



Napsu Karmitsa

LMBM — FORTRAN Subroutines for Large-Scale Nonsmooth Minimization: User's Manual

TURKU CENTRE *for* COMPUTER SCIENCE

TUCS Technical Report
No 856, December 2007



LMBM — FORTRAN Subroutines for Large-Scale Nonsmooth Minimization: User's Manual

Napsu Karmitsa

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

TUCS Technical Report
No 856, December 2007

Abstract

LMBM is a limited memory bundle method for large-scale nonsmooth, possibly nonconvex, optimization. It is intended for problems that are difficult or even impossible to solve with classical gradient-based optimization methods due to nonsmoothness and for problems that can not be solved efficiently with standard nonsmooth optimization methods (like proximal bundle and bundle trust methods) due to high dimension and/or nonconvexity. LMBM can also be used for solving large smooth problems in which information on the Hessian matrix is difficult to obtain. The algorithm to be described is implemented in FORTRAN77. Beside of the description of the method and the code, some results from numerical experiments are given that demonstrate the efficiency of the algorithm.

Keywords: Nondifferentiable programming, large-scale optimization, bundle methods, limited memory methods, algorithms.

TUCS Laboratory
Applied Mathematics

1 Introduction

The purpose of the limited memory bundle algorithm (LMBM) is to solve general, possibly nonconvex, large-scale nonsmooth (nondifferentiable) unconstrained optimization problems

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

where the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is supposed to be locally Lipschitz continuous (see e.g., [17]) and the number of variables n is supposed to be large. At every point $\mathbf{x} \in \mathbb{R}^n$, the user is supposed to supply the value of the objective function $f(\mathbf{x})$ and an arbitrary subgradient $\xi \in \mathbb{R}^n$ from the subdifferential [2]

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

where “conv” denotes the convex hull of a set. Nevertheless, the knowledge of the structure of the problem or its Hessian (does not need to exist) are not required and, thus, LMBM can be used to solve various types of problems. The algorithm is described in detail in [5, 6, 7]. Here, in Figure 1, a simple flowchart of LMBM is given to point out the basic ideas.

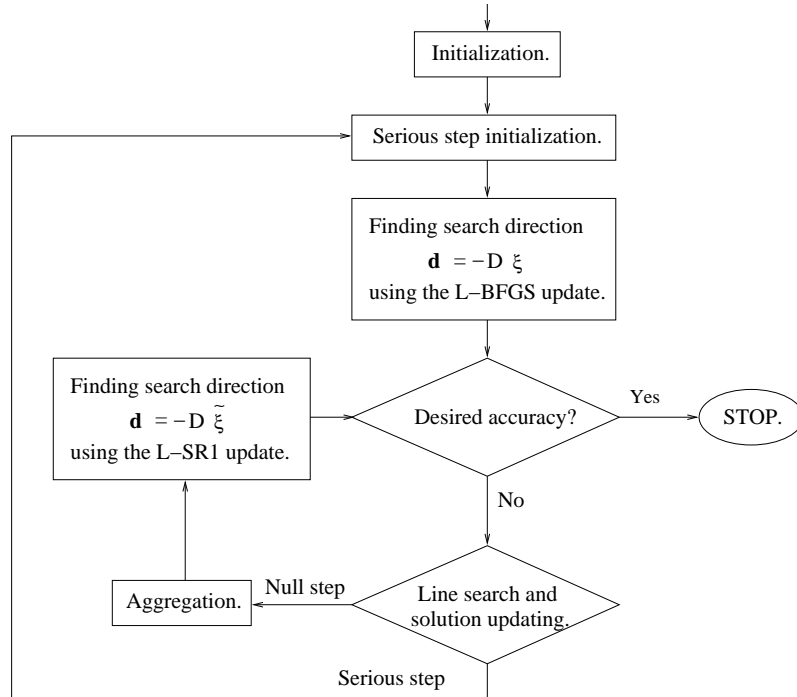


Figure 1: Flowchart of LMBM .

The idea of LMBM is to combine the variable metric bundle methods [13, 22] for small- and medium-scale nonsmooth optimization with the limited memory variable metric methods (see, e.g., [1, 18]) for large-scale smooth optimization. To be precise, LMBM exploits the ideas of the variable metric bundle methods, namely the utilization of null steps, simple aggregation of subgradients, and the subgradient locality measures, but the search direction is calculated using a limited memory approach. Therefore, the time-consuming quadratic direction finding problem appearing in standard bundle methods (see, e.g., [10, 17, 20]) need not to be solved and the number of stored subgradients (i.e., the size of the bundle, m_ξ) is independent of the dimension of the problem. Furthermore, the method uses only few vectors to produce the variable metric updates D (see Figure 1) that, in smooth case, represent the approximation of the inverse of the Hessian matrix. Thus, it avoids storing and manipulating large matrices as is the case in variable metric bundle method [13, 22].

For direction determination, LMBM uses the original subgradient ξ after serious steps and the aggregate subgradient $\tilde{\xi}$ after null steps (see Figure 1). The usage of null steps gives further information about the nonsmooth objective in the case the search direction d is not “good enough”. In its turn, the subgradient locality measure β , which generalizes the linearization error for nonconvex function (see, e.g., [17]), is used to approximate the accuracy of the current bundle. That is, to approximate how much the subgradient calculated at a new auxiliary point deviates from being member of the subdifferential of the current iteration point. This together with a simple aggregation of subgradients (see [5, 6, 7] for details) is used to guarantee the global convergence of the method.

In LMBM both the limited memory BFGS and the limited memory SR1 update formulae are used in the calculations of the search direction and the aggregate values (see Figure 1). These limited memory variable metric updates are presented in the compact matrix form originally described in [1]. The idea of limited memory matrix updating is that, instead of storing large $n \times n$ -matrices D , we store a certain (usually small) number 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 pair is deleted to make room for new one.

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., [4]). However, both the storage space required and the number of operations needed in the calculations are significantly smaller. Namely, LMBM requires roughly $n(11 + 2m_\xi + 2\hat{n}_u)$ storage locations and the number of operations needed is $O(n\hat{n}_u)$. Here \hat{n}_u denotes the maximum number of stored correction pairs. The best thing is that the user can control the amount of storage and operations required by selecting the size of the bundle m_ξ and the number of stored correction pairs \hat{n}_u . In both cases quite small values, say $2 \leq m_\xi \leq 10$ and $3 \leq \hat{n}_u \leq 20$, are recommended.

In our implementation of LMBM the number of stored correction pairs may change during the computation. This means that the optimization can be started with a small, say \hat{m}_c , number of stored correction pairs (values on range $3 \leq \hat{m}_c \leq 15$ are recommended) and when the current (intermediate) solution point is closer to the optimal point, \hat{m}_c may be increased until some predefined upper limit \hat{m}_u is achieved. The aim of this adaptability is to improve the accuracy of the method without losing much from efficiency, that is, without increasing computational costs too much.

To the best of my knowledge LMBM is the only bundle based solver available for general nonsmooth large-scale optimization. The advantages of LMBM are that the code is easy to use and the user need not supply information of the Hessian (it does not need to exist) or the structure of the objective function but only the value $f(\mathbf{x})$ and one arbitrary subgradient $\xi \in \partial f(\mathbf{x})$ at each point \mathbf{x} . Moreover, the storage requirement of the algorithm is modest and can be controlled by the user and, finally, the cost of iteration is low.

The drawbacks of the algorithm are that it is not very rapidly convergent and it sometimes fails to obtain high accuracy in the solution, especially, if the subgradient of the objective function is sparse (i.e., almost all components of the subgradient vector are zero).

The purpose of this manual is to provide a user's introduction to LMBM code. It starts by offering the easiest way to use the code: by using the default parameters. After that a more detailed description of the termination properties and the selection of parameters will be given. Finally, an example problem and some results from numerical experiments will be presented.

2 Getting Started

The basic LMBM (unconstrained version) is available online at

<http://napsu.karmita.fi/lmbm/>

The code is written in FORTRAN77 with double precision arithmetics. The software includes the following modules (or files):

<code>t_lmbm.f</code>	– test program for limited memory bundle method.
<code>lmbm.f</code>	– limited memory bundle method.
<code>lmsub.f</code>	– subroutines for limited memory bundle method.
<code>matcal.f</code>	– matrix and vector calculus.
<code>Makefile</code>	– makefile.
<code>tnsunc.f</code>	– large-scale nonsmooth test problems (for testing purposes).

At the beginning of each file, there is a list of the subroutines and functions included to that file. Moreover, at the beginning of each subroutine, there is a description of parameters used in the routine (at least those needed in the calling sequence). The types of the parameters (or arguments) are introduced with two letters. The first letter is either I for integer arguments or R for double precision real arguments. The second letter specifies whether the argument must have a value defined on the entry to the subroutine (I), whether it is a value which will be returned (O), or both (U), or whether it is an auxiliary value (A). Note that also the input arguments of the types “II” and “RI” can be changed on output under some circumstances: especially, if improper input values are given or if set zero. In the latter case the default values will be used (if applicable).

The easiest way to use the code is to edit test program TLMBM (in file `tlmbm.f`), the user supplied subroutine FUNDER (in file `tlmbm.f`), and `Makefile` that describes the relationship among the files in the program and provides commands to compile and link the program. Compiling the code can then be done simply by typing `make` and running the program by typing `tlmbm`¹.

Test program TLMBM calls subroutine LMBMU (in file `lmbm.f`) that is the initialization subroutine for LMBM code. The calling statement of LMBMU is

```
CALL LMBMU(N,NA,MCU,MC,NW,X,F,RPAR,IPAR,IOUT,
&          TIME,RTIM,W)
```

and following is the description of all parameters in this call:

<u>Argument</u>	<u>Type</u>	<u>Description</u>
N	II	Number of variables. Set by the user.
NA	II	Size of the bundle (m_ξ), $NA \geq 2$. Set by the user. The range $2 \leq NA \leq 10$ is recommended.
MCU	II	Upper limit for maximum number of stored correction pairs (\hat{m}_u), $MCU \geq 3$. Set by the user. The range $3 \leq MCU \leq 20$ is recommended.
MC	IU	Maximum number of stored correction pairs (\hat{m}_c), $MC \leq MCU$. On input: Initial maximum number of stored correction pairs. Set by the user. Also the initial value may be altered by the routine if either $MC \geq MCU$ or $MC \leq 0$. In the first case $MC = MCU$ and in the latter case $MC = 3$. The range $3 \leq MC \leq 15$ is recommended. On output: Number of correction pairs used at the

¹To use `Makefile` to delete the executable file `tlmbm` and all the object files `*.o` created by `make` command, type: `make clean`.

			solution.
X(N)	RU		Vector of variables (N-dimensional array). On input: Starting point for the optimization. Set by the user. On output: Final solution.
F	RO		Value of the objective function at the final solution.
RPAR(8)	RI		Real parameters (8-dimensional array). Set by the user (see Section 4). With choice $RPAR(I) = 0$ ($I = 1, \dots, 8$), the default values for parameters will be used.
IPAR(7)	II		Integer parameters (7-dimensional array). Set by the user (see Section 4). With choice $IPAR(I) = 0$ ($I = 1, \dots, 7$), the default values for parameters will be used.
IOUT(3)	IO		Output integer parameters (3-dimensional array): IOUT(1) – Number of used iterations. IOUT(2) – Number of used function evaluations. IOUT(3) – Cause of termination (see Section 3).
TIME	RI		Maximum CPU time in seconds (REAL argument). Set by the user. If $TIME \leq 0.0$ the maximum time is ignored.
RTIM(2)	RO		Time vector (2-dimensional REAL array). On output: $RTIM(1)$ (i.e., the first component of the array) contains the CPU time used (in seconds).
NW	II		Dimension of the work vector W: $NW \geq 1 + 9N + 2N \cdot NA + 3NA + 2N(MCU + 1) + 3(MCU + 2)(MCU + 1)/2 + 9(MCU + 1)$
W(NW)	RA		Array used as a workspace. This array must not be altered by the user.

REMARK. The input parameters $RPAR(I)$ ($I = 1, \dots, 8$) and $IPAR(I)$ ($I = 1, \dots, 7$) can be set to zero to obtain default parameters to be used. However, if the objective function is nonconvex, the value $RPAR(6)$ should be greater than 0.01. Moreover, parameter $IPAR(5)$ defines the standard output of the code. With value $IPAR(5) = 0$ only the error messages will be printed while, for instance, with value $IPAR(5) = 1$ the code prints the basic details in the final solution. For more details of this (and other) parameter(s) see Section 4.

In addition to the main program that calls subroutine LMBMU, the code requires the user supplied subroutine FUNDER that defines the value of the objective function and its (arbitrary) subgradient at any given point. An example of this routine using the collection of unconstrained nonsmooth test problems [6] (in file `tnsunc.f`) is given in file `t1mbm.f`. This subroutine should have the form

SUBROUTINE FUNDER(N, X, F, G, ITERM)

with the parameters defined as follows:

<u>Argument</u>	<u>Type</u>	<u>Description</u>
N	II	Number of variables.
X(N)	RI	Vector of variables (N-dimensional array).
F	RO	Value of the objective function at X.
G(N)	RO	Subgradient of the objective function at X (N-dimensional array).
ITERM	IO	Return value: 0 – Everything is ok. -3 – Failure in function or subgradient calculations.

Besides of these subroutines the code uses a REAL FUNCTION `ETIME` that determines the CPU time used and, thus, enables the termination after user specified maximum time (`TIME`). Function `ETIME` is a FORTRAN library routine available for most computing systems (for some f77-compilers the `+U77` option may be required). If `ETIME` is not available (or a better local timing system exists) the user will have to comment² (or rewrite) the line calling this function. That is, one line in subroutine `GETIME` that is the last subroutine in file `lmsub.f`. Moreover, if no other timing system exists, the maximum time parameter `TIME` should be set to zero.

The progress of LMBM can be monitored using standard output printed to the screen. The following is the sample output of the program with `IPAR(5) = 1`:

```
NIT= 128  NFE= 244  F= 0.88570382E-07
WK= 0.1932E-06  QK= 0.9660E-07  ITERM= 1
```

Here, `NIT` and `NFE` are the number of iterations and function evaluations used, `F` is the value of the objective function at the termination, `WK` and `QK` are the values of the stopping parameters, and `ITERM = IOUT(3)` is the cause of termination (see Section 3). Using `IPAR(5) = 4` similar information is printed out after each iteration. For more details of printing options see Section 4.

3 Termination and Error Messages

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 in nonsmooth case when we replace the gradient by an arbitrary subgradient. Due to the subgradient aggregation, we have

²Type letter “c” at the beginning of each line.

quite a useful approximation to the gradient at our disposal, namely the aggregate subgradient $\tilde{\xi}$. However, as a stopping criterion, the direct test $\|\tilde{\xi}\| < \varepsilon$, for some $\varepsilon > 0$, is often too uncertain. Therefore, we use the term $\tilde{\xi}^T D \tilde{\xi} = -\tilde{\xi}^T \mathbf{d}$ and the aggregate subgradient locality measure $\tilde{\beta}$ to improve the accuracy of $\|\tilde{\xi}\|$ ^{3,4}. Hence, the stopping parameter WK is defined by

$$\text{WK} = -\tilde{\xi}^T \mathbf{d} + 2\tilde{\beta}. \quad (2)$$

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 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 QK, which does not depend on the matrix D . The second stopping parameter is similar to that in proximal bundle methods (see, e.g., [17]), that is,

$$\text{QK} = \frac{1}{2} \tilde{\xi}^T \tilde{\xi} + \tilde{\beta}. \quad (3)$$

Now, the stopping criterion is given by:

If $\text{WK} < \text{RPAR}(4)$ and $\text{QK} < \text{RPAR}(5)$,
for given $\text{RPAR}(4), \text{RPAR}(5) > 0$, then stop.

In addition to this stopping criterion controlled by parameters $\text{RPAR}(4)$ and $\text{RPAR}(5)$, the code is designed to terminate when the change in the objective function values is sufficiently small (controlled by parameters $\text{RPAR}(1)$, $\text{RPAR}(2)$, and $\text{IPAR}(4)$) and when the maximum number of iterations ($\text{IPAR}(2)$), function calls ($\text{IPAR}(3)$) or the maximum time (TIME) is exceeded. Moreover, the code may terminate because of detected error. In all the cases the cause of termination will be returned in the final output parameter $\text{IOUT}(3)$ (i.e., in the third component of the array). This parameter returns an integer with the following meaning:

IOUT(3) Cause of termination

- 1 The problem has been solved with desired accuracy. That is, $\text{WK} \leq \text{RPAR}(4)$ and $\text{QK} \leq \text{RPAR}(5)$, where WK and QK are defined by (2) and (3), respectively .
- 2 Changes in the objective function values are sufficiently small (in serious steps). That is, $|\mathbf{F} - \mathbf{F}_{old}| < \text{RPAR}(1)$ in $\text{IPAR}(4)$

³We recall that D is the current variable metric update that, in smooth case, represent the approximation of the inverse of the Hessian matrix and \mathbf{d} is the search direction.

⁴The aggregate values $\tilde{\xi}$ and $\tilde{\beta}$ are computed only if the last step was a null step. Otherwise, we set $\tilde{\xi} = \xi \in \partial f(\mathbf{x})$ and $\tilde{\beta} = 0$.

- subsequent iterations.
- 3 The change in the objective function value is sufficiently small (in serious step). That is, $|F - F_{old}| < RPAR(2) \cdot SMALL \cdot \max(|F_{old}|, |F|, 1)$, where $SMALL$ is the smallest positive number such that $1.0 + SMALL > 1.0$ (automatically generated by the code).
 - 4 Number of function calls $> IPAR(3)$.
 - 5 Number of iterations $> IPAR(2)$.
 - 6 Time limit $TIME$ exceeded.
 - 7 The value of the objective function F is lower than the minimum acceptable function value $RPAR(3)$.
 - 1 Internal error: two consecutive restarts⁵ or $TMAX < TMIN$ in two subsequent iterations (here, $TMIN = 10^{-12}$ is an internal parameter and $TMAX$ that depends on user specified parameter $RPAR(8)$ is generated by the code). This error may be prevented by more careful choosing of the parameter $RPAR(8)$.
 - 2 Internal error: number of restarts is greater than the maximum number of restarts $MAXNRS$ ($MAXNRS = 2000$ is an internal parameter).
 - 3 Failure in function or subgradient calculations (assigned by the user).
 - 4 Internal error: Failure in attaining the demanded accuracy. That is, $|WK - WK_{old}| \leq SMALL$ in $MAXEPS$ subsequent null steps ($MAXEPS = 20$ is an internal parameter).
 - 5 Invalid input parameters. If $IPAR(5) \geq 0$ the verbose error message specifying the error will be printed.
 - 6 Not enough working space. The dimension NW of the work vector W should be increased (see Section 2 for proper value).

REMARK. During computations the final output parameter $IOUT(3)$ is called $ITERM$ and it should be equal to zero. Otherwise, the code will terminate. Thus, in subroutine $FUNDER$ (supplied by the user) the parameter $ITERM$ should be set to zero (instead of one) if everything is alright.

4 Selection of Parameters

The code includes two arrays of input parameters $RPAR$ and $IPAR$. In this section these parameters are introduced and some advice for the selection of these parameters are given.

The real parameters $RPAR(I)$, $I = 1, \dots, 8$, have the following meanings:

⁵We restart the computation (i.e., the number of stored correction pairs is reset to zero) in the case the direction vector d is almost orthogonal to the aggregate subgradient ξ .

<u>Argument</u>	<u>Description</u>
RPAR(1)	Tolerance for changes in function values. RPAR(1) is used to determine whether a significant progress in serious steps occurs or not (see IOU(3) = 2). Choosing RPAR(1) ≤ 0 forces the default value RPAR(1) = 10^{-8} instead.
RPAR(2)	Second tolerance for changes in function values (see IOU(3) = 3). Note that with this tolerance, the computation will terminate due to just one sufficiently small change in the function values after a serious step. Thus, to shun premature terminations very large values of this parameter should be avoided (note however, that this parameter is multiplied by SMALL). This parameter and the corresponding termination test may be ignored by choosing RPAR(2) < 0. Choosing RPAR(2) = 0 forces the default value RPAR(2) = 10^4 instead.
RPAR(3)	Minimum acceptable function value. Choosing RPAR(1) = 0 forces the default value RPAR(1) = 10^{-60} instead.
RPAR(4)	Tolerance for the first termination parameter WK (see IOU(3) = 1). Choosing RPAR(4) ≤ 0 forces the default value RPAR(4) = 10^{-6} instead.
RPAR(5)	Tolerance for the second termination parameter QK (see IOU(3) = 1). Prevents the premature terminations arising from the limited memory matrix updating. This second termination criterion is somewhat difficult to satisfy and, thus, the values larger or equal to RPAR(4) are recommended. Choosing RPAR(5) ≤ 0 forces the default value RPAR(5) = RPAR(4) instead.
RPAR(6)	Distance measure parameter. Used for the calculations of the subgradient locality measure. Small values on range $0 - 10^{-4}$ are recommended for convex problems while larger values on range $0.01 - 1.0$ are recommended for nonconvex problems. Choosing RPAR(6) < 0 forces the default value RPAR(6) = 0.5 instead.
RPAR(7)	Line search parameter; $0 < \text{RPAR}(7) < 0.25$. Parameter RPAR(7) affects on the acceptance of new serious step: the smaller RPAR(7) leads for smaller difference in the function values to be acceptable. Choosing RPAR(7) ≤ 0 forces the default value RPAR(7) = 10^{-4} instead.
RPAR(8)	Maximum step size, $1 < \text{RPAR}(8)$. Parameter RPAR(8) reduces the step size such that it plays an important role in the vicinity of a nonsmooth point. It may influence the efficiency of the method considerably and, thus, it should be tuned carefully. Choosing RPAR(8) ≤ 0 forces the default value RPAR(8) = 1.5 instead.

The integer parameters $\text{IPAR}(I)$, $I = 1, \dots, 7$, have the following meanings:

<u>Argument</u>	<u>Description</u>
$\text{IPAR}(1)$	Exponent for distance measure. Used for the calculations of the subgradient locality measure. Choosing $\text{IPAR}(1) \leq 0$ forces the default value $\text{IPAR}(1) = 2$ instead.
$\text{IPAR}(2)$	Maximum number of iterations. Choosing $\text{IPAR}(2) \leq 0$ forces the default value $\text{IPAR}(2) = 10\,000$ instead.
$\text{IPAR}(3)$	Maximum number of function evaluations. Choosing $\text{IPAR}(3) \leq 0$ forces the default value $\text{IPAR}(3) = 20\,000$ instead.
$\text{IPAR}(4)$	Maximum number of iterations with changes of function values smaller than $\text{RPAR}(1)$ (see $\text{IOUT}(3) = 2$). Choosing $\text{IPAR}(4) \leq 0$ forces the default value $\text{IPAR}(4) = 10$ instead.
$\text{IPAR}(5)$	Printout specification: <ul style="list-style-type: none"> -1 – No printout. 0 – Print only the error messages. 1 – Print basic details of the final solution. 2 – Print basic details of the final solution and the most serious warning messages. 3 – Print the whole final solution including the components of \mathbf{X} and the most serious warning messages. 4 – Print details (including warning messages) of every iteration excluding the components of \mathbf{X}. 5 – Print details of every iteration including the components of \mathbf{X} and the most serious warning messages.
$\text{IPAR}(6)$	Selection of the method (for more details of alternatives see Subsection 4.1): <ul style="list-style-type: none"> 0 – Limited memory bundle method. 1 – L-BFGS bundle method.
$\text{IPAR}(7)$	Selection of the scaling strategy for variable metric updates (for more details of alternatives see Subsection 4.2): <ul style="list-style-type: none"> 0 – Scaling at every iteration with $\mathbf{s}^T \mathbf{u} / \mathbf{u}^T \mathbf{u}$, where \mathbf{s} and \mathbf{u} denotes the difference on current and auxiliary iteration points and subgradients, respectively. 1 – Scaling at every iteration with $\mathbf{s}^T \mathbf{s} / \mathbf{s}^T \mathbf{u}$. 2 – Interval scaling with $\mathbf{s}^T \mathbf{u} / \mathbf{u}^T \mathbf{u}$.

- 3 – Interval scaling with $s^T s / s^T u$.
- 4 – Preliminary scaling with $s^T u / u^T u$.
- 5 – Preliminary scaling with $s^T s / s^T u$.
- 6 – No scaling.

The selection of the method and the scaling strategy will be described in next two subsections.

4.1 Selection of Method

In basic LMBM (with selection $\text{IPAR}(6) = 0$) both the limited memory BFGS and the limited memory SR1 formulae are used to update the variable metric matrices in the calculations of the search direction and aggregate values (see figure 1).

In practice, the limited memory SR1 update requires some more calculations than the limited memory BFGS update (e.g., checking the conditions needed for the positive definiteness of the update) and, further, it is quite frequently skipped⁶ due to violation of positive definiteness or an additional testing procedure that is used to guarantee the global convergence of the method [5, 7]. Therefore, a version where SR1 update is not used at all was developed [5]. This version is called L-BFGS bundle method (LBB) and it may be used by choosing $\text{IPAR}(6) = 1$.

In LBB, the limited memory BFGS update is used after a serious step and the update is simply skipped after each null step. Due to limited memory approach the matrix $D = D_{old}$ has to be recalculated also when the update is skipped. However, the additional testing procedure used with limited memory SR1 update is not needed (the convergence properties of LBB are, nevertheless, similar to those of the basic LMBM [5]) and the positive definiteness of the update can be guaranteed without additional calculations.

The procedures used in LBB are very similar to the basic LMBM. The only differences are in the limited memory matrix calculations (i.e., in the calculation of the search direction and the aggregate values). Moreover, the interval scaling strategy ($\text{IPAR}(7) = 2$, see Section 4.2) is recommended, since here this approach has found to be more advantageous than scaling at every iteration ($\text{IPAR}(7) = 0$).

4.2 Selection of Scaling

In this subsection, some possibilities to scale the limited memory variable metric updates will be described. We will concentrate on the scaling strategies similar to those developed for standard variable metric updates (see, e.g., [12, 19, 21]) and use a simple scaling matrix that is a scalar multiple of the identity matrix (i.e., ϑI).

⁶That is, the new correction pair is not stored but the same correction pairs are used as in the previous iteration.

Two different scaling parameters ϑ are used with the limited memory BFGS update. The first is defined by the formula

$$\vartheta = \frac{\mathbf{u}^T \mathbf{s}}{\mathbf{u}^T \mathbf{u}}, \quad (4)$$

and the second is given by

$$\vartheta = \frac{\mathbf{s}^T \mathbf{s}}{\mathbf{u}^T \mathbf{s}}. \quad (5)$$

Here, \mathbf{s} denotes the difference on current and auxiliary iteration points and \mathbf{u} denotes the difference on current and auxiliary subgradients, respectively. Both of them are obtained at the previous iteration of the algorithm (for more details, see [5, 7]).

Formulae (4) and (5) are unsuitable for scaling the limited memory SR1 update, since they might make the update formula undefined. Thus, the value $\vartheta = 1$ is always used with the limited memory SR1 update.

With the limited memory BFGS update the scaling parameters (4) and (5) are combined with the value $\vartheta = 1$ using some predefined scaling strategy controlled by parameter `IPAR(7)`. The following scaling strategies are available in the implementation of LMBM:

<u>IPAR(7)</u>	<u>Scaling strategy</u>
0, 1	<i>Scaling at every iteration</i> (AS). The scaling parameter ϑ is calculated by (4) (if <code>IPAR(7) = 2</code>) or (5) (if <code>IPAR(7) = 3</code>) at every iteration.
2, 3	<i>Interval scaling</i> (IS). The scaling parameter ϑ is calculated by (4) (if <code>IPAR(7) = 0</code>) or (5) (if <code>IPAR(7) = 1</code>) at every iteration. If the scaling parameter ϑ does not lie in the interval $[\vartheta_l, \vartheta_u]$, we set $\vartheta = 1$. Here, $0 < \vartheta_l < \vartheta_u$ and $\vartheta_u > 2$ are the given lower and upper bounds for the scaling parameter ϑ .
4, 5	<i>Preliminary scaling</i> (PS). The scaling parameter ϑ is calculated by (4) (if <code>IPAR(7) = 4</code>) or (5) (if <code>IPAR(7) = 5</code>) after the number of current correction pairs is reset to zero. Otherwise, we set $\vartheta = 1$. The number of stored correction pairs is reset to zero at the first iteration and in the case the direction vector \mathbf{d} is almost orthogonal to the aggregate subgradient $\tilde{\boldsymbol{\xi}}$ in which case we restart the computation.
6	<i>No scaling</i> (NS). Scaling parameter $\vartheta = 1$ at every iteration.

In interval scaling (IS), the internal parameter values $\vartheta_l = 0.6$ and $\vartheta_u = 6.0$ for the scaling parameter (4) and $\vartheta_l = 0.5$ and $\vartheta_u = 5.0$ for the scaling parameter (5) have been chosen experimentally.

The basic LMBM is obtained by selecting $\text{IPAR}(6) = 0$ and $\text{IPAR}(7) = 0$ (i.e., the default parameters). For LBB ($\text{IPAR}(6) = 1$) the interval scaling strategy $\text{IPAR}(7) = 2$ is recommended. Scaling of updates has no influence on the global convergence of LMBM (or LBB).

5 Problem Example

In this section the solution process of LMBM is demonstrated by solving a simple nonsmooth (small-scale) test problem LQ [23]. That is,

$$\begin{cases} \text{minimize} & f(\mathbf{x}) = \max\{-x_1 - x_2, -x_1 - x_2 + (x_1^2 + x_2^2 - 1)\} \\ \text{subject to} & \mathbf{x} = (x_1, x_2) \in \mathbb{R}^2. \end{cases}$$

The optimum point for this problem is $\mathbf{x}^* = (1/\sqrt{2}, 1/\sqrt{2})$ and the optimal value of the function is $f(\mathbf{x}^*) = -\sqrt{2}$. The given starting point for the optimization process is $\mathbf{x} = (-0.5, -0.5)$.

5.1 Program

In what follows we present the main program calling LMBMU and the subroutine FUNDER that defines the value and the subgradient of the objective function given above.

The parameters for the solver were fixed as follows. The size of the bundle m_ε (or NA) was set to 2, the upper limit for maximum number of stored correction pairs \hat{m}_u (or MCU) was set to 15 and the initial maximum number of stored correction pairs \hat{m}_c (or MC) was set to 7. The tolerance $\text{RPAR}(4)$ for the first termination parameter was set to 10^{-5} (i.e., we demand approximately 5 right digits in the solution) and the tolerance $\text{RPAR}(5)$ for the second parameter was set to 1.0 (different choices of this parameter are analyzed at the end of Subsection 5.2). The distance measure $\text{RPAR}(6)$ was set to 0.0 since LQ is a convex problem. Finally, the maximum time TIME was selected to be big enough (i.e., 30.0 sec) such that optimization will not terminate due to time limit. Otherwise, the default values of the parameters were used (see, Section 4). That is, we initialized these parameters to be equal to zero.

```
*****
*
*      * PROGRAM TLMBM *
*
*      Test program for limited memory bundle subroutine for
*      large-scale unconstrained nonsmooth optimization.
*
*      Napsu Karmitsa 2007
```

```

PROGRAM TLMBM

*   Parameters
    INTEGER N,NA,MCU,NW
    PARAMETER(
&      N = 2,
&      NA = 2,
&      MCU = 15,
&      NW = 1 + 9*N + 2*N*NA + 3*NA + 2*N*(MCU+1) +
&      3*(MCU+2)*(MCU+1)/2 + 9*(MCU+1))

*   Scalar Arguments
    INTEGER MC
    DOUBLE PRECISION F

*   Array Arguments
    INTEGER IPAR(7),IOUT(3)
    DOUBLE PRECISION W(NW),X(N),RPAR(8)

*   Local Scalars
    INTEGER I

*   CPU-time
    REAL TIME,RTIM(2)

*   External Subroutines
    EXTERNAL LMBMU

*   Maximum time
    TIME = 30.0E+00

*   Initial number of stored corrections
    MC = 7

*   Initiation of X
    X(1) = -0.50D+00
    X(2) = -0.50D+00

*   Choice of integer parameters
    DO 10 I = 1,7
        IPAR(I) = 0
10    CONTINUE

*   Printout specification
    IPAR(5) = 5

```

```

*      Choice of real parameters
      DO 20 I = 1,8
          RPAR(I) = 0.0D+00
20     CONTINUE

*      Desired accuracy
      RPAR(4) = 1.0D-05
      RPAR(5) = 1.0D+00

*      Solution
      CALL LBMU(N,NA,MCU,MC,NW,X,F,RPAR,IPAR,IOUT,TIME,
&          RTIM,W)

      STOP
      END

*****
*
*      * SUBROUTINE FUNDER *
*
*      Computation of the value and the subgradient of the
*      objective function LQ.
*
*      Napsu Karmita 2007
*
      SUBROUTINE FUNDER(N,X,F,G,ITERM)

*      Scalar Arguments
      INTEGER N,ITERM
      DOUBLE PRECISION F

*      Array Arguments
      DOUBLE PRECISION G(*),X(*)

*      Local Scalars
      DOUBLE PRECISION TMP

*      LQ: Function and subgradient evaluation
      F = -X(1)-X(2)
      TMP = -X(1)-X(2)+(X(1)*X(1)+X(2)*X(2)-1.0D+00)

      IF (F .GE. TMP) THEN
          G(1) = -1.0D+00
          G(2) = -1.0D+00

```

```

ELSE
  F = TMP
  G(1) = -1.0D+00 + 2.0D+00*X(1)
  G(2) = -1.0D+00 + 2.0D+00*X(2)
ENDIF

ITERM = 0

RETURN
END
*****

```

5.2 Result

The output of LMBM (with $IPAR(5) = 5$) is the following:

```

Entry to LMBM:
NIT= 1 NFE= 1 F= 0.10000000E+01 WK= 0.2000E+01 QK= 0.1000E+01
X= -0.50000000E+00 -0.50000000E+00
NIT= 2 NFE= 2 F=-0.10000000E+01 WK= 0.2000E+01 QK= 0.1000E+01
X= 0.50000000E+00 0.50000000E+00
NIT= 3 NFE= 3 F=-0.10000000E+01 WK= 0.1088E+01 QK= 0.6597E+00
X= 0.50000000E+00 0.50000000E+00
NIT= 4 NFE= 4 F=-0.12777778E+01 WK= 0.6667E+00 QK= 0.1000E+01
X= 0.63888889E+00 0.63888889E+00
NIT= 5 NFE= 5 F=-0.12777778E+01 WK= 0.3204E+00 QK= 0.2414E+00
X= 0.63888889E+00 0.63888889E+00
NIT= 6 NFE= 6 F=-0.13851247E+01 WK= 0.3429E+00 QK= 0.1000E+01
X= 0.6925624E+00 0.6925624E+00
NIT= 7 NFE= 7 F=-0.13851247E+01 WK= 0.1014E+00 QK= 0.7611E-01
X= 0.6925624E+00 0.6925624E+00
NIT= 8 NFE= 8 F=-0.14124521E+01 WK= 0.4113E-02 QK= 0.1751E+00
X= 0.7092223E+00 0.7092223E+00
NIT= 9 NFE= 9 F=-0.14139115E+01 WK= 0.3196E-02 QK= 0.1000E+01
X= 0.7069558E+00 0.7069558E+00
NIT= 10 NFE= 10 F=-0.14142063E+01 WK= 0.3878E-04 QK= 0.1716E+00
X= 0.7071156E+00 0.7071156E+00
NIT= 11 NFE= 11 F=-0.14142131E+01 WK= 0.1279E-04 QK= 0.1000E+01
X= 0.7071065E+00 0.7071065E+00
Exit from LMBM:
NIT= 12 NFE= 12 F=-0.14142136E+01
WK= 0.6301E-07 QK= 0.1716E+00 ITERM= 1
X = 0.7071068E+00 0.7071068E+00

```

Note that this standard output does not give the computational time. However, it can be printed simply by adding a line

```
PRINT*, 'Used time = ', RTIM(1)
```

at the end of main program TLMBM.

In this test run we had $\text{RPAR}(5) = 1.0$ and the problem was solved with the desired accuracy (i.e., $\text{ITERM} = 1$). If the default value $\text{RPAR}(5) = \text{RPAR}(4)$ was used, then the solver terminated after 15 iterations since the value of the objective function did not change (i.e., $\text{ITERM} = 3$) Nevertheless, the result is obtained with more that 5 right digits. In this case, the last lines of the output reads as follows:

```
Abnormal exit: The value of the function does not change.
NIT=   15  NFE=   16  F=-0.14142136E+01
      WK= 0.2146E-11  QK= 0.1000E+01  ITERM=   3
X=  0.7071068E+00  0.7071068E+00
```

On the other hand, if we ignore the termination with $\text{ITERM} = 3$ (i.e., we set $\text{RPAR}(2) < 0.0$) we again obtain $\text{ITERM} = 1$ (with approximately the same accuracy as before) but only after 64 function evaluations. Then the last lines of the output are

```
NIT=   20  NFE=   64  F=-0.14142136E+01
      WK= 0.3247E-15  QK= 0.1623E-15  ITERM=   1
X =  0.7071068E+00  0.7071068E+00
```

These examples show the difficulty in satisfying condition $\text{QK} \leq \varepsilon$ for some small $\varepsilon > 0$. However, ignoring this test may sometimes lead to premature terminations. Nevertheless, the practical experiments have shown that also in the case of $\text{ITERM} = 2$ or 3 the results are usually obtained with the desired accuracy if WK is small.

6 Numerical Experiments

In this section we compare the limited memory bundle solver LMBM to the proximal bundle solver PBNCGC . The solver PBNCGC has been used as a benchmark since the proximal bundle methods are the most frequently used bundle methods in nonsmooth optimization. The tested algorithm PBNCGC is from the software package NSOLIB (NonSmooth Optimization LIBrary, see [15]). The solver utilizes the subgradient aggregation strategy of [10] and it uses the quadratic solver QPDF4 based on the dual active-set method [11] to solve the quadratic direction finding problem. For a detailed description of the solver see [15, 16, 17].

The experiments were performed in a SGI Origin 2000/128 supercomputer (MIPS R12000, 600 Mflop/s/processor). Both the algorithms LMBM and PBNCGC were implemented in FORTRAN77 with double-precision arithmetic.

In what follows, we first compare LMBM to PBNCGC using some academic nonsmooth minimization problems. Then, we bring into comparison the different

scaling strategies and the different versions of the basic method. Finally, we analyze the performance of the solvers with two practical nonsmooth image restoration (or image denoising) problems [8, 9]. A more extensive numerical analysis concerning the performance of LMBM and its comparison with some other existing bundle methods in large-scale minimization problems can be found in [6]. Moreover, more experiments of using LMBM in image denoising applications can be found in [14].

In the test results to be reported, we say that optimization terminated successfully if for LMBM (and its different modifications) we had $\text{ITERM} = 1, 2$, or 3 (see return values of $\text{IOUT}(3)$ in Section 3) and with $\text{ITERM} = 2$ and 3 the result obtained was less than two significant digits greater than the desired accuracy of the solution. For PBNCGC the corresponding termination criteria were applied. In addition, 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 optimization failed.

The results are analyzed using the performance profiles introduced in [3]. As performance measures, we use computation times and numbers of function evaluations. 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 and reliability 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 [3].

Nonsmooth academic problems. The solvers LMBM and PBNCGC were first tested with 10 nonsmooth academic minimization problems described in [6] (the problems can be downloaded in file `tnsunc.f`). Half of these problems were convex and the rest were nonconvex. The number of variables was 1000, and the solvers were tested with relatively small sizes of the bundles, that is, $m_\xi = 10$ and $m_\xi = 100$. In what follows, we denote these different variants by LMBM(10), LMBM(100), PBNCGC(10), and PBNCGC(100). For LMBM, the following values of the parameters were used:

$$\begin{aligned} \text{MC} = 7, \quad \text{MCU} = 7, \quad \text{RPAR}(4) = 10^{-5}, \quad \text{RPAR}(5) = 10^{-5}, \quad \text{and} \\ \text{RPAR}(6) = 0.0 \text{ for convex and } \text{RPAR}(6) = 0.5 \text{ for nonconvex problems.} \end{aligned}$$

For PBNCGC the similar parameters were employed whenever applicable. Otherwise, the default parameters of the solvers were used.

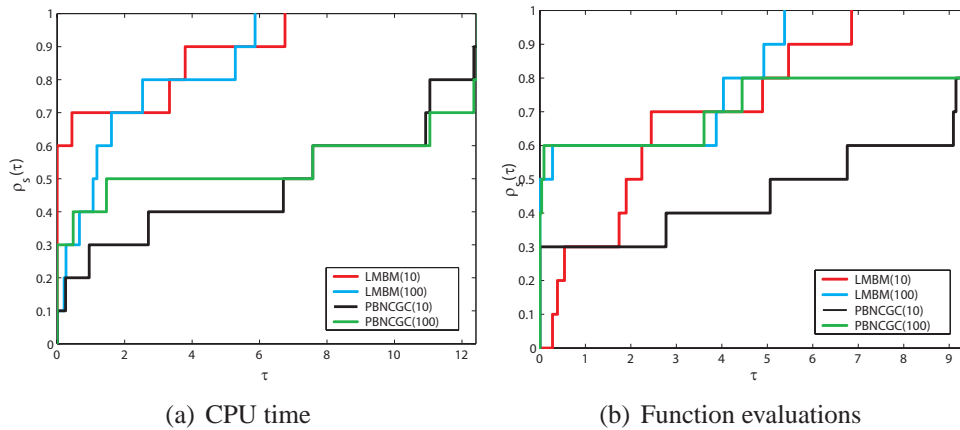


Figure 2: Nonsmooth problems with 1000 variables.

The results of the experiments are summarized in Figure 2. In Figure 2(a) we see that LMBM(10) was the most efficient solver on 60% of the problems. It also solved the rest of the problems really fast while, with PBNCGC (with both sizes of bundles) there was a great variation in the computation times of different problems. LMBM(100) was very efficient with these large-scale problems as well, although, it usually needed slightly more computation time than LMBM(10). The computation time elapsed with LMBM was on the average about 40 times shorter than that of PBNCGC .

The numbers of function evaluations used with LMBM(100) and PBNCGC(100) were approximately the same (see Figure 2(b)). With PBNCGC the required number of function evaluations was significantly larger when the size of the bundle was small (that is, $m_\xi = 10$). This was usually true also for LMBM but with LMBM the difference was not so substantial. However, 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.

Different scaling of updates. Next, we compared different scaling of limited memory variable metric updates. In order to do this, we combined the basic limited memory bundle solver LMBM(10) (i.e., $m_\xi = 10$) with the different scaling strategies (see Section 4.2). In what follows, we use the abbreviations (NS) for no scaling, (PS) for preliminary scaling, (AS) for scaling at every iteration, and (IS) for interval scaling. In addition, we denote by “1” the scaling parameter defined in (4) and by “2” the scaling parameter defined in (5). Note that with this notation, the basic limited memory bundle solver is denoted by LMBM(AS1).

The experiment was done by using a set of 20 nonsmooth academic minimization problems with 1000 variables (for details of the problems, see [5]). The parameter values were chosen similarly to previous experiment but the distance measure parameter $RPAR(6)$, which depends on the convexity of the objective

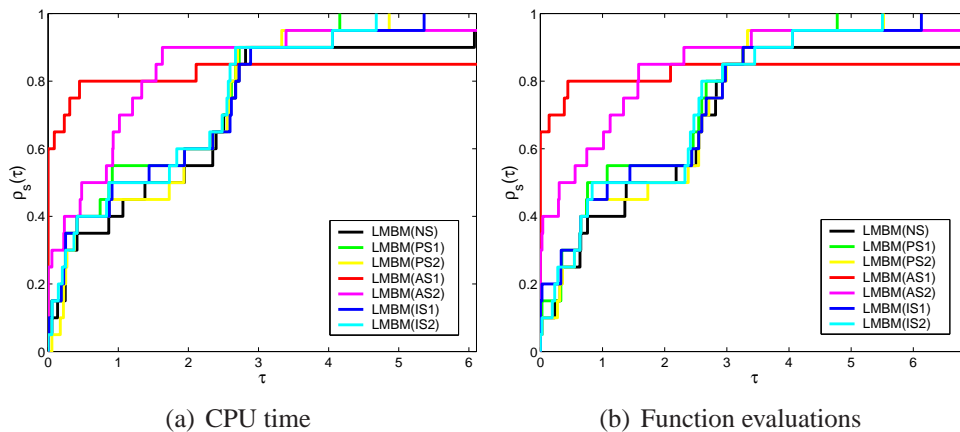


Figure 3: Results with different scaling of updates.

function, was chosen experimentally individual to the each problem. That is, all the problems were minimized with four different values of the parameter and the best results were selected to be reported here (we first checked the accuracy of the result and then the computation time elapsed). The tested values of the distance measure parameter were $RPAR(6) = 0.0, 0.25, 0.5, \text{ and } 0.9$.

The results of the experiments are summarized in Figure 3. Scaling at every iteration with the scaling parameter (4) (i.e., LMBM(AS1)) was clearly the most efficient scaling strategy tested (on 60% of problems, see Figure 3(a)). Unfortunately, it was also the most unreliably, and the solver failed to solve three of the problems when this scaling strategy was used. Note, however, that all these failures can be prevented by a suitable choice of the parameters.

Scaling at every iteration with the scaling parameter (5) (i.e., LMBM(AS2)) was the second best choice when comparing the computation times (see Figure 3(a)). Moreover, LMBM(AS2) failed to solve only one problem in the test set. However, the average computation time elapsed with LMBM(AS2) was about five times longer than that with LMBM(AS1).

All the other scaling strategies tested behaved quite similarly when compared to each other. With the (NS) strategy the solver failed to solve one problem but in all the other cases, the solver succeeded to solve all the problems in the test set. Unfortunately, the average computation times elapsed with these scaling strategies increased approximately sixfold to that used with (AS1).

Naturally, the differences between the computation times elapsed with the different scaling strategies correspond to the differences between the numbers of function evaluations required (see Figure 3).

We conclude that the different scaling of the limited memory variable metric updates may, in some cases, lead to more accurate results but it does not, in general, improve the behavior of the method. Moreover, the same kind of improvement on the accuracy properties may be obtained with a careful tuning of the parameters.

Different versions. Next, we compared the different versions of the method. As before, the experiment was done by using a set of 20 nonsmooth minimization problems with 1000 variables (for details of the problems, see [5]). In what follows we denote by LMBM the basic limited memory bundle method, LBB the limited memory BFGS bundle method and ALMBM the adaptive version of the method (i.e., the version in which the maximum number of stored correction pairs can change). With the adaptive version, we have used the initial maximum number of stored correction pairs MC equal to 7 (the same as that with the basic version LMBM) and the upper limit for the maximum number of stored correction pairs MCU equal to 50. Moreover, with LBB the interval scaling strategy ($IPAR(7) = 2$) was used. Otherwise, the parameter values similar to those used with different scaling of updates were chosen.

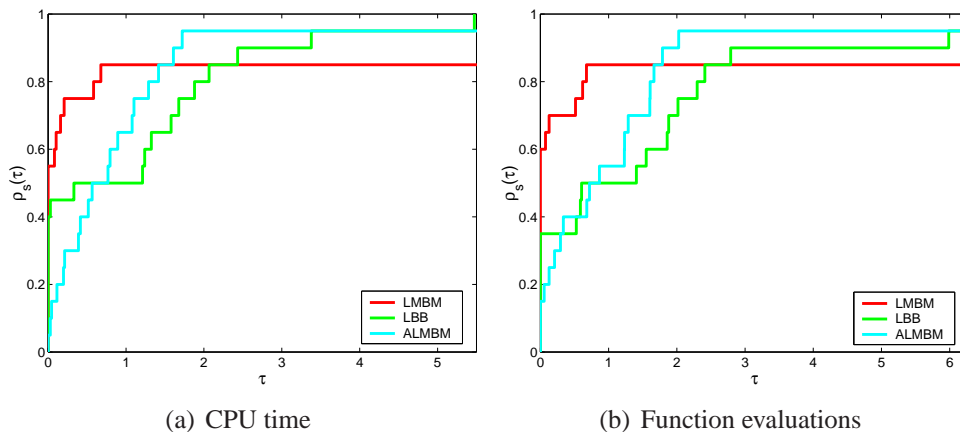


Figure 4: Results with different versions.

The results of the experiments are summarized in Figure 4. Again the basic version LMBM was usually the most efficient one (that is, on 55% of the problems) but also the most unreliable of the solvers tested (see Figures 4(a) and 4(b)). The adaptive version ALMBM succeeded to solve two of the three problems where the basic version failed but, then, it lost slightly on the efficiency. The L-BFGS bundle solver LBB succeeded to solve all the problems. However, with LBB, there was quite a big variation in the computation times (as well as in the number of function evaluations) needed for different problems and, on the average, LBB was the most time-consuming of different versions.

Image restoration problems. Finally, we tested the different modifications of LMBM and the proximal bundle solver PBNCGC with two convex image restoration problems [8, 9], which are typical nonsmooth large-scale optimization problems arising in optimal control applications. In what follows, we denote by (P1) the problem described in [9] and by (P2) the problem described in [8]. Both

of the problems are so-called noise reduction problems. The noise reduction problems are formulated as minimization problems consisting of a least squares fit and a nonsmooth bounded variational type regularization term (for more details of the formulations, see [8, 9]).

The solvers were tested with $m_\xi = 10$ for different modifications of LMBM and $m_\xi = 100$ for PBNCGC (since the previous experiments have shown that a larger bundle usually works better with PBNCGC). The stopping parameter $\text{RPAR}(4) = 10^{-4}$ was used with all the solvers and the value of the distance measure parameter $\text{RPAR}(6)$ was set to zero (since the objective functions are convex)⁷. We tested the basic version of LMBM with different maximum numbers of stored correction pairs, that is, $\text{MC} = \text{MCU} = 7$ and $\text{MC} = \text{MCU} = 15$. In what follows, we denote these different variants by LMBM[7] and LMBM[15], respectively. Otherwise, the default parameters of the solvers were used.

In Figure 5, we give the computation times elapsed for the problems with different number of variables. Furthermore, we give some more specified results for the problems with 100, 300, and 1000 variables in Tables 1 and 2, where NIT and NFE denote the numbers of iterations and function evaluations used, respectively, and F denotes the value of the objective function at termination. In addition to the results obtained in our experiment, we give (in Table 1) the reported final values of the objective function for problem (P1) obtained with the special active set method (ACS) [9]. The active set method has not been applied for problem (P2) in [8] and, thus, in Table 2, we only give the results obtained in our experiment.

Based on the numerical results, we conclude the superiority of the limited memory bundle solver LMBM and its modifications when comparing the computation times: in all the cases, they used significantly less CPU time than the proximal bundle solver PBNCGC (see Figure 5). With large number of variables, they also usually needed less iterations and function evaluations than PBNCGC (see Tables 1 and 2). However, the minima found with these solvers could be slightly greater than those of PBNCGC (especially, in case of LBB, see Tables 1 and 2). In all image restoration problems but one with 100 variables, the optimization with LMBM (and its modifications) was terminated because the value of the objective function did not change (i.e., $\text{ITERM} = 2$ or 3). Thus, even if the value of stopping parameter $\text{RPAR}(4)$ was decreased the results obtained would not become any smaller. Nevertheless, the results obtained with LMBM usually became more accurate when the maximum number of stored correction pairs was increased or, especially, when the adaptive version ALMBM was used (see Tables 1 and 2).

Moreover, in most cases, the results obtained for problem (P1) with LMBM[7], LMBM[15], and ALMBM were smaller than those obtained with the active-set method ACS used in [9] (see Table 1), and both visually and with

⁷The similar parameters for PBNCGC were used.

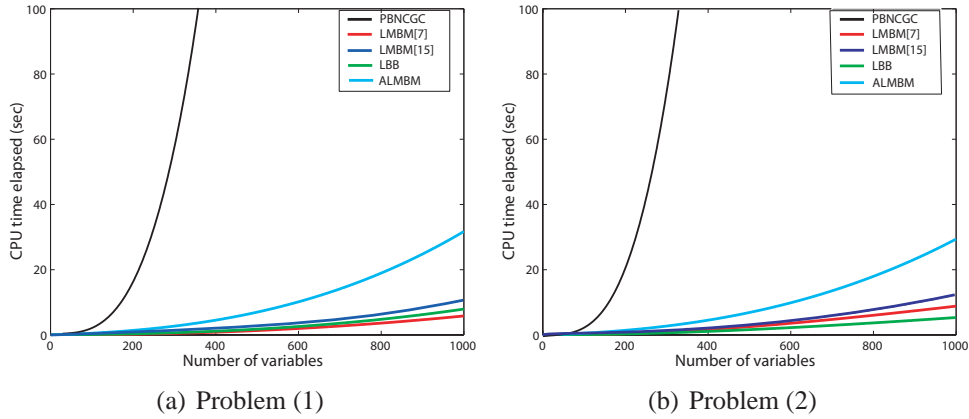


Figure 5: CPU times elapsed for the image restoration problems.

Table 1: Results for the image restoration problem (P1).

Solver/ n	100		300		1000	
	NIT/NFE	F	NIT/NFE	F	NIT/NFE	F
ACS	–	0.7981	–	2.7586	–	9.8950
PBNCGC	186/187	0.7976	1388/1389	2.7580	16777/16778	9.7617
LMBM[7]	442/669	0.7979	1707/1919	2.7666	6343/6373	9.8152
LMBM[15]	598/1451	0.7976	3980/5184	2.7632	8285/8508	9.8042
LBB	863/3714	0.7981	1819/6492	2.7682	6848/18721	9.9196
ALMBM	736/4953	0.7976	2601/5762	2.7602	11812/13326	9.7694

Table 2: Results for the image restoration problem (P2).

Solver/ n	100		300		1000	
	NIT/NFE	F	NIT/NFE	F	NIT/NFE	F
PBNCGC	249/250	0.5973	1597/1598	2.2517	17383/17384	7.9571
LMBM[7]	764/1062	0.5974	1476/1597	2.2603	10028/10073	7.9958
LMBM[15]	640/1515	0.5973	3324/4535	2.2568	9871/10054	7.9944
LBB	639/2833	0.5976	1713/6205	2.2635	4692/12305	8.0823
ALMBM	772/4721	0.5972	4814/10531	2.2536	10981/12388	7.9771

respect to the reconstruction error (that is, the average error between the true signal and the obtained result [8, 9]) all the results of LMBM[7], LMBM[15], ALMBM, and LBB were comparable to the results of the proximal bundle solver PBNCGC (for both the problems). In fact, in all large-scale cases the reconstruction errors obtained with the different modifications of the limited memory bundle method were smaller than those obtained with the proximal bundle method PBNCGC.

We can conclude that LMBM and its modifications were very efficient for large-scale nonsmooth optimization. For academic problems with 1000 variables, LMBM was on the average about 40 times faster than the proximal bundle solver PBNCGC. Moreover, LMBM found the (local) minimum in a reliable way for both convex and nonconvex optimization problems. For the demanding image restoration problems the differences in the computation times were even more perceptible: the limited memory bundle solver LMBM was about 50 times faster than PBNCGC already with 300 variables (see Figure 5).

7 Conclusions

In this paper, a limited memory bundle algorithm LMBM for nonsmooth large-scale optimization has been described. The code is written in FORTRAN77 and it is available online at

<http://napsu.karmita.fi/lmbm/>

The numerical experiments confirm that LMBM is efficient for both convex and nonconvex large-scale nonsmooth optimization problems. With large numbers of variables it used significantly less CPU time than the proximal bundle method tested.

Acknowledgements

I would like to thank Prof. Marko M. Mäkelä and Prof. Kaisa Miettinen for valuable comments and ideas throughout the development period of the code. I also would like to express my appreciation to Prof. Ladislav Lukšan and Dr. Jan Vlček for giving a permission to use and modify their variable metric bundle software PVAR to make the method suitable for large-scale optimization. The code follows many of the ideas and the style of their PVAR codes. Finally, I want to thank MSc. Tommi Ronkainen for advice concerning the computational implementations. The computational resources used in the experiments were provided by CSC, the Finnish IT Center for Science.

References

- [1] 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.
- [2] CLARKE, F. H. *Optimization and Nonsmooth Analysis*. Wiley-Interscience, New York, 1983.
- [3] DOLAN, E. D., AND MORÉ, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming* 91 (2002), 201–213.
- [4] FLETCHER, R. *Practical Methods of Optimization*, 2nd ed. John Wiley and Sons, Chichester, 1987.
- [5] HAARALA, M. *Large-Scale Nonsmooth Optimization: Variable Metric Bundle Method with Limited Memory*. PhD thesis, University of Jyväskylä, Department of Mathematical Information Technology, 2004.
- [6] 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.
- [7] HAARALA, N., MIETTINEN, K., AND MÄKELÄ, M. M. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Mathematical Programming* 109, 1 (2007), 181–205.
- [8] KÄRKKÄINEN, T., MAJAVA, K., AND MÄKELÄ, M. M. Comparison of formulations and solution methods for image restoration problems. Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B. 14/2000 University of Jyväskylä, Jyväskylä, 2000.
- [9] 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.
- [10] KIWIEL, K. C. *Methods of Descent for Nondifferentiable Optimization*. Lecture Notes in Mathematics 1133. Springer-Verlag, Berlin, 1985.
- [11] KIWIEL, K. C. A method for solving certain quadratic programming problems arising in nonsmooth optimization. *IMA Journal of Numerical Analysis* 6 (1986), 137–152.
- [12] LUKŠAN, L., AND VLČEK, J. Simple scaling for variable metric updates. Technical Report 611, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1995.

- [13] 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.
- [14] MAJAVA, K., HAARALA, N., AND KÄRKKÄINEN, T. Solving variational image denoising problems using limited memory bundle method. In *Recent Progress in Scientific Computing. Proceedings of SCPDE05*. (2007), W. Liu, M. Ng, and Z.-C. Shi, Eds., Science Press, Beijing, pp. 319–332.
- [15] MÄKELÄ, M. M. Issues of implementing a Fortran subroutine package NSOLIB for nonsmooth optimization. Technical Report 5/1993, Department of Mathematics, Laboratory of Scientific Computing, University of Jyväskylä, 1993.
- [16] 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.
- [17] MÄKELÄ, M. M., AND NEITTAANMÄKI, P. *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control*. World Scientific Publishing Co., Singapore, 1992.
- [18] NOCEDAL, J. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation* 35, 151 (1980), 773–782.
- [19] NOCEDAL, J., AND YUAN, Y. Analysis of a self-scaling quasi-Newton method. *Mathematical Programming* 61 (1993), 19–37.
- [20] 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.
- [21] SHANNO, D. F., AND PHUA, K. J. Matrix conditioning and nonlinear optimization. *Mathematical Programming* 14 (1978), 144–160.
- [22] 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.
- [23] WOMERSLEY, R. S. *Numerical Methods for Structured Problems in Nonsmooth Optimization*. PhD thesis, Department of Mathematics, University of Dundee, 1981.

TURKU
CENTRE *for*
COMPUTER
SCIENCE

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi



University of Turku

- Department of Information Technology
- Department of Mathematics



Åbo Akademi University

- Department of Computer Science
- Institute for Advanced Management Systems Research



Turku School of Economics and Business Administration

- Institute of Information Systems Sciences

ISBN ?

ISSN 1239-1891