Napsu Karmitsa | Marko M. Mäkelä

Globally Convergent Limited Memory Bundle Algorithm for Nondifferentiable Programming subject to Box Constraints

Turku Centre for Computer Science

TUCS Technical Report
No 882, March 2008

Globally Convergent Limited Memory Bundle Algorithm for Nondifferentiable Programming subject to Box Constraints

Napsu Karmitsa
Department of Mathematics
University of Turku
FI-20014 Turku, Finland
napsu@karmitsa.fi
Marko M. Mäkelä
Department of Mathematics
University of Turku
FI-20014 Turku, Finland
makela@utu.fi

TUCS Technical Report
No 882, March 2008

Abstract

Practical optimization problems often involve nonsmooth functions of hundreds or thousands of variables. As a rule, the variables in such problems are restricted to certain meaningful intervals. In the report [Haarala, Mäkelä, 2006] we have described an efficient adaptive limited memory bundle method for large-scale nonsmooth, possibly nonconvex, box constrained optimization. In this paper, a new variant of this method is proposed and its global convergence for locally Lipschitz continuous functions is proved. In addition, some numerical experiments are given in order to show the applicability of the method.

Keywords: Nonsmooth optimization, large-scale problems, bundle methods, limited memory methods, bound constraints, global convergence.

TUCS Laboratory
Applied Mathematics

1 Introduction

In this paper, a global convergence theory will be provided for a new version of adaptive limited memory bundle algorithm (LMBM-B) [13] that is developed for solving large nonsmooth (nondifferentiable) box constrained optimization problems. This kind of problems frequently appear, for instance, in many areas of industrial design, process control, and economic planning (see, e.g. [7, 27, 28, 30]), and, due to nonsmoothness, they are difficult or even impossible to solve with classical gradient-based optimization methods. On the other hand, none of the current general nonsmooth optimization solvers (especially, those capable for constraint handling) can be used efficiently when the number of variables is large, say 1000 or more. This means there is an evident need for methods able to solve large, possible nonconvex, nonsmooth, (box) constrained optimization problems.

We consider the problem

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

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is supposed to be locally Lipschitz continuous and the number of variables n is supposed to be large. Moreover, the vectors $\boldsymbol{x}^l$ and $\boldsymbol{x}^u$ representing the lower and the upper bounds on the variables, respectively, are fixed and the inequalities in (1) are taken component-wise.

Nowadays different variants of bundle methods (see, e.g. [17, 19, 23, 25, 32]) are recognized the most effective and reliable methods for solving nonsmooth optimization problems. Their basic assumption is that at every point $\boldsymbol{x} \in \mathbb{R}^n$, we can evaluate the value of the objective function $f(\boldsymbol{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \mathbb{R}^n$ from the subdifferential [6]

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

where "conv" denotes the convex hull of a set. The idea of bundle methods is to approximate the subdifferential or, to be exact, the Goldstein ε-subdifferential (see, e.g. [25])

$$\partial_\varepsilon^G f(\boldsymbol{x}) = \text{conv}\{ \partial f(\boldsymbol{y}) \mid \boldsymbol{y} \in \bar{B}(\boldsymbol{x}; \varepsilon) \}$$

by gathering subgradients from previous iterations into a bundle. Here, we have $\boldsymbol{y} \in \mathbb{R}^n$, $\varepsilon \geq 0$, and $\bar{B}(\boldsymbol{x}; \varepsilon)$ denotes a closed ball with center $\boldsymbol{x}$ and radius ε. Note that $\partial f(\boldsymbol{x}) \subseteq \partial_\varepsilon^G f(\boldsymbol{x})$ for all $\varepsilon \geq 0$.

While standard bundle methods are very efficient for small- and medium-scale problems, they are not, in general, competent in large-scale settings (see, e.g. [2, 14, 18]). In [12, 14, 15] we have proposed a limited memory bundle method (LMBM) for general, possibly nonconvex, nonsmooth large-scale unconstrained optimization. The idea of LMBM is to combine the variable metric bundle methods [22, 33] for small- and medium-scale nonsmooth optimization with the limited memory variable metric methods (see, e.g. [5, 11, 21, 29]) for large-scale smooth optimization. LMBM exploits the ideas of the variable metric bundle methods, namely the utilization of null steps and simple aggregation of subgradients, 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. [19, 25, 32]) need not to be solved and the number of stored subgradients (i.e. the size of the bundle) is independent of the dimension of the problem. Furthermore, LMBM uses only few vectors to represent the variable metric updates and, thus, it avoids storing and manipulating large matrices as is the case in variable metric bundle methods [22, 33].

The basic LMBM [14, 15] as well as its adaptive version [12] are only suitable for unconstrained problems. In [13], a new variant of the method, LMBM-B, suitable for solving box constrained problems was introduced. In LMBM-B the constraint handling is based on subgradient projection and dual subspace minimization and it is adopted from the smooth limited memory BFGS method for box constrained optimization [4]. Although numerically very efficient, the method described in [13] is not necessarily globally convergent in nonsmooth case.

In order to prove the global convergence of LMBM-B some modifications had to be made to the algorithm introduced in [13]. Namely, we have included so-called stark projections in aggregation procedure to guarantee the convergence of aggregate subgradients to zero. Moreover, we have added some corrections to limited memory matrices to preserve sufficient positive definiteness and boundedness of these matrices whenever necessary, and finally we took along a slightly modified line search procedure. Although this may sound like an easy task several open question had to be answered and many implementational challenges had to be solved before preserving even a hint of the efficiency of the previous version together with the convergence properties.

The rest of this paper is organized as follows. In Section 2, we describe an algorithm LMBM-B for box constrained optimization. We start by giving a brief introduction to the method. After that, we describe in detail the algorithm, limited memory matrix updating, identification of the active set, and the subspace minimization procedure used. We also give a special line search procedure, which is modified from that used in [13, 15]. In Section 3, we prove the global convergence of the method for locally Lipschitz continuous objective functions that are not necessary differentiable or convex. Some preliminary results of numerical experiments are presented in Section 4 and, finally, in Section 5, we conclude the paper.

In order to make this paper more self-contained, we recall (sometimes with the same words) many formulae and procedures given in the previous papers [12, 13, 14, 15].

2 Method

In this section, we describe the adaptive limited memory bundle method LMBM-B for box constrained large-scale nonsmooth optimization. We start by giving a simple flowchart (in Figure 1) to point out the basic ideas of the algorithm.

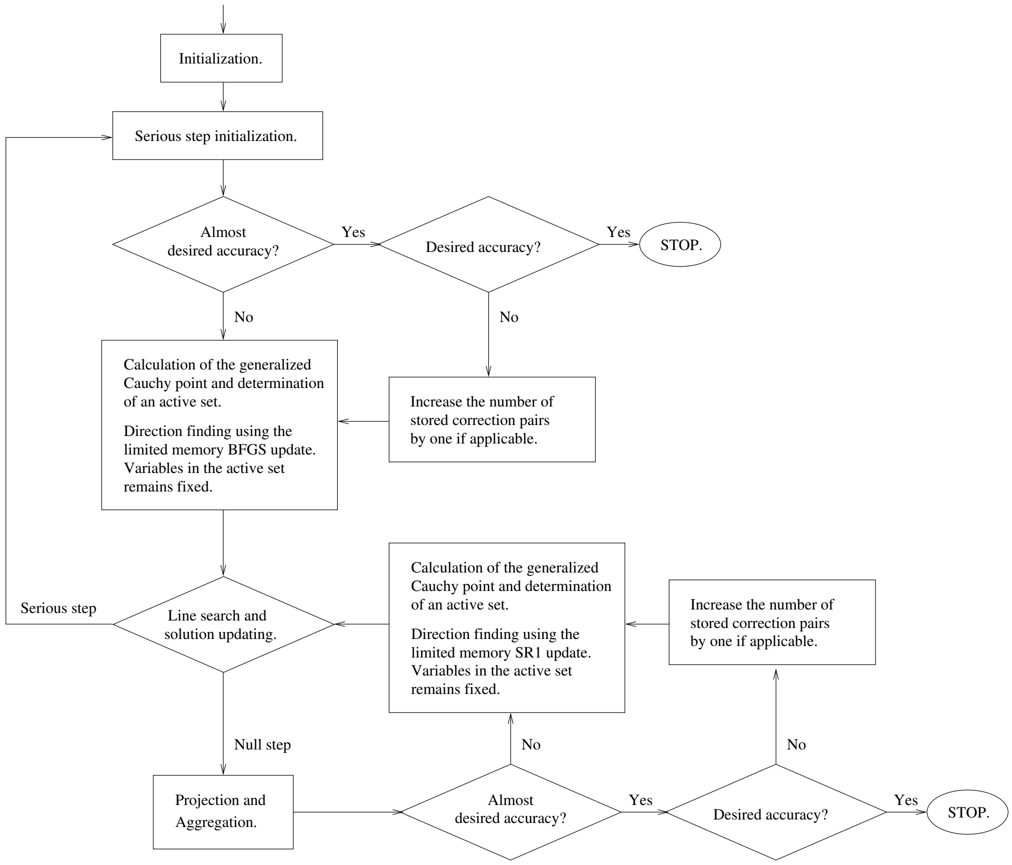

Figure 1: Adaptive limited memory bundle method with bounds.

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

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

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

Here $\tilde{\boldsymbol{\xi}}_k$ is the aggregate subgradient of the objective function from the previous iteration and B_k is a positive definite limited memory variable metric update that, in smooth case, represents the approximation of the Hessian matrix. Now, starting from $\boldsymbol{x}_k$ the generalized gradient projection method is used to find the generalized Cauchy point $\boldsymbol{x}_k^c$ [8] and, at the same time, to identify the active set $\mathcal{I}_A^k = \{i \mid x_{k,i}^c = x_i^l \text{ or } x_{k,i}^c = x_i^u\}$ of the problem. Here, we have denoted by $x_{k,i}^c$ the ith component of the vector $\boldsymbol{x}_k^c$. The calculation of the generalized Cauchy point makes it possible to add and delete several bounds from the active set during a single iteration, which may be an important feature for both nonsmooth [31] and large-scale [9] problems. After the active set has been identified, the quadratic model function (2) is approximately minimized with respect to free variables, in other words, the variables in the active set are fixed. The search direction is then defined to be the vector leading from the current iteration point $\boldsymbol{x}_k$ to this approximate minimizer. Finally, a line search that is guaranteed to produce feasible points is performed.

We utilize the limited memory approach (see, e.g. [5, 11, 21, 29]) in the calculation of the generalized Cauchy point, search direction and aggregate values. The idea of limited memory matrix updating is that instead of storing the large $n \times n$ -matrices B_k and D_k (we denote by D_k the update formula that is inverse of B_k), we store a certain (usually small constant) number $\hat{m}_c$ of vectors, so-called correction pairs obtained at the previous iterations of the algorithm, and we use these correction pairs to implicitly define the variable metric matrices. When the storage space available is used up, the oldest correction pairs are deleted to make room for new ones; thus, except for the first few iterations, we always have the $\hat{m}_c$ most recent correction pairs available.

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. [10]). However, both the storage space required and the number of operations needed in the calculations are significantly smaller. Namely, the number of operations needed is $O(n)$ while with standard variable metric updates used in original variable metric bundle methods [22, 33], it is $O(n^2)$. In the adaptive limited memory bundle method [12] the number of stored correction pairs $\hat{m}_c$ may change during the computation. This means that we can start the optimization with a small $\hat{m}_c$ and when we are closer to the optimal point, $\hat{m}_c$ may be increased until some predefined upper limit $\hat{m}_u$ is achieved. The aim of this adaptability is to improve the accuracy of the basic method without loosing much from efficiency, that is, without increasing computational costs too much.

2.1 LMBM-B algorithm

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

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

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

A necessary condition for a serious step is to have

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

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

Otherwise, we take a null step. In this case, the usage of special line search procedure guarantees that we have

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \mathcal{P}_{\boldsymbol{x}_k}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_k}[\boldsymbol{\xi}_{k+1}] \geq -\varepsilon_R w_k, \tag{5}$$

where $\varepsilon_R \in (\varepsilon_L, 1/2)$ is a line search parameter, $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, $\mathcal{P}_{\boldsymbol{x}}[\boldsymbol{\xi}]$ denotes a stark projection of $\boldsymbol{\xi}$ at $\boldsymbol{x}$ (to be described short after), and β_{k+1} is the subgradient locality measure [20, 26] similar to bundle methods. In the case of a null step, we set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ but information about the objective function is increased because we store the auxiliary point $\boldsymbol{y}_{k+1}$ and the corresponding auxiliary subgradient $\boldsymbol{\xi}_{k+1}$.

For direction finding LMBM-B uses the original subgradient $\boldsymbol{\xi}_k$ after the serious step and the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ after the null step. The aggregation procedure used in the previous versions of the limited memory bundle method [12, 13, 14, 15] is similar to that of the original variable metric bundle methods [22, 33] except that the variable metric updates are calculated using limited memory approach. However, in order to guarantee the global convergence of the box constrained version, we need to consider projections of subgradients instead of original subgradients in the aggregation procedure (see Step 6 in Algorithm 2.1). It may seem that utilization of active set $\mathcal{I}_A^k$ in the projection procedure would be a natural choice. However, active set $\mathcal{I}_A^k$ is calculated at the generalized Cauchy point $\boldsymbol{x}_k^c$ and it may change in consecutive null steps, which is highly undesirable from the view point of global convergence. Therefore, we instead calculate a very simple projection at point $\boldsymbol{x}_k$ that is not changing ($\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ in null steps). We define this stark projection operator $\mathcal{P}_{\boldsymbol{x}}[\cdot]$ at point $\boldsymbol{x}$ (componentwise) by

$$\mathcal{P}_{\boldsymbol{x}}[\boldsymbol{\xi}]_i = \begin{cases} 0, & \text{if } x_i^l - x_i \geq 0 \\ \xi_i, & \text{if } x_i \in (x_i^l, x_i^u) \\ 0, & \text{if } x_i^u - x_i \leq 0. \end{cases} \tag{6}$$

In what follows we call this projection the $\mathcal{P}_{\boldsymbol{x}}$-projection (at point $\boldsymbol{x}$).

We use both the limited memory BFGS and the limited memory SR1 update formulae in the calculations of the search direction and the aggregate values. If the previous step was a null step, the matrices D_k and B_k are formed using the limited memory SR1 updates (see (15) and (16)). The SR1 update formulae give us a possibility to preserve the boundedness and some other properties of generated matrices that guarantee the global convergence of the method. Otherwise, since these properties are not required after a serious step, the more efficient limited memory BFGS updates (see (13) and (14)) are employed. The individual updates that would violate positive definiteness are skipped (for more details, see [12, 14, 15] and Appendix).

We now present an algorithm for box constrained nonsmooth optimization. After that, we first describe the limited memory matrix updating and then how the generalized Cauchy point and the search direction can be determined. Finally, we show how to select the appropriate step sizes. In what follows, we assume that at every feasible point $\boldsymbol{x} \in \mathcal{F} \subset \mathbb{R}^n$ we can evaluate the value of the objective function $f(\boldsymbol{x})$ and the corresponding arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$.

ALGORITHM 2.1. (LMBM-B)

Data: Choose the final accuracy tolerance $\varepsilon > 0$, the positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1/2)$, and the distance measure parameter $\gamma \geq 0$ (with $\gamma = 0$ if f is convex). Select the lower and upper bounds $t_{min} \in (0, 1)$ and $t_{max} \geq 1$ for step sizes. Select the control parameter $C > 0$ for the length of the direction vector and a correction parameter $\sigma \in (0, 1/2)$. Select an upper limit $\hat{m}_u \geq 3$ for the number of stored correction pairs.

Step 0: (*Initialization.*) Choose a (feasible) starting point $\boldsymbol{x}_1 \in \mathcal{F} \subset \mathbb{R}^n$. Choose an initial maximum number of stored correction pairs $\hat{m}_c$ ($3 \leq \hat{m}_c \leq \hat{m}_u$) and initialize the limited memory matrices $S_1 = U_1 = [\,]$ (empty matrices) and the scaling parameter $\vartheta_1 = 1$. Set $\boldsymbol{y}_1 = \boldsymbol{x}_1$ and $\beta_1 = 0$. Compute $f_1 = f(\boldsymbol{x}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{x}_1)$. Set the iteration counter $k = 1$.

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

Step 2: (*Stopping criterion and correction.*) Calculate the values $\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]$ and $\sigma\|\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]\|^2$. Use the limited memory BFGS update formula (13) for the calculation of D_k if $m = k$. Else, use the limited memory SR1 update formula (15). If $\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] \leq \sigma\|\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]\|^2$ or $i_{CN} = 1$, set

$$w_k = \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] + \sigma\|\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]\|^2 + 2\tilde{\beta}_k \tag{7}$$

(i.e. $D_k = D_k + \sigma I$) and $i_{CN} = 1$. Otherwise, set

$$w_k = \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] + 2\tilde{\beta}_k. \tag{8}$$

If $w_k \leq \varepsilon$ and $\xi_{k,i} \leq 0$ for all i such that $x_{k,i} = x_i^u$ and $\xi_{k,i} \geq 0$ for all i such that $x_{k,i} = x_i^l$, then stop with $\boldsymbol{x}_k$ as the final solution. Otherwise, if $w_k \leq 10^3\varepsilon$ and $\hat{m}_c < \hat{m}_u$, set $\hat{m}_c = \hat{m}_c + 1$.

Step 3: (*Generalized Cauchy point.*) Compute the generalized Cauchy point $\boldsymbol{x}_k^c$ and determine the active set $\mathcal{I}_A^k = \{\, i \mid x_{k,i}^c = x_i^l \text{ or } x_{k,i}^c = x_i^u \}$ by Algorithm 2.2. Use the same type of update formula for B_k as in Step 2 for D_k. Note that $B_k = D_k^{-1}$ if $i_{CN} = 0$ and $B_k = (D_k + \sigma I)^{-1}$, otherwise.

Step 4: (*Direction finding.*) Compute the search direction $\boldsymbol{d}_k$ by Algorithm 2.3 using the same update formula for D_k as in Step 2 and setting $D_k = D_k + \sigma I$ if $i_{CN} = 1$. The variables in the active set $\mathcal{I}_A^k$ remains fixed.

Step 5: (*Line search and solution updating.*) Set the scaling parameter for the length of the direction vector and for line search $\theta_k = \min\{\,1, C/\|\boldsymbol{d}_k\|\,\}$. Determine the maximum step size $t_{max}^k \leq t_{max}$ such that $\boldsymbol{x}_k + t_{max}^k\theta_k\boldsymbol{d}_k$ is feasible. Choose the initial step size $t_I^k \in [t_{min}, t_{max}^k]$. Determine the step sizes $t_R^k \in (0, t_I^k]$ and $t_L^k \in [0, t_R^k]$ by Algorithm 2.4. Set the corresponding values

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

Set $\boldsymbol{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$ and $\boldsymbol{s}_k = \boldsymbol{y}_{k+1} - \boldsymbol{x}_k = t_R^k\theta_k\boldsymbol{d}_k$ and update the limited memory matrices U_{k+1} and S_{k+1}.

If condition (4) is valid (i.e. we take a serious step), set $\beta_{k+1} = 0$, $k = k + 1$, and go to Step 1. Otherwise (i.e. condition (5) is valid), calculate the locality measure

$$\beta_{k+1} = \max\{|f(\boldsymbol{x}_k) - f(\boldsymbol{y}_{k+1}) + (\boldsymbol{s}_k)^T\boldsymbol{\xi}_{k+1})|,\, \gamma\|\boldsymbol{s}_k\|^2\,\}. \tag{9}$$

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

$$\varphi(\lambda_1,\lambda_2,\lambda_3) = \mathcal{P}_{\boldsymbol{x}_m}[\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k] + 2(\lambda_2\beta_{k+1} + \lambda_3\tilde{\beta}_k), \tag{10}$$

where D_k is again calculated by the same updating formula as in Step 2 and $D_k = D_k + \sigma I$ if $i_{CN} = 1$. Set

$$\tilde{\boldsymbol{\xi}}_{k+1} = \lambda_1^k\boldsymbol{\xi}_m + \lambda_2^k\boldsymbol{\xi}_{k+1} + \lambda_3^k\tilde{\boldsymbol{\xi}}_k \qquad \text{and} \tag{11}$$

$$\tilde{\beta}_{k+1} = \lambda_2^k\beta_{k+1} + \lambda_3^k\tilde{\beta}_k. \tag{12}$$

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

The usage of $\mathcal{P}_{\boldsymbol{x}}$-projection depends only on point $\boldsymbol{x}$ and not on the subgradient $\boldsymbol{\xi}$ on focus. Thus, $\mathcal{P}_{\boldsymbol{x}_m}[\lambda_1\boldsymbol{\xi}_m + \lambda_2\boldsymbol{\xi}_{k+1} + \lambda_3\tilde{\boldsymbol{\xi}}_k] = \lambda_1\mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_m] + \lambda_2\mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_{k+1}] + \lambda_3\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]$ and this makes the minimization at Step 6 in Algorithm 2.1 rather an easy task. Note that in Steps 2, 3, 4, and 6 the matrices D_k and B_k are not formed explicitly but the limited memory expressions to be described in next subsection are used instead.

2.2 Limited memory matrices

The limited memory variable metric matrices used in our algorithm are represented in the compact matrix form originally described in [5].

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

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

where the correction pairs $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, $(i < k)$ are obtained at Step 5 in Algorithm 2.1.

The inverse limited memory BFGS update is defined by the formula

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T(C_k + \vartheta_k U_k^T U_k)R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}, \tag{13}$$

where R_k is an upper triangular matrix of order $\hat{m}_k$ given by the form

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

C_k is a diagonal matrix of order $\hat{m}_k$ such that

$$C_k = \operatorname{diag}\,[\boldsymbol{s}_{k-\hat{m}_k}^T \boldsymbol{u}_{k-\hat{m}_k}, \ldots, \boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1}],$$

and ϑ_k is a positive scaling parameter.

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

$$B_k = \frac{1}{\vartheta_k} I - \begin{bmatrix} \frac{1}{\vartheta_k} S_k & U_k \end{bmatrix} \begin{bmatrix} \frac{1}{\vartheta} S_k^T S_k & L_k \\ L_k^T & -C_k \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{\vartheta_k} S_k^T \\ U_k^T \end{bmatrix}, \tag{14}$$

where

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

Note that we have $\hat{m}_1 = 0$, $\vartheta_1 = 1$ and, thus, at the first iteration $B_1 = D_1 = I$.

The inverse limited memory SR1 update is defined by

$$D_k = \vartheta_k I - (\vartheta_k U_k - S_k)(\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k - S_k)^T \tag{15}$$

and, correspondingly, the direct SR1 update is defined by

$$B_k = \frac{1}{\vartheta_k} I + (U_k - \frac{1}{\vartheta_k} S_k)(L_k + L_k^T + C_k - \frac{1}{\vartheta_k} S_k^T S_k)^{-1}(U_k - \frac{1}{\vartheta_k} S_k)^T. \tag{16}$$

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

The SR1-update is not in general positive definite. Moreover, both the nonconvexity and bounds may prevent the condition $\boldsymbol{s}_i^T \boldsymbol{u}_i > 0$ for all $i = 1, ..., k-1$, that is the classical condition for the positive definiteness of the BFGS update, from satisfying. Thus, to maintain the positive definiteness of the generated matrices, we discard a correction pair $(\boldsymbol{s}_{k-1}, \boldsymbol{u}_{k-1})$ if some positive definiteness condition (see Appendix) is not satisfied. If this happens, we do not delete the oldest correction pair as is normally done in updating process. In practice, this means that correction matrices S_k and U_k may include some vectors with indices smaller than $k - \hat{m}_k$.

In order to guarantee the global convergence of LMBM-B, the boundedness of both the length of the direction vector (see Step 5 in Algorithm 2.1) and the matrices $B_i = D_i^{-1}$ are required (we say that a matrix is bounded if its eigenvalues lie in the compact interval that does not contain zero). The utilization of correction at Step 2 in Algorithm 2.1 is equivalent to adding a positive definite matrix σI to matrix D_k. Since the limited memory representation of matrix D_k (or B_k) does not contain information of the correction σI that may have been added, we have to add it explicitly at Steps 3, 4, and 6 if $i_{CN} = 1$. In case of inverse updates (13) and (15) (Steps 4 and 6 in Algorithm 2.1) this is obviously an easy task that does not require much additional computations. However, at Step 3 we have to calculate $B_k = (D_k + \sigma I)^{-1}$ instead of using formulae (14) or (16). Now, by using the

Sherman-Morrison-Woodbury formula to $(D_k + \sigma I)^{-1}$, omitting k and denoting $\rho = \vartheta/(\vartheta+\sigma)$ the limited memory BFGS type of update can be written in a form

$$B = \frac{\rho}{\vartheta} I - \rho^2 \begin{bmatrix} \frac{1}{\vartheta}S & U \end{bmatrix} \begin{bmatrix} \frac{\rho}{\vartheta} S^T S & \rho L - (1-\rho)R \\ \rho L^T - (1-\rho)R^T & -C - (1-\rho)\vartheta U^T U \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{\vartheta} S^T \\ U^T \end{bmatrix}, \quad (17)$$

and the limited memory SR1 type of update is given by

$$B = \tfrac{\rho}{\vartheta} I + \rho^2 (U - \tfrac{1}{\vartheta}S)((1-\rho)(U^T U - R - R^T + C) + \rho(L + L^T + C - \tfrac{1}{\vartheta} S^T S))^{-1} (U - \tfrac{1}{\vartheta}S)^T. \quad (18)$$

The middle matrix in (17) is indefinite. However, using the procedure rather similar to that given in [5] for (14), its inversion can be carried out using Cholesky factorization of the related matrix. By re-ordering the middle matrix in (17) we obtain

$$\begin{bmatrix} -(C + (1-\rho)\vartheta U^T U) & (\rho L - (1-\rho)R)^T \\ \rho L - (1-\rho)R & \frac{\rho}{\vartheta} S^T S \end{bmatrix} = \begin{bmatrix} Q & \mathbf{0} \\ -(\rho L - (1-\rho)R)(Q^{-1})^T & P \end{bmatrix} \begin{bmatrix} -Q^T & Q^{-1}(\rho L - (1-\rho)R)^T \\ \mathbf{0} & P^T \end{bmatrix},$$

where P and Q are lower triangular matrices such that $QQ^T = C + (1-\rho)\vartheta U^T U$ and $PP^T = \frac{\rho}{\vartheta} S^T S + (\rho L - (1-\rho)R)(QQ^T)^{-1}(\rho L - (1-\rho)R)^T$. Note that we have $\boldsymbol{s}_i^T \boldsymbol{u}_i > 0$ for all $i = 1, ..., k-1$ (the classical condition for the positive definiteness of the BFGS update) and, thus, $C + (1-\rho)\vartheta U^T U$ is symmetric and positive definite. Moreover, this condition guarantees the positive definiteness of the matrix $\frac{\rho}{\vartheta} S^T S + (\rho L - (1-\rho)R)(QQ^T)^{-1}(\rho L - (1-\rho)R)^T$ as well (the proof is very similar to the proof of Theorem 2.4 in [5]). Thus, to implement (17), only the Cholesky factorizations of two $\hat{m}_k \times \hat{m}_k$ symmetric positive definite matrices need to be computed.

Naturally, formulae (17) and (18) require some more computations than the traditional formulae (14) and (16) but also in these cases the calculations can be done within $O(n)$ operations.

Finally, we remark that the basic assumption for bundle method to converge, that is, after a null step we have $\boldsymbol{z}^T D_{k+1} \boldsymbol{z} \le \boldsymbol{z}^T D_k \boldsymbol{z}$ for all $\boldsymbol{z} \in \mathbb{R}^n$, is guaranteed by the special limited memory SR1 update [12, 15].

2.3 Generalized Cauchy Point and Active Set

In this subsection, we show how to calculate the generalized Cauchy point and, at the same time, how to identify the active set of problem (1). The procedure used here is in principle the same as that in [4]. We only use here the aggregate subgradient of the objective function instead of gradient and, in addition to the limited memory BFGS update formula, we utilize the limited memory SR1 update whenever necessary. That is, after a null step.

We define the second projection operator $\mathcal{P}_c[\cdot]$ (component-wise) by

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

This operator projects the point $\boldsymbol{x}$ into the feasible region $\mathcal{F}$ defined by bounds $\boldsymbol{x}^l$ and $\boldsymbol{x}^u$. Note that, contrary to (6), here the projected component x_i has an effect to the result.

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

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

(with q_k defined in (2)) along the projected gradient direction $\mathcal{P}_c[\boldsymbol{x}_k - t\tilde{\boldsymbol{\xi}}_k, \boldsymbol{x}^l, \boldsymbol{x}^u] - \boldsymbol{x}_k$ (see, e.g. [8]). That is, if we denote by t_k^c the value of t corresponding to the first local minimum of $\hat{q}_k(t)$, the generalized Cauchy point is given by

$$\boldsymbol{x}_k^c = \mathcal{P}_c[\boldsymbol{x}_k - t_k^c\tilde{\boldsymbol{\xi}}_k, \boldsymbol{x}^l, \boldsymbol{x}^u]. \tag{19}$$

The variables whose values at $\boldsymbol{x}_k^c$ are at lower or upper bound, comprise the active set $\mathcal{I}_A^k = \{i \mid x_{k,i}^c = x_i^l \text{ or } x_{k,i}^c = x_i^u\}$.

In practice, we first compute the values

$$t_i = \begin{cases} (x_{k,i} - x_i^l)/\tilde{\xi}_{k,i}, & \text{if } \tilde{\xi}_{k,i} > 0 \\ (x_{k,i} - x_i^u)/\tilde{\xi}_{k,i}, & \text{if } \tilde{\xi}_{k,i} < 0 \\ \infty, & \text{otherwise} \end{cases} \tag{20}$$

for all $i = 1, \ldots, n$ to define the breakpoints in each coordinate direction. We then sort these values t_i in increasing order to obtain the ordered set $\{t^j \mid t^j \leq t^{j+1}, j = 1, \ldots, n\}$. For finding the generalized Cauchy point we search trough the intervals $[t^j, t^{j+1}]$ in order of increasing j until the one containing $\boldsymbol{x}_k^c$ is located. Thus, we investigate the behavior of the quadratic function (2) for points lying on the piecewise linear path

$$x_{k,i}(t) = \begin{cases} x_{k,i} - t\tilde{\xi}_{k,i}, & \text{if } t \leq t_i \\ x_{k,i} - t_i\tilde{\xi}_{k,i}, & \text{otherwise.} \end{cases} \tag{21}$$

Let us define the jth break-point by $\boldsymbol{x}^j = \boldsymbol{x}_k(t^j)$. We can now express (21) in the interval $[t^j, t^{j+1}]$ as

$$\boldsymbol{x}_k(t) = \boldsymbol{x}^j + \Delta t\hat{\boldsymbol{d}}^j, \tag{22}$$

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

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

Defining

$$f_j = f(\boldsymbol{x}_k) + \tilde{\boldsymbol{\xi}}_k^T \boldsymbol{z}^j + \frac{1}{2} \boldsymbol{z}^{jT} B_k \boldsymbol{z}^j,$$

we have

$$\begin{aligned} f_j' &= \tilde{\boldsymbol{\xi}}_k^T \hat{\boldsymbol{d}}^j + \hat{\boldsymbol{d}}^{jT} B_k \boldsymbol{z}^j, \\ f_j'' &= \hat{\boldsymbol{d}}^{jT} B_k \hat{\boldsymbol{d}}^j, \end{aligned}$$

where $\boldsymbol{z} = \boldsymbol{x}^j - \boldsymbol{x}_k$. Now, combining (2) and (22), we obtain

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

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

We now give an algorithm for calculation of the generalized Cauchy point and determination of the active set $\mathcal{I}_A^k$. Note that except for minor details this is the algorithm given already in [4]. To simplify the notation, we, for a while, omit the iteration index k and use subscripts i and b to denote the ith and the bth component of a vector. Moreover, we denote by $\boldsymbol{e}_b$ the bth column of the identity matrix.

ALGORITHM 2.2. *(Generalized Cauchy Point.)*

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

Step 0: (*Initialization.*) Compute the breakpoints t_i in each coordinate direction by (20) and define the direction

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

Initialize

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

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

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

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

Step 2: (*Subsequent segments.*) Set

$b = i$ such that $t_i = t$. Shift b from $\mathcal{I}_F$ to $\mathcal{I}_A$.

Set

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

Calculate

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

Set

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

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

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

$$\Delta t_{min} = \max\{\Delta t_{min}, 0\},$$
$$t_{old} = t_{old} + \Delta t_{min},$$
$$x_i^c = x_i + t_{old}\hat{d}_i \text{ for all } i \text{ such that } t_i \geq t.$$

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

The only expensive computations in Algorithm 2.2 are

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

at Steps 1 and 2. However, these calculations can be done very efficiently (within $O(n)$ operations) by using limited memory approach. We do not give any details of these calculations here since, essentially, for the limited memory BFGS update, these calculations proceeds similar to those given in [4] and, for the limited memory SR1 update, the idea should be perspicuous as well.

2.4 Direction finding

When the generalized Cauchy point has been found, we approximately minimize the quadratic model function (2) over the space of free variables. The subspace minimization procedure used is in principal the same as the dual space method in [4] but, as before, we use the aggregate subgradient of the objective function and we utilize the limited memory SR1 update if the previous step taken was a null step (see Algorithm 2.1).

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

$$\begin{cases} \text{minimize} & \tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T B_k \boldsymbol{d} \\ \text{such that} & A_k^T \boldsymbol{d} = \boldsymbol{b}_k \qquad \text{and} \\ & \boldsymbol{x}^l \leq \boldsymbol{x}_k + \boldsymbol{d} \leq \boldsymbol{x}^u, \end{cases} \tag{23}$$

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

We first ignore the box constraints. The first order optimality conditions for problem (23) without bounds are

$$\tilde{\boldsymbol{\xi}}_k + B_k \boldsymbol{d}_k^* + A_k \boldsymbol{\mu}_k^* = \boldsymbol{0} \tag{24}$$

$$A_k^T \boldsymbol{d}_k^* = \boldsymbol{b}_k. \tag{25}$$

Now, by multiplying (24) by $A_k^T D_k$, where, as before, $D_k = B_k^{-1}$, and by using (25), we obtain

$$(A_k^T D_k A_k)\boldsymbol{\mu}_k^* = -A_k^T D_k \tilde{\boldsymbol{\xi}}_k - \boldsymbol{b}_k, \tag{26}$$

which determines Lagrange multipliers $\boldsymbol{\mu}_k^* \in \mathbb{R}^{n_A}$. The linear system (26) can be solved by utilizing the Sherman-Morrison-Woodbury formula and the compact representation of limited memory matrices (see [4]). Thus, $\boldsymbol{d}_k^*$ can be given by

$$B_k \boldsymbol{d}_k^* = -A_k \boldsymbol{\mu}_k^* - \tilde{\boldsymbol{\xi}}_k. \tag{27}$$

If there are no active variables, we simply obtain $\boldsymbol{d}_k^* = -D_k \tilde{\boldsymbol{\xi}}_k$, which is the formula used also in the original unconstrained version of the limited memory bundle method [12, 14]. In the case the vector $\boldsymbol{x}_k + \boldsymbol{d}_k^*$ violates the box constraints in (23), we, similarly to [4], backtrack along the line joining the infeasible point $\boldsymbol{x}_k + \boldsymbol{d}_k^*$ and the generalized Cauchy point $\boldsymbol{x}_k^c$ to regain the feasible region. That is, we compute

$$\alpha_k^* = \min \left\{ 1, \max\{ \alpha \mid x_i^l \leq x_{k,i}^c + \alpha(x_{k,i} + d_i^* - x_{k,i}^c) \leq x_i^u,\ i \in \mathcal{I}_F^k \} \right\} \tag{28}$$

and we set $\bar{\boldsymbol{x}} = \boldsymbol{x}_k^c + \alpha_k^*(\boldsymbol{x}_k + \boldsymbol{d}_k^* - \boldsymbol{x}_k^c)$ and $\boldsymbol{d}_k = \bar{\boldsymbol{x}} - \boldsymbol{x}_k$.

We now give an efficient algorithm for direction finding in box constrained case. For more details of the calculations using this limited memory approach, see [4].

ALGORITHM 2.3. *(Direction Finding.)*

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

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

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

and go to Step 4.

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

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

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

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

Step 3: (*Search direction.*) Compute

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

Step 4: (*Backtrack.*) Compute α_k^* by (28). Set $\bar{\boldsymbol{x}} = \boldsymbol{x}_k^c + \alpha_k^*(\boldsymbol{x}_k + \boldsymbol{d}_k^* - \boldsymbol{x}_k^c)$ and $\boldsymbol{d}_k = \bar{\boldsymbol{x}} - \boldsymbol{x}_k$.

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

2.5 Line Search Procedure

Now we consider how to choose the appropriate step sizes in LMBM-B. In the previous versions of LMBM [12, 13, 14, 15] the initial step size $t_I^k \in [t_{min}, t_{max}^k]$ (see Step 5 in Algorithm 2.1) is always selected by using a bundle containing auxiliary points and corresponding function values and subgradients. However, the numerical experiments confirmed that usually a simple choice $t_I^k = \max\{t_{min}, \min\{2, t_{max}^k\}\}$ if the previous step was a serious step and $t_I^k = \max\{t_{min}, \min\{1, t_{max}^k\}\}$, otherwise, works as well in box constrained case and, naturally, needs less calculations. Thus, we use this simple selection always but in case we obtain $w_k \leq \varepsilon$ but some components (at least one) of the subgradient vector $\boldsymbol{\xi}_k$ are of the wrong sign (see Step 2 of Algorithm 2.1). In this particular case, we use the procedure similar to previous versions in order to guarantee that the serious step will be taken (see Lemma 3.5). Likewise in previous versions and original variable metric bundle method [33], we have here the possibility to use step sizes greater than one after serious steps since information about objective function included in matrix D_k may not be sufficient for a proper step size determination in nonsmooth case.

Next, we present the line search algorithm, which is used to determine the step sizes $t_R^k \in (0, t_I^k]$ and $t_L^k \in [0, t_R^k]$. The line search procedure used in LMBM-B is rather similar to that given in [12, 15] which, on the other hand, was derived from [33]. However, in order to guarantee global convergence also in box constrained case, we needed to modify the line search procedure as well as the semi-smoothness assumption used in the previous variants of the limited memory bundle method.

ALGORITHM 2.4. *(Modified line search for box constrained problems).*

Data: Suppose that we have the current iteration point $\boldsymbol{x}_k$, the current search direction $\boldsymbol{d}_k$, the current scaling parameter $\theta_k \in (0, 1]$, and the current vector $\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k$ available. Suppose also that we have the initial step size t_I^k, an auxiliary lower bound for serious steps $t_{min} \in (0, 1)$, the distance measure parameter $\gamma \geq 0$, the desirable amount of descent w_k, and the positive line search parameters $\varepsilon_L \in (0, 1/2)$, $\varepsilon_R \in (\varepsilon_L, 1/2)$, $\varepsilon_A \in (0, \varepsilon_R - \varepsilon_L)$, and $\varepsilon_T \in (\varepsilon_L, \varepsilon_R - \varepsilon_A)$ available. In addition, suppose that we have given the number of consecutive null steps $i_{null} \geq 0$ and the maximum number of additional interpolations i_{max}.

Step 0: (*Initialization.*) Set $t_A = 0$, $t = t_U = t_I^k$, and $i_I = 0$, and calculate the interpolation parameter

$$\kappa = 1 - \frac{1}{2(1-\varepsilon_T)}.$$

Step 1: (*New values.*) Compute $f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k)$, $\boldsymbol{\xi} \in \partial f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k)$, and

$$\beta = \max\{\, |f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k) + t\theta_k \boldsymbol{d}_k^T \boldsymbol{\xi}|,\ \gamma\,(t\theta_k \|\boldsymbol{d}_k\|)^2\,\}.$$

If $f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) - \varepsilon_T t w_k$, then set $t_A = t$. Otherwise, set $t_U = t$.

Step 2: (*Serious step.*) If

$$f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) - \varepsilon_L t w_k,$$

and either

$$t \geq t_{min} \qquad \text{or} \qquad \beta > \varepsilon_A w_k,$$

then set $t_R^k = t_L^k = t$ and stop.

Step 3: (*Test for additional interpolation.*) If $f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k) > f(\boldsymbol{x}_k)$, $i_{null} > 0$, and $i_I < i_{max}$, then set $i_I = i_I + 1$ and go to Step 5.

Step 4: (*Null step.*) If

$$-\beta - \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}] \geq -\varepsilon_R w_k,$$

then set $t_R^k = t$, $t_L^k = 0$ and stop.

Step 5: (*Interpolation.*) If $t_A = 0$, then set

$$t = \max\left\{\kappa t_U, \frac{-\frac{1}{2} t_U^2 w_k}{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + t\theta_k \boldsymbol{d}_k) - t_U w_k}\right\}.$$

Otherwise, set $t = \frac{1}{2}(t_A + t_U)$. Go to Step 1.

The line search algorithm 2.4 terminates in a finite number of iterations if the problem satisfies the following modified semi-smoothness assumption: For all $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\boldsymbol{x})$ with some small $\varepsilon \geq 0$ and for any $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{d} \in \mathbb{R}^n$, and sequences $(\hat{\boldsymbol{\xi}}_j) \subset \mathbb{R}^n$ and $(t_j) \subset \mathbb{R}_+$ satisfying $\hat{\boldsymbol{\xi}}_j \in \partial f(\boldsymbol{x} + t_j \boldsymbol{d})$ and $t_j \downarrow 0$, we have

$$-\limsup_{j\to\infty} \mathcal{P}_{\boldsymbol{x}}[\boldsymbol{\xi}]^T D \mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j] \geq \liminf_{j\to\infty} \frac{f(\boldsymbol{x} + t_j \boldsymbol{d}) - f(\boldsymbol{x})}{t_j}, \tag{29}$$

where $\mathcal{P}_{\boldsymbol{x}}[\cdot]$ denotes the projection at point $\boldsymbol{x}$ and D is the inverse variable metric approximation calculated at this very same point. Note that if none of the variables is on the boundary at $\boldsymbol{x}$ this modified semi-smoothness assumption reverts very similar to the classical semi-smoothness assumption [3].

LEMMA 2.1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$ and let the above mentioned modified semi-smoothness assumption hold. Then Algorithm 2.4 finds, after a finite number of steps, the step sizes t_L^k and t_R^k such that either a serious step or a null step occurs.*

PROOF. If Algorithm 2.4 terminates, it is obvious that either a serious step or a null step occurs. Thus, it is enough to prove that the algorithm terminates after a finite number of iterations.

For a contradiction, let us assume that the algorithm does not terminate. To simplify the notation, we now omit the iteration number k and the scaling parameter θ_k. That is, we denote by $\boldsymbol{d}$ the scaled direction vector $\theta_k \boldsymbol{d}_k$.

Let (t_j), (t_A^j), (t_U^j), $(\hat{\boldsymbol{\xi}}_j)$, and (β_j) be the sequences of values generated by Algorithm 2.4 so far ($t_j = t_A^j$ or $t_j = t_U^j$). Since $t_A^j \leq t_A^{j+1} \leq t_U^{j+1} \leq t_U^j$, and $t_U^{j+1} - t_A^{j+1} \leq (1-\kappa)(t_U^j - t_A^j)$ for all indices j, there exists a value t^* such that $t_A^j \uparrow t^*$, $t_U^j \downarrow t^*$, and $t_j \to t^*$. Let us denote $\mathcal{T} = \{t \geq 0 \mid f(\boldsymbol{x} + t\boldsymbol{d}) \leq f(\boldsymbol{x}) - \varepsilon_T t w\}$. Now, since $(t_A^j) \subset \mathcal{T}$, $t_A^j \uparrow t^*$ and the function f is continuous, we have

$$f(\boldsymbol{x} + t^*\boldsymbol{d}) \leq f(\boldsymbol{x}) - \varepsilon_T t^* w. \tag{30}$$

Thus, $(t^*) \subset \mathcal{T}$. Let $\mathcal{J} = \{j \mid t_j \notin \mathcal{T}\}$. We will first show that the set $\mathcal{J}$ is infinite. If there exists an index $\bar{j} \in \mathcal{J}$ such that $t_j \in \mathcal{T}$ for all $j > \bar{j}$, then $t_U^{\bar{j}} = t_U^j \downarrow t^*$ for all $j > \bar{j}$ or $t^* = t_U^j \notin \mathcal{T}$ would have to hold. This, however, is in contradiction to $t^* \in \mathcal{T}$. Therefore, the set $\mathcal{J}$ is infinite and we have $f(\boldsymbol{x} + t_j\boldsymbol{d}) > f(\boldsymbol{x}) - \varepsilon_T t_j w$ for all $j \in \mathcal{J}$. Now, combining the previous result with (30) we obtain

$$\frac{f(\boldsymbol{x} + t_j\boldsymbol{d}) - f(\boldsymbol{x} + t^*\boldsymbol{d})}{t_j - t^*} > -\varepsilon_T w$$

for all $j \in \mathcal{J}$ and by using the assumption (29) we obtain

$$\begin{aligned} -\varepsilon_T w &< \liminf_{j \xrightarrow{\mathcal{J}} \infty} \frac{f(\boldsymbol{x} + t^*\boldsymbol{d} + (t_j - t^*)\boldsymbol{d}) - f(\boldsymbol{x} + t^*\boldsymbol{d})}{t_j - t^*} \\ &\leq -\limsup_{j \xrightarrow{\mathcal{J}} \infty} \mathcal{P}_{\boldsymbol{x}}[\boldsymbol{\xi}\,]^T D \mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j\,] \end{aligned} \tag{31}$$

for all $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\boldsymbol{x})$.

Let us first suppose that $t^* > 0$. By (30), condition $f(\boldsymbol{x} + t^*\boldsymbol{d}) \leq f(\boldsymbol{x}) - \varepsilon_L t^* w$ holds for sufficiently large indices, since $\varepsilon_L < \varepsilon_T$, $t_j \to t^*$, and function f is continuous. Since the algorithm does not terminate, both $\beta_j \leq \varepsilon_A w$ (Step 2 in Algorithm 2.4) and $\beta_j + \mathcal{P}_{\boldsymbol{x}}[\tilde{\boldsymbol{\xi}}\,]^T D \mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j\,] > \varepsilon_R w$ (Step 4 in Algorithm 2.4) have to be satisfied for sufficiently large indices. Combining these two and using the fact that $\varepsilon_T < \varepsilon_R - \varepsilon_A$, we obtain

$$-\mathcal{P}_{\boldsymbol{x}}[\tilde{\boldsymbol{\xi}}\,]^T D \mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j\,] < \beta_j - \varepsilon_R w \leq \varepsilon_A w - \varepsilon_R w < -\varepsilon_T w$$

but this is in contradiction to (31) for $j \in \mathcal{J}$ and $\mathcal{J}$ infinite due to fact that $\tilde{\boldsymbol{\xi}} \in \partial^G \varepsilon f(\boldsymbol{x})$ as a convex combination of previous subgradients.

Let us now suppose that $t^* = 0$. Then $t_j \to 0$ implies $\beta_j \to 0$ due to fact that the function f is locally Lipschitz continuous and the local boundedness of subgradients. Since algorithm does not terminate, $\beta_j + \mathcal{P}_{\boldsymbol{x}}[\tilde{\boldsymbol{\xi}}]^T D\mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j] > \varepsilon_R w$ has to hold for sufficiently large indices so that

$$-\limsup_{j\to\infty} \mathcal{P}_{\boldsymbol{x}}[\tilde{\boldsymbol{\xi}}]^T D\mathcal{P}_{\boldsymbol{x}}[\hat{\boldsymbol{\xi}}_j] < -\varepsilon_R w < -\varepsilon_T w.$$

However, this is again in contradiction to (31). Therefore, Algorithm 2.4 terminates after a finite number of iterations. $\square$

On the output of Algorithm 2.4 (Steps 2 and 4), the step sizes t_L^k and t_R^k satisfy the serious descent criterion

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \le -\varepsilon_L t_L^k w_k^1 \tag{32}$$

and, in the case of $t_L^k = 0$ (a null step), also condition (5).

3 Convergence Analysis

In this section, we prove the global convergence of Algorithm 2.1. The main differences between the algorithm proposed here and the previous version [13] are in the usage of stark projections of subgradients in the aggregation procedure (see Step 6 of Algorithm 2.1) and in the calculation of the stopping criterion (see Step 2 of Algorithm 2.1). Moreover, we here use the corrections for the limited memory matrices whenever necessary (see Step 2 of Algorithm 2.1) and the length of the direction vector is kept bounded (see Step 5 of Algorithm 2.1). Also the line search procedure (see Algorithm 2.4) has been changed.

We now recall a necessary KKT-type optimality condition for locally Lipschitz continuous objective functions in box constrained case (i.e. for problem (1)). Assuming the convexity of the objective function, this condition is also sufficient and the minimum is global.

THEOREM 3.1. (KKT-type optimality condition for box constrained problems.) *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$ and let us denote by $\boldsymbol{g}^l(\boldsymbol{x}) = \boldsymbol{x}^l - \boldsymbol{x}$ and $\boldsymbol{g}^u(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{x}^u$. If $\boldsymbol{x}$ is a local minimum of problem (1), then there exist Lagrange multipliers $\mu_i^l, \mu_i^u \ge 0$ such that $\mu_i^l g_i^l(\boldsymbol{x}) = 0$ and $\mu_i^u g_i^u(\boldsymbol{x}) = 0$ for all $i \in \{1, \dots, n\}$ and*

$$\mathbf{0} \in \partial f(\boldsymbol{x}) + \sum_{i=1}^n \mu_i^l \partial g_i^l(\boldsymbol{x}) + \sum_{i=1}^n \mu_i^u \partial g_i^u(\boldsymbol{x}).$$

A feasible point $\boldsymbol{x}$ satisfying the KKT optimality condition above is said to be a KKT point *associated to problem (1).*

Note that in the previous theorem we have $\partial g_i^l(\boldsymbol{x}) = \{\nabla g_i^l(\boldsymbol{x})\}$ and $\partial g_i^u(\boldsymbol{x}) = \{\nabla g_i^u(\boldsymbol{x})\}$ due to differentiability of constraints (see [6]).

Now, in addition to assuming that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous, the set $\mathcal{F} \cap \{ \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \le f(\boldsymbol{x}_1) \}$ is supposed to be compact. Furthermore, we assume that each execution of the line search procedure is finite (i.e. the modified semismoothness assumption (29) is valid).

We start the theoretical analysis of LMBM-B by studying the case when the algorithm terminates after a finite number of iterations: we prove that if Algorithm 2.1 stops at iteration k, then the point $\boldsymbol{x}_k$ is a KKT point for problem (1). Then, we prove that Algorithm 2.1 does not stop at a non-KKT-point and, finally, we prove that in case of infinite sequence $(\boldsymbol{x}_k) \subset \mathcal{F}$, every accumulation point $\bar{\boldsymbol{x}}$ of the sequence $(\boldsymbol{x}_k)$ generated by the algorithm is a KKT point for problem (1). In order to do this, we assume that the final accuracy tolerances ε is equal to zero.

REMARK 3.1. The sequence $(\boldsymbol{x}_k)$ generated by Algorithm 2.1 is bounded by assumption and the monotonicity of the sequence (f_k) obtained due to serious descent criterion (32). Since $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ for serious steps and $\|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\| \le t_{max}C$ for null steps by (3) and due to facts that $t_{max}^k \le t_{max}$ and we use the scaled direction vector $\theta_k \boldsymbol{d}_k$ with $\theta_k = \min\{ 1, C/\|\boldsymbol{d}_k\| \}$ and predefined $C > 0$ in the line search, the sequence $(\boldsymbol{y}_k)$ is also bounded. By the local boundedness and the upper semi-continuity of subdifferential we obtain the boundedness of subgradients $\boldsymbol{\xi}_k$ as well as all their convex combinations (see [6]). Finally, also the $\mathcal{P}_{\boldsymbol{x}}$-projections of subgradients are bounded by definition.

LEMMA 3.2. *Suppose that Algorithm 2.1 is not terminated before the kth iteration. Then, there exist numbers* $\lambda^{k,j} \ge 0$ *for* $j = 1, \dots, k$ *and* $\tilde{\alpha}_k \ge 0$ *such that*

$$(\tilde{\boldsymbol{\xi}}_k, \tilde{\alpha}_k) = \sum_{j=1}^{k} \lambda^{k,j} (\boldsymbol{\xi}_j, \|\boldsymbol{y}_j - \boldsymbol{x}_k\|), \qquad \sum_{j=1}^{k} \lambda^{k,j} = 1, \quad \textit{and} \quad \tilde{\beta}_k \ge \gamma \tilde{\alpha}_k^2.$$

PROOF. See the proof of Lemma 3.2 in [33]. $\square$

LEMMA 3.3. *Let* $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ *be given and suppose that there exist vectors* $\bar{\boldsymbol{\zeta}}$, $\bar{\boldsymbol{\xi}}_j$, $\bar{\boldsymbol{y}}_j$, *and numbers* $\bar{\lambda}_j \ge 0$ *for* $j = 1, \dots, l$, $l \ge 1$, *such that*

$$\begin{aligned}
&(\bar{\boldsymbol{\zeta}}, 0) = \sum_{j=1}^{l} \bar{\lambda}_j (\bar{\boldsymbol{\xi}}_j, \|\bar{\boldsymbol{y}}_j - \bar{\boldsymbol{x}}\|), \\
&\bar{\boldsymbol{\xi}}_j \in \partial f(\bar{\boldsymbol{y}}_j), \quad j = 1, \dots, l, \qquad \textit{and} \\
&\sum_{j=1}^{l} \bar{\lambda}_j = 1.
\end{aligned}$$

Then $\bar{\boldsymbol{\zeta}} \in \partial f(\bar{\boldsymbol{x}})$.

PROOF. See the proof of Lemma 3.3 in [33]. $\square$

THEOREM 3.4. *If Algorithm 2.1 terminates at the kth iteration, then the point $\boldsymbol{x}_k$ is a KKT point for problem (1).*

PROOF. Let us first point out that $\tilde{\beta}_k \geq 0$ for all k by (9), (12), and Step 1 in Algorithm 2.1. Due to (7), (8), and the positive definiteness of D_k, we have

$$w_k \geq 2\tilde{\beta}_k \qquad \text{and} \qquad w_k \geq \sigma \| \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] \|^2. \tag{33}$$

Since Algorithm 2.1 terminates at Step 2, the fact $\varepsilon = 0$ implies that we have $w_k = 0$. Thus, $\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] = \mathbf{0}$ and $\tilde{\beta}_k = \tilde{\alpha}_k = 0$ by (33) and Lemma 3.2. Moreover, by Lemma 3.2 and by using Lemma 3.3 with

$$\begin{aligned} &\bar{\boldsymbol{x}} = \boldsymbol{x}_k, \qquad l = k, \qquad \bar{\boldsymbol{\zeta}} = \tilde{\boldsymbol{\xi}}_k, \\ &\bar{\boldsymbol{\xi}}_j = \boldsymbol{\xi}_j, \qquad \bar{\boldsymbol{y}}_j = \boldsymbol{y}_j, \qquad \bar{\lambda}_j = \lambda^{k,j} \quad \text{for } j \leq k, \end{aligned}$$

we obtain $\tilde{\boldsymbol{\xi}}_k \in \partial f(\boldsymbol{x}_k)$.

Let us first consider the case none of the variables is on the boundary at point $\boldsymbol{x}_k$ (apropos, we always have $\boldsymbol{x}_k = \boldsymbol{x}_m$ for both serious and null steps). Now, $\tilde{\boldsymbol{\xi}}_k = \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]$ by (6). Thus, $\mathbf{0} = \tilde{\boldsymbol{\xi}}_k \in \partial f(\boldsymbol{x}_k)$ and $\boldsymbol{x}_k$ is (an unconstrained) KKT point for problem (1).

If some of the variables $x_{k,i}$, $i \in \{1, \dots, n\}$ are on the boundary at point $\boldsymbol{x}_k$ we may have $\tilde{\xi}_{k,i} \neq 0$ for those variables (for the other variables we have $\tilde{\xi}_{k,i} = 0$ by (6)). Now, due to fact that $g_i^l(\boldsymbol{x}_k) = 0$ for the variables such that $x_{k,i} = x_i^l$ and $g_i^u(\boldsymbol{x}_k) = 0$ for $x_{k,i} = x_i^u$, respectively, and by choosing $\mu_i^l, \mu_i^u = 0$ for free variables (subject to bounds at $\boldsymbol{x}_k$), we obtain $\mu_i^l g_i^l(\boldsymbol{x}_k) = 0$ and $\mu_i^u g_i^u(\boldsymbol{x}_k) = 0$ for all $i \in \{1, \dots, n\}$.

Since Algorithm 2.1 stops at Step 2, we have $\tilde{\xi}_{k,i} \geq 0$ for all i such that $x_{k,i} = x_i^l$ and $\tilde{\xi}_{k,i} \leq 0$ for all i such that $x_{k,i} = x_i^u$. Now, by first noting that $\nabla g_i^l(\boldsymbol{x}_k)$ is a vector with ith component equal to -1 and others zero while $\nabla g_i^u(\boldsymbol{x}_k)$ has its ith component equal to 1 and others zero and, then, by choosing $\mu_i^l = \tilde{\xi}_{k,i}$, if $x_{k,i} = x_i^l$, and $\mu_i^u = -\tilde{\xi}_{k,i}$, if $x_{k,i} = x_i^u$, we obtain

$$\mathbf{0} = \tilde{\boldsymbol{\xi}}_k + \sum_{i=1}^{n} \mu_i^l \nabla g_i^l(\boldsymbol{x}_k) + \sum_{i=1}^{n} \mu_i^u \nabla g_i^u(\boldsymbol{x}_k)$$

with $\mu_i^l, \mu_i^u \geq 0$ for all $i \in \{1, \dots, n\}$. Thus, by Theorem (3.1) the point $\boldsymbol{x}_k$ is a KKT point for problem (1). $\square$

From now on, we suppose that Algorithm 2.1 does not terminate. We first study the case when $w_k = 0$ for some k but at least one component of the subgradient vector $\tilde{\boldsymbol{\xi}}_k$ is of the wrong sign.

LEMMA 3.5. *Suppose that Algorithm 2.1 generates a point $\boldsymbol{x}_k \in \mathcal{F}$ such that $w_k = 0$ but either $\tilde{\xi}_{k,i} < 0$ for some i such that $x_{k,i} = x_i^l$, or $\tilde{\xi}_{k,i} > 0$ for some i such that $x_{k,i} = x_i^u$ (or both). Then, the next step is a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$.*

PROOF. For simplicity, we first consider the case only one variable, say $x_{k,i}$, is on the boundary. From this point of view, the proof can be easily generalized.

Due to fact $w_k = 0$, we have $\mathcal{P}_{\boldsymbol{x}_k}[\tilde{\boldsymbol{\xi}}_k] = \mathbf{0}$ (i.e. we have $\tilde{\xi}_{k,j} = 0$ for all $j \neq i$) and, thus, we can restrict our consideration to a single component i. Let us first suppose that $x_{k,i} = x_i^u$. Since now $\xi_{k,i} > 0$, the generalized Cauchy point $x_{k,i}^c < x_{k,i} = x_i^u$ (see (19)). Thus, either i is not in $\mathcal{I}_A^k$ or $x_{k,i}^c$ is on the lower bound. In the first case, we simply obtain $\boldsymbol{d}_k^* = -D_k\tilde{\boldsymbol{\xi}}_k$. In the latter case, we have $\boldsymbol{b}_k = A_k^T(\boldsymbol{x}_k^c - \boldsymbol{x}_k) = x_i^l - x_i^u$. From (25) we obtain $d_i^* = x_i^l - x_i^u < 0$ and by (27) we have $d_j^* = -(D_k)_j\tilde{\boldsymbol{\xi}}_k$ for all $j \neq i$ ($(D_k)_j$ denotes the jth row of matrix D_k). In both cases we, obviously, have $(\boldsymbol{d}_k^*)^T\tilde{\boldsymbol{\xi}}_k < 0$.

Now, the search direction $\boldsymbol{d}_k$ is calculated by $\boldsymbol{d}_k = \boldsymbol{x}_k^c + \alpha_k^*(\boldsymbol{x}_k + \boldsymbol{d}_k^* - \boldsymbol{x}_k^c) - \boldsymbol{x}_k$ with $\alpha_k^* \in (0,1]$ defined in (28). Since $\tilde{\xi}_{k,j} = 0$ for all $j \neq i$ and $\xi_{k,i} > 0$ we obtain

$$\boldsymbol{d}_k^T\tilde{\boldsymbol{\xi}}_k = ((1-\alpha_k^*)(x_{k,i}^c - x_{k,i}) + \alpha_k^* d_i^*)\xi_{k,i} < 0.$$

That is, $\boldsymbol{d}_k$ is a descent direction for the objective function.

In line search procedure, we have the initial step size $t_k^I \in [t_{min}, t_{max}^k]$ chosen such that it minimized an approximation of $f(\boldsymbol{x}_k + t\theta_k\boldsymbol{d}_k)$ (for details see [33]). Now, we have $w_k = 0$, $\boldsymbol{d}_k$ is a descent direction, and we initialize $t = t_k^I$. Thus, for sufficiently small $t_{min} > 0$, we have

$$f(\boldsymbol{x}_k + t\theta_k\boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) - \varepsilon_L t w_k = f(\boldsymbol{x}_k), \qquad \text{and} \qquad t \geq t_{min} \tag{34}$$

at Step 2 in Algorithm 2.4 (at the first iteration of the algorithm). Therefore, a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$ is taken.

Now suppose that, in addition to $x_{k,i}$, some other variable $x_{k,j}$ is on the boundary. Suppose also that the difference between these variables is that $\tilde{\xi}_{k,j}$ is of the right sign, let us say $x_{k,j} = x_j^u$ and $\tilde{\xi}_{k,j} \leq 0$. The generalized Cauchy point is now given $x_{k,j}^c = x_{k,j} = x_j^u$. That is, we have $j \in \mathcal{I}_A^k$ and $\boldsymbol{b}_k = A_k^T(\boldsymbol{x}_k^c - \boldsymbol{x}_k) = 0$ (for simplicity we assume $i \notin \mathcal{I}_A^k$). By (25) we obtain $d_{k,j} = 0$ and the result above holds also in this case.

The similar procedure can be followed in case $x_{k,i} = x_i^l$ and/or $x_{k,j} = x_j^l$ and it can be easily generalized to the case where more variables are on the boundary. □

The last Lemma proves that Algorithm 2.1 does not stop at a non-KKT-point but moves away from the boundary when necessary. Now, we suppose that Algorithm 2.1 does not terminate and $w_k > 0$ for all k.

LEMMA 3.6. *Suppose that there exist a point $\bar{\boldsymbol{x}} \in \mathcal{F}$ and an infinite set $\mathcal{K} \subset \{1,2,\ldots\}$ such that $(\boldsymbol{x}_k)_{k\in\mathcal{K}} \to \bar{\boldsymbol{x}}$ and $(w_k)_{k\in\mathcal{K}} \to 0$. Then $\bar{\boldsymbol{x}}$ is a KKT point for problem (1).*

PROOF. Let us denote $\mathcal{J} = \{1,\ldots,n+2\}$. Using the fact that $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{y}_k)$ for all $k \geq 1$, Lemma 3.2, and Carathéodory's theorem (see, e.g. [16]), we deduce that there exist vectors $\boldsymbol{y}^{k,j}$, $\boldsymbol{\xi}^{k,j}$, and numbers $\lambda^{k,j} \geq 0$ and $\tilde{\alpha}_k$ for $j \in \mathcal{J}$ and

$k \geq 1$, such that

$$\begin{aligned}
&(\tilde{\boldsymbol{\xi}}_k, \tilde{\alpha}_k) = \sum_{j \in \mathcal{J}} \lambda^{k,j} (\boldsymbol{\xi}^{k,j}, \|\boldsymbol{y}^{k,j} - \boldsymbol{x}_k\|), \\
&\boldsymbol{\xi}^{k,j} \in \partial f(\boldsymbol{y}^{k,j}), \qquad \text{and} \\
&\sum_{j \in \mathcal{J}} \lambda^{k,j} = 1,
\end{aligned} \tag{35}$$

with

$$(\boldsymbol{y}^{k,j}, \boldsymbol{\xi}^{k,j}) \in \{(\boldsymbol{y}_i, \boldsymbol{\xi}_i) \mid i = 1, \dots, k\}.$$

From the boundedness of $(\boldsymbol{y}_k)$ (see Remark 3.1), we obtain the existence of points $\boldsymbol{y}_j^*$ ($j \in \mathcal{J}$), and an infinite set $\mathcal{K}_0 \subset \mathcal{K}$ satisfying $(\boldsymbol{y}^{k,j})_{k \in \mathcal{K}_0} \to \boldsymbol{y}_j^*$ for $j \in \mathcal{J}$. The boundedness of $(\boldsymbol{\xi}_k)$ and $(\lambda^{k,j})$ gives us the existence of vectors $\boldsymbol{\xi}_j^* \in \partial f(\boldsymbol{y}_j^*)$, numbers λ_j^* for $j \in \mathcal{J}$, and an infinite set $\mathcal{K}_1 \subset \mathcal{K}_0$ satisfying $(\boldsymbol{\xi}^{k,j})_{k \in \mathcal{K}_1} \to \boldsymbol{\xi}_j^*$ and $(\lambda^{k,j})_{k \in \mathcal{K}_1} \to \lambda_j^*$ for $j \in \mathcal{J}$.

It can be seen from (35) that

$$\lambda_j^* \geq 0 \quad \text{for } j \in \mathcal{J}, \qquad \text{and} \qquad \sum_{j \in \mathcal{J}} \lambda_j^* = 1.$$

From the fact $(w_k)_{k \in \mathcal{K}} \to 0$, equation (33), and Lemma 3.2, we obtain

$$(\tilde{\beta}_k)_{k \in \mathcal{K}} \to 0 \qquad \text{and} \qquad (\tilde{\alpha}_k)_{k \in \mathcal{K}} \to 0.$$

By letting $k \in \mathcal{K}_1$ approach infinity in (35), and by using Lemma 3.3 with

$$\begin{aligned}
&\bar{\boldsymbol{\zeta}} = \tilde{\boldsymbol{\xi}}_k, && \bar{\boldsymbol{\xi}}_j = \boldsymbol{\xi}_j^*, && \bar{\boldsymbol{y}}_j = \boldsymbol{y}_j^*, \\
&l = n + 2, && \bar{\lambda}_j = \lambda_j^* && \text{for } j \leq l,
\end{aligned}$$

we obtain $\tilde{\boldsymbol{\xi}}_k \in \partial f(\bar{\boldsymbol{x}})$.

If none of the variables is on the boundary[1] at $\bar{\boldsymbol{x}}$, we have $\tilde{\boldsymbol{\xi}}_k = \mathcal{P}_{\bar{\boldsymbol{x}}}[\tilde{\boldsymbol{\xi}}_k]$ by (6). Since $(w_k)_{k \in \mathcal{K}} \to 0$, we have $(\mathcal{P}_{\bar{\boldsymbol{x}}}[\tilde{\boldsymbol{\xi}}_k])_{k \in \mathcal{K}} \to \mathbf{0}$ by (33). Thus, $\mathbf{0} \in \partial f(\bar{\boldsymbol{x}})$ and $\bar{\boldsymbol{x}}$ is (an unconstrained) KKT point for problem (1).

Suppose now that some of the variables are on the boundary at $\bar{\boldsymbol{x}}$. By Lemma 3.5 a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$ occurs (i.e. a point $\boldsymbol{x}_k$ is not an accumulation point) if $w_k = 0$ but some components of the aggregate subgradient are of the wrong sign. Thus, we can restrict our consideration to the case the components of $\tilde{\boldsymbol{\xi}}_k$ are of the right sign or zero. Since $(w_k)_{k \in \mathcal{K}} \to 0$, we have $(\mathcal{P}_{\bar{\boldsymbol{x}}}[\tilde{\boldsymbol{\xi}}_k])_{k \in \mathcal{K}} \to \mathbf{0}$ by (33). Now, similarly to the proof of Theorem 3.4 we can

[1] If none of the variables is on the boundary, this proof is in fact exactly similar to the proof of Lemma 3.4 in [33].

choose $\mu_i^l, \mu_i^u \geq 0$ such that $\mu_i^l g_i^l(\bar{\boldsymbol{x}}) = 0$ and $\mu_i^u g_i^u(\bar{\boldsymbol{x}}) = 0$ for all $i \in \{1, \dots, n\}$ and

$$\mathbf{0} = \tilde{\boldsymbol{\xi}}_k + \sum_{i=1}^{n} \mu_i^l \nabla g_i^l(\bar{\boldsymbol{x}}) + \sum_{i=1}^{n} \mu_i^u \nabla g_i^u(\bar{\boldsymbol{x}}).$$

Thus, by Theorem (3.1) the point $\bar{\boldsymbol{x}}$ is a KKT point for problem (1). □

LEMMA 3.7. *Suppose that the number of serious steps in Algorithm 2.1 is finite and the last serious step occurred at the iteration* $m-1$. *Then there exists a number* $k^* \geq m$, *such that*

$$\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}]^T D_{k+1} \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}] \leq \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}] \qquad \textit{and}$$
$$\operatorname{tr}(D_k) < \frac{3}{2}n$$

for all $k \geq k^*$, *where* $\operatorname{tr}(D_k)$ *denotes the trace of matrix* D_k.

PROOF. The result is due to safeguarded SR1 update used (see [12, 15]). The proof is similar to the proof of Lemma 7 in [15]. □

LEMMA 3.8. *Suppose that the number of serious steps is finite and the last serious step occurred at the iteration* $m-1$. *Then,* $\boldsymbol{x}_m$ *is a KKT point for problem (1).*

PROOF. From (7), (8), (10), (11), (12), and Lemma 3.7 we obtain

$$\begin{aligned} w_{k+1} &= \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}]^T D_{k+1} \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}] + 2\tilde{\beta}_{k+1} \\ &\leq \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}] + 2\tilde{\beta}_{k+1} \\ &\leq \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k] + 2\tilde{\beta}_k = w_k \end{aligned} \tag{36}$$

for $k \geq k^*$ with k^* defined in Lemma 3.7 (for simplicity, we denote $D_k = D_k + \sigma I$ if $i_{CN} = 1$, i.e. (7) and (8) coincide). The last inequality in (36) follows from the fact that the pair $(\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_{k+1}], \tilde{\beta}_{k+1})$ minimizes function (10) over all convex combinations of pairs $(\mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_m], \beta_m)$, $(\mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_{k+1}], \beta_{k+1})$, and $(\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k], \tilde{\beta}_k)$. In addition, the line search procedure guarantees that we have

$$-\beta_{k+1} - \mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]^T D_k \mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_{k+1}] \geq -\varepsilon_R w_k$$

for all $k \geq m$. Now, due to boundedness of $\mathcal{P}_{\boldsymbol{x}_m}[\boldsymbol{\xi}_{k+1}]$, $\mathcal{P}_{\boldsymbol{x}_m}[\tilde{\boldsymbol{\xi}}_k]$, and D_k (see Remark 3.1 and Lemma 3.7) it can be proved that $w_k \to 0$ (the proof is rather similar to the proof, part(ii), of Lemma 3.6 in [33]). Now, since $\boldsymbol{x}_k = \boldsymbol{x}_m$ for all $k \geq m$, we have $\boldsymbol{x}_k \to \boldsymbol{x}_m$. Therefore, by Lemma 3.6, $\boldsymbol{x}_m$ is a KKT point for problem (1). □

REMARK 3.2. In the proof of Lemma 3.8 we showed that in case of consecutive null steps $w_k \to 0$. On the other hand, by Lemma 3.5 a serious step occurs if $w_k = 0$ but some components of the aggregate subgradient vector are of the wrong sign. Thus, if $w_k \to 0$ we either have a KKT point for the problem or a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$ occurs.

THEOREM 3.9. *Every accumulation point $\bar{\boldsymbol{x}}$ of the sequence $(\boldsymbol{x}_k)$ generated by Algorithm 2.1 is a KKT point for problem (1).*

PROOF. Let $\bar{\boldsymbol{x}}$ be an accumulation point of $(\boldsymbol{x}_k)$, and let $\mathcal{K} \subset \{1, 2, \ldots\}$ be an infinite set such that $(\boldsymbol{x}_k)_{k\in\mathcal{K}} \to \bar{\boldsymbol{x}}$. In view of Lemma 3.8, we can restrict our consideration to the case where the number of serious steps (with $t_L^k > 0$) is infinite. Let us denote

$$\mathcal{K}' = \{k \mid t_L^k > 0, \text{ there exists } i \in \mathcal{K},\ i \leq k \text{ such that } \boldsymbol{x}_i = \boldsymbol{x}_k\}.$$

Obviously, $\mathcal{K}'$ is infinite, $(\boldsymbol{x}_k)_{k\in\mathcal{K}'} \to \bar{\boldsymbol{x}}$. The continuity of f implies that $(f_k)_{k\in\mathcal{K}'} \to f(\bar{\boldsymbol{x}})$ and, thus, $f_k \downarrow f(\bar{\boldsymbol{x}})$ by the monotonicity of the sequence (f_k) obtained due to serious descent criterion (32). Using the fact that $t_L^k \geq 0$ for all $k \geq 1$ and condition (32), we obtain

$$0 \leq \varepsilon_L t_L^k w_k \leq f_k - f_{k+1} \to 0 \qquad \text{for } k \geq 1. \tag{37}$$

If the set $\mathcal{K}_1 = \{k \in \mathcal{K}' \mid t_L^k \geq t_{min}\}$ is infinite, then $(w_k)_{k\in\mathcal{K}_1} \to 0$ and $(\boldsymbol{x}_k)_{k\in\mathcal{K}_1} \to \bar{\boldsymbol{x}}$ by (37). Thus, by Lemma 3.6 $\bar{\boldsymbol{x}}$ is a KKT point for problem (1).

If the set $\mathcal{K}_1$ is finite, then the set $\mathcal{K}_2 = \{k \in \mathcal{K}' \mid \beta_{k+1} > \varepsilon_A w_k\}$ has to be infinite (see Algorithm 2.4, Step 2). To the contrary, let us assume that

$$w_k \geq \delta > 0, \qquad \text{for all } k \in \mathcal{K}_2.$$

From (37), we have $(t_L^k)_{k\in\mathcal{K}_2} \to 0$ and Step 5 in Algorithm 2.1 implies

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| = t_L^k \theta_k \|\boldsymbol{d}_k\| \leq t_L^k C$$

for all $k \geq 1$. Thus, we have $(\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|)_{k\in\mathcal{K}_2} \to 0$. By (9), (37), the boundedness of $\boldsymbol{\xi}_k$ (see Remark 3.1) and the fact that we have $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ for serious steps, we obtain $(\beta_{k+1})_{k\in\mathcal{K}_2} \to 0$, which is in contradiction with

$$\varepsilon_A \delta \leq \varepsilon_A w_k < \beta_{k+1}, \qquad k \in \mathcal{K}_2.$$

Therefore, there exists an infinite set $\mathcal{K}_3 \subset \mathcal{K}_2$ such that $(w_k)_{k\in\mathcal{K}_3} \to 0$ and $(\boldsymbol{x}_k)_{k\in\mathcal{K}_3} \to \bar{\boldsymbol{x}}$. By Lemma 3.6 $\bar{\boldsymbol{x}}$ is a KKT point for problem (1). □

4 Numerical Experiments

We now compare the proposed globally convergent limited memory bundle method `LMBM-B` to the proximal bundle method `PBNCGC` (version 2.0, [24, 25]) in a limited number of nonsmooth large-scale test problems. We used the solver `PBNCGC` as a benchmark since the proximal bundle method is the most frequently used bundle method in nonsmooth optimization. In addition, we compared the new version of `LMBM-B` to the older, non-globally convergent, version `LMBM-B-OLD` [13]. The experiments were performed in a Intel® Core™ 2

CPU 1.80GHz and all the algorithms were implemented in Fortran77 with double-precision arithmetic.

The solvers were tested with 10 nonsmooth academic minimization problems described in [14][2]. The number of variables used in our experiments were 1000, 2000, and 4000. The problems in [14] are unconstrained but we inclosed the additional bounds

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

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

The solvers were tested with relatively small amount of stored subgradient information. That is, the size of the bundle m_ξ was set to 10 for `LMBM-B` and `LMBM-B-OLD` and to 100 for `PBNCGC` (since the previous experiments [12, 15] have shown that a larger bundle usually works better with `PBNCGC`). For `LMBM-B` the bundle size larger to 2 affects only if $w_k = 0$ but the current iteration point x_k is not a KKT-point. With both `LMBM-B` and `LMBM-B-OLD`, we used the values $\hat{m}_u = 15$ and $\hat{m}_c = 7$ due to good results of previous experiments with the older version [13]. For convex problems (problems 1 – 5 in [14]), we used the distance measure parameter $\gamma = 0$ and for nonconvex problems (problems 6 – 10 in [14]), we used the value $\gamma = 0.5$ with all the solvers. The final accuracy parameter $\varepsilon = 10^{-5}$ was used in all the cases. Otherwise, we used the default parameters of the solvers.

In addition to the usual stopping criteria of the solvers, we terminated the experiments if the CPU time elapsed exceeded half an hour.

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

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

The proximal bundle solver `PBNCGC` always needed less function evaluations than the different variants of LMBM-B. However, as can be seen when comparing

[2] All these problems can be downloaded from the website http://napsu.karmitsa.fi/lmbm

Table 1: Results for box constrained problems with 1000 variables.

Solver	LMBM-B		LMBM-B-OLD		PBNCGC	
Problem	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
1	-/-	fail	11346/11385	0.01	-/-	fail
2	-/-	fail	676/825	0.26651	18/19	$8.2 \cdot 10^{-6}$
3	286/2029	-1396.08	35/62	-1395.45	18/23	-1396.12
4	157/921	2334.75	64/86	2334.75	25/26	2334.75
5	95/632	2042.62	63/252	2042.63	25/26	2042.62
6	278/291	0.09547	526/526	0.09531	9/25	0.40523
7	293/3106	99.9000	160/694	99.9001	300/395	99.9192
8	120/648	-698.121	74/310	-698.099	35/36	-698.172
9	129/895	8.45406	55/123	8.45415	46/74	8.45407
10	142/812	147.301	63/137	147.299	22/33	147.299
Time	1.07		0.21*		67.48	

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

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

Solver	LMBM-B		LMBM-B-OLD		PBNCGC	
Problem	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
1	-/-	fail	-/-	fail	-/-	fail
2	-/-	fail	3375/3453	0.38118	20/22	$3.0 \cdot 10^{-6}$
3	112/644	-2793.50	36/71	-2791.41	19/24	-2793.63
4	150/749	4671.97	74/139	4671.97	22/23	4671.97
5	101/421	4087.25	74/276	4087.26	28/29	4087.25
6	517/522	0.09531	-/-	fail	-/-	fail
7	63/427	200.717	53/91	199.979	69/82	200.251
8	138/686	-1396.05	68/164	-1396.68	31/32	-1396.93
9	86/500	16.9159	63/177	16.9068	35/55	16.9065
10	146/812	294.911	108/110	294.792	36/48	294.793
Time	1.11		0.68		540.14	

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

Solver	LMBM-B		LMBM-B-OLD		PBNCGC	
Problem	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
1	-/-	fail	-/-	fail	-/-	fail
2	-/-	fail	-/-	fail	24/28	$1.8 \cdot 10^{-6}$
3	66/403	-5555.69	42/71	-5587.79	15/20	-5588.65
4	172/946	9346.40	88/126	9346.40	13/14	9346.54
5	149/686	8176.57	100/152	8176.57	13/14	8176.63
6	-/-	fail	-/-	fail	19/38	28.5408
7	281/2369	399.900	-/-	fail	14/17	404.086
8	120/521	-2794.24	123/329	-2794.39	11/12	-2784.70
9	129/639	33.8113	76/128	33.8270	13/17	35.6548
10	204/1117	589.780	95/230	589.946	13/17	593.702
Time	5.57		1.90		1562.28	

the computational times, each individual iteration with PBNCGC was much more costly than that with LMBM-B or LMBM-B-OLD. Indeed, with PBNCGC all the problems with 4000 variables but two (problems 2 and 6) were terminated because the time limit exceeded. This also explains the inaccurate results obtained with PBNCGC (see Table 3).

The new variant LMBM-B failed to solve two of the problems (problems 1 and 2) with any number of variables tested. These failures were quite predictable, since both of these problems are reported to be difficult to solve with limited memory bundle method even without the box constraints [14]. However, the termination conditions in problem 1 deserves to be better known: a point where solver LMBM-B stopped was a non-KKT point with $w_k = 0$. In Lemma 3.5 we have proved that Algorithm 2.1 does not stop at a non-KKT-point but a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$ occurs. When studying these cases more carefully, we noticed that, indeed, a serious step with $\boldsymbol{x}_{k+1} \neq \boldsymbol{x}_k$ occurred but the differences between consecutive function values were so small that the solver stopped because the the value of objective function did not change enough in 10 straight iterations. If we removed the stopping condition that allows us stop if no significant improvement in function values occurs, the solver did progress but, unfortunately, very slowly.

In the case of failure or inaccurate result, the most common reason for termination with PBNCGC was that the time was up, while with LMBM-B and LMBM-B-OLD it was that the value of objective function did not change enough in last 10 iterations (with serious steps). The final values of the objective function for problem 6 with $n = 4000$ obtained with LMBM-B and LMBM-B-OLD were smaller than that of PBNCGC implying they were converging to different local minima before the failure occurred. This was also the case (without failures) with this same problem and 1000 variables.

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

5 Conclusions

In this paper, we have described a new variant LMBM-B of the limited memory bundle method for box constrained nonsmooth large-scale optimization. We have proved the global convergence of the method for locally Lipschitz continuous functions, which are not necessarily differentiable or convex.

The preliminary numerical experiments confirm that LMBM-B is efficient for both convex and nonconvex large-scale nonsmooth optimization problems. With large numbers of variables it used significantly less CPU time than the proximal bundle method tested. Moreover, the difference between the computational times of this globally convergent variant and the previous (non convergent) version of the method was not as large as we foreboded.

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

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

Acknowledgements

This work was financially supported by University of Jyväskylä (Finland) and University of Turku (Finland).

Appendix

LEMMA 5.1. *The condition*

$$-\boldsymbol{s}_i^T\boldsymbol{u}_i + (t_R^i\theta_i)^2 \max\{(\boldsymbol{x}_i^c - \boldsymbol{x}_i)^T B_i(\boldsymbol{x}_i^c - \boldsymbol{x}_i), (-A_i\boldsymbol{\mu}_i^* - \tilde{\boldsymbol{\xi}}_i)^T\boldsymbol{d}_i^*\} < 0 \quad (38)$$

for all $i = 1, \ldots, k-1$ *assures the positive definiteness of matrices obtained by limited memory SR1 updates.*

PROOF. The limited memory SR1 updates D_k and B_k ($k \geq 1$, $D_1 = B_1 = I$) are positive definite if

$$\boldsymbol{s}_i^T(\boldsymbol{u}_i - B_i\boldsymbol{s}_i) > 0$$

for all $i = 1, \ldots, k$ (see the proof of Lemma 10 in [15]). Now, since $\boldsymbol{d}_i = (1 - \alpha_i^*)(\boldsymbol{x}_i^c - \boldsymbol{x}_i) + \alpha_i^*\boldsymbol{d}_i^*$, $B_i\boldsymbol{d}_i^* = -A_i\boldsymbol{\mu}_i^* - \tilde{\boldsymbol{\xi}}_i$ (see, Subsection 2.4), and due to convexity of function $\|\cdot\|^2$, we obtain

$$
\begin{aligned}
\boldsymbol{s}_i^T B_i \boldsymbol{s}_i &= (t_R^i\theta_i)^2 \boldsymbol{d}_i^T B_i \boldsymbol{d}_i \\
&= (t_R^i\theta_i)^2((1-\alpha_i^*)(\boldsymbol{x}_i^c - \boldsymbol{x}_i) + \alpha_i^* \boldsymbol{d}_i^*)^T B_i((1-\alpha_i^*)(\boldsymbol{x}_i^c - \boldsymbol{x}_i) + \alpha_i^* \boldsymbol{d}_i^*) \\
&= (t_R^i\theta_i)^2 \|(1-\alpha_i^*)N_i(\boldsymbol{x}_i^c - \boldsymbol{x}_i) + \alpha_i^* N_i \boldsymbol{d}_i^*\|^2 \\
&\leq (t_R^i\theta_i)^2 \max\{\|N_i(\boldsymbol{x}_i^c - \boldsymbol{x}_i)\|^2, \|N_i \boldsymbol{d}_i^*\|^2\} \\
&= (t_R^i\theta_i)^2 \max\{(\boldsymbol{x}_i^c - \boldsymbol{x}_i)^T B_i (\boldsymbol{x}_i^c - \boldsymbol{x}_i), (\boldsymbol{d}_i^*)^T B_i \boldsymbol{d}_i^*\} \\
&= (t_R^i\theta_i)^2 \max\{(\boldsymbol{x}_i^c - \boldsymbol{x}_i)^T B_i (\boldsymbol{x}_i^c - \boldsymbol{x}_i), (-A_i\boldsymbol{\mu}_i^* - \tilde{\boldsymbol{\xi}}_i)^T \boldsymbol{d}_i^*\}
\end{aligned}
$$

where we have denoted $B_i = N_i^T N_i$. Therefore, if condition (38) is valid, we have $\boldsymbol{s}_i^T(\boldsymbol{u}_i - B_i\boldsymbol{s}_i) > 0$ for all $i = 1, \ldots, k-1$. □

Condition (38) also guarantees $\boldsymbol{s}_i^T \boldsymbol{u}_i > 0$ for all $i = 1, \ldots, k-1$ that is the classical condition for the positive definiteness of the (limited memory) BFGS updates. Since we check condition (38) also during the limited memory BFGS update before updating matrices, all the matrices D_k and B_k formed in LMBM-B are positive definite.

Note that testing of the positive definiteness does not require any additional matrix calculations since $(\boldsymbol{x}_i^c - \boldsymbol{x}_i)^T B_i (\boldsymbol{x}_i^c - \boldsymbol{x}_i)$ has already been calculated at previous iteration during the computation of generalized Cauchy point. If there is no bounds on the variables this positive definiteness condition reverts to the condition of unconstrained version (see [15]).

References

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

[2] Ben-Tal, A., and Nemirovski, A. Non-Euclidean restricted memory level method for large-scale convex optimization. *Mathematical Programming 102*, 3 (2005), 407–456.

[3] Bihain, A. Optimization of upper semidifferentiable functions. *Journal of Optimization Theory and Applications 4* (1984), 545–568.

[4] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing 16*, 5 (1995), 1190–1208.

[5] 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.

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

[7] CLARKE, F. H., LEDYAEV, Y. S., STERN, R. J., AND WOLENSKI, P. R. *Nonsmooth Analysis and Control Theory*. Springer, New York, 1998.

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

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

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

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

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

[13] HAARALA, M., AND MÄKELÄ, M. M. Limited memory bundle algorithm for large bound constrained nonsmooth minimization problems. Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B. 1/2006 University of Jyväskylä, Jyväskylä, 2006.

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

[15] 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.

[16] HIRIART-URRUTY, J.-B., AND LEMARÉCHAL, C. *Convex Analysis and Minimization Algorithms I*. Springer-Verlag, Berlin, 1993.

[17] HIRIART-URRUTY, J.-B., AND LEMARÉCHAL, C. *Convex Analysis and Minimization Algorithms II*. Springer-Verlag, Berlin, 1993.

[18] 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.

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

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

[21] LIU, D. C., AND NOCEDAL, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming 45* (1989), 503–528.

[22] 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.

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

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

[25] MÄKELÄ, M. M., AND NEITTAANMÄKI, P. *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control*. World Scientific Publishing Co., Singapore, 1992.

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

[27] MISTAKIDIS, E. S., AND STAVROULAKIS, G. E. *Nonconvex Optimization in Mechanics. Smooth and Nonsmooth Algorithms, Heuristics and Engineering Applications by the F.E.M.* Kluwert Academic Publishers, Dordrecht, 1998.

[28] MOREAU, J. J., PANAGIOTOPOULOS, P. D., AND STRANG, G., Eds. *Topics in Nonsmooth Mechanics*. Birkhäuser Verlag, Basel, 1988.

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

[30] OUTRATA, J., KOČVARA, M., AND ZOWE, J. *Nonsmooth Approach to Optimization Problems With Equilibrium Constraints. Theory, Applications and Numerical Results*. Kluwert Academic Publisher, Dordrecht, 1998.

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

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

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

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 978-952-12-2063-0
ISSN 1239-1891