NEW LIMITED MEMORY BUNDLE METHOD FOR LARGE-SCALE NONSMOOTH OPTIMIZATION

M. Haarala[1] K. Miettinen[2] M. M. Mäkelä[3]

[1,3]*Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland.*

[2]*Helsinki School of Economics, P.O. Box 1210, FIN-00101 Helsinki, Finland.*

Many practical optimization problems involve nonsmooth (that is, not necessarily differentiable) functions of hundreds or thousands of variables. In such problems, the direct application of smooth gradient-based methods may lead to a failure due to the nonsmooth nature of the problem. On the other hand, none of the current general nonsmooth optimization methods is efficient in large-scale settings. In this paper, we describe a new limited memory variable metric based bundle method for nonsmooth large-scale optimization. In addition, we introduce a new set of academic test problems for large-scale nonsmooth minimization. Finally, we give some encouraging results from numerical experiments using both academic and practical test problems.

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

1 INTRODUCTION

In this paper, we describe a limited memory bundle algorithm for solving large nonsmooth (nondifferentiable) unconstrained optimization problems. We write this problem as

$$\begin{cases} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathbb{R}^n, \end{cases} \tag{1}$$

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.

Nonsmooth optimization problems of type (1) arise in many fields of applications, for example, in image restoration (see, e.g., [1]) and in optimal control (see, e.g., [2]). 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., [3]). On the other hand, direct methods, for example, Powel's method (see, e.g., [4]) employing no derivative information, are quite unreliable and become inefficient when the size of the problem increases. Thus, we need special tools for solving nonsmooth optimization problems.

1. E-mail: hamasi@mit.jyu.fi
2. E-mail: miettine@mit.jyu.fi
3. E-mail: makela@mit.jyu.fi

At the moment, various versions of bundle methods (see, e.g., [2, 5, 6, 7]) are regarded as the most effective and reliable methods for nonsmooth optimization. They are based on the assumption that at every point $\mathbf{x} \in \mathbb{R}^n$, we can evaluate the value of the objective function $f(\mathbf{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \mathbb{R}^n$ from the subdifferential (see [8])

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

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

The basic idea of bundle methods is to approximate the subdifferential of the objective function by gathering subgradients from previous iterations into a bundle. This subgradient information serves for the construction of a piecewise linear local approximation to the objective function. A descent direction for this approximation and, thus, also for the objective function can then be determined by solving a quadratic direction finding problem (see, e.g., [2]). The global convergence of bundle methods with a limited number of stored subgradients can be guaranteed by using a subgradient aggregation strategy [6], which accumulates information from the previous iterations.

In their present form, bundle methods are efficient for small- and medium-scale problems. However, their computational demand expands in large-scale problems with more than 1000 variables. This is explained by the fact that bundle methods need relatively large bundles to be capable of solving the problems efficiently. In other words, the size of the bundle has to be approximately the same as the number of variables (see [6]) and, thus, the quadratic direction finding problem becomes very time-consuming to solve.

In variable metric bundle methods introduced by Lukšan and Vlček [9, 10], the search direction is calculated by using the variable metric approximation of the inverse of the Hessian matrix. Thus, the quite complicated quadratic direction finding problem appearing in standard bundle methods does not need to be solved. Furthermore, the subgradient aggregation is done by using only three subgradients and, thus, the size of the bundle does not need to grow with the dimension of the problem. However, variable metric bundle methods use dense approximations of the Hessian matrix to calculate the search direction and, thus, due to matrix operations also these methods become inefficient when the dimension of the problem increases.

We have not found any general bundle-based solver for large-scale nonsmooth optimization problems in the literature. Thus, we can say that at the moment the only possibility to optimize nonsmooth large-scale problems is to use some subgradient methods (see, e.g., [11]). However, the basic subgradient methods suffer from some serious disadvantages: a nondescent search direction may occur, there exists no implementable stopping criterion, and the convergence speed is poor (not even linear) (see, e.g., [3]). On the other hand, the more advanced variable metric based subgradient methods (see, e.g., [11]) using dense matrices suffer from the same drawbacks than the variable metric bundle methods. This means that there is an evident need of reliable and efficient solvers for nonsmooth large-scale optimization problems.

In this paper, we introduce a new limited memory bundle method for large-scale nonsmooth unconstrained minimization. The method is a hybrid of the variable metric bundle methods [9, 10] and the limited memory variable metric methods (see, e.g., [12, 13]), where the latter have been developed for smooth large-scale optimization. The new method combines in a novel way the ideas of the variable metric bundle method with the search direction calculation of limited memory approach. Therefore, the time-consuming quadratic direction finding problem appearing in the standard bundle methods does not need to be solved, nor the number of stored subgradients needs to grow with the dimension of the problem. Furthermore, the method uses only few vectors to represent the variable metric approximation of the Hessian matrix and, thus, it avoids storing and manipulating large matrices as is the case in variable metric bundle methods (see [9, 10]). These improvements make the limited memory bundle method suitable for large-scale optimization. Namely, the number of operations needed for the calculation of the search direction and the aggregate values is only linearly dependent on the number of variables while, for example, with the original variable metric bundle method, this dependence is quadratic.

This paper is organized as follows. In the following section, we describe the new limited memory bundle method. In Section 3, we introduce a new set of academic test problems for large-scale nonsmooth minimization. Then, in Section 4, we analyze numerical experiments concerning some existing bundle methods and our new method. The numerical results demonstrate the usability of the new method with both smooth and nonsmooth large-scale minimization problems. Finally, in Section 5, we conclude by giving a short summary of the performance of the methods tested.

2 LIMITED MEMORY BUNDLE METHOD

In this section, we present a new method for large-scale nonsmooth unconstrained optimization. The basic idea of the method is the following: We use some ideas essential to bundle methods, namely, the utilization of null steps, simple aggregation of subgradients, and subgradient locality measures, but the search direction is calculated by using the limited memory variable metric updates. The idea of this limited memory approach is that the variable metric update of the approximated Hessian is not constructed explicitly. The updates use the information of the last few iterations to implicitly define a variable metric approximation. In practice, this means that the approximation of the Hessian matrix is not as accurate as that of the original variable metric bundle methods but both the storage space required and the number of operations used are significantly smaller. In smooth large-scale settings, there exist also some other possibilities to deal with the variable metric approximation of the Hessian matrix (see, e.g., [14, 15]). However, we chose this limited memory approach to be adopted for nonsmooth problems because it does not need any information of the structure of the problem or its Hessian. Thus, the only assumption set is that the objective function f is locally Lipschitz continuous.

Next, we go through the algorithm step by step and describe both its theoretical properties and some details of the implementation. In what follows, we use the following notations: The current iteration point at iteration k is denoted by $\mathbf{x}_k$, and the current auxiliary point is denoted by $\mathbf{y}_k$. The approximation of the inverse of the Hessian matrix is denoted by D_k, the subgradient of the objective function is denoted by $\boldsymbol{\xi}_k$, and an aggregate subgradient to be described in Subsection 2.3 is denoted by $\tilde{\boldsymbol{\xi}}_k$.

2.1 Direction Finding

The basic idea in finding the search direction is the same as with the limited memory variable metric methods (see, e.g., [12]) and the approximations D_k are formed implicitly by using the limited memory variable metric updates. However, due to the usage of null steps some modifications similar to variable metric bundle methods [9, 10] have to be made.

After a null step, the approximation D_k is formed by using the limited memory SR1 update (see, e.g., [12]), since this update formula gives us a possibility of preserving the boundedness of the generated matrices (that is, the eigenvalues of the matrix lie in a compact interval not containing zero) (see, e.g., [10]). In addition, we use an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to calculate the search direction

$$\mathbf{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k. \tag{2}$$

Because the boundedness of the generated matrices is not required after a serious step (see [10]), then, the more efficient limited memory BFGS update formula (see, e.g., [12]) is used to compute the approximation D_k. The search direction is calculated by using the original subgradient $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$. Thus, after a serious step, the search direction is defined by $\mathbf{d}_k = -D_k \boldsymbol{\xi}_k$.

2.2 Line Search

Next, we consider how to calculate a new iteration point $\mathbf{x}_{k+1}$ when the search direction $\mathbf{d}_k$ has been calculated. Similarly to the standard bundle methods and the variable metric bundle methods, we use the line search procedure (see, e.g., [10]) that generates two points

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + t_L^k \mathbf{d}_k \qquad && \text{and} \\ \mathbf{y}_{k+1} &= \mathbf{x}_k + t_R^k \mathbf{d}_k, \qquad && \text{for } k \geq 1 \end{aligned} \tag{3}$$

with $\mathbf{y}_1 = \mathbf{x}_1$, where $t_R^k \in (0, t_I^k]$, $t_L^k \in [0, t_R^k]$ are step sizes, $t_I^k \in [t_{min}, t_{max})$ is the initial step size, and $t_{min} \in (0,1)$ and $t_{max} > 1$ are the lower and the upper bounds for the initial step size t_I^k, respectively. The initial step size t_I^k is selected exactly the same way as in the original variable metric bundle method for nonconvex objective functions (see [10]).

A necessary condition for a serious step to be taken is to have

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\mathbf{y}_{k+1}) \leq f(\mathbf{x}_k) - \varepsilon_L t_R^k w_k, \tag{4}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ (see (10)) represents the desirable amount of descent of f at $\mathbf{x}_k$. If condition (4) is satisfied, we set $\mathbf{x}_{k+1} = \mathbf{y}_{k+1}$ and a serious step is taken.

A null step is taken if

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \mathbf{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{5}$$

where $\varepsilon_R \in (\varepsilon_L, 1)$ is a fixed line search parameter, $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$, and β_{k+1} is the subgradient locality measure similar to bundle methods (see, e.g., [7]), that is,

$$\beta_{k+1} = \max\{|f(\mathbf{x}_k) - f(\mathbf{y}_{k+1}) + (\mathbf{y}_{k+1} - \mathbf{x}_k)^T \boldsymbol{\xi}_{k+1})|, \gamma \|\mathbf{y}_{k+1} - \mathbf{x}_k\|^\omega \}. \tag{6}$$

Here $\gamma \geq 0$ is a distance measure parameter and $\omega \geq 1$ is a locality measure parameter supplied by the user. Parameter γ can be set to zero when f is convex.

In the case of a null step, we set $\mathbf{x}_{k+1} = \mathbf{x}_k$ but information about the objective function is increased because we store the auxiliary point $\mathbf{y}_{k+1}$ and the auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$.

The step sizes t_L^k and t_R^k can be determined by using the line search procedure quite similar to that in the original variable metric bundle method [10]. However, in order to avoid many consecutive null steps and, thus, to make the method more efficient, we have added an additional interpolation step. That is, we look for more suitable step sizes t_L^k and t_R^k by using an extra interpolation loop if necessary. The role of this additional step is that if we have already taken a null step at the previous iteration, we rather try to find a step size suitable for a serious step (that is, so that (4) is valid) even if condition (5) required for a null step was satisfied.

Note that under some semismoothness assumptions, the line search procedure used is guaranteed to find the step sizes t_L^k and t_R^k such that exactly one of the two possibilities, a serious step or a null step, occurs (see [10]).

2.3 Subgradient Aggregation

In principle, the aggregation procedure used with the limited memory bundle method is the same as that with the original variable metric bundle methods [9, 10]. However, since the matrix D_k is not formed explicitly here, the practical implementation of the aggregation procedure differs from that of the original method.

The aggregation procedure uses three subgradients and two locality measures. We denote by m the lowest index j satisfying $\mathbf{x}_j = \mathbf{x}_k$ (that is, m is the index of the iteration after the latest serious step). Suppose that we have the current subgradient $\boldsymbol{\xi}_m \in \partial f(\mathbf{x}_k)$, the auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{y}_{k+1})$, and the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ (note that $\tilde{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1$) available. In addition, suppose that we have the current locality measure β_{k+1} (see (6)) and the current aggregate locality measure $\tilde{\beta}_k$ from the previous iteration (note that $\tilde{\beta}_1 = 0$). The quite complicated quadratic direction finding problem (see, e.g., [2]) appearing in the standard bundle methods reduces to the minimization of the function

$$\begin{aligned}\varphi(\lambda_1, \lambda_2, \lambda_3) = {} & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k) \\ & + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k),\end{aligned} \tag{7}$$

where $\lambda_i \geq 0$ for $i \in \{1, 2, 3\}$ and $\sum_{i=1}^{3} \lambda_i = 1$. The optimal values λ_i^k, $i \in \{1, 2, 3\}$ can be calculated by using a simple formulae.

The next $\tilde{\boldsymbol{\xi}}_{k+1}$ is defined as a convex combination of the subgradients mentioned above:

$$\tilde{\boldsymbol{\xi}}_{k+1} = \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k \tag{8}$$

and the next $\tilde{\beta}_{k+1}$ as a convex combination of the locality measures:

$$\tilde{\beta}_{k+1} = \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k. \tag{9}$$

Note that the aggregate values are computed only if the last step was a null step. Otherwise, we set $\tilde{\boldsymbol{\xi}}_{k+1} = \boldsymbol{\xi}_{k+1} \in \partial f(\mathbf{x}_{k+1})$ and $\tilde{\beta}_{k+1} = 0$.

2.4 Stopping Criterion

For smooth functions, a necessary condition for a local minimum is that the gradient has to be zero and by continuity it becomes small when we are close to an optimal point. This is no longer true when we replace the gradient by an arbitrary subgradient. Due to the subgradient aggregation, we have quite a useful approximation to the gradient at our disposal, namely the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$. However, as a stopping criterion, the direct test $\|\tilde{\boldsymbol{\xi}}_k\| < \varepsilon$, for some $\varepsilon > 0$, is too uncertain, if the current piecewise linear approximation of the objective function is too rough. Therefore, we use the term $\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k = -\tilde{\boldsymbol{\xi}}_k^T \mathbf{d}_k$ and the aggregate subgradient locality measure $\tilde{\beta}_k$ to improve the accuracy of $\|\tilde{\boldsymbol{\xi}}_k\|$ (see, e.g., [2]). Hence, the stopping parameter w_k at iteration k is defined by

$$w_k = -\tilde{\boldsymbol{\xi}}_k^T \mathbf{d}_k + 2\tilde{\beta}_k. \tag{10}$$

Note that the parameter w_k is also used during the line search procedure (see (4)) to represent the desirable amount of descent.

This first part of our stopping criterion is similar to that of the original variable metric bundle method. However, in practice the limited memory approximation D_k is not very accurate and computational experiments have shown that some accidental terminations may occur (that is, the optimization may terminate before the minimum point has been actually found). Thus, we added a second stopping parameter q_k, which does not depend on the matrix D_k. The second stopping parameter is similar to that in proximal bundle methods (see, e.g., [2]), that is,

$$q_k = \frac{1}{2}\tilde{\boldsymbol{\xi}}_k^T \tilde{\boldsymbol{\xi}}_k + \tilde{\beta}_k. \tag{11}$$

Now, the stopping criterion is given by:

$$\text{If } w_k < \varepsilon \text{ and } q_k < 10^2\varepsilon, \text{ for given } \varepsilon > 0, \text{ then stop.} \tag{12}$$

The multiplier 10^2 above has been chosen experimentally such that the accuracy of the new method would be approximately the same as with other bundle methods.

2.5 Algorithm

We are now ready to present the limited memory bundle method for nonsmooth large-scale unconstrained optimization.

Algorithm 1. (Limited Memory Bundle Method).

Data: Select the lower and the upper bounds $t_{min} \in (0,1)$ and $t_{max} > 1$ for serious steps. Select positive line search parameters $\varepsilon_L \in (0,1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$. Choose the final accuracy tolerance $\varepsilon > 0$, the distance measure parameter $\gamma \geq 0$ ($\gamma = 0$ if f is convex), and the locality measure parameter $\omega \geq 1$.

Step 0: (*Initialization.*) Choose a starting point $\mathbf{x}_1 \in \mathbb{R}^n$. Set $\beta_1 = 0$ and $\mathbf{y}_1 = \mathbf{x}_1$. Compute $f_1 = f(\mathbf{x}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\mathbf{x}_1)$. 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: (*Direction finding.*) Compute

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

by a limited memory BFGS update if $m = k$ (Algorithm 2) and by a limited memory SR1 update, otherwise (Algorithm 3). Note that $\mathbf{d}_1 = -\boldsymbol{\xi}_1$.

Step 3: (*Stopping criterion.*) Calculate w_k and q_k by (10) and (11), respectively. If $w_k < \varepsilon$ and $q_k < 10^2\varepsilon$, then stop with $\mathbf{x}_k$ as the final solution.

Step 4: (*Line search.*) Calculate the initial step size $t_I^k \in [t_{min}, t_{max})$. Determine the step sizes $t_R^k \in (0, t_I^k]$ and $t_L^k \in [0, t_R^k]$ to take either a serious step or a null step (that is, check whether (4) or (5) is valid). Set the corresponding values

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

Set $\mathbf{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$ and $\mathbf{s}_k = \mathbf{y}_{k+1} - \mathbf{x}_k = t_R^k \mathbf{d}_k$. If $t_L^k > 0$ (serious step), set $\beta_{k+1} = 0$, $k = k+1$, and go to Step 1. Otherwise, calculate the locality measure β_{k+1} by (6).

Step 5: (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$, $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function (7), where D_k is calculated by the same updating formula as in Step 2. Set

$$\begin{aligned}\tilde{\boldsymbol{\xi}}_{k+1} &= \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k \qquad \text{and}\\ \tilde{\beta}_{k+1} &= \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.\end{aligned}$$

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

Note that in Steps 2 and 5, the matrices D_k are not formed explicitly but expressions (14) and (18) are used instead.

As mentioned in Subsection 2.3, the aggregation procedure uses only three subgradients and two locality measures to calculate the new aggregate values. In practice, this means that the minimum size of the bundle m_ξ is 2 and a larger bundle is used only for the selection of the initial step size (see [10]).

2.6 Matrix Updating

Finally, we need to consider how to update the approximation D_k of the inverse of the Hessian matrix and, thus, how to find the search direction $\mathbf{d}_k$. Note that until this point, the procedures described are rather similar to those of the original variable metric bundle method [10].

We use a compact representation of limited memory matrices (see [12]), since in addition to the BFGS updating formula, we need the SR1 updating formula and for SR1 updates, there exists no recursive updating formula analogous to that given in [13]. Moreover, the compact representation of limited memory matrices facilitates the possibility to generalize the method for constrained optimization.

The basic idea of the limited memory matrix updating is that instead of storing the matrices D_k, we use the information of the last few iterations to implicitly define the approximation. This is done by storing a certain number of correction pairs $(\mathbf{s}_i, \mathbf{u}_i)$, $(i < k)$, obtained in Step 4 of Algorithm 1. When the storage space available is used up, the oldest corrections are deleted to make room for new ones.

Let us denote by m_c the maximum number of stored corrections supplied by the user $(3 \leq m_c)$ and by $m_k = \min\{\, k-1, m_c \,\}$ the current number of stored corrections. We assume that m_c is constant, although it is possible to adapt all the formulae of this subsection so that m_c varies at every iteration (see, e.g., [16]).

The $n \times m_k$ -dimensional correction matrices S_k and U_k are defined by

$$\begin{aligned} S_k &= \begin{bmatrix}\mathbf{s}_{k-m_k} & \dots & \mathbf{s}_{k-1}\end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix}\mathbf{u}_{k-m_k} & \dots & \mathbf{u}_{k-1}\end{bmatrix}. \end{aligned} \tag{13}$$

These matrices are used to implicitly define D_k at each iteration. When a new auxiliary iteration point $\mathbf{y}_{k+1}$ is generated, the new matrices S_{k+1} and U_{k+1} are obtained by deleting the oldest corrections $\mathbf{s}_{k-m_k}$ and $\mathbf{u}_{k-m_k}$ from S_k and U_k if $m_{k+1} = m_k$ (that is, $k > m_c$) and by adding the most recent corrections $\mathbf{s}_k$ and $\mathbf{u}_k$ to the matrices. Thus, except for the first few iterations, we always have the m_c most recent correction pairs $(\mathbf{s}_i, \mathbf{u}_i)$ available.

We define the inverse limited memory BFGS update by the formula (see [12])

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T(C_k + \vartheta_k U_k^T U_k)R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}. \tag{14}$$

Here, R_k is an upper triangular matrix of order m_k given as

$$(R_k)_{ij} = \begin{cases} (\mathbf{s}_{k-m_k-1+i})^T(\mathbf{u}_{k-m_k-1+j}), & \text{if } i \le j \\ 0, & \text{otherwise,} \end{cases} \tag{15}$$

C_k is a diagonal matrix of order m_k such that

$$C_k = \operatorname{diag}[\mathbf{s}_{k-m_k}^T \mathbf{u}_{k-m_k}, \dots, \mathbf{s}_{k-1}^T \mathbf{u}_{k-1}], \tag{16}$$

and the scaling parameter $\vartheta_k > 0$ is given by

$$\vartheta_k = \frac{\mathbf{u}_{k-1}^T \mathbf{s}_{k-1}}{\mathbf{u}_{k-1}^T \mathbf{u}_{k-1}}. \tag{17}$$

In addition, we define the inverse limited memory SR1 update (see [12]) by

$$D_k = \vartheta_k I - (\vartheta_k U_k - S_k)(\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k - S_k)^T, \tag{18}$$

where instead of (17) we use the value $\vartheta_k = 1$ for every k.

Next, we describe some procedures for updating the limited memory BFGS and SR1 matrices. In addition to the two $n \times m_k$ -matrices S_k and U_k, we store the $m_k \times m_k$ -matrices R_k, $U_k^T U_k$, and C_k and the m_k-vectors $S_k^T \boldsymbol{\xi}_m$ and $U_k^T \boldsymbol{\xi}_m$ from the previous iteration. Since in practice m_k is clearly smaller than n, the storage space required by these three auxiliary matrices and two vectors is insignificant but the savings in computational efforts are considerable. A detailed description of updating these matrices and vectors can be found in [12].

We first give an efficient algorithm for updating the limited memory BFGS matrix D_k and for computing the search direction $\mathbf{d}_k = -D_k \boldsymbol{\xi}_k$. This algorithm is used whenever the previous step was a serious step. Note that after a serious step the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$ and the correction vectors obtained at the previous iteration in Step 4 of Algorithm 1 can be equally expressed as $\mathbf{s}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$ and $\mathbf{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1}$. Therefore, the calculations used are very similar to those given in [12]. In fact, all the calculations in [12] could be done by replacing the gradient $\nabla f(\mathbf{x})$ by an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$.

Algorithm 2. (BFGS Updating and Direction Finding).

Data: Suppose that the number of current corrections is m_{k-1} and the maximum number of stored corrections is m_c. Suppose that we have the most recent corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$ (from the previous iteration), the current subgradient $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$, the previous subgradient $\boldsymbol{\xi}_{k-1} \in \partial f(\mathbf{x}_{k-1})$, the $n \times m_{k-1}$ -matrices S_{k-1} and U_{k-1}, the $m_{k-1} \times m_{k-1}$ -matrices R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}, the previous scaling parameter ϑ_{k-1}, and the m_{k-1}-vectors $S_{k-1}^T \boldsymbol{\xi}_{k-1}$ and $U_{k-1}^T \boldsymbol{\xi}_{k-1}$ available.

Step 1: (*Positive definiteness*) If

$$\mathbf{u}_{k-1}^T \mathbf{s}_{k-1} > 0, \tag{19}$$

then set $m_k = \min\{m_{k-1}+1, m_c\}$ and update the matrices (i.e., go to Step 2). Otherwise, skip the updates, that is, set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$, $C_k = C_{k-1}$, $\vartheta_k = \vartheta_{k-1}$, and $m_k = m_{k-1}$, compute $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$, and go to Step 7.

Step 2: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 3: Compute and store m_k-vectors $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$.

Step 4: Compute m_k-vectors $S_k^T \mathbf{u}_{k-1}$ and $U_k^T \mathbf{u}_{k-1}$ by using the fact

$$\mathbf{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1}.$$

Step 5: Update $m_k \times m_k$ -matrices R_k, $U_k^T U_k$, and C_k.

Step 6: If $\mathbf{u}_{k-1}^T \mathbf{u}_{k-1} > 0$, compute ϑ_k according to (17). Otherwise, set $\vartheta_k = 1.0$.

Step 7: (*Intermediate values*) Solve the values $\mathbf{p}_1 \in \mathbb{R}^{m_k}$ and $\mathbf{p}_2 \in \mathbb{R}^{m_k}$ satisfying the linear equations

$$\begin{aligned} R_k \mathbf{p}_1 &= S_k^T \boldsymbol{\xi}_k, \\ R_k^T \mathbf{p}_2 &= C_k \mathbf{p}_1 + \vartheta_k U_k^T U_k \mathbf{p}_1 - \vartheta_k U_k^T \boldsymbol{\xi}_k. \end{aligned}$$

Step 8: (*Search direction*) Compute

$$\mathbf{d}_k = \vartheta_k U_k \mathbf{p}_1 - S_k \mathbf{p}_2 - \vartheta_k \boldsymbol{\xi}_k.$$

Note that condition (19) assures the positive definiteness of the matrices obtained by the limited memory BFGS update (see, e.g., [12]).

Next, we give an efficient algorithm for updating the limited memory SR1 matrix D_k and for computing the search direction $\mathbf{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k$. This algorithm is used whenever the previous step was a null step.

Algorithm 3. (SR1 Updating and Direction Finding).

Data: Suppose that the number of current corrections is m_{k-1} and the maximum number of stored corrections is m_c. Suppose that we have the most recent corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$, the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$, the previous aggregate subgradient $\tilde{\boldsymbol{\xi}}_{k-1}$, the current subgradients $\boldsymbol{\xi}_k \in \partial f(\mathbf{y}_k)$ and $\boldsymbol{\xi}_m \in \partial f(\mathbf{x}_k)$, the previous search direction $\mathbf{d}_{k-1}$, the matrices S_{k-1}, U_{k-1}, R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}, and the vectors $S_{k-1}^T \boldsymbol{\xi}_m$ and $U_{k-1}^T \boldsymbol{\xi}_m$ available.

Step 1: (*Positive definiteness*) If

$$-\mathbf{d}_{k-1}^T \mathbf{u}_{k-1} - \tilde{\boldsymbol{\xi}}_{k-1}^T \mathbf{s}_{k-1} < 0, \tag{20}$$

then set $m_k = \min\{m_{k-1}+1, m_c\}$ and update the matrices (i.e., go to Step 2). Otherwise, skip the updates, that is, set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$, $C_k = C_{k-1}$, $S_k^T \boldsymbol{\xi}_m = S_{k-1}^T \boldsymbol{\xi}_m$, $U_k^T \boldsymbol{\xi}_m = U_{k-1}^T \boldsymbol{\xi}_m$, and $m_k = m_{k-1}$ and go to Step 6.

Step 2: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 3: Compute m_k-vectors $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$.

Step 4: Compute m_k-vectors $S_k^T \mathbf{u}_{k-1}$ and $U_k^T \mathbf{u}_{k-1}$ by using the fact

$$\mathbf{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_m.$$

Store m_k-vectors $S_k^T \boldsymbol{\xi}_m$ and $U_k^T \boldsymbol{\xi}_m$.

Step 5: Update $m_k \times m_k$ -matrices R_k, $U_k^T U_k$, and C_k.

Step 6: Set $\vartheta_k = 1.0$.

Step 7: Compute m_k-vectors $S_k^T \tilde{\boldsymbol{\xi}}_k$ and $U_k^T \tilde{\boldsymbol{\xi}}_k$.

Step 8: (*Intermediate value*) Compute

$$\mathbf{p} = (\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k^T \tilde{\boldsymbol{\xi}}_k - S_k^T \tilde{\boldsymbol{\xi}}_k).$$

Step 9: (*Search direction*) Compute

$$\mathbf{d}_k = -\vartheta_k \tilde{\boldsymbol{\xi}}_k + (\vartheta_k U_k - S_k)\mathbf{p}.$$

Note that condition (20) assures the positive definiteness of the matrices obtained by the limited memory SR1 update (see [17]).

In order to use both Algorithms 2 and 3 with the same stored information, some modifications have to be made. For example, since we use the same correction matrices S_k and U_k in both of the BFGS and the SR1 updates, both the positive definiteness conditions (19) and (20) have to be valid in each case before we update the matrices. Nevertheless, it can be shown (see [17]) that condition (20) implies (19) and, thus, we only have to check the validity of (20). However, numerical experiments have showed that the simple skipping of the BFGS update (see Algorithm 2, Step 1) if condition (20) required for the SR1 update is not satisfied, makes the method quite inefficient. Therefore, in the case of BFGS update, the new search direction $\mathbf{d}_k$ is calculated conventionally using the most recent corrections $\mathbf{s}_{k-1}$ and $\mathbf{u}_{k-1}$ whenever the required positive definiteness condition (19) is valid but the matrices are not updated, that is, the correction vectors are not stored, unless both the conditions (19) and (20) are satisfied. In practice, this means that the correction matrices S_k and U_k may actually include some indices smaller than $k - m_k$ due to skipping of updates and that, in the case of the BFGS update, the number of the current corrections used may be $m_k = m_c + 1$.

The slightly modified version of this limited memory bundle method can be proved to be globally convergent. A detailed description of the convergence analysis can be found in [17].

3 LARGE-SCALE NONSMOOTH TEST PROBLEMS

Many practical optimization applications involve nonsmooth functions of many variables. However, there exist only few large-scale academic test problems for the nonsmooth case. For this reason, we now introduce a new set of large-scale unconstrained minimization problems for nonsmooth optimization.

We present 10 nonsmooth problems which all can be formulated with any number of variables. The problems have been constructed either by chaining and extending small existing nonsmooth problems or by "nonsmoothing" large smooth problems (that is, for example, by replacing the term x_i^2 with $|x_i|$). First, we give the formulation of the objective function f and the starting point $\mathbf{x}_1 = (x_1^{(1)}, \dots, x_n^{(1)})^T$ for each problem. Then, we collect some details of the problems as well as the references to the original problems in Table 1.

1. Generalization of MAXQ

$$f(\mathbf{x}) = \max_{1 \le i \le n} x_i^2.$$

$x_i^{(1)} = i, \qquad$ for $i = 1, \dots, n/2$ and
$x_i^{(1)} = -i, \qquad$ for $i = n/2+1, \dots, n$.

2. Generalization of MXHILB

$$f(\mathbf{x}) = \max_{1 \le i \le n} \left| \sum_{j=1}^{n} \frac{x_j}{i+j-1} \right|.$$

$x_i^{(1)} = 1, \qquad$ for all $i = 1, \dots, n$.

3. Chained LQ

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \max \left\{ -x_i - x_{i+1}, -x_i - x_{i+1} + (x_i^2 + x_{i+1}^2 - 1) \right\}.$$

$x_i^{(1)} = -0.5, \;$ for all $i = 1, \dots, n$.

4. Chained CB3 I

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \max \left\{ x_i^4 + x_{i+1}^2, (2-x_i)^2 + (2-x_{i+1})^2, 2e^{-x_i + x_{i+1}} \right\}.$$

$x_i^{(1)} = 2, \qquad$ for all $i = 1, \dots, n$.

5. Chained CB3 II

$$f(\mathbf{x}) = \max \Big\{ \sum_{i=1}^{n-1} \left(x_i^4 + x_{i+1}^2 \right), \sum_{i=1}^{n-1} \left((2-x_i)^2 + (2-x_{i+1})^2 \right), \sum_{i=1}^{n-1} \left(2e^{-x_i + x_{i+1}} \right) \Big\}.$$

$x_i^{(1)} = 2, \qquad$ for all $i = 1, \dots, n$.

6. Number of Active Faces

$$f(\mathbf{x}) = \max_{1 \le i \le n} \left\{ g\left(-\sum_{i=1}^{n} x_i \right), g(x_i) \right\},$$

where $g(y) = \ln(|y| + 1)$.
$x_i^{(1)} = 1, \qquad$ for all $i = 1, \dots, n$.

7. Nonsmooth generalization of Brown function 2

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left(|x_i|^{x_{i+1}^2+1} + |x_{i+1}|^{x_i^2+1} \right).$$

$$\begin{array}{lll} x_i^{(1)} = -1, & \text{when} & \text{mod}\,(i,2) = 1,\ (i = 1,\ldots,n) \text{ and} \\ x_i^{(1)} = 1, & \text{when} & \text{mod}\,(i,2) = 0,\ (i = 1,\ldots,n). \end{array}$$

8. Chained Mifflin 2

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left(-x_i + 2\left(x_i^2 + x_{i+1}^2 - 1\right) + 1.75\left|x_i^2 + x_{i+1}^2 - 1\right| \right).$$

$$x_i^{(1)} = -1, \quad \text{for all } i = 1,\ldots,n.$$

9. Chained Crescent I

$$\begin{aligned} f(\mathbf{x}) = \max\Big\{ & \sum_{i=1}^{n-1} \left(x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1 \right), \\ & \sum_{i=1}^{n-1} \left(-x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1 \right) \Big\}. \end{aligned}$$

$$\begin{array}{lll} x_i^{(1)} = -1.5, & \text{when} & \text{mod}\,(i,2) = 1,\ (i = 1,\ldots,n) \text{ and} \\ x_i^{(1)} = 2.0, & \text{when} & \text{mod}\,(i,2) = 0,\ (i = 1,\ldots,n). \end{array}$$

10. Chained Crescent II

$$\begin{aligned} f(\mathbf{x}) = \sum_{i=1}^{n-1} \max\Big\{ & x_i^2 + (x_{i+1} - 1)^2 + x_{i+1} - 1, \\ & -x_i^2 - (x_{i+1} - 1)^2 + x_{i+1} + 1 \Big\}. \end{aligned}$$

$$\begin{array}{lll} x_i^{(1)} = -1.5, & \text{when} & \text{mod}\,(i,2) = 1,\ (i = 1,\ldots,n) \text{ and} \\ x_i^{(1)} = 2.0, & \text{when} & \text{mod}\,(i,2) = 0,\ (i = 1,\ldots,n). \end{array}$$

The details of the problems are given in Table 1, where p denotes the problem number, $f(\mathbf{x}^*)$ is the minimum value of the objective function, and the symbols "$-$" (nonconvex) and "$+$" (convex) denote the convexity of the problems. Also the references to the original problems in each case are given in Table 1.

4 NUMERICAL EXPERIMENTS

In order to get some information of how the new method works in practice when compared to other nonsmooth methods, we tested different existing solvers with several problems. In this section, we first introduce the software tested and the testing environment. Then, we give the results of the numerical experiments and draw some conclusions.

Table 1: Test problems

p	$f(\mathbf{x}^*)$	Convex	Original problem and reference
1	0.0	+	MAXQ, $n = 20$, see [18]
2	0.0	+	MXHILB, $n = 50$, see [19]
3	$-(n-1)2^{1/2}$	+	LQ, $n = 2$, see [20]
4	$2(n-1)$	+	CB3, $n = 2$, see [21]
5	$2(n-1)$	+	CB3, $n = 2$, see [21]
6	0.0	−	See [22]
7	0.0	−	Generalization of Brown function, see [23]
8	varies*	−	Mifflin 2, $n = 2$, see [24]
9	0.0	−	Crescent, $n = 2$, see [6]
10	0.0	−	Crescent, $n = 2$, see [6]

* $f(\mathbf{x}^*) \approx -6.51$ for $n = 10$, $f(\mathbf{x}^*) \approx -70.15$ for $n = 100$ and $f(\mathbf{x}^*) \approx -706.55$ for $n = 1000$.

4.1 Software Tested and Testing Environment

The experiments were performed in a SGI Origin 2000/128 supercomputer (MIPS R12000, 600 Mflop/s/processor). The algorithms were implemented in FORTRAN77 with double-precision arithmetic. The solvers used in our experiments are presented in Table 2.

Table 2: Tested pieces of software

Software	Author(s)	Method	Reference
`PVAR`	Lukšan & Vlček	Variable metric bundle	[9, 10]
`PNEW`	Lukšan & Vlček	Bundle-Newton	[25]
`PBUN`	Lukšan & Vlček	Proximal bundle	[26]
`PBNCGC`	Mäkelä	Proximal bundle	[2]
`L-BFGS`	Nocedal	Limited memory BFGS	[27, 13]
`LMBM`	Haarala	Limited memory bundle	

None of the nonsmooth optimization solvers `PVAR`, `PBUN`, `PNEW`, and `PBNCGC` has been developed for large-scale optimization. On the other hand, we wanted to get some information of the behavior of the new solver `LMBM` especially with large-scale problems. For this reason, we used the smooth large-scale optimization solver `L-BFGS` as a benchmark. Thus, all the solvers were first tested with 22 smooth minimization problems given in [28]. However, we removed two problems (problems 2 and 10 in [28]) where the solvers converged to different local minima.

Next, the solvers for nonsmooth optimization (that is, all the solvers in Table 2 except `L-BFGS`) were tested with 10 nonsmooth minimization problems described earlier, and with a practical nonsmooth image restoration problem described in [1].

In the test results to be reported, we say that the optimization terminated successfully if

- the problem was solved with the desired accuracy. That is, $w_k \leq \varepsilon$, where $w_k = 1/2\|\tilde{\boldsymbol{\xi}}_k\|^2 + \tilde{\beta}_k$ in `PNEW`, `PBUN`, and `PBNCGC`, $w_k = \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k$ in `PVAR` and `LMBM` (note that also $1/2\|\tilde{\boldsymbol{\xi}}_k\|^2 + \tilde{\beta}_k \leq 10^2\varepsilon$ in `LMBM`), and $w_k = \|\nabla f(\mathbf{x}_k)\|/\max\{1, \|\mathbf{x}_k\|\}$ in `L-BFGS`.

In addition, for the solvers `PVAR`, `PNEW`, `PBUN` and `LMBM` we say that the optimization terminated successfully if

- $|f_{k+1} - f_k| \leq 1.0 \cdot 10^{-8}$ in 10 subsequent iterations and the result obtained was less than two significant digits greater than the desired accuracy of the solution.

For `L-BFGS` we accepted all the results, since in all cases they were obtained with the desired accuracy (even if the solver claimed that it failed).

In addition to the stopping criteria already mentioned, we terminated the experiments if the CPU time elapsed exceeded half an hour. Also in these cases, the results were accepted if they were less than two significant digits greater than the desired accuracy of the solution.

Otherwise, we say that the optimization failed.

The test results are analyzed using the performance profiles introduced in [29]. We use the computational times of the solvers as a performance measure, although, we report also some other properties. In the following graphs $\rho_s(\tau)$ denotes the logarithmic performance profile

$$\rho_s(\tau) = \frac{\text{no. of problems where } \log_2(r_{p,s}) \leq \tau}{\text{total no. of problems}} \tag{21}$$

with $\tau \geq 0$, where $r_{p,s}$ is the performance ratio between the time to solve problem p by solver s over the lowest time required by any of the solvers. The ratio $r_{p,s}$ is set to infinity (or some sufficiently large number) if solver s fails to solve problem p.

There are two important facts to be kept in mind to have a good interpretation of the performance profiles: The value of $\rho_s(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the corresponding solver s is the best and the value of $\rho_s(\tau)$ at the rightmost abscissa is the percentage of test problems that the corresponding solver s can solve (this does not depend on the measured performance). Finally, 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 [29].

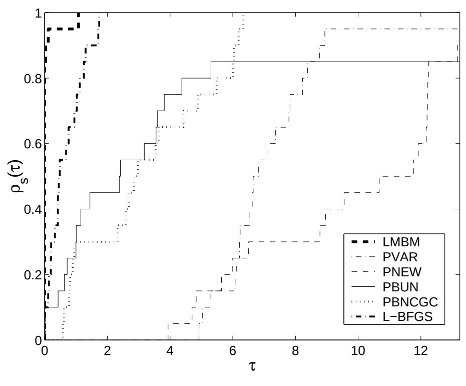
4.2 Numerical Results

Smooth test problems. All the solvers given in Table 2 were first tested with 20 smooth problems with 1000 variables and, in case of the limited memory solvers `L-BFGS` and `LMBM`, also with 10 000 variables.

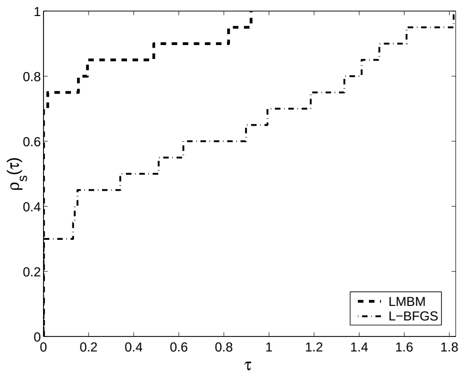
We tested the bundle solvers `PVAR`, `PNEW`, `PBUN`, `PBNCGC`, and `LMBM` with relatively small sizes of the bundle, that is, $m_\xi = 10$. For the limited memory solvers `L-BFGS` and `LMBM`, the maximum number of stored corrections (m_c) was set to 7 and for `LMBM` the maximum number of additional interpolations used at each iteration was set to 200 (default value). As a stopping parameter, we used $\varepsilon = 10^{-6}$ in all the cases. The other parameters used were chosen experimentally.

The results of the smooth experiments are summarized in Figure 1. In Figure 1(a) we give the performance profile for each of the six solvers with 1000 variables and in Figure 1(b) for limited memory solvers `LMBM` and `L-BFGS` with 10 000 variables.

To sum up, the new limited memory bundle solver `LMBM` was usually the most efficient method tested: on 80% of the problems with $n = 1000$ and on 70% of the problems with $n = 10\ 000$ (see Figure 1). The solver `LMBM` found the local minimum in a reliable way and in our experiments it was also very robust while, for example, `PVAR` and `PBUN` were very sensitive to the choice of internal parameters. The limited memory BFGS solver `L-BFGS` was also very efficient with these large-scale



(a) $n = 1000$



(b) $n = 10\ 000$

Figure 1: Smooth problems.

smooth problems. However, it usually needed much more iterations and function evaluations and, thus, also CPU time than LMBM. All the other tested solvers were computationally inefficient with these large-scale problems (see Figure 1(a)).

Nonsmooth test problems. The solvers for nonsmooth optimization were tested with the nonsmooth minimization problems 1–10 described in Section 3. The numbers of variables used were 10, 100, and 1000, and the sizes of bundles (m_ξ) used were 10 and 100 for all the solvers. For LMBM, the maximum number of stored corrections (m_c) was set to 7 since this value seems to be suitable for the limited memory bundle method.

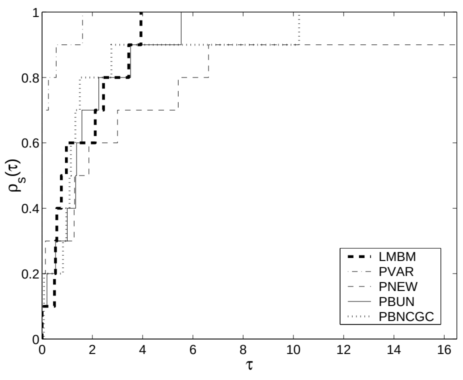
In our nonsmooth experiments, the following values of parameters were used: For convex problems 1–5, the distance measure parameter $\gamma = 0$ was used with solvers PBUN, PBNCGC, and LMBM. With PNEW, the value $\gamma = 0.001$ was used since it has to be positive (see [25]) and with PVAR, the convex version of the solver was used. For nonconvex problems 6–10, the value $\gamma = 0.50$ was used with all the solvers and we used the nonconvex version of the solver PVAR. The stopping parameter $\varepsilon = 10^{-5}$ was used in all the cases. Otherwise, the default parameters of the solvers were used.

The results of the nonsmooth experiments are summarized in Figures 2 – 4. In these figures, we give the performance profiles of the five solvers. In all cases, we give the results obtained with the smaller bundle (Figures (a)) and the results obtained with the larger bundle (Figures (b)).

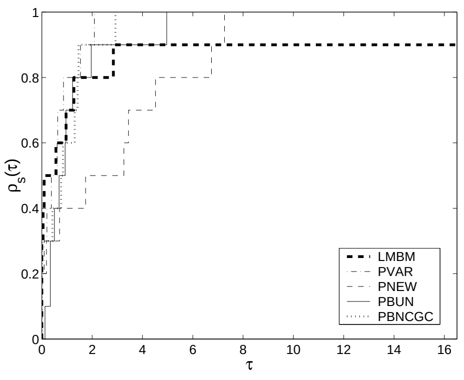
With small problems the computational times with our new solver LMBM were comparable to those of the other solvers tested (see Figure 2), and, already with 100 variables, LMBM was usually the most efficient solver (see Figure 3) although the differences were not substantial in these medium-scale cases.

With large-scale problems (see Figure 4) LMBM outperformed the original variable metric bundle solver PVAR and the bundle Newton solver PNEW in all the cases. It was the most efficient solver on 50% of the problems, and it also solved the rest of the problems really fast while, for example, with proximal bundle solvers PBUN and PBNCGC, there was a great variation in the computation times of different problems.

Besides being usually the most efficient, LMBM was also the most reliable solver (sometimes together with other solvers) in medium- and large-scale settings (see

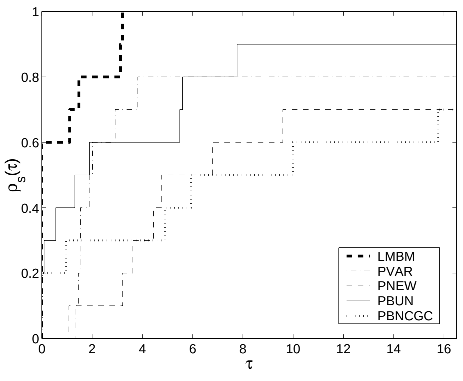


(a) $m_\xi = 10$

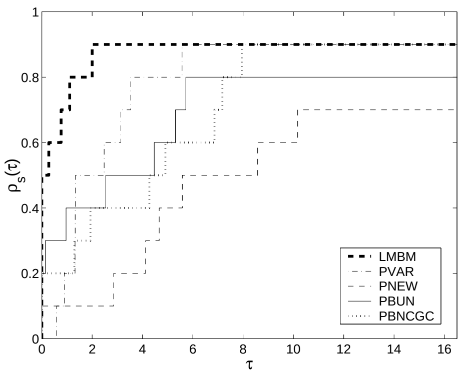


(b) $m_\xi = 100$

Figure 2: Nonsmooth problems with 10 variables.

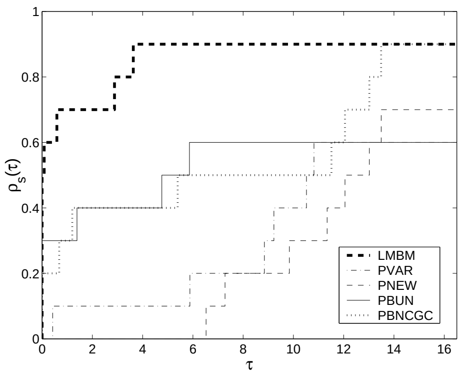


(a) $m_\xi = 10$

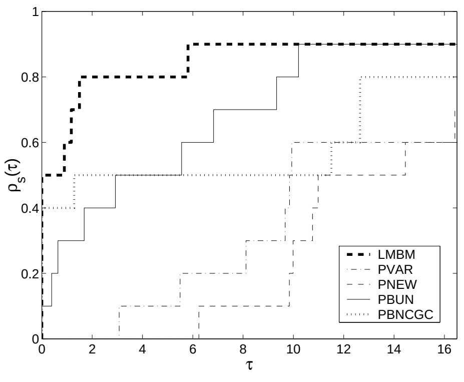


(b) $m_\xi = 100$

Figure 3: Nonsmooth problems with 100 variables.



(a) $m_\xi = 10$



(b) $m_\xi = 100$

Figure 4: Nonsmooth problems with 1000 variables.

Figures 3 and 4). However, it had some serious difficulties with problem 2: In large-scale settings the optimization was terminated before the desired accuracy was achieved (that is, the value of the objective function did not change). The probable reason for this premature termination is the piecewise linear nature of the problem. Moreover, `LMBM` had some difficulties with problem 1: The numbers of iterations and function evaluations were enormously larger ($>$ 20 000 with $n = 1000$) than those required with the other problems ($<$ 1400 in all the cases). These difficulties were quite predictable, since there exists only one nonzero component in the subgradient vector of the objective function at each iteration. In practice, this means that D_k becomes sparse and, thus, the search direction may be quite inaccurate. Also smooth limited memory methods have been reported to be best suited for problems where the Hessian matrix is not very sparse [27].

All the other problems 3–10 were solved successfully with `LMBM`. In all these cases, the subgradient vector of the objective function contains many nonzero entries and the values of these components depend on the current iteration point $\mathbf{x}_k$.

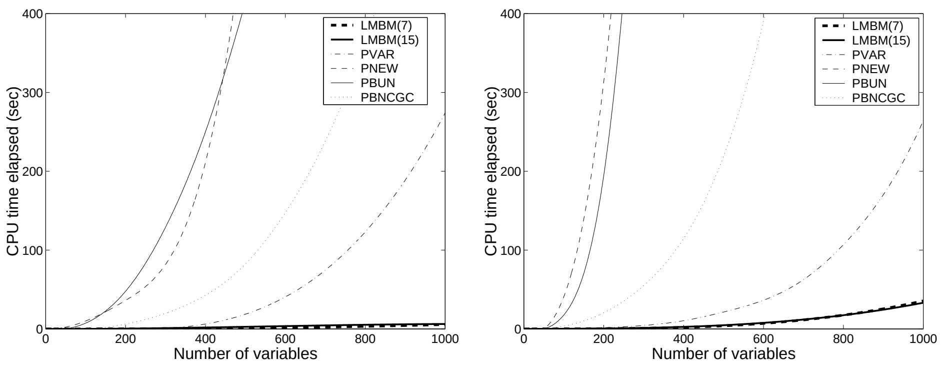
With large-scale problems, `LMBM` usually needed less iterations and function evaluations than the other solvers except the bundle Newton solver `PNEW` which, however, was the most time-consuming due to matrix operations. Thus, `LMBM` should be an efficient method also in the cases where the function and the subgradient evaluations are expensive.

The numbers of iterations and function evaluations required with `LMBM` were usually significantly smaller when the size of the bundle was large. This is due to the fact that the selection of the initial step size is more accurate when a larger bundle is used. On the other hand, each individual iteration was more costly when the size of the bundle was increased. In practice, this means that for problems with expensive objective function and subgradient evaluations, it is better to use larger bundles and, thus, fewer iterations and function evaluations.

Image restoration problem. Finally, we tested the nonsmooth optimization solvers with a convex image restoration problem (see [1]), which is a typical nonsmooth large-scale optimization problem arising in optimal control applications. We tested `LMBM` with different maximum numbers of stored corrections, that is, $m_c = 7$ and $m_c = 15$. In what follows, we denote these different modifications by `LMBM(7)` and `LMBM(15)`. The stopping parameter $\varepsilon = 10^{-4}$ was used with all the solvers. Otherwise, parameter values similar to the convex problems 1–5 were used.

In Figure 5, we give the CPU times elapsed with different number of variables. In addition, we give some more specified results for the problem with 100, 500, and 1000 variables in Table 3, where Ni and Nf denote the numbers of iterations and function evaluations used, respectively, and f denotes the value of the objective function at termination. For all the solvers, we first give the results obtained with the smaller bundle ($m_\xi = 10$) and then below the results obtained with the larger bundle ($m_\xi = 100$).

From the numerical results, we can conclude the superiority of the limited memory bundle solver `LMBM` when comparing the computational times (see Figure 5). In all the cases, it used significantly less CPU time than the other solvers. However, the accuracy of the new solver was somewhat disappointing: The minima of the objective function found with `LMBM` were usually a little bit greater than with the other solvers (especially those found with the proximal bundle solvers `PBNCGC` and `PBUN`). However, the result obtained with `LMBM` usually became more accurate when



(a) $m_\xi = 10$ (b) $m_\xi = 100$

Figure 5: CPU time elapsed for the image restoration problem.

Table 3: Results for the image restoration problem.

Solver/n	100		500		1000	
	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
PVAR	203/203	0.79758	1687/1687	4.86200	3171/3171	9.78941
	266/266	0.79750	1689/1689	4.86216	2932/2932	9.80222
PNEW	993/1138	0.79752	998/1185	4.87698	267/293	11.39220
	242/250	0.79750	773/861	4.87243	110/121	12.36375
PBUN	45232/51461	0.79750	196206/233754	4.86189	377092/480182	9.76172
	2281/2310	0.79750	71081/79933	4.86200	43136/48470	9.76818
PBNCGC	5900/5901	0.79751	25568/25569	4.86194	116058/116059	9.76172
	300/301	0.79751	3590/3591	4.86194	9913/9914	9.76184
LMBM(7)	416/474	0.79778	2479/2544	4.88256	6456/6472	9.80494
	508/537	0.79791	2851/2879	4.88373	12762/12790	9.80076
LMBM(15)	350/566	0.79778	4373/4588	4.87457	5287/5326	9.80519
	583/687	0.79760	3144/3219	4.87275	10988/11036	9.77978

the maximum number of stored corrections or the size of the bundle was increased (see Table 3). Moreover, in all cases, the results obtained with LMBM were better than those obtained with the active-set method used in [1], and both visually and with respect to the reconstruction error (see [1]) the results of LMBM were comparable to the results of proximal bundle methods. In fact, in all cases the reconstruction errors obtained with LMBM were smaller than those of other solvers tested.

Note that the results obtained with this image restoration problem differ a lot from those obtained with academic problems especially with the solver PBUN. Now PBUN used much more iterations and function evaluations than the other proximal bundle solver PBNCGC (see Table 3). It was also very time-consuming (see Figure 5) while with smooth problems and with problems 1–10 it usually was the most efficient of the bundle solvers tested except LMBM. On the other hand, the variable metric bundle solver PVAR was rather efficient, while with academic problems it was very time-consuming.

Also LMBM behaved a little bit differently with the image restoration problem than before: In most cases it used more iterations and function evaluations when the larger bundle was used. However, as said before, also the results obtained with

LMBM usually became more accurate when the size of the bundle was increased.

We conclude from these experiments that our new method was usually the most efficient method for large-scale problems. With smooth problems, LMBM was on the average one and a half times faster than the limited memory variable metric solver L-BFGS, which has been developed for smooth large-scale minimization. In addition, with 1000 variables, LMBM was on the average about 10 times faster than the fastest bundle solver PBUN and 160 times faster than the original variable metric bundle solver PVAR. For nonsmooth problems these differences were even more perceptible. For example, for the image restoration problem with 500 variables, LMBM was about 20 times faster than PVAR, 70 times faster than PBNCGC, 350 times faster than PBUN and 570 times faster than PNEW.

5 CONCLUSIONS

In this paper, we have introduced a new limited memory bundle method for nonsmooth large-scale optimization. This method is intended to fill the gap, which exist in the field of nonsmooth optimization with large numbers of variables. We have tested the performance of the new method with different minimization problems. The numerical experiments confirm that the new method is efficient and reliable for both smooth and nonsmooth optimization problems. With large numbers of variables it usually used significantly less CPU time than the other solvers tested.

Our numerical experiments showed that the limited memory bundle method works well for both convex and nonconvex minimization problems. Yet, it is best suited for problems with dense subgradient vectors where components depend on the current iteration point.

With smooth problems and with academic nonsmooth problems, the accuracy of the new solver was comparable to the other solvers tested. However, with the nonsmooth image restoration problem the minima found with the limited memory bundle solver were often slightly greater than those of the other solvers and with a large number of variables some inaccurate results occurred. Thus, some further research is required.

Possible areas of future development include the following: alternative ways of scaling and skipping the updates (especially, the SR1 update), constraint handling, and parallelization.

Acknowledgements

The authors would like to thank Dr. Ladislav Lukšan and Dr. Jan Vlček for the permission to use and modify their variable metric bundle software to make the method suitable for large-scale optimization.

This work was financially supported by COMAS Graduate School of the University of Jyväskylä and Academy of Finland grants #65760 and #104641. The computational resources used in experiments were provided by CSC, the Finnish IT Center for Science.

References

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

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

[3] C. Lemaréchal (1989). Nondifferentiable optimization. In: G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J Todd (Eds.), *Optimization*, pp 529–572. North-Holland, Amsterdam.

[4] R. Fletcher (1987). *Practical Methods of Optimization.* John Wiley and Sons, Chichester, second edition.

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

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

[7] M.M. Mäkelä (2002). Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Software*, 17(1), 1–29.

[8] F.H. Clarke (1983). *Optimization and Nonsmooth Analysis.* Wiley-Interscience, New York.

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

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

[11] N.Z. Shor (1985). *Minimization Methods for Non-Differentiable Functions.* Springer-Verlag, Berlin.

[12] R.H. Byrd, J. Nocedal, and R.B. Schnabel (1994). Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63, 129–156.

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

[14] A. Griewank and Ph.L. Toint (1982). Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik*, 39, 119–137.

[15] Ph.L. Toint (1977). On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation*, 31(140), 954–961.

[16] T.G. Kolda, D.P. O'Leary, and L. Nazareth (1998). BFGS with update skipping and varying memory. *SIAM Journal on Optimization*, 8(4), 1060–1083.

[17] M. Haarala, K. Miettinen, and M.M. Mäkelä (2003). Limited memory bundle method for large-scale nonsmooth optimization: Convergence analysis. Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B 12/2003 University of Jyväskylä, Jyväskylä.

[18] H. Schramm (1989). *Eine Kombination von Bundle- und Trust-Region-Verfahren zur Lösung nichtdifferenzierbarer Optimierungsprobleme.* PhD thesis, Bayreuther Mathematische Schriften, No. 30, Universität Bayreuth.

[19] K.C. Kiwiel (1989). An ellipsoid trust region bundle method for nonsmooth convex optimization. *SIAM Journal on Control and Optimization*, 27, 737–757.

[20] R.S. Womersley (1981). *Numerical Methods for Structured Problems in Nonsmooth Optimization.* PhD thesis, Department of Mathematics, University of Dundee.

[21] C. Charalambous and A.R. Conn (1978). An efficient method to solve the minimax problem directly. *SIAM Journal on Numerical Analysis*, 15, 162–187.

[22] A. Grothey (2001). *Decomposition Methods for Nonlinear Nonconvex Optimization Problems.* PhD thesis, University of Edinburgh.

[23] A.R. Conn, N.I.M. Gould, and Ph.L. Toint (1988). Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50(182), 399–430.

[24] N. Gupta (1985). *A Higher than First Order Algorithm for Nonsmooth Constrained Optimization.* PhD thesis, Washington State University.

[25] L. Lukšan and J. Vlček (1998). A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming*, 83, 373–391.

[26] L. Lukšan and J. Vlček (2000). NDA: Algorithms for nondifferentiable optimization. Technical Report 797, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague.

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

[28] L. Lukšan and J. Vlček (1999). Sparse and partially separable test problems for unconstrained and equality constrained optimization. Technical Report 767, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague.

[29] E.D. Dolan and J.J. Moré (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91, 201–213.