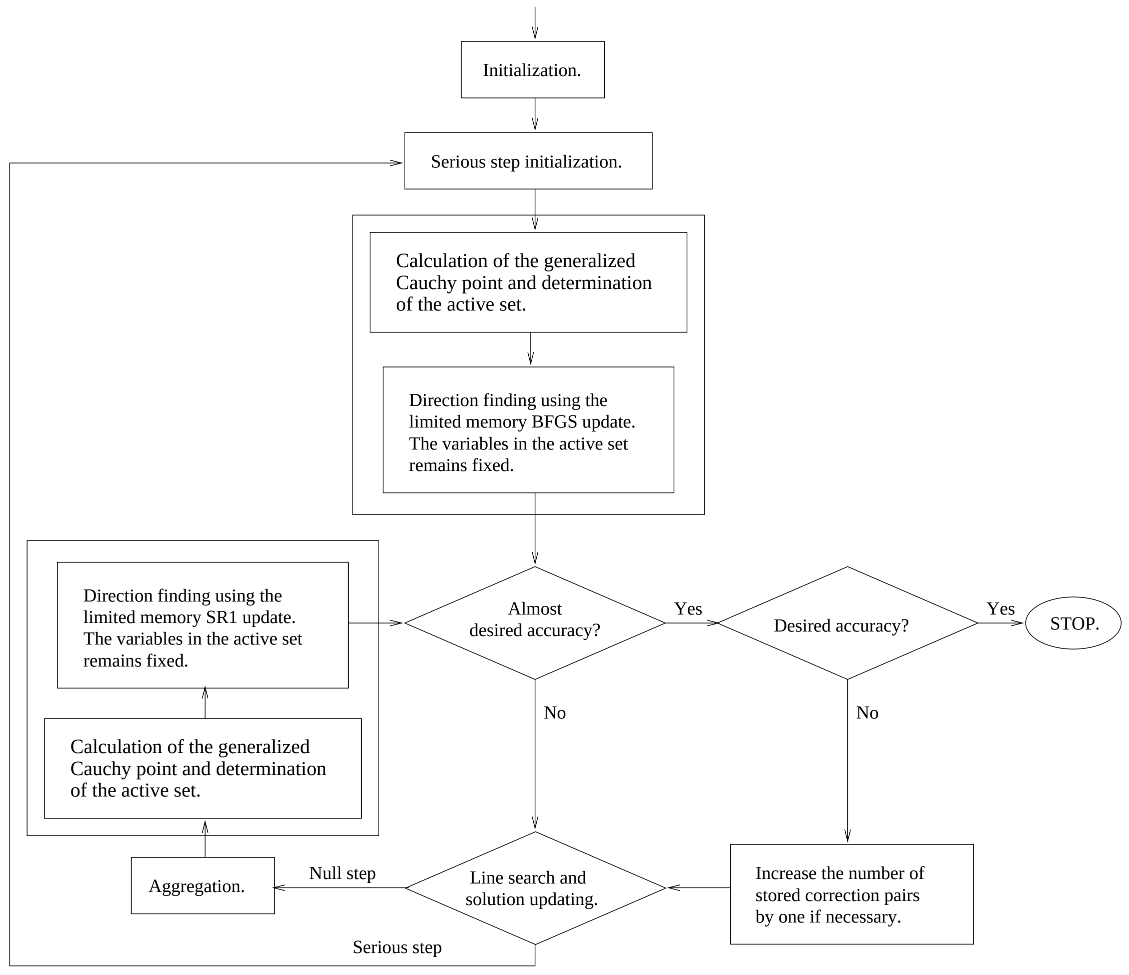

Initialization.
Serious step initialization.
Calculation of the generalized Cauchy point and determination of the active set.
Direction finding using the limited memory BFGS update. The variables in the active set remains fixed.
Direction finding using the limited memory SR1 update. The variables in the active set remains fixed.
Calculation of the generalized Cauchy point and determination of the active set.
Almost desired accuracy?
Yes
Desired accuracy?
Yes
STOP.
No
No
Aggregation.
Null step
Line search and solution updating.
Increase the number of stored correction pairs by one if necessary.
Serious step

Figure 1: Adaptive limited memory bundle method with bounds.

the quadratic model function (2) is approximately minimized with respect to free variables, in other words, the variables in the active set remain fixed throughout the process. The search direction is then defined to be the vector leading from the current iteration point $\boldsymbol{x}_k$ to this approximate minimizer. Finally, a line search that is guaranteed to produce feasible points is performed.

As already mentioned, we utilize the limited memory approach (see, e.g., [4, 10, 22, 29]) in the calculation of the search direction and the aggregate values. The idea of limited memory matrix updating is that instead of storing the large matrices B_k and D_k (we denote by D_k the update formula that is inverse of B_k), we store a certain (usually small constant) number $\hat{m}_c$ of vectors, so-called correction pairs obtained at the previous iterations of the algorithm, and we use these correction pairs to implicitly define the variable metric matrices. When the storage space available is used up, the oldest correction pairs are deleted to make room for new ones; thus, except for the first few iterations, we always have the $\hat{m}_c$ most recent correction pairs available.

In practice, the utilization of limited memory approach means that the variable metric updates are not as accurate as if we used standard variable metric updates (see, e.g., [9]). However, both the storage space required and the number of operations needed in the calculations are significantly smaller. In the adaptive limited memory bundle method [12] the number of stored correction pairs $\hat{m}_c$ may change during the computation. This means that we can start the optimization with a small $\hat{m}_c$ and when we are closer to the optimal point, $\hat{m}_c$ may be increased until some upper limit $\hat{m}_u$ is achieved. The aim of this adaptability is to improve the accuracy of the basic method without loosing much from efficiency, that is, without increasing computational costs too much.

For more details of the limited memory updating formulae refer to Appendix.

2.1 Limited memory bundle method.

In this subsection, we describe within more details the limited memory bundle algorithm for solving nonsmooth optimization problems of type (1). The algorithm to be presented generates a sequence of basic points $(\boldsymbol{x}_k) \subset S$ together with a sequence of auxiliary points $(\boldsymbol{y}_k) \subset S$, where the set $S = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u\}$ is the feasible region of problem (1). A new iteration point $\boldsymbol{x}_{k+1}$ and a new auxiliary point $\boldsymbol{y}_{k+1}$ are produced using a special line search procedure [12, 13] such that

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + t_L^k \boldsymbol{d}_k \qquad \text{and} \\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_k + t_R^k \boldsymbol{d}_k, \qquad \text{for } k \geq 1 \end{aligned}$$

with $\boldsymbol{y}_1 = \boldsymbol{x}_1$, where $t_R^k \in (0, t_{max}^k]$ and $t_L^k \in [0, t_R^k]$ are step sizes, $t_{max}^k \geq 1$ is the upper bound for the step size that assures the feasibility of produced points, and $\boldsymbol{d}_k$ is a search direction.

A necessary condition for a serious step is to have

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\boldsymbol{y}_{k+1}) \leq f(\boldsymbol{x}_k) - \varepsilon_L t_R^k w_k, \tag{3}$$

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 $\boldsymbol{x}_k$. If condition (3) is satisfied, we set $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ 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

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{4}$$

where $\varepsilon_R \in (\varepsilon_L, 1/2)$ is a line search parameter, $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, and β_{k+1} is the subgradient locality measure [21, 27] similar to bundle methods. In the case of a null step, we set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ but information about the objective function is increased because we store the auxiliary point $\boldsymbol{y}_{k+1}$ and the corresponding auxiliary subgradient $\boldsymbol{\xi}_{k+1}$.

For the direction finding, the limited memory bundle method uses the original subgradient $\boldsymbol{\xi}_k$ after the serious step and the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ after the null step. The aggregation procedure (see, Step 6 in Algorithm 2.1) is similar to that of the original variable metric bundle methods [23, 33] except that the variable metric updates are now calculated using limited memory approach.

In our algorithm 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, eqs. (16) and (17)), since these update formulae give us a possibility to preserve the boundedness and some other properties of generated matrices needed for global convergence. Otherwise, since these properties are not required after a serious step, the more efficient limited memory BFGS updates (see Appendix, eqs. (14) and (15)) are employed. The individual updates that would violate positive definiteness are skipped (for more details, see [12, 13, 14]).

We now present a model algorithm for the adaptive limited memory bundle method for solving the bound constrained optimization problems. After that, we describe how the generalized Cauchy point and the search direction can be determined. In what follows, we assume that at every feasible point $\boldsymbol{x} \in S \subset \mathbb{R}^n$ we can evaluate the value of the objective function $f(\boldsymbol{x})$ and the corresponding arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$.

ALGORITHM 2.1. *(Adaptive Limited Memory Bundle Method with Bounds.)*

Data: Choose the final accuracy tolerance $\varepsilon > 0$, the positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1/2)$, the distance measure parameter $\gamma \geq 0$ (with $\gamma = 0$ if f is convex), and the locality measure parameter $\omega \geq 1$. Select an upper limit $\hat{m}_u \geq 3$ for the number of stored correction pairs and the size of the bundle $m_\xi \geq 2$.

Step 0: (*Initialization.*) Choose a (feasible) starting point $\boldsymbol{x}_1 \in S \subset \mathbb{R}^n$ and set the initial matrices $D_1 = B_1 = I$. Choose an initial maximum number of stored correction pairs $\hat{m}_c$ ($3 \leq \hat{m}_c \leq \hat{m}_u$). Set $\boldsymbol{y}_1 = \boldsymbol{x}_1$ and $\beta_1 = 0$. Compute $f_1 = f(\boldsymbol{x}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{x}_1)$. Set the iteration counter $k = 1$.

Step 1: (*Serious step initialization.*) Set the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ and the aggregate subgradient locality measure $\tilde{\beta}_k = 0$. Set an index for the serious step $m = k$.

Step 2: (*Generalized Cauchy point.*) Compute the generalized Cauchy point and determine the active set by Algorithm 2.2 using the limited memory BFGS update formula (14) and (15) if $m = k$ and using the limited memory SR1 update formula (16) and (17), otherwise.

Step 3: (*Direction finding.*) Compute the search direction $\boldsymbol{d}_k$ by Algorithm 2.3 using the same update formula as in Step 2.

Step 4: (*Stopping criterion.*) Calculate

$$w_k = -\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d}_k + 2\tilde{\beta}_k. \tag{5}$$

If $w_k \le \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution. Otherwise, if $w_k \le 10^3\varepsilon$ and $\hat{m}_c < \hat{m}_u$, set $\hat{m}_c = \hat{m}_c + 1$.

Step 5: (*Line search.*) Determine the maximum step size t^k_{max} such that $\boldsymbol{x}_k + t^k_{max}\boldsymbol{d}_k$ is feasible. Determine the step sizes $t^k_R \in (0, t^k_{max}]$ and $t^k_L \in [0, t^k_R]$ to take either a serious step or a null step (that is, check whether (3) or (4) is valid). Set the corresponding values

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + t^k_L \boldsymbol{d}_k,\\
\boldsymbol{y}_{k+1} &= \boldsymbol{x}_k + t^k_R \boldsymbol{d}_k,\\
f_{k+1} &= f(\boldsymbol{x}_{k+1}),\\
\boldsymbol{\xi}_{k+1} &\in \partial f(\boldsymbol{y}_{k+1}).
\end{aligned}$$

Set $\boldsymbol{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$ and $\boldsymbol{s}_k = \boldsymbol{y}_{k+1} - \boldsymbol{x}_k = t^k_R \boldsymbol{d}_k$. If $t^k_L > 0$ (serious step), set $\beta_{k+1} = 0$, $k = k+1$, and go to Step 1. Otherwise, calculate the locality measure

$$\beta_{k+1} = \max\{|f(\boldsymbol{x}_k) - f(\boldsymbol{y}_{k+1}) + \boldsymbol{s}_k^T \boldsymbol{\xi}_{k+1})|, \gamma\|\boldsymbol{s}_k\|^{\omega}\}.$$

Step 6: (*Aggregation.*) Determine multipliers λ^k_i satisfying $\lambda^k_i \ge 0$ for all $i \in \{1,2,3\}$, and $\sum_{i=1}^3 \lambda^k_i = 1$ that minimize the function

$$\begin{aligned}
\varphi(\lambda_1, \lambda_2, \lambda_3) = \ & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)\\
& + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k),
\end{aligned}$$

where D_k is calculated by the same updating formula as in Steps 2 and 3.

Set

$$\begin{aligned}
\tilde{\boldsymbol{\xi}}_{k+1} &= \lambda^k_1 \boldsymbol{\xi}_m + \lambda^k_2 \boldsymbol{\xi}_{k+1} + \lambda^k_3 \tilde{\boldsymbol{\xi}}_k \qquad \text{and}\\
\tilde{\beta}_{k+1} &= \lambda^k_2 \beta_{k+1} + \lambda^k_3 \tilde{\beta}_k.
\end{aligned}$$

Set $k = k+1$ and go to Step 2.

To ensure the global convergence of the method, we assume that matrices D_k are 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 requires some modifications to the model algorithm, for instance, corrections of matrices D_k when necessary. In this way we obtain more complicated algorithm which, in unconstrained case, is described in detail in [12, 13]. The basic assumption for bundle method to converge, that is, after a null step we have $\boldsymbol{z}^T D_{k+1} \boldsymbol{z} \le \boldsymbol{z}^T D_k \boldsymbol{z}$ for all $\boldsymbol{z} \in \mathbb{R}^n$, is guaranteed by the special limited memory SR1 update [12, 13].

2.2 Generalized Cauchy Point

In this subsection, we show how to calculate the generalized Cauchy point and, at the same time, how to identify the active set of problem (1). In principle, the procedure used here is the same as that in [3]. 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.

Let us first define the projection operator $\mathcal{P}[\cdot]$ (componentwise) by

$$\mathcal{P}[\boldsymbol{x}, \boldsymbol{x}^l, \boldsymbol{x}^u]_i = \begin{cases} x_i^l, & \text{if } x_i < x_i^l \\ x_i, & \text{if } x_i \in [x_i^l, x_i^u] \\ x_i^u, & \text{if } x_i > x_i^u. \end{cases}$$

This operator projects the point $\boldsymbol{x}$ into the feasible region S defined by bounds $\boldsymbol{x}^l$ and $\boldsymbol{x}^u$.

The generalized Cauchy point at iteration k is defined as the first local minimizer of the univariate piecewise quadratic function

$$\hat{q}_k(t) = q_k(\mathcal{P}[\boldsymbol{x}_k - t\tilde{\boldsymbol{\xi}}_k, \boldsymbol{x}^l, \boldsymbol{x}^u]),$$

(with q_k defined in (2)) along the projected gradient direction $\mathcal{P}[\boldsymbol{x}_k - t\tilde{\boldsymbol{\xi}}_k, \boldsymbol{x}^l, \boldsymbol{x}^u] - \boldsymbol{x}_k$ (see, e.g., [6]). That is, if we denote by t_k^{cp} the value of t corresponding to the first local minimum of $\hat{q}_k(t)$, the generalized Cauchy point is given by

$$\boldsymbol{x}_k^{cp} = \mathcal{P}[\boldsymbol{x}_k - t_k^{cp}\tilde{\boldsymbol{\xi}}_k, \boldsymbol{x}^l, \boldsymbol{x}^u].$$

The variables whose values at $\boldsymbol{x}_k^{cp}$ are at lower or upper bound, comprise the active set $\mathcal{I}_A$.

In practice, we first compute the values

$$t_i = \begin{cases} (x_{k,i} - x_i^l)/\tilde{\xi}_{k,i}, & \text{if } \tilde{\xi}_{k,i} > 0 \\ (x_{k,i} - x_i^u)/\tilde{\xi}_{k,i}, & \text{if } \tilde{\xi}_{k,i} < 0 \\ \infty, & \text{otherwise} \end{cases} \tag{6}$$

for all $i = 1, \dots, n$ to define the breakpoints in each coordinate direction. Here, we have denoted by $x_{k,i}$ and $\tilde{\xi}_{k,i}$ the ith components of vectors $\boldsymbol{x}_k$ and $\tilde{\boldsymbol{\xi}}_k$, respectively. We then sort these values t_i in increasing order to obtain the ordered set $\{t^j \mid t^j \leq t^{j+1}, j = 1, \dots, n\}$. For finding the generalized Cauchy point we search trough the intervals $[t^j, t^{j+1}]$ in order of increasing j until the one containing $\boldsymbol{x}_k^{cp}$ is located. Thus, we investigate the behavior of the quadratic function (2) for points lying on the piecewise linear path

$$x_{k,i}(t) = \begin{cases} x_{k,i} - t\tilde{\xi}_{k,i}, & \text{if } t \leq t_i \\ x_{k,i} - t_i\tilde{\xi}_{k,i}, & \text{otherwise.} \end{cases} \tag{7}$$

Let us define the jth breakpoint by $\boldsymbol{x}^j = \boldsymbol{x}_k(t^j)$. We can now express (7) in the interval $[t^j, t^{j+1}]$ as

$$\boldsymbol{x}_k(t) = \boldsymbol{x}^j + \Delta t \hat{\boldsymbol{d}}^j, \tag{8}$$

where $\Delta t = t - t^j$ and

$$\hat{d}_i^j = \begin{cases} -\tilde{\xi}_{k,i}, & \text{if } t^j \le t_i \\ 0, & \text{otherwise.} \end{cases}$$

Now, defining

$$\begin{aligned} f_j &= f(\boldsymbol{x}_k) + \tilde{\boldsymbol{\xi}}_k^T \boldsymbol{z}^j + \frac{1}{2} \boldsymbol{z}^{jT} B_k \boldsymbol{z}^j, \\ f_j' &= \tilde{\boldsymbol{\xi}}_k^T \hat{\boldsymbol{d}}^j + \hat{\boldsymbol{d}}^{jT} B_k \boldsymbol{z}^j, \\ f_j'' &= \hat{\boldsymbol{d}}^{jT} B_k \hat{\boldsymbol{d}}^j, \end{aligned}$$

where $\boldsymbol{z} = \boldsymbol{x}^j - \boldsymbol{x}_k$, and combining (2) and (8), we obtain

$$q_k(\boldsymbol{x}_k(t)) = f_j + \Delta t f_j' + \frac{1}{2} \Delta t^2 f_j''.$$

By calculating the derivative of $q_k(\boldsymbol{x}_k(t))$ and setting it to zero, we obtain $t = t^j - f_j'/f_j''$ (note that $f_j'' \neq 0$ since in our algorithm B_k is positive definite and $\hat{\boldsymbol{d}}^j \neq \boldsymbol{0}$). Thus, due to positive definiteness of matrices B_k used in our algorithm, the generalized Cauchy point lies at $\boldsymbol{x}_k(t^j - f_j'/f_j'')$ if the point $t^j - f_j'/f_j''$ lies in the interval $[t^j, t^{j+1})$. Otherwise, the generalized Cauchy point lies at $\boldsymbol{x}_k(t^j)$ if we have $f_j' \ge 0$, and it lies at or beyond $\boldsymbol{x}_k(t^{j+1})$ in all the other cases.

We now give an algorithm for calculation of the generalized Cauchy point and determination of the active set $\mathcal{I}_A$. To simplify the notation, we, for a while, omit the iteration index k and use subscripts i and b to denote the ith and the bth component of a vector. Moreover, we denote by $\boldsymbol{e}_b$ the bth column of the identity matrix.

ALGORITHM 2.2. *(Generalized Cauchy Point.)*

Data: Suppose we have available the current (feasible) iteration point $\boldsymbol{x}$, the lower and the upper bounds $\boldsymbol{x}^l$ and $\boldsymbol{x}^u$ for $\boldsymbol{x}$, the current aggregate subgradient $\tilde{\boldsymbol{\xi}}$, and the limited memory representation of matrix B (either the BFGS or the SR1 formulation).

Step 0: (*Initialization.*) Compute the breakpoints t_i in each coordinate direction by (6) and define the direction

$$\hat{d}_i = \begin{cases} 0, & \text{if } t_i = 0 \\ -\tilde{\xi}_i, & \text{otherwise.} \end{cases}$$

Initialize

$$\begin{aligned}
&\boldsymbol{x}^{cp} = \boldsymbol{x}, \\
&\mathcal{I}_F = \{i \mid t_i > 0\} \text{ (set of indices corresponding to the free variables)}, \\
&\mathcal{I}_A = \{i \mid t_i = 0\} \text{ (set of indices corresponding to the active bounds)}, \\
&t = \min\{t_i \mid i \in \mathcal{I}_F\}, \\
&t_{old} = 0, \\
&\Delta t = t - t_{old} = t, \quad \text{and} \\
&b = i \text{ such that } t_i = t. \text{ Shift } b \text{ from } \mathcal{I}_F \text{ to } \mathcal{I}_A.
\end{aligned}$$

Step 1: (*Examining the first interval.*) Calculate

$$\begin{aligned}
&f' = \tilde{\boldsymbol{\xi}}^T \hat{\boldsymbol{d}} = -\hat{\boldsymbol{d}}^T \hat{\boldsymbol{d}}, \\
&f'' = \hat{\boldsymbol{d}}^T B \hat{\boldsymbol{d}}, \qquad \text{and} \\
&\Delta t_{min} = -f'/f''.
\end{aligned}$$

If $\Delta t_{min} < \Delta t$ go to Step 4.

Step 2: (*Subsequent segments.*) Set

$$x_b^{cp} = \begin{cases} x_b^u, & \text{if } \hat{d}_b > 0 \\ x_b^l, & \text{if } \hat{d}_b < 0. \end{cases}$$

Calculate

$$\begin{aligned}
&z_b = x_b^{cp} - x_b, \\
&f' = f' + \Delta t f'' + \tilde{\xi}_b^2 + \tilde{\xi}_b \boldsymbol{e}_b^T B \boldsymbol{z}, \quad \text{and} \\
&f'' = f'' + 2\tilde{\xi}_b \boldsymbol{e}_b^T B \hat{\boldsymbol{d}} + \tilde{\xi}_b^2 \boldsymbol{e}_b^T B \boldsymbol{e}_b.
\end{aligned}$$

Set

$$\begin{aligned}
&\hat{d}_b = 0, \\
&\Delta t_{min} = -f'/f'', \\
&t_{old} = t, \\
&t = \min\{t_i \mid i \in \mathcal{I}_F\} \text{ (using the heapsort algorithm [1])} \\
&\Delta t = t - t_{old}, \text{ and} \\
&b = i \text{ such that } t_i = t. \text{ Shift } b \text{ from } \mathcal{I}_F \text{ to } \mathcal{I}_A.
\end{aligned}$$

Step 3: (*Loop.*) If $\Delta t_{min} \geq \Delta t$ go to Step 2.

Step 4: (*Generalized Cauchy point.*) Set

$$\begin{aligned}&\Delta t_{min} = \max\{\Delta t_{min}, 0\},\\&t_{old} = t_{old} + \Delta t_{min},\\&x_i^{cp} = x_i + t_{old}\hat{d}_i \text{ for all } i \text{ such that } t_i \ge t.\end{aligned}$$

For all $i \in \mathcal{I}_F$ such that $t_i = t$, shift i from $\mathcal{I}_F$ to $\mathcal{I}_A$.

The only expensive computations in Algorithm 2.2 are

$$\hat{\boldsymbol{d}}^T B\hat{\boldsymbol{d}}, \qquad \boldsymbol{e}_b^T B\boldsymbol{z}, \qquad \boldsymbol{e}_b^T B\hat{\boldsymbol{d}}, \qquad \text{and} \qquad \boldsymbol{e}_b^T B\boldsymbol{e}_b$$

at Steps 1 and 2. However, these calculations can be done very efficiently (within $O(n)$ operations) by using limited memory approach. We do not give any details of these calculations here since, essentially, for the limited memory BFGS update, these calculations proceeds similar to those given in [3] and, for the limited memory SR1 update, the idea should be perspicuous as well.

2.3 Direction finding

When the generalized Cauchy point has been found, we approximately minimize the quadratic model function (2) over the space of free variables. The subspace minimization procedure used in our approach is in principal the same as the dual space method in [3] 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 (see Algorithm 2.1).

We solve $\boldsymbol{d}$ from smooth quadratic problem

$$\begin{cases}\text{minimize} & \tilde{\boldsymbol{\xi}}_k^{\,T}\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T B_k\boldsymbol{d}\\ \text{such that} & A_k^T\boldsymbol{d} = \boldsymbol{b}_k \qquad \text{and}\\ & \boldsymbol{x}^l \le \boldsymbol{x}_k + \boldsymbol{d} \le \boldsymbol{x}^u,\end{cases} \tag{9}$$

where A_k is the matrix of active constraints gradients at $\boldsymbol{x}_k^{cp}$ and $\boldsymbol{b}_k = A_k^T(\boldsymbol{x}_k^{cp} - \boldsymbol{x}_k)$. Note that A_k consists of n_A unit vectors (here n_A is the number of elements in the active set $\mathcal{I}_A$) and $A_k^T A_k$ is equal to identity.

We first ignore the bound constraints. The first order optimality conditions for problem (9) without bounds are

$$\tilde{\boldsymbol{\xi}}_k + B_k\boldsymbol{d}^* + A_k\boldsymbol{\mu}^* = \boldsymbol{0} \tag{10}$$

$$A_k^T\boldsymbol{d}^* = \boldsymbol{b}_k. \tag{11}$$

Now, by multiplying (10) by $A_k^T D_k$, where, as before, $D_k = B_k^{-1}$, and by using (11), we obtain

$$(A_k^T D_k A_k)\boldsymbol{\mu}^* = -A_k^T D_k\tilde{\boldsymbol{\xi}}_k - \boldsymbol{b}_k, \tag{12}$$

which determines Lagrange multipliers $\boldsymbol{\mu}^*$. The linear system (12) can be solved by utilizing the Sherman-Morrison-Woodbury formula and the compact representation of limited memory matrices (see [3]). Thus, $\boldsymbol{d}^*$ can be given by

$$B_k \boldsymbol{d}^* = -A_k \boldsymbol{\mu}^* - \tilde{\boldsymbol{\xi}}_k. \tag{13}$$

If there are no active bounds present, we simply obtain $\boldsymbol{d}^* = -D_k \tilde{\boldsymbol{\xi}}_k$, which is the formula used also in the original unconstrained version of the limited memory bundle method [12, 14]. In the case the vector $\boldsymbol{x}_k + \boldsymbol{d}^*$ violates the bounds in (9), we, similarly to [3], backtrack along the line joining the infeasible point $\boldsymbol{x}_k + \boldsymbol{d}^*$ and the generalized Cauchy point $\boldsymbol{x}_k^{cp}$ to regain the feasible region.

We now give an efficient algorithm for direction finding in bound constrained case. For more details of the calculations using the limited memory approach, see [3].

ALGORITHM 2.3. *(Direction Finding.)*

Data: Suppose that we have available the current (feasible) iteration point $\boldsymbol{x}_k$, the Cauchy point $\boldsymbol{x}_k^{cp}$, the number of active variables n_A at $\boldsymbol{x}_k^{cp}$, the $n \times n_A$ matrix A_k of the active constraints gradients at $\boldsymbol{x}_k^{cp}$, the limited memory representation of matrix D_k (either the BFGS or the SR1 formulation), and the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$.

Step 1: (*No active bounds.*) If $n_A = 0$, compute

$$\boldsymbol{d}^* = -D_k \tilde{\boldsymbol{\xi}}_k$$

and go to Step 4.

Step 2: (*Lagrange multipliers.*) Compute the intermediate n_A-vector

$$\boldsymbol{p} = -A_k^T D_k \tilde{\boldsymbol{\xi}}_k - \boldsymbol{b}_k,$$

where $\boldsymbol{b}_k = A_k^T(\boldsymbol{x}_k^{cp} - \boldsymbol{x}_k)$. Calculate the n_A-vector of Lagrange multipliers

$$\boldsymbol{\mu}^* = (A_k^T D_k A_k)^{-1} \boldsymbol{p}.$$

Step 3: (*Search direction.*) Compute

$$\boldsymbol{d}^* = -D_k(A_k \boldsymbol{\mu}^* + \tilde{\boldsymbol{\xi}}_k).$$

Step 4: (*Backtrack.*) Compute

$$\alpha^* = \min\left\{1, \max\{\alpha \mid x_i^l \le x_{k,i}^{cp} + \alpha(x_{k,i} + d_i^* - x_{k,i}^{cp}) \le x_i^u,\ i \in \mathcal{I}_F\}\right\}.$$

Set $\bar{\boldsymbol{x}} = \boldsymbol{x}_k^{cp} + \alpha^*(\boldsymbol{x}_k + \boldsymbol{d}^* - \boldsymbol{x}_k^{cp})$ and $\boldsymbol{d}_k = \bar{\boldsymbol{x}} - \boldsymbol{x}_k$.

Note that since the columns of A_k are unit vectors, the operations between A_k and a vector amount to select the appropriate elements from the vector and change of sign if necessary. Hence, and due to usage of limited memory approach, the search direction can be calculated within $O(n)$ operations.

3 Numerical Experiments

In this section we compare the proposed limited memory bundle method (`LMBM-B`) to the proximal bundle method (`PBNCGC` [24, 26]) in a limited number of academic large-scale test problems and one practical application. We use the solver `PBNCGC` as a benchmark since the proximal bundle method is the most frequently used bundle method in nonsmooth optimization. A more extensive numerical analysis concerning the performance of the unconstrained version of the limited memory bundle method and its comparison with some other existing bundle methods in large-scale minimization problems can be found in [14].

The academic test experiments were performed in a Intel® Pentium® 4 CPU 3.20GHz. For the practical application the test runs were performed on an HP9000/ J5600 workstation $2\times$PA8600 CPU 552MHz because of the libraries required. Both the algorithms were implemented in Fortran77 with double-precision arithmetic.

The solvers were first tested with 10 nonsmooth academic minimization problems described in [14]. The problems in [14] are unconstrained but we inclosed the additional bounds

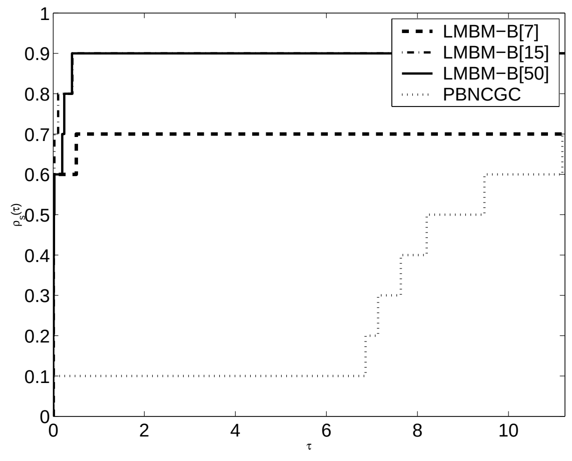
$$x_i^* + 0.1 \le x_i \le x_i^* + 1.1 \quad \text{for all odd } i,$$

where $\boldsymbol{x}^*$ denotes the solution for the unconstrained problem. If the original starting point given in [14] was not feasible, we simply projected it to the feasible region.

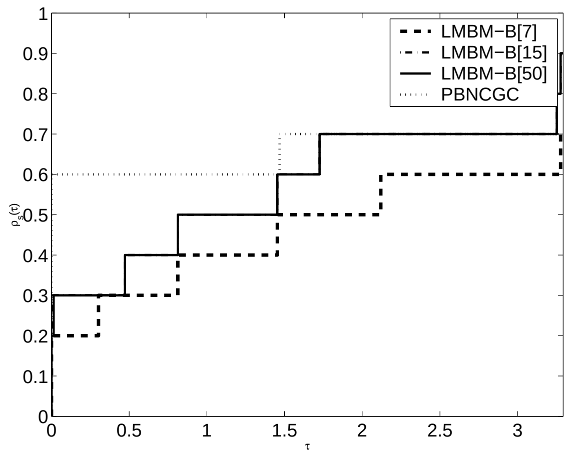
The number of variables used in our academic experiment was 1000, and the solvers were tested with relatively small amount of stored subgradient information. That is, the size of the bundle m_ξ was set to 10 for `LMBM-B` and to 100 for `PBNCGC` (since the previous experiments [12, 13] have shown that a larger bundle usually works better with `PBNCGC`). We tested `LMBM-B` with different upper limits for the stored correction pairs, that is, $\hat{m}_u = 7$, $\hat{m}_u = 15$, and $\hat{m}_u = 50$. In all cases the initial maximum number of stored corrections pairs $\hat{m}_c$ was set to 7. In what follows, we denote these different variants by `LMBM-B[7]`, `LMBM-B[15]`, and `LMBM-B[50]`, respectively. For convex problems (problems 1 – 5 in [14]), we used the distance measure parameter $\gamma = 0$ and for nonconvex problems (problems 6 – 10 in [14]), we used the value $\gamma = 0.5$ with both `LMBM-B` and `PBNCGC`. The final accuracy parameter $\varepsilon = 10^{-5}$ was used in all the cases. 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. In these cases, the results were accepted if they were less than two significant digits greater than the desired accuracy of the solution.

The results of the academic experiments are summarized in Figure 2. The results are analyzed using the performance profiles introduced in [8]. As performance measures, we use computation times (in Figure 2(a)) and numbers of function evaluations (in Figure 2(b)). In the performance profiles the value of $\rho(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the corresponding solver is the best and the value of $\rho(\tau)$ at the rightmost abscissa gives the percentage of test problems that



(a) CPU time



(b) Function evaluations

Figure 2: Nonsmooth bound constrained problems with 1000 variables.

the corresponding solver can solve (this does not depend on the measured performance). Moreover, the relative efficiency of each solver can be directly seen from the performance profiles: the higher is the particular curve, the better is the corresponding solver. For more information of performance profiles, see [8].

In Figure 2(a) we see the superiority of the different variants of the limited memory bundle solver when comparing the computational times; the computation time elapsed with any of these variants was on the average about 100 times shorter than that of PBNCGC and that only if we, for each solver, removed the most time consuming problem from the test set. Otherwise, the differences between computational times of PBNCGC and different variants of LMBM-B were even greater. On the other hand, there was not a big difference in the computational times between the different variants of LMBM-B (see Figure 2(a)). Although, quite surprisingly, LMBM-B[7] usually needed slightly more computation time than the other two. This is due to fewer function evaluations required with LMBM-B[15] and LMBM-B[50] (see Figure 2(b)). Note that, with all the problems the numbers of function evaluations used with LMBM-B[15] were exactly the same as those used with LMBM-B[50] and, thus, they can not be directly seen in Figure 2(b). The proximal bundle solver PBNCGC usually needed less function evaluations than the different variants of LMBM-B (see Figure 2(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.

All the variants of LMBM-B failed to solve one of the problems (problem 2 in [14]). This failure was quite predictable, since the problem is reported to be difficult to solve with limited memory bundle method even without the bound constraints [14]. In addition, with solver LMBM-B[7] there were some difficulties to reach the desired accuracy in two problems (problems 1 and 8 in [14]). Both these problems were solved successfully when the upper limit for the number of stored correction pairs was increased. Also the proximal bundle solver PBNCGC failed to solve three of the problems in the test set (problems 1, 6, and 10 in [14]).

In addition to the academic test problems, we tested LMBM-B and PBNCGC with a hemivariational inequality problem [25] that is a nonsmooth nonconvex real-world large-scale bound constrained optimization problem. Hemivariational inequalities can be considered as generalizations of variational inequalities. The origin of these kind of problems is in nonsmooth mechanics of solids, especially in nonmonotone contact problems. Typically, hemivariational inequalities can be formulated as substationary point problems of the corresponding nonsmooth nonconvex energy functionals. For more details of the mathematical theory an applications of hemivariational inequalities in general, see [28, 30] and of the formulation of this particular problem, see [25].

Similarly to previous experiments, the solvers were tested with $m_\xi = 10$ for LMBM-B[7], LMBM-B[15], and LMBM-B[50] and with $m_\xi = 100$ for PBNCGC. In all cases the value of the distance measure parameter γ was set to 0.5 (since the objective function is nonconvex). Otherwise, the parameter values similar to academic problems were used. The number of variables in the problem was equal to 840.

The results of the experiment are given in Table 1, where Ni and Nf denote the numbers of iterations and function evaluations used, respectively, f denotes the value of the objective function at termination, and CPU time is given in seconds.

Table 1: Results for hemivariational inequalities.

Solver	Ni/Nf	f	CPU
PBNCGC	694/1454	-0.01240639	48.23
LMBM-B[7]	121/126	-0.01249884	1.49
LMBM-B[15]	118/124	-0.01242044	3.04
LMBM-B[50]	121/128	-0.01248906	5.66

Again, the different variants of LMBM-B were superior when comparing the computation times. Moreover, in this problem also the numbers of function evaluations used with the limited memory bundle solvers were significantly smaller than those used with the proximal bundle solver PBNCGC (see Table 1).

4 Conclusions

In this paper, we have described a new limited memory bundle method for bound constrained nonsmooth large-scale optimization. The preliminary numerical experiments confirm that the limited memory bundle solver 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 other solver tested.

Although the new method appears to be useful already, there are some further ideas to study in order to make the method more efficient and applicative. The future development of limited memory bundle method includes the study of alternative ways of constraint handling (with more complicated constraints), for instance, those based on interior point methods.

Acknowledgements

The authors would like to thank Prof. Kaisa Miettinen for her support.

Appendix

Limited memory matrices The limited memory variable metric matrices used in our algorithm are represented in the compact matrix form originally described in [4].

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

$$\begin{aligned} S_k &= \begin{bmatrix} \boldsymbol{s}_{k-\hat{m}_k} & \dots & \boldsymbol{s}_{k-1} \end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix} \boldsymbol{u}_{k-\hat{m}_k} & \dots & \boldsymbol{u}_{k-1} \end{bmatrix}. \end{aligned}$$

The inverse limited memory BFGS update is defined by the formula

$$D_k = \vartheta_k I + Q_k N_k Q_k^T, \tag{14}$$

where ϑ_k is a positive scaling parameter,

$$\begin{aligned} Q_k &= \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix}, \qquad \text{and} \\ N_k &= \begin{bmatrix} (R_k^{-1})^T (C_k + \vartheta_k U_k^T U_k) R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix}. \end{aligned}$$

Here, on the other hand, R_k and C_k are matrices of order $\hat{m}_k$ given by the form

$$\begin{aligned} (R_k)_{ij} &= \begin{cases} (\boldsymbol{s}_{k-\hat{m}_k-1+i})^T (\boldsymbol{u}_{k-\hat{m}_k-1+j}), & \text{if } i \leq j \\ 0, & \text{otherwise,} \end{cases} \\ C_k &= \operatorname{diag}\,[\boldsymbol{s}_{k-\hat{m}_k}^T \boldsymbol{u}_{k-\hat{m}_k}, \dots, \boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1}]. \end{aligned}$$

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

$$B_k = \frac{1}{\vartheta_k} I - \bar{Q}_k \bar{N}_k \bar{Q}_k^T, \tag{15}$$

where

$$\begin{aligned}
\bar{Q}_k &= \begin{bmatrix} U_k & \frac{1}{\vartheta_k} S_k \end{bmatrix}, \\
\bar{N}_k &= \begin{bmatrix} -C_k & L_k^T \\ L_k & \frac{1}{\vartheta} S_k^T S_k \end{bmatrix}^{-1} \\
&= \begin{bmatrix} -C_k^{1/2} & C_k^{-1/2} L_k^T \\ 0 & J_k^T \end{bmatrix}^{-1} \begin{bmatrix} C_k^{1/2} & 0 \\ -L_k C_k^{-1/2} & J_k \end{bmatrix}^{-1}, \\
(L_k)_{ij} &= \begin{cases} (\boldsymbol{s}_{k-\hat{m}_k-1+i})^T (\boldsymbol{u}_{k-\hat{m}_k-1+j}), & \text{if } i > j \\ 0, & \text{otherwise,} \end{cases}
\end{aligned}$$

and

$$J_k J_k^T = \frac{1}{\vartheta} S_k^T S_k + L_k C_k^{-1} L_k^T.$$

Note that here we have

$$L_k = S_k^T U_k - R_k.$$

In addition, the inverse limited memory SR1 update is defined by

$$D_k = \vartheta_k I - W_k M_k^{-1} W_k^T, \qquad (16)$$

where

$$\begin{aligned}
W_k &= \vartheta_k U_k - S_k \qquad \text{and} \\
M_k &= \vartheta_k U_k^T U_k - R_k - R_k^T + C_k,
\end{aligned}$$

and, correspondingly, the direct SR1 update is defined by

$$B_k = \frac{1}{\vartheta_k} I + \bar{W}_k \bar{M}_k^{-1} \bar{W}_k^T, \qquad (17)$$

where

$$\begin{aligned}
\bar{W}_k &= U_k - \frac{1}{\vartheta_k} S_k \qquad \text{and} \\
\bar{M}_k &= L_k + L_k^T + C_k - \frac{1}{\vartheta_k} S_k^T S_k.
\end{aligned}$$

References

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1974.

[2] BEN-TAL, A., AND NEMIROVSKI, A. Non-Euclidean restricted memory level method for large-scale convex optimization. Available in web page <URL: `http://iew3.technion.ac.il/Labs/Opt/index.php?4`>, 2004. (September 9th, 2004).

[3] BYRD, R. H., LU, P., NOCEDAL, J., AND ZHU, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing 16*, 5 (1995), 1190–1208.

[4] BYRD, R. H., NOCEDAL, J., AND SCHNABEL, R. B. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming 63* (1994), 129–156.

[5] CLARKE, F. H. *Optimization and Nonsmooth Analysis*. Wiley-Interscience, New York, 1983.

[6] CONN, A. R., GOULD, N. I. M., AND TOINT, P. L. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis 25*, 2 (1988), 433–460.

[7] CONN, A. R., GOULD, N. I. M., AND TOINT, P. L. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation 50*, 182 (1988), 399–430.

[8] DOLAN, E. D., AND MORÉ, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming 91* (2002), 201–213.

[9] FLETCHER, R. *Practical Methods of Optimization*, 2nd ed. John Wiley and Sons, Chichester, 1987.

[10] GILBERT, J.-C., AND LEMARÉCHAL, C. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming 45* (1989), 407–435.

[11] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1998.

[12] HAARALA, M. *Large-Scale Nonsmooth Optimization: Variable Metric Bundle Method with Limited Memory*. PhD thesis, University of Jyväskylä, Department of Mathematical Information Technology, 2004.

[13] HAARALA, M., MIETTINEN, K., AND MÄKELÄ, M. M. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. Conditionally accepted for publication in *Mathematical Programming A*.

[14] HAARALA, M., MIETTINEN, K., AND MÄKELÄ, M. M. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software 19*, 6 (2004), 673–692.

[15] Hiriart-Urruty, J.-B., and Lemaréchal, C. *Convex Analysis and Minimization Algorithms II*. Springer-Verlag, Berlin, 1993.

[16] Kärkkäinen, T., Majava, K., and Mäkelä, M. M. Comparison of formulations and solution methods for image restoration problems. *Inverse Problems 17*, 6 (2001), 1977–1995.

[17] Kiwiel, K. C. *Methods of Descent for Nondifferentiable Optimization*. Lecture Notes in Mathematics 1133. Springer-Verlag, Berlin, 1985.

[18] Kiwiel, K. C. A method of linearizations for linearly constrained nonconvex nonsmooth minimization. *Mathematical Programming 34* (1986), 175–187.

[19] Kiwiel, K. C. A constraint linearization method for nondifferentiable convex minimization. *Numeriche Mathematik 51* (1987), 395–414.

[20] Lemaréchal, C. Nondifferentiable optimization. In *Optimization*, G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, Eds. Elsevier North-Holland, Inc., New York, 1989, pp. 529–572.

[21] Lemaréchal, C., Strodiot, J.-J., and Bihain, A. On a bundle algorithm for nonsmooth optimization. In *Nonlinear Programming*, O. L. Mangasarian, R. R. Mayer, and S. M. Robinson, Eds. Academic Press, New York, 1981, pp. 285–281.

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

[23] Lukšan, L., and Vlček, J. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications 102* (1999), 593–613.

[24] Mäkelä, M. M. Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0. Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B 13/2003 University of Jyväskylä, Jyväskylä, 2003.

[25] Mäkelä, M. M., Miettinen, M., Lukšan, L., and Vlček, J. Comparing nonsmooth nonconvex bundle methods in solving hemivariational inequalities. *Journal of Global Optimization 14* (1999), 117–135.

[26] Mäkelä, M. M., and Neittaanmäki, P. *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control*. World Scientific Publishing Co., Singapore, 1992.

[27] Mifflin, R. A modification and an extension of Lemaréchal's algorithm for nonsmooth minimization. *Matematical Programming Study 17* (1982), 77–90.

[28] Naniewicz, Z., and Panagiotopoulos, P. D. *Mathematical Theory of Hemivariational Inequalities and Applications*. Marcel Dekker, New York, 1995.

[29] NOCEDAL, J. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation 35*, 151 (1980), 773–782.

[30] PANAGIOTOPOULOS, P. D. *Hemivariational Inequalities*. Springer-Verlag, New York, 1993.

[31] PANIER, E. R. An active set method for solving linearly constrained nonsmooth optimization problems. *Mathematical Programming 37* (1987), 269–292.

[32] SCHRAMM, H., AND ZOWE, J. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM Journal on Optimization 2*, 1 (1992), 121–152.

[33] VLČEK, J., AND LUKŠAN, L. Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. *Journal of Optimization Theory and Applications 111*, 2 (2001), 407–430.