JYVÄSKYLÄ STUDIES IN COMPUTING 40

Marjo Haarala

Large-Scale Nonsmooth Optimization

Variable Metric Bundle Method with Limited Memory

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi Villa Rana -rakennuksen Paulaharju-salissa
marraskuun 13. päivänä 2004 kello 14.

Academic dissertation to be publicly discussed, by permission of
the the Faculty of Information Technology of the University of Jyväskylä,
in the Building Villa Rana, Paulaharju Hall, on November 13, 2004 at 14 o'clock.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2004

Large-Scale Nonsmooth Optimization

Variable Metric Bundle Method with Limited Memory

JYVÄSKYLÄ STUDIES IN COMPUTING 40

Marjo Haarala

Large-Scale Nonsmooth Optimization

Variable Metric Bundle Method with Limited Memory

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2004

Editors
Tommi Kärkkäinen
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Marja-Leena Tynkkynen
Publishing Unit, University Library of Jyväskylä

URN:ISBN 9513919277
ISBN 951-39-1927-7 (PDF)

ISBN 951-39-1908-0 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä
and ER-Paino Ky, Lievestuore 2004

ABSTRACT

Haarala, Marjo
Large-Scale Nonsmooth Optimization: Variable Metric Bundle Method with Limited Memory
Jyväskylä: University of Jyväskylä, 2004, 107 pages
(Jyväskylä Studies in Computing,
ISSN 1456-5390; 40)
ISBN 951-39-1927-7
Finnish summary
diss.

Many practical optimization problems involve nonsmooth (that is, not necessarily differentiable) functions of hundreds or thousands of variables. In such problems, the direct application of smooth gradient-based methods may lead to a failure due to the nonsmooth nature of the problem. On the other hand, none of the current general nonsmooth optimization methods is efficient in large-scale settings. The motivation of this work is to develop efficient and reliable solvers for large-scale nonsmooth optimization problems.

In this thesis, we introduce a new limited memory bundle method for nonsmooth large-scale optimization. The new method is a hybrid of the variable metric bundle method and the limited memory variable metric methods, where the former has been developed for small- and medium-scale nonsmooth optimization and the latter have been developed for large-scale smooth optimization. The new limited memory bundle method aims at filling the gap that exists in the field of nonsmooth optimization with large numbers of variables.

Besides describing the new limited memory bundle method in detail, we prove its global convergence for locally Lipschitz continuous objective functions, which are not supposed to be differentiable or convex. In addition, we give some modifications to the basic method in order to improve the accuracy of the method without losing much in its efficiency.

The efficiency and reliability of the new method and its modifications are demonstrated with numerical experiments. The problems included in our experiments contain both academic test problems and practical applications arising in the field of nonsmooth large-scale optimization.

Keywords: Nonsmooth optimization, large-scale optimization, bundle methods, variable metric methods, limited memory methods, nondifferentiable programming.

Author Marjo Haarala

Department of Mathematical Information Technology
University of Jyväskylä
Finland

Supervisors Doctor Marko M. Mäkelä

Department of Mathematical Information Technology
University of Jyväskylä
Finland

Professor Kaisa Miettinen

Helsinki School of Economics
Finland

or

Department of Mathematical Information Technology
University of Jyväskylä
Finland

Reviewers Professor Manlio Gaudioso

Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria
Italia

Doctor Ladislav Lukšan

Department of Computational Methods
Institute of Computer Science
Academy of Sciences of the Czech Republic
Czech Republic

Opponent Professor Per Olov Lindberg

Department of Mathematics
Linköping University
Sweden

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisors Dr. Marko Mäkelä and Prof. Kaisa Miettinen for excellent guidance and continuous support throughout my research. I am very grateful to Marko Mäkelä for introducing me into this field of nonsmooth optimization. I am also grateful to Dr. Ladislav Lukšan and Dr. Jan Vlček for giving a permission to use and modify their variable metric bundle software to make the method suitable for large-scale optimization. I would like to thank Prof. Manlio Gaudioso and Dr. Ladislav Lukšan for reviewing the manuscript and giving encouraging feedback. Dr. Ladislav Lukšan deserves special thanks for enlightening discussions concerning the limited memory bundle method.

I would like to express my appreciation to the Optimization Group at the University of Jyväskylä for their feedback and support. I also want to thank my common-law husband Tommi Ronkainen for advice concerning the computational implementations and Dr. Kirsi Majava for providing some practical applications for testing purposes.

This work was financially supported by COMAS Graduate School of the University of Jyväskylä, Academy of Finland grant #104641 (project of Kaisa Miettinen), and Emil Aaltonen Foundation. The computational resources used in experiments were provided by CSC, the Finnish IT Center for Science.

Finally, I am deeply indebted to my family and friends for their support, especially to Tommi for his patience and understanding.

Jyväskylä, 13 September, 2004
Marjo Haarala

LIST OF SYMBOLS

$\mathbb{R}^n$	n-dimensional Euclidean space
$\mathbb{N}$	set of natural numbers
$a, b, c, \varepsilon, \lambda$	scalars
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	(column) vectors
$\boldsymbol{x}^T$	transposed vector
$\boldsymbol{x}^T\boldsymbol{y}$	inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$
$\|\boldsymbol{x}\|$	norm of $\boldsymbol{x}$ in $\mathbb{R}^n$, $\|\boldsymbol{x}\| = (\boldsymbol{x}^T\boldsymbol{x})^{\frac{1}{2}}$
x_i	ith component of vector $\boldsymbol{x}$
$(\boldsymbol{x}_k)$	sequence of vectors
$B(\boldsymbol{x}, r)$	open ball with radius r and central point $\boldsymbol{x}$
(a, b)	open interval
$[a, b]$	closed interval
$[a, b)$, $(a, b]$	half-open intervals
S	set
$\cap_{i=1}^m S_i$	intersection of sets S_i, $i = 1, \ldots, m$
$\operatorname{conv} S$	convex hull of set S
$\mathcal{P}(S)$	set of all subset in S
$f(\boldsymbol{x})$	objective function value at $\boldsymbol{x}$
$\nabla f(\boldsymbol{x})$	gradient of function f at $\boldsymbol{x}$
$\partial f(\boldsymbol{x})/\partial x_i$	partial derivative of function f with respect to x_i
$C^m(\mathbb{R}^n)$	the space of functions $f : \mathbb{R}^n \to \mathbb{R}$ with continuous partial derivatives up to order m
$f'(\boldsymbol{x}; \boldsymbol{d})$	directional derivative of function f at $\boldsymbol{x}$ in the direction $\boldsymbol{d}$
$f^\circ(\boldsymbol{x}; \boldsymbol{d})$	generalized directional derivative of function f at $\boldsymbol{x}$ in the direction $\boldsymbol{d}$
$\partial_c f(\boldsymbol{x})$	subdifferential of convex function f at $\boldsymbol{x}$
$\partial f(\boldsymbol{x})$	subdifferential of function f at $\boldsymbol{x}$
$\partial_\varepsilon^G f(\boldsymbol{x})$	Goldstein ε-subdifferential of function f at $\boldsymbol{x}$
$\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$	subgradient of function f at $\boldsymbol{x}$
Ω_f	a set in $\mathbb{R}^n$ where function f is not differentiable
$\arg\min f(\boldsymbol{x})$	point where function f attains its minimum value
$\bar{f}_\xi$	ξ-linearization of function f
$\hat{f}_{\boldsymbol{x}}$	linearization of function f at $\boldsymbol{x}$
A, B, G	matrices
$(A)_{ij}$	element of matrix A in row i of column j
H	Hessian matrix $H = \nabla^2 f(\boldsymbol{x})$
B	(generalized) approximation of the Hessian matrix
D	(generalized) approximation of the inverse of the Hessian matrix
I	identity matrix
A^T	transposed matrix
A^{-1}	inverse of matrix A

$\mathrm{tr}(A)$	trace of matrix A
$\mathrm{diag}[\theta_1, \ldots, \theta_n]$	diagonal matrix with diagonal elements $\theta_1, \ldots, \theta_n$
$\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$	sets of indices
$\lvert\mathcal{I}\rvert$	number of elements in set $\mathcal{I}$
$x \downarrow 0$	$x \to 0_+$
$O(\cdot)$	time complexity or space requirement of algorithm
$\mathrm{div}(i, j)$	integer division for positive integers i and j
$\mathrm{mod}(i, j)$	remainder after integer division

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The classical theory of optimization presumes certain differentiability and strong regularity assumptions (see, e.g., [16]). However, these assumptions are too demanding for many practical applications, since the functions involved are often nonsmooth, that is, they are not necessarily differentiable. The source of the nonsmoothness may be the objective function itself, its possible interior function, or both. For example, in economics, tax models typically consist of several different pieces that have no continuous gradients at their intersections (see, e.g., [43]), in steel industry, the phase changes of material typically contain various discontinuities and irregularities in the original phenomena (see, e.g., [64]), and, in optimal control problems, the nonsmoothness is usually caused by some extra technological constraints (see, e.g., [63]). Moreover, there exist so-called stiff problems that are analytically smooth but numerically nonsmooth. This means that the gradient varies too rapidly and, thus, these problems behave like nonsmooth problems.

In addition to the nonsmoothness, many practical optimization problems involve large numbers of variables (see, e.g., [31, 62, 63, 69]). The intention of this work is to develop efficient and reliable solvers for large-scale nonsmooth optimization problems. Let us, however, start by giving a general overview of the field of research to point out the basic ideas and to motivate the need of solvers for large-scale nonsmooth optimization problems.

Let us consider a nonlinear unconstrained optimization problem of the form

$$\begin{cases} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n, \end{cases} \tag{1.1}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is supposed to be locally Lipschitz continuous at $\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^n$. If f is continuously differentiable, then problem (1.1) is said to be smooth. Otherwise, problem (1.1) is said to be nonsmooth.

Methods for solving problem (1.1) are usually iterative (see, e.g., [16, 28, 76]). The basic idea of iterative minimization methods is, by starting from a given initial point $\boldsymbol{x}_1 \in \mathbb{R}^n$, to generate a sequence $(\boldsymbol{x}_k) \subset \mathbb{R}^n$ that converges to a (local) minimum point $\boldsymbol{x}^*$ of the objective function f. In other words, $\boldsymbol{x}_k \to \boldsymbol{x}^*$ whenever

$k \to \infty$. If $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ for all k, then an iterative method is called a descent method. The next iteration point $\boldsymbol{x}_{k+1}$ for a descent method is defined by the formula $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_k \boldsymbol{d}_k$, where $\boldsymbol{d}_k$ is a descent direction for the objective function (that is, there exists $\varepsilon > 0$ such that $f(\boldsymbol{x}_k + t\boldsymbol{d}_k) < f(\boldsymbol{x}_k)$ for all $t \in (0, \varepsilon]$), and t_k is the step size such that $t_k \approx \arg\min_{t>0} f(\boldsymbol{x}_k + t\boldsymbol{d}_k)$.

Most optimization algorithms are constructed for the smooth version of problem (1.1). For smooth objective functions, a descent direction may be generated by exploiting the fact that the direction opposite to the gradient is locally the direction of steepest descent. Then, the step size can be determined, for example, by using some line search techniques, which usually employ some efficient univariate smooth optimization method or some polynomial interpolation (see, e.g., [16]). Furthermore, for smooth objective functions, a necessary condition for local optimality is that the gradient has to be zero at each local solution and by continuity it becomes small whenever we approach an optimal point. This fact provides a useful stopping criterion for smooth iterative methods.

In this thesis, we consider the nonsmooth version of problem (1.1). The direct application of smooth gradient-based methods to nonsmooth problems may lead to a failure in convergence, in optimality conditions, or in gradient approximation (see, e.g., [43]). On the other hand, derivative free methods, for example, Powel's method (see, e.g., [16]) are quite unreliable and become inefficient when the size of the problem increases. We can say that nonsmooth optimization deals with a broader class of problems than smooth optimization in the sense that nonsmooth optimization techniques can be successfully applied to smooth problems but not vice versa.

In nonsmooth optimization, we may get into a situation, where the objective function fails to have a derivative for some values of the variables. Thus, we have to use so-called generalized gradients (or subgradients) instead of gradients. This allows us to generalize the effective smooth gradient-based methods for nonsmooth problems. The methods for solving nonsmooth optimization problems can be divided into two main classes: subgradient methods (see, e.g., [76]) and bundle methods (see, e.g., [20, 29, 33, 60, 63, 73]). All of these methods are based on the assumptions that the objective function is locally Lipschitz continuous and the value of the function and its arbitrary subgradient at each point are available.

The basic idea behind the subgradient methods (Kiev methods) is to generalize smooth methods by replacing the gradient with an arbitrary subgradient. Due to this simple structure, they are widely used methods in nonsmooth optimization. However, there exist some serious drawbacks in these methods. Firstly, a nondescent search direction may occur, because the direction opposite to an arbitrary subgradient is not necessarily a descent one (see, e.g., [43]). Thus, any standard line search operation can not be applied for step size selection. Secondly, due to the fact that the norm of an arbitrary subgradient does not necessarily become small in the neighborhood of an optimal point, there exists no subgradient based implementable stopping criterion. Moreover, the convergence speed of subgradient methods is poor (not even linear) (see, e.g., [43]). An extensive overview of various subgradient methods can be found in [76].

At the moment, bundle methods are regarded as the most effective and reliable methods for nonsmooth optimization (see, e.g., [63]). They are based on the subdifferential theory developed by Rockafellar [70] and Clarke [11], where the classical differential theory is generalized for convex and locally Lipschitz continuous functions, respectively. The basic idea of bundle methods is to approximate the subdifferential (that is, the set of subgradients) of the objective function by gathering subgradients from previous iterations into a bundle. In this way, we obtain more information about the local behavior of the function than what an individual arbitrary subgradient can yield.

The history of bundle methods was initiated with the ε-steepest descent method introduced by Lemaréchal [40]. The idea of this method is to combine the cutting plane model [32] with the conjugate subgradient method [39]. The main difficulty of the ε-steepest descent method is the selection of the approximation tolerance that controls the radius of the ball, in which the cutting plane model is supposed to be a good approximation to the objective function. To avoid this difficulty, the idea of the generalized cutting plane method was defined by Lemaréchal [41] and the method was further developed by Kiwiel [33]. The basic idea of the generalized cutting plane method is to construct a convex piecewise linear approximation to the objective function by using linearizations generated by the subgradients. In [33], Kiwiel also introduced two strategies to bound the number of stored subgradients, namely the subgradient selection and the subgradient aggregation. Since then, especially, the subgradient aggregation strategy has been widely applied in various bundle methods.

In spite of the different backgrounds, both Lemaréchal's ε-steepest descent method and Kiwiel's generalized cutting plane method generate the search direction by solving quadratic direction finding problems being closely related. The generalized cutting plane method avoids the difficulties of the ε-steepest descent method but, then, it has been found to be very sensitive to the scaling of the objective function (that is, multiplication of f by a positive constant) [58].

The next improvement of bundle methods was the development of the proximal bundle method by Kiwiel [35] and the bundle trust region method by Schramm and Zowe [73]. The proximal bundle method is based on the proximal point algorithm of [71] and the work of [1], while the idea of the bundle trust region method is to combine the bundle idea with the classical trust region method (see, e.g., [16]). There exist strong similarities between these methods and, in fact, they use approximately the same algorithm that differs only in technical details.

Later, various different bundle methods have been proposed. Among them are tilted bundle methods [36], where the cutting plane model is replaced by a so-called tilted cutting plane model, level bundle methods [4, 44] which are based on the minimization of the stabilizing quadratic term subject to some level set of the linearization, and a DC piecewise affine model with bundling technique [18] that uses the difference of two piecewise linear convex functions to construct a model for a nonconvex objective function. In addition, there exist variable metric bundle methods [3, 20, 54, 78] that exploit the second-order information of the objective function in the form of an approximated Hessian matrix, and the bundle-Newton

method [53] that uses second-order information in order to construct a quadratic model of the objective function. For a thorough overview of various bundle methods we refer to [60].

In their present form, bundle methods are efficient for small- and medium-scale problems. However, their computational demand expands in large-scale problems with more than 500 variables [27]. This is explained by the fact that bundle methods need relatively large bundles to be capable of solving the problems effectively. In other words, the size of the bundle has to be approximately the same as the number of variables [33] and, thus, the quadratic direction finding problem becomes very time-consuming to solve.

In variable metric bundle methods introduced by Lukšan and Vlček [54, 78], the search direction is calculated by using the variable metric updates. Thus, the quite complicated quadratic direction finding problem appearing in standard bundle methods is not solved at all. Furthermore, the subgradient aggregation is done by using only three subgradients and, thus, the size of the bundle is independent of the dimension of the problem. However, variable metric bundle methods use dense matrices to calculate the search direction and, thus, due to massive matrix operations, also these methods become inefficient when the dimension of the problem increases.

We have not found any general bundle-based solver for large-scale nonsmooth optimization problems in the literature. Thus, we can say that at the moment the only possibility to optimize nonsmooth large-scale problems is to use some variant of subgradient methods. However, as mentioned before, the basic subgradient methods suffer from some serious disadvantages and, on the other hand, the more advanced variable metric based subgradient methods (see, e.g., [76]) using dense matrices suffer from the same drawbacks as the variable metric bundle methods. Lately, some subgradient methods based on mirror descent algorithms have been proposed (see e.g., [2]). However, in these methods the objective function is required to be convex and Lipchitz continuous. This means that there is an evident need of reliable and efficient solvers for general, possible nonconvex, nonsmooth large-scale optimization problems.

In this thesis, we introduce a new limited memory bundle method for finding the unconstrained local minimum of a nonsmooth and possible nonconvex objective function with large numbers of variables. The new method is a hybrid of the variable metric bundle method [54, 78] and the limited memory variable metric methods (see, e.g., [8, 65]), where the latter have been developed for smooth large-scale optimization. The new method combines in a novel way the ideas of the variable metric bundle method with the search direction calculation of the limited memory approach. Therefore, the time-consuming quadratic direction finding problem appearing in the standard bundle methods does not need to be solved and the number of stored subgradients does not depend on the dimension of the problem. Furthermore, the method 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 method [54, 78]. These properties make the limited memory bundle method suitable for large-scale optimization. As a matter of

fact, the number of operations needed for the calculation of the search direction and the aggregate values is only linearly dependent on the number of variables while, for example, with the original variable metric bundle method, this dependence is quadratic.

In addition to the basic limited memory bundle method, we give some modifications of it based on the different scaling and skipping strategies of the limited memory variable metric updates and the adaptability of the storage space required. However, before introducing the new method, we present some essential results of nonsmooth analysis and give a short description of some basic methods for (small-scale) nonsmooth optimization. Moreover, for the convenience of the reader, we give a description of the smooth limited memory variable metric methods.

In order to get some impression about how the different optimization methods (including our new method and its different modifications) operate in practice, we have tested some of them with large-scale minimization problems. Thus, in addition to the descriptions of the methods, we are able to give some details of the performance of the methods. The numerical results to be presented demonstrate the usability and the reliability of the new limited memory bundle methods with both convex and nonconvex large-scale nonsmooth minimization problems.

This thesis is organized as follows. In Chapter 2, we first recall some notations and basic results of smooth analysis. Then we generalize the concepts of differential calculus for convex and locally Lipschitz continuous functions, and present some basic results. At the end of the chapter, we generalize the classical optimality conditions to the nonsmooth case.

In Chapter 3, we give a short review of some smooth optimization methods. First, we consider the standard variable metric methods for small- and medium-scale problems. Then, we give the basic ideas of the limited memory BFGS method based on [65]. At the end of Chapter 3, we discuss compact representations of matrices generated by the limited memory variable metric updates [8], and show how to use them efficiently in limited memory methods.

Chapter 4 is devoted to basic methods for (small-scale) nonsmooth optimization. First, we give a survey of standard bundle methods, and then, we present the basic ideas of the variable metric bundle method that is a hybrid of variable metric and bundle methods.

The ideas of the variable metric bundle method and the limited memory variable metric methods are then used in Chapter 5, where we construct the new method for large-scale nonsmooth unconstrained optimization. In this chapter, we first present the basic limited memory bundle algorithm and prove the global convergence of the method for locally Lipschitz continuous functions that are not supposed to be differentiable or convex. Then, we give some modifications to the basic algorithm.

In Chapter 6, we analyze some numerical experiments concerning some of the methods presented in Chapters 3, 4, and the different modifications of the new method presented in Chapter 5. The problems solved contain both smooth and nonsmooth academic test problems as well as some practical applications arising in the field of nonsmooth large-scale optimization.

Finally, in Chapter 7, we conclude by giving a short summary of the performance of the methods described. Furthermore, we give some ideas of further development.

2 THEORETICAL BACKGROUND

In this chapter, we first collect some notations and basic results of smooth analysis. Then we generalize the concepts of differential calculus for convex, not necessarily differentiable functions [70]. We define subgradients and subdifferentials and present some basic results. After that, we generalize the convex differential theory to locally Lipschitz continuous functions [11] and define so-called ε-subdifferentials that approximate the ordinary subdifferentials. In the third part of this chapter, we generalize the classical optimality conditions: We give the necessary conditions for a locally Lipschitz continuous function to attain its minimum in an unconstrained case. Moreover, we define some notions of linearizations for locally Lipschitz continuous functions and present their basic properties.

The proofs of this chapter are omitted since they can be found, for example, in [63].

2.1 Notations and Definitions

All the vectors $\boldsymbol{x}$ are considered as column vectors and, correspondingly, all the transposed vectors $\boldsymbol{x}^T$ are considered as row vectors. We denote by $\boldsymbol{x}^T\boldsymbol{y}$ the usual inner product and by $\|\boldsymbol{x}\|$ the norm in the n-dimensional real Euclidean space $\mathbb{R}^n$. In other words,

$$\boldsymbol{x}^T\boldsymbol{y} = \sum_{i=1}^{n} x_i y_i \qquad \text{and} \qquad \|\boldsymbol{x}\| = \left(\boldsymbol{x}^T\boldsymbol{x}\right)^{\frac{1}{2}},$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are in $\mathbb{R}^n$ and $x_i, y_i \in \mathbb{R}$ are the ith components of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively.

An open ball with center $\boldsymbol{x} \in \mathbb{R}^n$ and radius $r > 0$ is denoted by $B(\boldsymbol{x}; r)$, that is,

$$B(\boldsymbol{x}; r) = \{\, \boldsymbol{y} \in \mathbb{R}^n \mid \|\boldsymbol{y} - \boldsymbol{x}\| < r \,\}.$$

We denote by $[\boldsymbol{x}, \boldsymbol{y}]$ the closed line-segment joining $\boldsymbol{x}$ and $\boldsymbol{y}$, that is,

$$[\boldsymbol{x}, \boldsymbol{y}] = \{\, \boldsymbol{z} \in \mathbb{R}^n \mid \boldsymbol{z} = \lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \ \text{ for } \ 0 \le \lambda \le 1 \,\},$$

and by $(\boldsymbol{x}, \boldsymbol{y})$ the corresponding open line-segment.

A set $S \subset \mathbb{R}^n$ is said to be *convex* if

$$\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in S$$

whenever $\boldsymbol{x}$ and $\boldsymbol{y}$ are in S and $\lambda \in [0,1]$. Geometrically this means that the closed line-segment $[\boldsymbol{x}, \boldsymbol{y}]$ is entirely contained in S whenever its endpoints $\boldsymbol{x}$ and $\boldsymbol{y}$ are in S. If $S_i \subset \mathbb{R}^n$ are convex sets for $i = 1, \ldots, m$, then their intersection $\cap_{i=1}^m S_i$ is also convex.

A linear combination $\sum_{i=1}^k \lambda_i \boldsymbol{x}_i$ is called a *convex combination* of elements $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k \in \mathbb{R}^n$ if each $\lambda_i \ge 0$ and $\sum_{i=1}^k \lambda_i = 1$.

The intersection of all the convex sets containing a given subset $S \subset \mathbb{R}^n$ is called the *convex hull* of set S and it is denoted by $\operatorname{conv} S$. For any $S \subset \mathbb{R}^n$, $\operatorname{conv} S$ consists of all the convex combinations of the elements of S, that is,

$$\operatorname{conv} S = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} = \sum_{i=1}^k \lambda_i \boldsymbol{x}_i, \ \sum_{i=1}^k \lambda_i = 1, \ \boldsymbol{x}_i \in S, \ \lambda_i \ge 0 \,\}.$$

The convex hull of set S is the smallest convex set containing S, and S is convex if and only if $S = \operatorname{conv} S$. Furthermore, the convex hull of a compact set is compact.

The *power set* of a given set $S \subset \mathbb{R}^n$ is denoted by $\mathcal{P}(S)$ and it is the set of all subsets of S.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *convex* if

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \le \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}) \tag{2.1}$$

whenever $\boldsymbol{x}$ and $\boldsymbol{y}$ are in $\mathbb{R}^n$ and $\lambda \in [0,1]$. If a strict inequality holds in (2.1) for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{x} \ne \boldsymbol{y}$ and $\lambda \in (0,1)$, the function f is said to be *strictly convex*.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *Lipschitz continuous* if

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \le L \|\boldsymbol{x} - \boldsymbol{y}\|$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, where $L > 0$ is a constant independent of $\boldsymbol{x}$ and $\boldsymbol{y}$. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *locally Lipschitz continuous* at $\boldsymbol{x} \in \mathbb{R}^n$ with a constant $L > 0$ if there exists a positive number ε such that

$$|f(\boldsymbol{y}) - f(\boldsymbol{z})| \le L \|\boldsymbol{y} - \boldsymbol{z}\|$$

for all $\boldsymbol{y}, \boldsymbol{z} \in B(\boldsymbol{x}; \varepsilon)$. A convex function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous at $\boldsymbol{x}$ for any $\boldsymbol{x} \in \mathbb{R}^n$.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *positively homogeneous* if

$$f(\lambda \boldsymbol{x}) = \lambda f(\boldsymbol{x})$$

for all $\lambda \geq 0$ and *subadditive* if

$$f(\boldsymbol{x}+\boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$$

for all $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^n$. A function is said to be *sublinear* if it is both positively homogeneous and subadditive. A sublinear function is always convex.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *upper semicontinuous* at $\boldsymbol{x} \in \mathbb{R}^n$ if for every sequence $(\boldsymbol{x}_k)$ converging to $\boldsymbol{x}$ the following holds

$$\limsup_{k\to\infty} f(\boldsymbol{x}_k) \leq f(\boldsymbol{x})$$

and *lower semicontinuous* if

$$f(\boldsymbol{x}) \leq \liminf_{k\to\infty} f(\boldsymbol{x}_k).$$

A both upper and lower semicontinuous function is *continuous*.

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *differentiable* at $\boldsymbol{x} \in \mathbb{R}^n$ if there exists a vector $\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$ and a function $\varepsilon : \mathbb{R}^n \to \mathbb{R}$ such that for all $\boldsymbol{d} \in \mathbb{R}^n$

$$f(\boldsymbol{x}+\boldsymbol{d}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T\boldsymbol{d} + \|\boldsymbol{d}\|\varepsilon(\boldsymbol{d})$$

and $\varepsilon(\boldsymbol{d}) \to 0$ whenever $\|\boldsymbol{d}\| \to 0$. The vector $\nabla f(\boldsymbol{x})$ is called the *gradient vector* of the function f at $\boldsymbol{x}$ and it has the following formula

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right)^T,$$

where the components $\partial f(\boldsymbol{x})/\partial x_i$ for $i = 1, \ldots, n$, are called *partial derivatives* of the function f. If the function is differentiable and all the partial derivatives are continuous, then the function is said to be *continuously differentiable* or *smooth* ($f \in C^1(\mathbb{R}^n)$).

The limit

$$f'(\boldsymbol{x};\boldsymbol{d}) = \lim_{t\downarrow 0} \frac{f(\boldsymbol{x}+t\boldsymbol{d}) - f(\boldsymbol{x})}{t}$$

(if it exists) is called the *directional derivative* of f at $\boldsymbol{x} \in \mathbb{R}^n$ in the direction $\boldsymbol{d} \in \mathbb{R}^n$. As a function of $\boldsymbol{d}$, the directional derivative $f'(\boldsymbol{x};\boldsymbol{d})$ is positively homogeneous and subadditive, in other words, it is sublinear. If a function f is differentiable at $\boldsymbol{x}$, then the directional derivative exists in every direction $\boldsymbol{d} \in \mathbb{R}^n$ and

$$f'(\boldsymbol{x};\boldsymbol{d}) = \nabla f(\boldsymbol{x})^T\boldsymbol{d}.$$

If, in addition, f is convex, then for all $\boldsymbol{y} \in \mathbb{R}^n$

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x}).$$

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *twice differentiable* at $\boldsymbol{x} \in \mathbb{R}^n$ if there exists a vector $\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$, a symmetric matrix $\nabla^2 f(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$, and a function $\varepsilon : \mathbb{R}^n \to \mathbb{R}$ such that for all $\boldsymbol{d} \in \mathbb{R}^n$

$$f(\boldsymbol{x} + \boldsymbol{d}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \boldsymbol{d} + \frac{1}{2} \boldsymbol{d}^T \nabla^2 f(\boldsymbol{x}) \boldsymbol{d} + \|\boldsymbol{d}\|^2 \varepsilon(\boldsymbol{d}),$$

where $\varepsilon(\boldsymbol{d}) \to 0$ whenever $\|\boldsymbol{d}\| \to 0$. The matrix $\nabla^2 f(\boldsymbol{x})$ is called the *Hessian matrix* of the function f at $\boldsymbol{x}$ and it is defined to consist of *second partial derivatives* of f, that is,

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n^2} \end{bmatrix}.$$

If the function is twice differentiable and all the second partial derivatives are continuous, then the function is said to be *twice continuously differentiable* ($f \in C^2(\mathbb{R}^n)$).

A matrix, for which horizontal and vertical dimensions are the same (that is, an $n \times n$ matrix), is called a *square matrix of order n*.

A square matrix $A \in \mathbb{R}^{n \times n}$ is called *symmetric* if $A = A^T$, that is, $(A)_{ij} = (A)_{ji}$ for all $i, j \in \{1, \ldots, n\}$ and $(A)_{ij}$ is the element of matrix A in row i of column j. The matrix $A^T \in \mathbb{R}^{n \times n}$ is called the *transpose* of A.

A square matrix $A \in \mathbb{R}^{n \times n}$ is called *positive definite* if

$$\boldsymbol{x}^T A \boldsymbol{x} > 0$$

for all nonzero $\boldsymbol{x} \in \mathbb{R}^n$ and *negative definite* if

$$\boldsymbol{x}^T A \boldsymbol{x} < 0$$

for all nonzero $\boldsymbol{x} \in \mathbb{R}^n$. Correspondingly, a square matrix $A \in \mathbb{R}^{n \times n}$ is called *positive semidefinite* if

$$\boldsymbol{x}^T A \boldsymbol{x} \geq 0$$

for all $\boldsymbol{x} \in \mathbb{R}^n$ and *negative semidefinite* if

$$\boldsymbol{x}^T A \boldsymbol{x} \leq 0$$

for all $\boldsymbol{x} \in \mathbb{R}^n$. A matrix which is neither positive or negative semidefinite is called *indefinite*.

If the matrix $A \in \mathbb{R}^{n \times n}$ is positive definite, then all the submatrices of the matrix A obtained by deleting the corresponding rows and columns of the matrix are positive definite and all the elements on the leading diagonal of the matrix are positive (that is, $(A)_{ii} > 0$ for all $i \in \{1, \ldots, n\}$). If the square matrices A and B are positive definite, then so is $A + B$.

An *inverse* of matrix $A \in \mathbb{R}^{n\times n}$ is a matrix $A^{-1} \in \mathbb{R}^{n\times n}$ such that

$$AA^{-1} = A^{-1}A = I,$$

where I is the *identity matrix*. A square matrix that has an inverse is called *invertible* or *nonsingular*. Otherwise, it is called *singular*. A positive definite matrix is always nonsingular and its inverse is positive definite.

A scalar λ is called an *eigenvalue* of the matrix $A \in \mathbb{R}^{n\times n}$ if

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

for some nonzero vector $\boldsymbol{x} \in \mathbb{R}^n$. The vector $\boldsymbol{x}$ is called an *eigenvector* associated to the eigenvalue λ. The eigenvalues of a symmetric matrix are real and a symmetric matrix is positive definite if and only if all its eigenvalues are positive. A matrix is said to be *bounded* if its eigenvalues lie in the compact interval that does not contain zero.

The *trace* of matrix $A \in \mathbb{R}^{n\times n}$ is denoted by $\mathrm{tr}(A)$ and it is the sum of the diagonal elements of the matrix, that is,

$$\mathrm{tr}(A) = \sum_{i=1}^{n} (A)_{ii}.$$

The trace of a matrix equals to the sum of its eigenvalues. For square matrices A and B, we have $\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B)$.

From now on, we use some special notations for special matrices: the Hessian matrix of the objective function is denoted by H, the approximation of the Hessian matrix is denoted by B, and the approximation of the inverse of the Hessian matrix is denoted by D.

2.2 Nonsmooth Analysis

The theory of nonsmooth analysis is based on convex analysis. Thus, we start this section by giving some definitions and results for convex (not necessarily differentiable) functions. We define the *subgradient* and the *subdifferential* of a convex function as they are defined in [70]. Then we generalize these results to nonconvex locally Lipschitz continuous functions. The aim of this section is not to give any detailed descriptions of nonsmooth analysis (for that we refer e.g., to [11, 63, 70]) but rather to collect some basic definitions and results needed in the following chapters of this thesis.

Definition 2.2.1. *The* subdifferential of a convex function $f : \mathbb{R}^n \to \mathbb{R}$ *at* $\boldsymbol{x} \in \mathbb{R}^n$ *is the set* $\partial_c f(\boldsymbol{x})$ *of vectors* $\boldsymbol{\xi} \in \mathbb{R}^n$ *such that*

$$\partial_c f(\boldsymbol{x}) = \left\{ \boldsymbol{\xi} \in \mathbb{R}^n \mid f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \boldsymbol{\xi}^T(\boldsymbol{y} - \boldsymbol{x}) \text{ for all } \boldsymbol{y} \in \mathbb{R}^n \right\}.$$

Each vector $\boldsymbol{\xi} \in \partial_c f(\boldsymbol{x})$ *is called a* subgradient *of* f *at* $\boldsymbol{x}$.

The subdifferential $\partial_c f(\boldsymbol{x})$ is a nonempty, convex, and compact set such that $\partial_c f(\boldsymbol{x}) \subset B(0; L)$, where $L > 0$ is the Lipschitz constant of f at $\boldsymbol{x}$.

THEOREM 2.2.2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then the directional derivative $f'(\boldsymbol{x}; \boldsymbol{d})$ exists in every direction $\boldsymbol{d} \in \mathbb{R}^n$ and it satisfies*

$$f'(\boldsymbol{x}; \boldsymbol{d}) = \inf_{t>0} \frac{f(\boldsymbol{x} + t\boldsymbol{d}) - f(\boldsymbol{x})}{t}.$$

The next theorem shows the relationship between the subdifferential and the directional derivative. It turns out that knowing $f'(\boldsymbol{x}; \boldsymbol{d})$ is equivalent to knowing $\partial_c f(\boldsymbol{x})$.

THEOREM 2.2.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then for all $\boldsymbol{x} \in \mathbb{R}^n$*

(i) $f'(\boldsymbol{x}; \boldsymbol{d}) = \max\{\, \boldsymbol{\xi}^T \boldsymbol{d} \mid \boldsymbol{\xi} \in \partial_c f(\boldsymbol{x}) \,\}$ *for all* $\boldsymbol{d} \in \mathbb{R}^n$, *and*

(ii) $\partial_c f(\boldsymbol{x}) = \{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid f'(\boldsymbol{x}, \boldsymbol{d}) \geq \boldsymbol{\xi}^T \boldsymbol{d}$ *for all* $\boldsymbol{d} \in \mathbb{R}^n \,\}$.

Since for locally Lipschitz continuous functions there does not necessarily exist any classical directional derivatives, we first define a *generalized directional derivative* [11]. Then we generalize the subdifferential for nonconvex locally Lipschitz continuous functions.

DEFINITION 2.2.4. *(Clarke). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. The* generalized directional derivative *of f at $\boldsymbol{x}$ in the direction $\boldsymbol{d} \in \mathbb{R}^n$ is defined by*

$$f^\circ(\boldsymbol{x}; \boldsymbol{d}) = \limsup_{\substack{\boldsymbol{y} \to \boldsymbol{x} \\ t \downarrow 0}} \frac{f(\boldsymbol{y} + t\boldsymbol{d}) - f(\boldsymbol{y})}{t}.$$

Note that this generalized directional derivate always exists for locally Lipschitz continuous functions and, as a function of $\boldsymbol{d}$, it is sublinear. Therefore, we can now define the subdifferential for nonconvex locally Lipschitz continuous functions analogous to Theorem 2.2.3 (ii) with the directional derivate replaced by the generalized directional derivative.

DEFINITION 2.2.5. *(Clarke). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at a point $\boldsymbol{x} \in \mathbb{R}^n$. Then the* subdifferential *of f at $\boldsymbol{x}$ is the set $\partial f(\boldsymbol{x})$ of vectors $\boldsymbol{\xi} \in \mathbb{R}^n$ such that*

$$\partial f(\boldsymbol{x}) = \{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid f^\circ(\boldsymbol{x}; \boldsymbol{d}) \geq \boldsymbol{\xi}^T \boldsymbol{d}$$ *for all* $\boldsymbol{d} \in \mathbb{R}^n \,\}$.

Each vector $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ is called a subgradient *of f at $\boldsymbol{x}$.*

The subdifferential has the following basic properties.

THEOREM 2.2.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$ with a Lipschitz constant L. Then*

(i) $f^\circ(\boldsymbol{x}; \boldsymbol{d}) = \max\{\, \boldsymbol{\xi}^T \boldsymbol{d} \mid \boldsymbol{\xi} \in \partial f(\boldsymbol{x}) \,\}$ *for all* $\boldsymbol{d} \in \mathbb{R}^n$,

(ii) *$\partial f(\boldsymbol{x})$ is a nonempty, convex, and compact set such that $\partial f(\boldsymbol{x}) \subset B(0; L)$, and*

(iii) *the mapping $\partial f : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ is upper semicontinuous.*

The subdifferential for locally Lipschitz continuous functions is a generalization of the subdifferential for convex functions: If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then $f'(\boldsymbol{x}; \boldsymbol{d}) = f^\circ(\boldsymbol{x}; \boldsymbol{d})$ for all $\boldsymbol{d} \in \mathbb{R}^n$, and $\partial_c f(\boldsymbol{x}) = \partial f(\boldsymbol{x})$. Furthermore, the subdifferential for locally Lipschitz continuous functions is a generalization of the classical derivative: If $f : \mathbb{R}^n \to \mathbb{R}$ is both locally Lipschitz continuous and differentiable at $\boldsymbol{x} \in \mathbb{R}^n$, then $\nabla f(\boldsymbol{x}) \in \partial f(\boldsymbol{x})$. If, in addition, $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable at $\boldsymbol{x} \in \mathbb{R}^n$, then $\partial f(\boldsymbol{x}) = \{\nabla f(\boldsymbol{x})\}$.

THEOREM 2.2.7. *(Rademacher). Let $S \subset \mathbb{R}^n$ be an open set. A function $f : S \to \mathbb{R}$ that is locally Lipschitz continuous on S is differentiable almost everywhere on S.*

By Rademacher's Theorem we know that a locally Lipschitz continuous function is differentiable almost everywhere and, thus, the gradient exists almost everywhere. Now, the subdifferential can be reconstructed as a convex hull of all possible limits of gradients at points $(\boldsymbol{x}_i)$ converging to $\boldsymbol{x}$. The set of points in which a given function f fails to be differentiable is denoted by Ω_f.

THEOREM 2.2.8. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at $\boldsymbol{x} \in \mathbb{R}^n$. Then*

$$\partial f(\boldsymbol{x}) = \operatorname{conv}\{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid \text{there exists } (\boldsymbol{x}_i) \subset \mathbb{R}^n \setminus \Omega_f \text{ such that } \boldsymbol{x}_i \to \boldsymbol{x} \text{ and } \nabla f(\boldsymbol{x}_i) \to \boldsymbol{\xi} \,\}.$$

In nonsmooth optimization, bundle methods are based on the theory of an *ε-subdifferential*, which is a modification of the ordinary subdifferential. Thus, we now define the *Goldstein ε-subdifferential* for nonconvex functions.

DEFINITION 2.2.9. *Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at $\boldsymbol{x} \in \mathbb{R}^n$ and let $\varepsilon \geq 0$. Then the* Goldstein ε-subdifferential *of f is the set*

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

Each element $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\boldsymbol{x})$ is called an ε-subgradient *of the function f at $\boldsymbol{x}$.*

The following theorem summarizes some basic properties of the Goldstein ε-subdifferential.

THEOREM 2.2.10. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$ with a Lipschitz constant L. Then*

(i) *$\partial_0^G f(\boldsymbol{x}) = \partial f(\boldsymbol{x})$,*

(ii) *if $\varepsilon_1 \leq \varepsilon_2$, then $\partial_{\varepsilon_1}^G f(\boldsymbol{x}) \subset \partial_{\varepsilon_2}^G f(\boldsymbol{x})$, and*

(iii) *$\partial_\varepsilon^G f(\boldsymbol{x})$ is a nonempty, convex, and compact set such that $\|\boldsymbol{\xi}\| \leq L$ for all $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\boldsymbol{x})$.*

As a corollary to Theorem 2.2.8, we obtain the following result.

COROLLARY 2.2.11. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. Then*

$$\partial_\varepsilon^G f(\boldsymbol{x}) = \operatorname{conv}\{\, \boldsymbol{\xi} \in \mathbb{R}^n \mid \text{there exists } (\boldsymbol{y}_i) \subset \mathbb{R}^n \setminus \Omega_f \text{ such that } \boldsymbol{y}_i \to \boldsymbol{y},\ \nabla f(\boldsymbol{y}_i) \to \boldsymbol{\xi},\ \text{and } \boldsymbol{y} \in B(\boldsymbol{x};\varepsilon)\,\}.$$

We conclude this section by observing that the Goldstein ε-subdifferential contains in a condensed form the subgradient information in the whole neighborhood of $\boldsymbol{x}$.

THEOREM 2.2.12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. If $\varepsilon \geq 0$, then*

$$\partial f(\boldsymbol{y}) \subset \partial_\varepsilon^G f(\boldsymbol{x})$$

for all $\boldsymbol{y} \in B(\boldsymbol{x};\varepsilon)$.

2.3 Nonsmooth Optimization Theory

In this section, we present some results connecting the theories of nonsmooth analysis and optimization. We first define global and local minima of functions. After that, we generalize the classical first order optimality conditions for unconstrained nonsmooth optimization. Furthermore, we define linearizations for locally Lipschitz continuous functions by using subgradient information, and present their basic properties. These linearizations are suitable for function approximation and they will be used in nonsmooth optimization methods in Chapter 4. At the end of this section, we define the notion of a descent direction and show how to find it for a locally Lipschitz continuous function.

As already mentioned in the introduction, we consider a nonsmooth unconstrained optimization problem of the form

$$\begin{cases} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathbb{R}^n, \end{cases} \tag{2.2}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous at $\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^n$.

DEFINITION 2.3.1. *A point $\boldsymbol{x} \in \mathbb{R}^n$ is a* global minimum *of f if it satisfies*

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}) \quad \text{for all } \boldsymbol{y} \in \mathbb{R}^n.$$

DEFINITION 2.3.2. *A point $\boldsymbol{x} \in \mathbb{R}^n$ is a* local minimum *of f if there exists $\varepsilon > 0$ such that*

$$f(\boldsymbol{x}) \leq f(\boldsymbol{y}) \quad \text{for all } \boldsymbol{y} \in B(\boldsymbol{x};\varepsilon).$$

The necessary conditions for a locally Lipschitz continuous function to attain its local minimum in an unconstrained case are given in the next theorem. For convex functions these conditions are also sufficient and the minimum is global.

THEOREM 2.3.3. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. If f attains its local minimal value at $\boldsymbol{x}$, then*

(i) $\mathbf{0} \in \partial f(\boldsymbol{x})$ *and*

(ii) $f^\circ(\boldsymbol{x}; \boldsymbol{d}) \geq 0$ *for all* $\boldsymbol{d} \in \mathbb{R}^n$.

THEOREM 2.3.4. *If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then the following conditions are equivalent:*

(i) *Function f attains its global minimal value at $\boldsymbol{x}$,*

(ii) $\mathbf{0} \in \partial_c f(\boldsymbol{x})$, *and*

(iii) $f'(\boldsymbol{x}; \boldsymbol{d}) \geq 0$ *for all* $\boldsymbol{d} \in \mathbb{R}^n$.

DEFINITION 2.3.5. *A point $\boldsymbol{x} \in \mathbb{R}^n$ satisfying $\mathbf{0} \in \partial f(\boldsymbol{x})$ is called a* stationary point *of f.*

The optimality condition can be presented also with the aid of the Goldstein ε-subdifferential. The result follows directly from Theorems 2.2.10 and 2.3.3.

COROLLARY 2.3.6. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. If f attains its local minimal value at $\boldsymbol{x}$, then*

$$\mathbf{0} \in \partial_\varepsilon^G f(\boldsymbol{x}).$$

Next we define some notions of *linearization* for locally Lipschitz continuous functions. With these linearizations we are able to construct a piecewise linear local approximation to the unconstrained optimization problem (2.2).

DEFINITION 2.3.7. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$ and let $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ be an arbitrary subgradient. Then the* ξ-linearization *of f at $\boldsymbol{x}$ is the function $\bar{f}_\xi : \mathbb{R}^n \to \mathbb{R}$ defined by*

$$\bar{f}_\xi(\boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{\xi}^T(\boldsymbol{y} - \boldsymbol{x})$$

for all $\boldsymbol{y} \in \mathbb{R}^n$, and the linearization *of f at $\boldsymbol{x}$ is the function $\hat{f}_{\boldsymbol{x}} : \mathbb{R}^n \to \mathbb{R}$ such that*

$$\hat{f}_{\boldsymbol{x}}(\boldsymbol{y}) = \max\{\, \bar{f}_\xi(\boldsymbol{y}) \mid \boldsymbol{\xi} \in \partial f(\boldsymbol{x}) \,\} \tag{2.3}$$

for all $\boldsymbol{y} \in \mathbb{R}^n$.

In the next two theorem, we collect some basic properties of the linearization $\hat{f}_{\boldsymbol{x}}$.

THEOREM 2.3.8. *Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at $\boldsymbol{x} \in \mathbb{R}^n$. Then the linearization $\hat{f}_{\boldsymbol{x}}$ is convex and*

(i) $\hat{f}_{\boldsymbol{x}}(\boldsymbol{x}) = f(\boldsymbol{x})$,

(ii) $\hat{f}_{\boldsymbol{x}}(\boldsymbol{y}) = f(\boldsymbol{x}) + f^\circ(\boldsymbol{x}; \boldsymbol{y} - \boldsymbol{x})$ *for all* $\boldsymbol{y} \in \mathbb{R}^n$, *and*

(iii) $\partial_c \hat{f}_{\boldsymbol{x}}(\boldsymbol{x}) = \partial f(\boldsymbol{x})$.

THEOREM 2.3.9. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Then*

(i) $f(\boldsymbol{y}) = \max\{\, \hat{f}_{\boldsymbol{x}}(\boldsymbol{y}) \mid \boldsymbol{x} \in \mathbb{R}^n \,\}$ *for all* $\boldsymbol{y} \in \mathbb{R}^n$, *and*

(ii) $\hat{f}_{\boldsymbol{x}}(\boldsymbol{y}) \leq f(\boldsymbol{y})$ *for all* $\boldsymbol{y} \in \mathbb{R}^n$.

An essential part of iterative optimization methods is finding a direction such that the objective function values decrease when moving in that direction. Next we define *a descent direction* for an objective function and show how to find it for a locally Lipschitz continuous function.

DEFINITION 2.3.10. *The direction $\boldsymbol{d} \in \mathbb{R}^n$ is said to be a* descent direction *for $f : \mathbb{R}^n \to \mathbb{R}$ at $\boldsymbol{x} \in \mathbb{R}^n$, if there exists $\varepsilon > 0$ such that for all $t \in (0, \varepsilon]$*

$$f(\boldsymbol{x} + t\boldsymbol{d}) < f(\boldsymbol{x}).$$

THEOREM 2.3.11. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz continuous function at $\boldsymbol{x} \in \mathbb{R}^n$. The direction $\boldsymbol{d} \in \mathbb{R}^n$ is a descent direction for f if any of the following holds:*

(i) $f^\circ(\boldsymbol{x}; \boldsymbol{d}) < 0$,

(ii) $\boldsymbol{\xi}^T \boldsymbol{d} < 0$ *for all* $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$,

(iii) $\boldsymbol{\xi}^T \boldsymbol{d} < 0$ *for all* $\boldsymbol{\xi} \in \partial_\varepsilon^G f(\boldsymbol{x})$, *or*

(iv) $\boldsymbol{d}$ *is a descent direction for the linearization* $\hat{f}_{\boldsymbol{x}}$ *at* $\boldsymbol{x}$.

The following theorem tells how to find a descent direction for the linearization function. By Theorem 2.3.11 this direction is a descent direction also for the original function. This fact is utilized in bundle methods to be described in Chapter 4.

THEOREM 2.3.12. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz continuous at $\boldsymbol{x} \in \mathbb{R}^n$ and let $\boldsymbol{\xi}^* \in \partial f(\boldsymbol{x})$ exist such that $\boldsymbol{\xi}^* = \arg\min\{\, \|\boldsymbol{\xi}\| \mid \boldsymbol{\xi} \in \partial f(\boldsymbol{x}) \,\}$. Let us consider the problem*

$$\begin{cases} \textit{minimize} & \hat{f}_{\boldsymbol{x}}(\boldsymbol{x} + \boldsymbol{d}) + \frac{1}{2}\|\boldsymbol{d}\|^2 \\ \textit{subject to} & \boldsymbol{d} \in \mathbb{R}^n. \end{cases} \tag{2.4}$$

Then

(i) *problem (2.4) has a unique solution* $\boldsymbol{d}^* \in \mathbb{R}^n$ *such that* $\boldsymbol{d}^* = -\boldsymbol{\xi}^*$,

(ii) $f^\circ(\boldsymbol{x}; \boldsymbol{d}^*) = -\|\boldsymbol{d}^*\|^2$,

(iii) $\hat{f}_{\boldsymbol{x}}(\boldsymbol{x} + \lambda \boldsymbol{d}^*) = \hat{f}_{\boldsymbol{x}}(\boldsymbol{x}) - \lambda \|\boldsymbol{\xi}^*\|^2$ *for all* $\lambda \in [0, 1]$,

(iv) $0 \notin \partial f(\boldsymbol{x})$ *if and only if* $\boldsymbol{d}^* \neq 0$, *and*

(v) $0 \in \partial f(\boldsymbol{x})$ *if and only if* $\hat{f}_{\boldsymbol{x}}$ *attains its global minimum at* $\boldsymbol{x}$.

Finally, we say a few words about convergence. In iterative optimization methods, we try to generate a sequence $(\boldsymbol{x}_k)$ that converges to a minimum point $\boldsymbol{x}^*$ of the objective function, that is, $(\boldsymbol{x}_k) \to \boldsymbol{x}^*$ whenever $k \to \infty$. If an iterative method converges to a (local) minimum $\boldsymbol{x}^*$ from any arbitrary starting point $\boldsymbol{x}_1$, it is said to be *globally convergent*. If it converges to a (local) minimum in some neighborhood of $\boldsymbol{x}^*$, it is said to be *locally convergent*. Note that the methods described in this thesis are local methods, that is, they do not attempt to find the global minimum of the objective function.

3 VARIABLE METRIC METHODS

In this chapter, we consider variable metric (or quasi-Newton) methods for unconstrained smooth optimization. These methods are well known and widely used for solving smooth small- and medium-scale optimization problems. Moreover, their modifications based either on partitioned, sparse, or limited memory updates are very efficient in smooth large-scale settings. In this thesis, we concentrate on limited memory variable metric methods since the basic knowledge of these methods is needed in Chapter 5, where we construct a new method for nonsmooth large-scale optimization.

We start this chapter by giving a short review of standard variable metric methods, and then, in Section 3.2, we give the basic ideas of the limited memory BFGS method as they are given in [65]. At the end of this chapter (that is, in Section 3.3), we present compact representations for matrices generated by the limited memory variable metric updates [8]. We also show how to compute them efficiently.

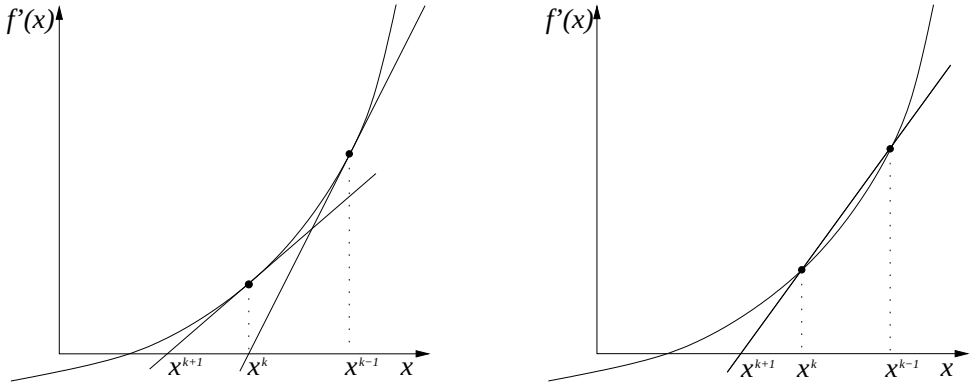
Throughout this chapter, we assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function whose gradient $\nabla f(\boldsymbol{x})$ is available for all $\boldsymbol{x} \in \mathbb{R}^n$.

3.1 Standard Variable Metric Methods

Variable metric methods are iterative methods based on the Newton's method (see, e.g., [16]). In these methods, the Hessian matrix of the objective function is approximated by symmetric, positive definite matrices B_k (where $k \in \mathbb{N}$ is the iteration number). At each iteration, the current approximation B_k is updated to a new approximation B_{k+1} satisfying the so-called *secant* or *quasi-Newton equation*

$$B_{k+1}\boldsymbol{s}_k = \boldsymbol{u}_k, \tag{3.1}$$

where $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{u}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$, and $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k+1}$ are the current and the next iteration points, respectively. The idea of the secant equation (3.1) is that the approximation of the Hessian matrix B_k acts as closely as possible like



(a) Newton's Method (b) Quasi-Newton Method

FIGURE 1: Secant equation.

the true Hessian matrix of the objective function. In a one dimensional case, this means that the tangent of the objective function f at the point $\boldsymbol{x}_k$ is approximated by a secant crossing the graph of f at two points $\boldsymbol{x}_k$ and $\boldsymbol{x}_{k-1}$ (see Figure 1).

Once we have the matrix B_k and the gradient $\nabla f(\boldsymbol{x}_k)$ available, the search direction $\boldsymbol{d}_k$ can be determined by solving a linear equation

$$B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k).$$

Alternatively, instead of matrices B_k we can approximate matrices $D_k \approx \nabla^2 f(\boldsymbol{x}_k)^{-1}$. In this case, the search direction $\boldsymbol{d}_k$ can be computed directly by

$$\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k),$$

where D_k is a symmetric positive definite approximation of the inverse of the Hessian matrix.

The next iteration point is defined by the formula

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_k \boldsymbol{d}_k$$

with $t_k \approx \arg\min_{t>0} f(\boldsymbol{x}_k + t\boldsymbol{d}_k)$. The positive step size t_k is chosen by some line search procedure such that it satisfies the *Wolfe conditions*

$$f(\boldsymbol{x}_k + t_k \boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + \varepsilon_L t_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) \tag{3.2}$$

and

$$\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k + t_k \boldsymbol{d}_k) \geq \varepsilon_R \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k), \tag{3.3}$$

where $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$ are some fixed line search parameters. Conditions (3.2) and (3.3) guarantee that the step size t_k, which provides a significant

reduction in the value of f, always exists and that it can be determined in a finite number of operations (see, e.g., [16]).

The approximation D_k of the inverse of the Hessian matrix is updated recursively by the formula

$$D_{k+1} = D_k + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{u}_k^T \boldsymbol{s}_k} - \frac{D_k \boldsymbol{u}_k \boldsymbol{u}_k^T D_k}{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k} + \frac{\eta_k}{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k} \left(\frac{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k}{\boldsymbol{u}_k^T \boldsymbol{s}_k} \boldsymbol{s}_k - D_k \boldsymbol{u}_k \right) \left(\frac{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k}{\boldsymbol{u}_k^T \boldsymbol{s}_k} \boldsymbol{s}_k - D_k \boldsymbol{u}_k \right)^T, \quad (3.4)$$

where η_k is a free parameter. Formula (3.4) defines a one-parameter class, the so-called *Broyden class*, of variable metric updates (see, e.g., [49]).

There exist three classical values for parameter η_k that are in common use. By setting $\eta_k = 0$, we get the *Davidon-Fletcher-Powel* (DFP) update

$$D_{k+1} = D_k + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{u}_k^T \boldsymbol{s}_k} - \frac{D_k \boldsymbol{u}_k \boldsymbol{u}_k^T D_k}{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k}. \quad (3.5)$$

The original formula was first suggested as a part of a method by Davidon [13] and later it was presented in the form (3.5) by Fletcher and Powell [17].

By setting $\eta_k = 1$, we get the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) update

$$D_{k+1} = D_k + \left(1 + \frac{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k}{\boldsymbol{u}_k^T \boldsymbol{s}_k} \right) \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{u}_k^T \boldsymbol{s}_k} - \frac{D_k \boldsymbol{u}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{u}_k^T D_k}{\boldsymbol{u}_k^T \boldsymbol{s}_k}. \quad (3.6)$$

The original formula was developed by Broyden [5], Fletcher [15], Goldfarb [22], and Shanno [74], and since then, the BFGS method has been generally considered to be the most effective of many variable metric methods (see, e.g., [16, 66]).

Finally, by setting $\eta_k = 1/(1 - \boldsymbol{u}_k^T D_k \boldsymbol{u}_k / \boldsymbol{u}_k^T \boldsymbol{s}_k)$, we get the *symmetric rank-one* (SR1) update

$$D_{k+1} = D_k - \frac{(D_k \boldsymbol{u}_k - \boldsymbol{s}_k)(D_k \boldsymbol{u}_k - \boldsymbol{s}_k)^T}{\boldsymbol{u}_k^T (D_k \boldsymbol{u}_k - \boldsymbol{s}_k)}. \quad (3.7)$$

The original formula was first discovered by Davidon in his seminal paper of quasi-Newton methods [13] and it was re-discovered by several authors in the late 1960s.

The interval given by $0 \leq \eta_k \leq 1$ defines the so-called *restricted Broyden class*, whose updates can be written as convex combinations of the DFP and the BFGS updates. In [9], the global convergence for the restricted Broyden class excluding the DFP method has been proved for convex objective functions with line search satisfying the Wolfe conditions (3.2) and (3.3). Note that, the exact line search (that is, $t_k = \arg\min_{t>0} f(\boldsymbol{x}_k + t\boldsymbol{d}_k)$) is required for DFP method to converge globally. Moreover, in [45], the global convergence of the BFGS method with cautious updating and line search satisfying the Wolfe conditions has been proved for nonconvex objective functions assuming that the function to be minimized has Lipschitz continuous gradients.

The update formulae (3.5) and (3.6) have the advantage of the hereditary positive definiteness. That is, if the matrix D_k is positive definite and the *curvature condition* $\boldsymbol{u}_k^T \boldsymbol{s}_k > 0$ is valid, then also D_{k+1} is positive definite. In fact, the hereditary positive definiteness is common to all members of the Broyden class with $\eta_k \geq 0$ (see, e.g., [16]). On the other hand, the SR1 update (3.7) does not, in general, maintain the hereditary positive definiteness of updates and, further, the denominator in (3.7) may become zero and the update formula is no longer defined. Thus, in practical algorithms, this formula requires some safeguards to avoid these difficulties. However, when compared to the other update formulae, the SR1 update is simpler and may require less computation per iteration.

The details of updating matrices B_k and more information on variable metric methods for smooth small- and medium-scale optimization can be found, for example, in [16, 49, 66].

3.2 Limited Memory BFGS Method

Many practical optimization problems involve functions of hundreds or thousands of variables. In these cases, the optimization algorithms that have been developed for small-scale problems may become inefficient. For example, standard variable metric methods are unsuitable for large-scale problems since they utilize dense approximations of the Hessian matrices and, thus, when the dimension of the problem increases, both the storage space required and the number of operations needed expand rapidly. Anyhow, for variable metric methods, there exist three basic approaches to deal with large-scale problems. The first idea consists of exploiting the structure of partially separable functions. This approach was initiated in [23]. The second approach is to preserve the sparsity pattern of the Hessian matrix by special updates. This approach was proposed in [77]. The third possibility is that of limited memory updating, in which only a few vectors are used and stored to represent the variable metric approximation of the Hessian matrix. This approach was first introduced in [65].

In this thesis, we consider the limited memory variable metric methods. In many cases, the limited memory approach is more useful than the other two possibilities, since it does not require any knowledge of the sparsity structure of the Hessian matrix (i.e., sparse variable metric updates) and it ignores the structure of the problem (needed in partitioned variable metric methods) (see, e.g., [67]).

The basic idea of limited memory methods is that the variable metric update of the approximated Hessian matrix is not constructed explicitly. Instead, the updates use information of the last few iterations to implicitly define the variable metric approximation. In practice, this means that the approximation of the Hessian matrix is not as accurate as that of standard variable metric methods but both the storage space required and the number of operations needed are significantly smaller. Various limited memory methods have been proposed in the literature; some of them combine conjugate gradient and quasi-Newton steps (see, e.g., [6]), and others are very closely related to standard variable metric methods (see, e.g., [46, 65]).

The most commonly used limited memory method is the limited memory BFGS method (L-BFGS) (see, e.g., [65]). This method is very similar to the standard BFGS method. The only difference is the matrix update. Instead of storing the matrix D_k approximating the inverse of the Hessian matrix at the iteration k, we store a certain number of so-called *correction pairs* $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, $(i < k)$, from previous iterations. Here, as before, $\boldsymbol{s}_i = \boldsymbol{x}_{i+1} - \boldsymbol{x}_i$ and $\boldsymbol{u}_i = \nabla f(\boldsymbol{x}_{i+1}) - \nabla f(\boldsymbol{x}_i)$. When the storage space available is used up, the oldest correction pair is deleted and a new one is inserted.

The stored correction pairs are used to define the matrix D_k implicitly using the following form of the BFGS update (see, e.g., [16])

$$D_{k+1} = V_k^T D_k V_k + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{u}_k^T \boldsymbol{s}_k}, \tag{3.8}$$

where

$$V_k = I - \frac{\boldsymbol{u}_k \boldsymbol{s}_k^T}{\boldsymbol{u}_k^T \boldsymbol{s}_k}.$$

Let us denote by m_c the *maximum number of stored correction pairs*. This number is supplied by the user and usually we have $3 \leq m_c \leq 20$ (see, e.g., [21, 46]). In addition, let us denote by $m_k = \min\{k-1, m_c\}$ the *current number of stored correction pairs*. Then, suppose that we have stored m_k pairs $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, where $i = k - m_k, \ldots, k-1$. Suppose also that we have defined the *basic matrix* $D_k^{(0)}$, which is usually a diagonal matrix of the form $D_k^{(0)} = \vartheta_k I$ with the *scaling parameter* $\vartheta_k > 0$ (see, e.g., [67]). Now, the basic matrix $D_k^{(0)}$ is updated m_k times by using the BFGS formula (3.8) with the vectors $\boldsymbol{s}_i$ and $\boldsymbol{u}_i$. Thus, the approximation D_k can be written as

$$D_k = \left(\prod_{i=k-m_k}^{k-1} V_i \right)^T D_k^{(0)} \left(\prod_{i=k-m_k}^{k-1} V_i \right) + \sum_{l=k-m_k}^{k-1} \left(\prod_{i=l+1}^{k-1} V_i \right)^T \frac{\boldsymbol{s}_l \boldsymbol{s}_l^T}{\boldsymbol{u}_l^T \boldsymbol{s}_l} \left(\prod_{i=l+1}^{k-1} V_i \right). \tag{3.9}$$

If the basic matrix $D_k^{(0)}$ is positive definite, then the matrix D_k defined by (3.9) is positive definite provided that we have $\boldsymbol{u}_i^T \boldsymbol{s}_i > 0$ for all i (see, e.g., [65]).

We shall now present an algorithm for the limited memory BFGS method [46]. It utilizes a direction finding algorithm to be described immediately after it.

Algorithm 3.1. *(L-BFGS Method).*

Data: Choose the final accuracy tolerance $\varepsilon > 0$, the positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$, and the maximum number of stored correction pairs $m_c > 1$.

Step 0: (*Initialization.*) Choose a starting point $\boldsymbol{x}_1 \in \mathbb{R}^n$ and a symmetric, positive definite matrix $D_1^{(0)}$ (e.g., $D_1^{(0)} = I$). Compute $f_1 = f(\boldsymbol{x}_1)$ and $\nabla f_1 = \nabla f(\boldsymbol{x}_1)$. Set the iteration counter $k = 1$.

Step 1: (*Stopping criterion.*) If $\|\nabla f_k\| \leq \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution.

Step 2: (*Direction finding.*) Compute

$$\boldsymbol{d}_k = -D_k \nabla f_k$$

by Algorithm 3.2.

Step 3: (*Line search.*) Determine the step size $t_k > 0$ satisfying the Wolfe conditions (3.2) and (3.3) (start with the step size $t_k = 1$). Set the corresponding values

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + t_k \boldsymbol{d}_k, \\ f_{k+1} &= f(\boldsymbol{x}_{k+1}), \quad \text{and} \\ \nabla f_{k+1} &= \nabla f(\boldsymbol{x}_{k+1}). \end{aligned}$$

Step 4: (*Update.*) If $k > m_c$, delete the oldest correction pair $(\boldsymbol{s}_{k-m_c}, \boldsymbol{u}_{k-m_c})$. Compute and store $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ and $\boldsymbol{u}_k = \nabla f_{k+1} - \nabla f_k$. Calculate the basic matrix $D_{k+1}^{(0)} = \vartheta_{k+1} I$ by using, for example,

$$\vartheta_{k+1} = \frac{\boldsymbol{u}_k^T \boldsymbol{s}_k}{\boldsymbol{u}_k^T \boldsymbol{u}_k}.$$

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

Note that in Step 2 the matrix D_k is not formed explicitly but the search direction $\boldsymbol{d}_k = -D_k \nabla f_k$ is calculated recursively. We now give an efficient algorithm by Nocedal [65] for finding the search direction using the limited memory BFGS update.

ALGORITHM 3.2. *(L-BFGS Direction Finding I).*

Data: Let the current iteration be k and the number of stored correction pairs $m_k = \min\{\, k-1, m_c \,\}$. Suppose that we have the m_k correction pairs $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, $(i = k - m_k, \ldots, k-1)$, an initial positive definite diagonal matrix $D_k^{(0)}$, and the current gradient ∇f_k available.

Step 0: Set $\boldsymbol{y} = \nabla f_k$.

Step 1: (*Backward recurrence.*) For $i = k - 1$ to $k - m_k$ set

$$\begin{aligned} \sigma_i &= \frac{\boldsymbol{s}_i^T \boldsymbol{y}}{\boldsymbol{u}_i^T \boldsymbol{s}_i} \qquad (\text{store } \sigma_i) \text{ and} \\ \boldsymbol{y} &= \boldsymbol{y} - \sigma_i \boldsymbol{u}_i. \end{aligned}$$

Step 2: Set $\boldsymbol{r} = D_k^{(0)}\boldsymbol{y}$.

Step 3: (*Forward recurrence.*) For $i = k - m_k$ to $k - 1$ set

$$\nu = \frac{\boldsymbol{u}_i^T \boldsymbol{r}}{\boldsymbol{u}_i^T \boldsymbol{s}_i} \qquad \text{and}$$
$$\boldsymbol{r} = \boldsymbol{r} + (\sigma_i - \nu)\boldsymbol{s}_i.$$

Step 4: (*Search direction.*) Set $\boldsymbol{d}_k = -\boldsymbol{r}$.

Note that, if $m_k = 0$, Steps 1 and 3 are not executed and, thus, the search direction at the first iteration is $\boldsymbol{d}_1 = -D_1^{(0)}\nabla f_1$.

This representation of the BFGS update requires only $O(nm_c)$ storage space (see, e.g., [8]). Assuming we have $m_c \ll n$ this is much less than the $O(n^2)$ storage space required for the standard BFGS implementation. The search direction can be computed implicitly using at most $O(nm_c)$ operations, which is much less than the $O(n^2)$ operations that are needed if the whole matrix D_k is stored. Therefore, the limited memory BFGS method is suitable for solving large-scale optimization problems, since it has been observed in practice that even small values of m_c (that is, $m_c \in [3, 7]$) give satisfactory results (see, e.g., [21, 46]). Furthermore, the limited memory BFGS method described above has been proved to be globally convergent for convex objective functions [46].

3.3 Compact Representation of Limited Memory Matrices

In this section, we consider the limited memory updating process using compact representations of matrices. The recursive formula given in the previous section as Algorithm 3.2 is very efficient for unconstrained optimization. However, if we wish to use update formulae other than BFGS (as we do in Chapter 5) or if we need to solve problems with constraints (which we probably want to do some other time), there are many advantages of using compact representations of limited memory matrices (see, e.g., [8]).

In this section, we show that both the limited memory BFGS and the limited memory SR1 updates can be presented in a compact matrix form. We also describe some procedures for the calculation of the search direction when compact representations of limited memory matrices are used.

As in the previous section, we keep (at most) m_c most recent correction pairs $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, $i < k$, in order to implicitly define the approximation of the inverse of the Hessian matrix at each iteration. At every iteration this set of pairs is updated by deleting the oldest correction pair and adding the new one. We assume that the maximum number of stored correction pairs m_c is constant, even though it is possible to adapt all the formulae of this section to the case where m_c varies at every iteration (see, e.g., [37]).

So far, we have dealt only with correction vectors $\boldsymbol{s}_i$ and $\boldsymbol{u}_i$ and avoided storing any matrices. Now we define $n \times m_k$ dimensional *correction matrices* S_k and U_k by

$$\begin{aligned} S_k &= \begin{bmatrix} \boldsymbol{s}_{k-m_k} & \dots & \boldsymbol{s}_{k-1} \end{bmatrix} \qquad \text{and} \\ U_k &= \begin{bmatrix} \boldsymbol{u}_{k-m_k} & \dots & \boldsymbol{u}_{k-1} \end{bmatrix}, \end{aligned} \tag{3.10}$$

where, as before, $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{u}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and $m_k = \min\{\, k-1, m_c \,\}$ is the current number of stored correction pairs. When the new iteration point $\boldsymbol{x}_{k+1}$ is generated, the new correction matrices S_{k+1} and U_{k+1} are obtained by deleting the oldest vectors $\boldsymbol{s}_{k-m_k}$ and $\boldsymbol{u}_{k-m_k}$ from S_k and U_k if $m_{k+1} = m_k$ (that is, $k > m_c$) and by adding the most recent vectors $\boldsymbol{s}_k$ and $\boldsymbol{u}_k$ to the matrices.

We assume that the basic matrix $D_k^{(0)}$ is given in the form $D_k^{(0)} = \vartheta_k I$ for some $\vartheta_k > 0$, as is commonly done in practice (see, e.g., [21, 46]). If we assume that $\boldsymbol{u}_i^T \boldsymbol{s}_i > 0$ for all $i = 1, \dots, k-1$, then the limited memory BFGS update can be expressed as

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T (C_k + \vartheta_k U_k^T U_k) R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}, \tag{3.11}$$

where R_k is an upper triangular matrix of order m_k given in the form

$$(R_k)_{ij} = \begin{cases} (\boldsymbol{s}_{k-m_k-1+i})^T (\boldsymbol{u}_{k-m_k-1+j}), & \text{if } i \le j \\ 0, & \text{otherwise} \end{cases} \tag{3.12}$$

and where C_k is a diagonal matrix of order m_k such that

$$C_k = \operatorname{diag}\left[\boldsymbol{s}_{k-m_k}^T \boldsymbol{u}_{k-m_k}, \dots, \boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1} \right]. \tag{3.13}$$

The condition $\boldsymbol{u}_i^T \boldsymbol{s}_i > 0$ for $i = 1, \dots, k-1$, ensures that R_k is nonsingular and, thus, (3.11) is well defined [8]. This is consistent with the well-known result that the BFGS update formula preserves positive definiteness if $\boldsymbol{u}_i^T \boldsymbol{s}_i > 0$ for all i (see, e.g., [16]).

Next we describe some procedures for the calculation of the search direction $\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k)$ when the compact representation of the limited memory BFGS matrix is used. In addition to the two $n \times m_k$ matrices S_k and U_k, the $m_k \times m_k$ matrices R_k, $U_k^T U_k$, and C_k are stored. Since in practice m_k is clearly smaller than n, the storage space required by these three auxiliary matrices is insignificant but the savings in computational costs are considerable when compared to the standard BFGS implementation.

At the kth iteration, we have to update the limited memory representation of D_{k-1} to get D_k and calculate the search direction $\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k)$. To get D_k, we delete the first column from S_{k-1} and U_{k-1} if $m_k = m_{k-1}$ and add a new column to the right of each of these matrices to get S_k and U_k, respectively. Then we make the corresponding updates to R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1} to get R_k, $U_k^T U_k$, and C_k, respectively. These updates can be done in $O(m_k^2)$ operations by storing a small amount of additional information, namely the m_k-vectors $S_{k-1}^T \nabla f(\boldsymbol{x}_{k-1})$ and $U_{k-1}^T \nabla f(\boldsymbol{x}_{k-1})$ from the previous iteration.

The new triangular matrix R_k (see (3.12)) is formed of R_{k-1} by deleting the first row and the first column if $m_k = m_{k-1}$ and by adding a new column to the right and a new row to the bottom of the matrix. The new column is given by

$$S_k^T \boldsymbol{u}_{k-1} = S_k^T (\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1}))$$

and the new row has the value zero in its first $m_k - 1$ components. The product $S_k^T \boldsymbol{u}_{k-1}$ can be computed efficiently since we already know $m_k - 1$ components of $S_k^T \nabla f(\boldsymbol{x}_{k-1})$ from $S_{k-1}^T \nabla f(\boldsymbol{x}_{k-1})$. Thus, we only need to calculate $\boldsymbol{s}_{k-1}^T \nabla f(\boldsymbol{x}_{k-1})$ and do the subtractions. The product $\boldsymbol{s}_{k-1}^T \nabla f(\boldsymbol{x}_{k-1})$ can be calculated in $O(m_k^2)$ operations by using the formulae given in [8]. The matrix $U_k^T U_k$ can be updated in a similar way. In this case, both the new column and the new row are given by $U_k^T \boldsymbol{u}_{k-1}$. Finally, the matrix C_k (see (3.13)) is updated by deleting the first element of C_{k-1} if $m_k = m_{k-1}$ and by adding $\boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1}$ as the last element (note that C_k is a diagonal matrix and, thus, stored as a vector).

Now we give an efficient algorithm by Byrd et al. [8] for updating the matrix D_k by the limited memory BFGS formula and for computing the search direction $\boldsymbol{d}_k = -D_k \nabla f(\boldsymbol{x}_k)$.

ALGORITHM 3.3. *(L-BFGS Direction Finding II).*

Data: Suppose that the number of current correction pairs is m_{k-1} and the maximum number of stored correction pairs is m_c. Suppose that we have the most recent vectors $\boldsymbol{s}_{k-1}$ and $\boldsymbol{u}_{k-1}$ (from the previous iteration), the current gradient $\nabla f_k = \nabla f(\boldsymbol{x}_k)$, the $n \times m_{k-1}$ matrices S_{k-1} and U_{k-1}, the $m_{k-1} \times m_{k-1}$ matrices R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}, and the m_{k-1}-vectors $S_{k-1}^T \nabla f_{k-1}$ and $U_{k-1}^T \nabla f_{k-1}$ from the previous iteration available. In addition, suppose that the initial matrix $D_k^{(0)} = \vartheta_k I$ is available.

Step 0: (*Initialization.*) Set $m_k = \min\{\, m_{k-1} + 1, m_c \,\}$.

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

Step 2: Compute and store m_k-vectors $S_k^T \nabla f_k$ and $U_k^T \nabla f_k$.

Step 3: Compute m_k-vectors $S_k^T \boldsymbol{u}_{k-1}$ and $U_k^T \boldsymbol{u}_{k-1}$ by using the fact

$$\boldsymbol{u}_{k-1} = \nabla f_k - \nabla f_{k-1}.$$

Step 4: Update the $m_k \times m_k$ matrices R_k, $U_k^T U_k$, and C_k.

Step 5: Compute ϑ_k, for example,

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

Note that both $\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1}$ and $\boldsymbol{u}_{k-1}^T \boldsymbol{u}_{k-1}$ have already been calculated.

Step 6: (*Intermediate values.*) Compute the vectors

$$\begin{aligned}\boldsymbol{p}_1 &= R_k^{-1} S_k^T \nabla f_k \quad \text{and} \\ \boldsymbol{p}_2 &= (R_k^{-1})^T (C_k \boldsymbol{p}_1 + \vartheta_k U_k^T U_k \boldsymbol{p}_1 - \vartheta_k U_k^T \nabla f_k).\end{aligned}$$

Step 7: (*Search direction.*) Compute

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

Note that in the first iteration (that is, $k = 1$) the search direction is not calculated by the Algorithm 3.3 but it is directly defined as $\boldsymbol{d}_1 = -\nabla f(\boldsymbol{x}_1)$.

The algorithm given above requires the same amount of work per iteration as the recursion with two loops in Algorithm 3.2; thus, the two algorithms are equally efficient for unconstrained problems. However, in constrained optimization it is very common to have problems where the gradients of the constraints are sparse. In these cases, compact representations of matrices are more useful than the recursive formula, since Algorithm 3.2 does not take advantage of the sparsity of the vectors involved. For further studies of constrained optimization with limited memory variable metric methods we refer to [7, 8, 38].

So far, we have only given a representation to the limited memory BFGS update. However, the limited memory DFP and SR1 updates can be expressed in the compact matrix form as well (see, e.g., [8, 37, 49]). For example, the limited memory SR1 update can be written as

$$D_k = D_k^{(0)} - (D_k^{(0)} U_k - S_k)(U_k^T D_k^{(0)} U_k - R_k - R_k^T + C_k)^{-1} (D_k^{(0)} U_k - S_k)^T, \quad (3.14)$$

where S_k, U_k, R_k, and C_k are defined as in (3.10), (3.12), and (3.13).

Since the SR1 update does not in general preserve the positive definiteness of the generated matrices, there is no reason to enforce the curvature condition $\boldsymbol{u}_i^T \boldsymbol{s}_i > 0$ for all i, as with the BFGS update. Thus, we only assume that the update is well defined, that is, $\boldsymbol{u}_i^T (D_i \boldsymbol{u}_i - \boldsymbol{s}_i) \neq 0$ for all $i = 1, \ldots, k-1$. Here, we do not give any algorithms for computing products involving limited memory SR1 matrices, because the ideas are very similar to those described above with the limited memory BFGS method.

Note that, if $D_k^{(0)}$ is kept fixed for all k in (3.14), we achieve some savings in storage and computational costs when compared to the limited memory BFGS update (3.11). If, on the other hand, $D_k^{(0)}$ is a scalar multiple of the identity matrix (that is, $D_k^{(0)} = \vartheta_k I$) and the scaling parameter ϑ_k is changed at each iteration, then the storage space required and the updating costs of the limited memory SR1 and BFGS methods are equivalent [8].

4 BUNDLE METHODS

In this chapter, we get back to the situation where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is not supposed to be differentiable. The standard variable metric methods presented in the previous chapter can be used to solve nonsmooth optimization problems by replacing the gradient with an arbitrary subgradient (see, e.g., [42, 54, 57]). However, the global convergence of these methods has not been proved when applied to nonsmooth problems, and some failures and inaccurate results can occur in practical computations.

At the moment, various versions of bundle methods (see, e.g., [29, 33, 60, 63, 73]) are regarded as the most effective and reliable methods for nonsmooth optimization. They are based on the assumptions that the objective function is locally Lipchitz continuous and 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 $\partial f(\boldsymbol{x})$ (see Theorem 2.2.8).

The basic idea of bundle methods is to approximate the subdifferential of the objective function by gathering subgradients from previous iterations into a bundle. This subgradient information serves for the construction of a piecewise linear local approximation to the objective function. A descent direction for this approximation and, thus, also for the objective function (see Theorem 2.3.11) can then be determined by solving a quadratic direction finding problem (see, e.g., [63]). The global convergence of bundle methods with a limited number of stored subgradients can be guaranteed by using a subgradient aggregation strategy [33], which accumulates information from previous iterations.

We begin this chapter by giving a survey of standard bundle methods. Then, in Section 4.2, we present the variable metric bundle method developed by Lukšan and Vlček [54, 78]. The idea of the variable metric bundle method is to use some properties of bundle methods to improve the robustness and efficiency of variable metric methods. Furthermore, the quite complicated quadratic direction finding problem appearing in standard bundle methods can be avoided. In Chapter 5, we use the ideas of the variable metric bundle method to construct a new method for large-scale nonsmooth optimization.

4.1 Standard Bundle Methods

In this section, we describe a general bundle method that produces a sequence of so-called *basic points* $(\boldsymbol{x}_k) \in \mathbb{R}^n$ that, in the convex case, converges to a global minimum of an objective function $f : \mathbb{R}^n \to \mathbb{R}$ (if it exists). In the nonconvex case, since the optimality condition of Theorem 2.3.3 is not sufficient without some convexity assumptions, the method is only guaranteed to find a stationary point of the objective function.

We start this section by describing how to find a search direction $\boldsymbol{d}_k \in \mathbb{R}^n$ at iteration k for a locally Lipschitz continuous objective function. We assume that, in addition to the current iteration point $\boldsymbol{x}_k$, we have some *auxiliary points* $\boldsymbol{y}_j \in \mathbb{R}^n$ from previous iterations and subgradients $\boldsymbol{\xi}_j \in \partial f(\boldsymbol{y}_j)$ for $j \in \mathcal{J}_k$, where the index set $\mathcal{J}_k$ is a nonempty subset of $\{1, \ldots, k\}$.

In (2.3) we have defined the linearization of the objective function f at $\boldsymbol{x}$. However, for this representation we need to know the whole subdifferential $\partial f(\boldsymbol{x})$, which is normally unknown. Therefore, we have to approximate it somehow. By using the auxiliary points $\boldsymbol{y}_j \in \mathbb{R}^n$ and the subgradients $\boldsymbol{\xi}_j \in \partial f(\boldsymbol{y}_j)$ for $j \in \mathcal{J}_k$, we can define a convex piecewise linear approximation of the original objective function f by

$$\hat{f}_k(\boldsymbol{x}) = \max\{\, f(\boldsymbol{y}_j) + \boldsymbol{\xi}_j^T(\boldsymbol{x} - \boldsymbol{y}_j) \mid j \in \mathcal{J}_k \,\}. \tag{4.1}$$

This approximation can be written in an equivalent form

$$\hat{f}_k(\boldsymbol{x}) = \max\{\, f(\boldsymbol{x}_k) + \boldsymbol{\xi}_j^T(\boldsymbol{x} - \boldsymbol{x}_k) - \alpha_j^k \mid j \in \mathcal{J}_k \,\},$$

where

$$\alpha_j^k = f(\boldsymbol{x}_k) - f(\boldsymbol{y}_j) + \boldsymbol{\xi}_j^T(\boldsymbol{y}_j - \boldsymbol{x}_k) \qquad \text{for all } j \in \mathcal{J}_k \tag{4.2}$$

is a so-called *linearization error*. If the function f is convex, then the linearization error is nonnegative, that is, $\alpha_j^k \geq 0$ for all $j \in \mathcal{J}_k$ and $f(\boldsymbol{x}) \geq \hat{f}_k(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$ (see, e.g., [63]).

For nonconvex functions the linearization error (4.2) can be negative. Therefore, we use so-called *subgradient locality measures* β_j^k to generalize the linearization errors for nonconvex functions. The subgradient locality measure is defined as

$$\beta_j^k = \max_{j \in \mathcal{J}_k}\{\, |\alpha_j^k|,\, \gamma(s_j^k)^\omega \,\} \geq 0, \tag{4.3}$$

where $\gamma \geq 0$ is a *distance measure parameter* and $\omega \geq 1$ is a *locality measure parameter* supplied by the user, and s_j^k is a *distance measure* such that

$$s_j^k = \begin{cases} \|\boldsymbol{x}_j - \boldsymbol{y}_j\| + \sum_{i=j}^{k-1} \|\boldsymbol{x}_{i+1} - \boldsymbol{x}_i\| & \text{for } j = 1, \ldots, k-1, \\ \|\boldsymbol{x}_k - \boldsymbol{y}_j\| & \text{for } j = k. \end{cases}$$

For convex objective functions the distance measure parameter γ can be set to zero and, thus, the subgradient locality measure and the linearization error coincide [63].

The linearization $\hat{f}_k$ defined in (4.1) is not suitable as such for determining a new approximation to the objective function in order to find its minimum: the minimum of $\hat{f}_k$ may not exist, since the function $\hat{f}_k$ is piecewise linear and, if it exists, it can be too far from the minimum of the objective function. For this reason, a stabilizing term $1/2\boldsymbol{d}^T G_k \boldsymbol{d}$, which keeps the approximation local enough, is added to function $\hat{f}_k$. The symmetric, positive definite $n \times n$ matrix G_k is intended to accumulate information about the curvature of the objective function f in a ball around the iteration point $\boldsymbol{x}_k$ (see, e.g., [60]).

Now, a search direction can be determined as

$$\boldsymbol{d}_k = \arg\min_{\boldsymbol{d}\in\mathbb{R}^n} \{\, \hat{f}_k(\boldsymbol{x}_k + \boldsymbol{d}) + \tfrac{1}{2}\boldsymbol{d}^T G_k \boldsymbol{d} \,\}. \tag{4.4}$$

The minimization in problem (4.4) is equivalent (see, e.g., [60]) to the (smooth) *quadratic direction finding problem*

$$\begin{cases} \text{minimize} & \frac{1}{2}\boldsymbol{d}^T G_k \boldsymbol{d} + v \\ \text{subject to} & -\beta_j^k + \boldsymbol{d}^T \boldsymbol{\xi}_j \le v \quad \text{for all } j \in \mathcal{J}_k, \end{cases} \tag{4.5}$$

where $\boldsymbol{d} \in \mathbb{R}^n$ and $v \in \mathbb{R}$ are the variables.

By duality this is equivalent to finding Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$ that solve the *quadratic dual problem*

$$\begin{cases} \text{minimize} & \frac{1}{2}\left(\sum\limits_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j\right)^T G_k^{-1} \left(\sum\limits_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j\right) + \sum\limits_{j\in\mathcal{J}_k} \lambda_j \beta_j^k \\ \text{subject to} & \sum\limits_{j\in\mathcal{J}_k} \lambda_j = 1 \quad \text{and} \\ & \lambda_j \ge 0 \quad \text{for all } j \in \mathcal{J}_k. \end{cases}$$

The solution $(\boldsymbol{d}_k, v_k)$ of problem (4.5) can be expressed in the form (see, e.g., [60])

$$\begin{aligned} \boldsymbol{d}_k &= -\sum_{j\in\mathcal{J}_k} \lambda_j^k G_k^{-1} \boldsymbol{\xi}_j \qquad \text{and} \\ v_k &= -\boldsymbol{d}_k^T G_k \boldsymbol{d}_k - \sum_{j\in\mathcal{J}_k} \lambda_j^k \beta_j^k. \end{aligned}$$

The direction finding problem (4.5) seems to be suitable for generating a descent direction but, we still have to decide, how to choose the index set $\mathcal{J}_k$. As mentioned at the beginning of this section, the index set $\mathcal{J}_k$ must be a nonempty subset of $\{1, \ldots, k\}$. Since, in practice, the choice $\mathcal{J}_k = \{1, \ldots, k\}$ would cause serious difficulties with storage and computations after a large number of iterations, the size of the index set $\mathcal{J}_k$ have to be bounded. That is, we set $|\mathcal{J}_k| \le m_\xi$, where $|\mathcal{J}_k|$ is the number of the elements in the set $\mathcal{J}_k$. The set $\mathcal{J}_k$ is usually determined such that, if the iteration number $k \le m_\xi$, then the index set is chosen as

$$\mathcal{J}_k = \{1, \ldots, k\}$$

and, if $k > m_\xi$, then the index set is chosen as

$$\mathcal{J}_k = \mathcal{J}_{k-1} \cup \{k\} \setminus \{k - m_\xi\}.$$

If we have $\mathcal{J}_k \neq \{1, \ldots, k\}$, then one possibility to guarantee the global convergence of the bundle method is to use the *subgradient aggregation strategy*, which accumulates information from previous iterations [33]. We define the so-called *aggregate values* f_a^k, $\boldsymbol{\xi}_a^k$, and s_a^k as follows. Let $\boldsymbol{x}_1 \in \mathbb{R}^n$ be a starting point supplied by the user. Then we initialize the algorithm by setting

$$\begin{aligned}
&\boldsymbol{y}_1 = \boldsymbol{x}_1, \\
&f_a^1 = f(\boldsymbol{y}_1), \\
&\boldsymbol{\xi}_a^1 = \boldsymbol{\xi}_1 \in \partial f(\boldsymbol{y}_1), \\
&s_a^1 = s_1^1 = 0, \qquad \text{and} \\
&\mathcal{J}_1 = \{1\}.
\end{aligned}$$

At iteration $k+1$, the new aggregate values f_a^{k+1}, $\boldsymbol{\xi}_a^{k+1}$, and s_a^{k+1} are defined by the formulae

$$\begin{aligned}
f_a^{k+1} &= \tilde{f}_a^k + (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)^T \tilde{\boldsymbol{\xi}}_a^k, \\
\boldsymbol{\xi}_a^{k+1} &= \tilde{\boldsymbol{\xi}}_a^k, \qquad \text{and} \\
s_a^{k+1} &= \tilde{s}_a^k + \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|,
\end{aligned}$$

where the values $\tilde{f}_a^k$, $\tilde{\boldsymbol{\xi}}_a^k$, and $\tilde{s}_a^k$ can be obtained after solving the Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$, and λ_a^k of the direction finding problem to be given below. However, first we have to define the *aggregate locality measure* by

$$\beta_a^k = \max\{\, |f(\boldsymbol{x}_k) - f_a^k|,\, \gamma(s_a^k)^\omega \,\}.$$

Now, by using these aggregate values, the search direction $\boldsymbol{d}_k$ can be determined by minimizing the quadratic direction finding problem (compare with (4.5))

$$\begin{cases}
\text{minimize} & \frac{1}{2}\boldsymbol{d}^T G_k \boldsymbol{d} + v \\
\text{subject to} & -\beta_j^k + \boldsymbol{d}^T \boldsymbol{\xi}_j \leq v \quad \text{for all } j \in \mathcal{J}_k, \text{ and} \\
& -\beta_a^k + \boldsymbol{d}^T \boldsymbol{\xi}_a^k \leq v,
\end{cases} \tag{4.6}$$

which by duality is equivalent to finding the Lagrange multipliers λ_j^k for $j \in \mathcal{J}_k$, and λ_a^k that solve the problem

$$\begin{cases}
\text{minimize} & \frac{1}{2}\left(\sum\limits_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j + \lambda_a \boldsymbol{\xi}_a^k\right)^T G_k^{-1} \left(\sum\limits_{j\in\mathcal{J}_k} \lambda_j \boldsymbol{\xi}_j + \lambda_a \boldsymbol{\xi}_a^k\right) \\
& \quad + \sum\limits_{j\in\mathcal{J}_k} \lambda_j \beta_j^k + \lambda_a \beta_a^k \\
\text{subject to} & \sum\limits_{j\in\mathcal{J}_k} \lambda_j + \lambda_a = 1, \\
& \lambda_j \geq 0 \quad \text{for all } j \in \mathcal{J}_k, \text{ and} \\
& \lambda_a \geq 0.
\end{cases} \tag{4.7}$$

The solution $(\boldsymbol{d}_k, v_k)$ of problem (4.6) can now be expressed in the form (see, e.g., [56])

$$\begin{aligned} \boldsymbol{d}_k &= -G_k^{-1}\tilde{\boldsymbol{\xi}}_a^k \qquad \text{and} \\ v_k &= -\boldsymbol{d}_k^T G_k \boldsymbol{d}_k - \tilde{\beta}_a^k, \end{aligned}$$

where

$$\begin{aligned} \tilde{\boldsymbol{\xi}}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k \boldsymbol{\xi}_j + \lambda_a^k \boldsymbol{\xi}_a^k \qquad \text{and} \\ \tilde{\beta}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k \beta_j^k + \lambda_a^k \beta_a^k. \end{aligned}$$

In addition to the values $\tilde{\boldsymbol{\xi}}_a^k$ and $\tilde{\beta}_a^k$, the Lagrange multipliers $\lambda_j^k \geq 0$ for $j \in \mathcal{J}_k$, and $\lambda_a^k \geq 0$ are used to define the aggregate values

$$\begin{aligned} \tilde{f}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k f_j^k + \lambda_a^k f_a^k \qquad \text{and} \\ \tilde{s}_a^k &= \sum_{j\in\mathcal{J}_k} \lambda_j^k s_j^k + \lambda_a^k s_a^k, \end{aligned}$$

where $f_j^k = f(\boldsymbol{y}_j) + \boldsymbol{\xi}_j^T(\boldsymbol{x}_k - \boldsymbol{y}_j)$ for $j \in \mathcal{J}_k$.

The minimal value of the dual problem (4.7) is given by

$$w_k = \tfrac{1}{2}(\tilde{\boldsymbol{\xi}}_a^k)^T G_k^{-1} \tilde{\boldsymbol{\xi}}_a^k + \tilde{\beta}_a^k = -v_k - \tfrac{1}{2}(\tilde{\boldsymbol{\xi}}_a^k)^T G_k^{-1} \tilde{\boldsymbol{\xi}}_a^k \geq 0. \tag{4.8}$$

The value w_k is used as a stopping parameter and the stopping criterion at iteration k is given in the form

$$\text{If } w_k \leq \varepsilon \text{ for a given } \varepsilon > 0, \text{ then stop.}$$

When the direction vector $\boldsymbol{d}_k$ has been determined, we have to calculate a new iteration point $\boldsymbol{x}_{k+1}$. To guarantee the global convergence of the bundle method it is not possible to simply set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_k$, but it is necessary to use some special line search procedure that generates two points

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

with $\boldsymbol{y}_1 = \boldsymbol{x}_1$, where $t_R^k \in (0, 1]$ and $t_L^k \in [0, t_R^k]$ are step sizes in such a way that exactly one of the two possibilities, a *serious step* or a *null step* occurs. There exist several different line search procedures suitable for bundle methods (see, e.g., [33]). In this work, we describe one of them, since it is suitable also for the variable metric bundle method to be described in the next section.

A necessary condition for a serious step to be taken is to have (compare with (3.2))

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\boldsymbol{y}_{k+1}) \leq f(\boldsymbol{x}_k) - \varepsilon_L t_R^k w_k, \tag{4.9}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k \geq 0$ (see (4.8)) represents the desirable amount of descent of f at $\boldsymbol{x}_k$. If condition (4.9) is satisfied, we set $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ and a serious step is taken.

Otherwise, a null step is taken. In that case, the usage of special line search procedure guarantees (see e.g., [33]) that we have (compare with (3.3))

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1}^{k+1} + \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{4.10}$$

where $\varepsilon_R \in (\varepsilon_L, 1)$ is a fixed line search parameter and $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$. In the case of a null step, we set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ but the piecewise linear approximation $\hat{f}_{k+1}$ of the objective function is improved, since we set $\mathcal{J}_{k+1} = \mathcal{J}_k \cup \{k+1\} \setminus \{k - m_\xi + 1\}$.

In the case of a serious step, there occurs a significant decrease in the value of the objective function. Therefore, there is no need to detect the possible discontinuities in the gradient and we may set $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{x}_{k+1})$. On the other hand, if a null step occurs, then there exists a discontinuity in the gradient of the objective function. In this case, the last requirement in (4.10) ensures that $\boldsymbol{x}_k$ and $\boldsymbol{y}_{k+1}$ lie on the opposite sides of the discontinuity, and the new subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$ will force a significant modification to the next direction finding problem.

In the convex case, the line search procedure can be replaced by a simple step size selection and the resulting step size is either accepted (serious step) or not (null step) (see, e.g., [60]).

The last open question is how to choose the stabilizing matrix G_k. As mentioned before, all the matrices G_k are supposed to be symmetric and positive definite. In addition, to ensure the global convergence of the bundle method, we have to assume that all the matrices G_k are bounded. Moreover, if the kth step is a null step, then we assume that

$$\boldsymbol{h}^T G_{k+1}^{-1} \boldsymbol{h} \leq \boldsymbol{h}^T G_k^{-1} \boldsymbol{h} \tag{4.11}$$

for all $\boldsymbol{h} \in \mathbb{R}^n$. These assumptions are relative strong, but they can be weakened for different versions of bundle methods.

There exist several versions of bundle methods that mainly differ with the choice of the matrix G_k. For example, in the most frequently used *proximal bundle* and *bundle trust region* methods (see, e.g., [35, 63, 73]), the matrix G_k is diagonal and of the form $G_k = u_k I$. The assumptions given above are satisfied if the weights $u_k > 0$ lie in a compact interval that does not contain zero and the condition $u_{k+1} \geq u_k$ is valid in the null step. A detailed overview of various bundle methods is given in [60].

Now we present a general algorithm for bundle methods. In addition to the algorithm given below, the line search algorithm (to be given as Algorithm 4.3), is also needed.

ALGORITHM 4.1. *(Bundle Method).*

Data: Choose the final accuracy tolerance $\varepsilon > 0$, the positive line search parameters $\varepsilon_L \in (0, 1/2)$ and $\varepsilon_R \in (\varepsilon_L, 1)$, the distance measure parameter $\gamma \geq 0$ ($\gamma = 0$ if f is convex), the locality measure parameter $\omega \geq 1$ and the maximum number of stored subgradients $m_\xi \geq 1$.

Step 0: (*Initialization.*) Choose a starting point $\boldsymbol{x}_1 \in \mathbb{R}^n$ and a symmetric, positive definite matrix G_1 (e.g., $G_1 = I$). Set $\boldsymbol{y}_1 = \boldsymbol{x}_1$ and compute $f_1 = f(\boldsymbol{y}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{y}_1)$. Set

$$s_1^1 = s_a^1 = 0, \qquad f_1^1 = f_a^1 = f_1, \qquad \boldsymbol{\xi}_a^1 = \boldsymbol{\xi}_1,$$

and $\mathcal{J}_1 = \{1\}$. Set the iteration counter $k = 1$.

Step 1: (*Direction finding.*) Determine multipliers λ_j^k for $j \in \mathcal{J}_k$, and λ_a^k ($\lambda_a^k \neq 0$ only if $\mathcal{J}_k \neq \{1, \ldots, k\}$), aggregate values $\tilde{\boldsymbol{\xi}}_a^k$, $\tilde{\beta}_a^k$, $\tilde{f}_a^k$, $\tilde{s}_a^k$, the direction vector $\boldsymbol{d}_k$, and the value v_k by solving the quadratic direction finding problem (4.6) (where the last constraint is used only if $\mathcal{J}_k \neq \{1, \ldots, k\}$).

Step 2: (*Stopping criterion.*) Calculate w_k by (4.8). If $w_k \leq \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution.

Step 3: (*Line search.*) Determine the step sizes $t_R^k \in (0, 1]$ and $t_L^k \in [0, t_R^k]$ by the line search Algorithm 4.3. Set the corresponding values

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

Step 4: (*Linearization updating.*) Calculate the new linearization values

$$\begin{aligned}
f_j^{k+1} &= f_j^k + (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)^T \boldsymbol{\xi}_j \quad \text{for } j \in \mathcal{J}_k, \\
f_a^{k+1} &= \tilde{f}_a^k + (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)^T \tilde{\boldsymbol{\xi}}_a^k, \\
f_{k+1}^{k+1} &= f_{k+1} + (\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1})^T \boldsymbol{\xi}_{k+1}, \\
\boldsymbol{\xi}_a^{k+1} &= \tilde{\boldsymbol{\xi}}_a^k, \\
s_j^{k+1} &= s_j^k + \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \quad \text{for } j \in \mathcal{J}_k, \\
s_a^{k+1} &= \tilde{s}_a^k + \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|, \quad \text{and} \\
s_{k+1}^{k+1} &= \|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\|.
\end{aligned}$$

Step 5: (*Matrix updating.*) Determine the stabilizing matrix G_{k+1} satisfying the assumptions discussed above.

Step 6: (*Bundle updating.*) If $|\mathcal{J}_k| < m_\xi$, then set

$$\mathcal{J}_{k+1} = \mathcal{J}_k \cup \{k + 1\}.$$

If $|\mathcal{J}_k| = m_\xi$, then set

$$\mathcal{J}_{k+1} = \mathcal{J}_k \cup \{k + 1\} \setminus \{k + 1 - m_\xi\}.$$

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

Under mild assumptions, it can be proved that the number of consecutive null steps in Algorithm 4.1 is finite and that every accumulation point of the sequence $(\boldsymbol{x}_k)$ is a stationary point of the objective function (see, e.g., [33]).

Note that Algorithm 4.1 requires relatively large bundles (with $m_\xi \sim n$) to be computationally efficient (see, e.g., [33]). Thus, we need to solve the quadratic direction finding problem (4.6) with a relatively large number of constraints, which is a time-consuming task.

For further studies of bundle methods we refer to [29, 33, 56, 63].

4.2 Variable Metric Bundle Method

In this section, we present variable metric bundle method by Lukšan and Vlček [54, 78]. The method presented is a hybrid of the variable metric methods described in Section 3.1 and the bundle methods described in Section 4.1. The basic idea of the variable metric bundle method is to use some properties of bundle methods to improve the robustness and the efficiency of variable metric methods. The differences when comparing the variable metric bundle method to the standard variable metric methods are the usage of null steps together with the aggregation of subgradients and the application of locality measures defined in (4.3). Using null steps gives sufficient information about the nonsmooth objective function in the cases the descent condition (4.9) is not satisfied. On the other hand, a simple aggregation of subgradients and the application of locality measures guarantee the convergence of the aggregate subgradients to zero and make it possible to evaluate a termination criterion.

Similarly to the bundle method, the variable metric bundle method generates a sequence of basic points $(\boldsymbol{x}_k) \in \mathbb{R}^n$ that, in the convex case, converges to the global minimum of an objective function $f : \mathbb{R}^n \to \mathbb{R}$ (if it exists). In the nonconvex case, the method is only guaranteed to find a stationary point of the objective function. In addition to the basic points, the algorithm generates a sequence of auxiliary points $(\boldsymbol{y}_k) \in \mathbb{R}^n$. A new iteration point $\boldsymbol{x}_{k+1}$ and a new auxiliary point $\boldsymbol{y}_{k+1}$ are produced by using a line search procedure such that

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

with $\boldsymbol{y}_1 = \boldsymbol{x}_1$, where $t_R^k \in (0, t_{max}]$, $t_L^k \in [0, t_R^k]$ are step sizes, $t_{max} > 1$ is the upper bound for the step size, $\boldsymbol{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k$ is a search direction vector, $\tilde{\boldsymbol{\xi}}_k$ is an aggregate subgradient, and matrix D_k is is formed by using the usual variable metric updates (see Section 3.1) with gradients replaced with subgradients. The role of matrix D_k is to accumulate information about previous subgradients and it represents an approximation of the inverse of the Hessian matrix if the objective function is smooth.

A necessary condition for a serious step to be taken is similar to that of bundle methods (see (4.9)), that is,

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\boldsymbol{y}_{k+1}) \leq f(\boldsymbol{x}_k) - \varepsilon_L t_R^k w_k, \tag{4.12}$$

where $\varepsilon_L \in (0, 1/2)$ is a fixed line search parameter and $w_k > 0$ represents the desirable amount of descent of f at $\boldsymbol{x}_k$. If condition (4.12) is satisfied, we set $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ and a serious step is taken. A condition for a null step to be taken is also similar to that of bundle methods (see (4.10)). Thus, a null step is taken, if

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R w_k, \tag{4.13}$$

where $\varepsilon_R \in (\varepsilon_L, 1/2)$ is a fixed line search parameter, $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, and β_{k+1} is a locality measure similar to bundle methods (see (4.3)), that is,

$$\beta_{k+1} = \max\{\, |f(\boldsymbol{x}_k) - f(\boldsymbol{y}_{k+1}) + \boldsymbol{\xi}_{k+1}^T (\boldsymbol{y}_{k+1} - \boldsymbol{x}_k)|,\, \gamma \|\boldsymbol{y}_{k+1} - \boldsymbol{x}_k\|^\omega \,\}. \tag{4.14}$$

Here, as before, $\gamma \geq 0$ is a distance measure parameter and $\omega \geq 1$ is a locality measure parameter supplied by the user.

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 auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$.

As with the standard bundle methods, there is no need to use any two-point line search procedure in the convex case. The procedure can be replaced by a simple step size selection and the resulting step size is either accepted (serious step) or not (null step) [54].

The aggregation procedure used with the variable metric bundle method utilizes only three subgradients and two locality measures. We denote by m the lowest index j satisfying $\boldsymbol{x}_j = \boldsymbol{x}_k$ (that is, m is the index of the iteration after the latest serious step). Suppose that we have the current subgradient $\boldsymbol{\xi}_m \in \partial f(\boldsymbol{x}_k)$, the auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, and the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ (note that $\tilde{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1$) available. In addition, suppose that we have the current locality measure β_{k+1} (see (4.14)) and the current aggregate locality measure $\tilde{\beta}_k$ from the previous iteration (note that $\tilde{\beta}_1 = 0$). Now the quite complicated quadratic direction finding problem (4.6) appearing in the standard bundle methods reduces to the minimization of the function

$$\begin{aligned} \varphi(\lambda_1, \lambda_2, \lambda_3) = {} & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k) \\ & + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k), \end{aligned} \tag{4.15}$$

where we have $\lambda_i \geq 0$ for all $i \in \{1, 2, 3\}$, and $\sum_{i=1}^3 \lambda_i = 1$. The optimal values λ_i^k, for $i \in \{1, 2, 3\}$, can be calculated by using a simple formula [78].

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

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

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

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

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

As mentioned before, the matrices D_k are formed by using the usual variable metric updates. After a null step, the symmetric rank-one (SR1) update (3.7) is used, since this update formula gives us a possibility to preserve the boundedness of the matrices generated as required for the proof of global convergence [54, 78]. In addition, we use an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to calculate the search direction

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

Because the boundedness of the generated matrices is not required after a serious step, the more efficient Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (3.6) is used together with the original subgradient $\boldsymbol{\xi}_k$ (note that after a serious step $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$). Thus, after a serious step, the search direction is defined as $\boldsymbol{d}_k = -D_k\boldsymbol{\xi}_k$.

We now present a variable metric bundle algorithm by Lukšan and Vlček [54, 78]. The algorithm is suitable for minimizing both convex and nonconvex but locally Lipchitz continuous objective functions, though, it can still be simplified a little bit for convex functions.

ALGORITHM 4.2. *(Variable Metric Bundle Method).*

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

Step 0: (*Initialization.*) Choose a starting point $\boldsymbol{x}_1 \in \mathbb{R}^n$ and a symmetric, positive definite matrix D_1 (e.g., $D_1 = I$). Set $\boldsymbol{y}_1 = \boldsymbol{x}_1$ and $\beta_1 = 0$. Compute $f_1 = f(\boldsymbol{x}_1)$ and $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{x}_1)$. Set the iteration counter $k = 1$.

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

Step 2: (*Stopping criterion.*) Calculate

$$w_k = \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k.$$

If $w_k \leq \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution.

Step 3: (*Direction finding.*) Compute

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

Step 4: (*Line search.*) Calculate the initial step size $t_I^k \in [t_{min}, t_{max})$. Determine the step sizes $t_R^k \in (0, t_I^k]$ and $t_L^k \in [0, t_R^k]$ by the line search Algorithm 4.3. Set the corresponding values

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

Set $\boldsymbol{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$. If condition (4.12) is valid (i.e., we take a serious step), then set $\beta_{k+1} = 0$ and go to Step 7. Otherwise (i.e., condition (4.13) is valid), calculate the locality measure β_{k+1} by (4.14).

Step 5: (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function (4.15). Set

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

Step 6: (*SR1 update.*) Let $\boldsymbol{v}_k = D_k \boldsymbol{u}_k - t_R^k \boldsymbol{d}_k$. If $\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{v}_k < 0$, then set

$$D_{k+1} = D_k - \frac{\boldsymbol{v}_k \boldsymbol{v}_k^T}{\boldsymbol{u}_k^T \boldsymbol{v}_k}.$$

Otherwise, set $D_{k+1} = D_k$. Set $k = k + 1$ and go to Step 2.

Step 7: (*BFGS update.*) If $\boldsymbol{u}_k^T \boldsymbol{d}_k > 0$, then set

$$D_{k+1} = D_k + \left(t_L^k + \frac{\boldsymbol{u}_k^T D_k \boldsymbol{u}_k}{\boldsymbol{u}_k^T \boldsymbol{d}_k} \right) \frac{\boldsymbol{d}_k \boldsymbol{d}_k^T}{\boldsymbol{u}_k^T \boldsymbol{d}_k} - \frac{D_k \boldsymbol{u}_k \boldsymbol{d}_k^T + \boldsymbol{d}_k \boldsymbol{u}_k^T D_k}{\boldsymbol{u}_k^T \boldsymbol{d}_k}.$$

Otherwise, set $D_{k+1} = D_k$. Set $k = k + 1$ and go to Step 1.

The condition

$$\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{v}_k < 0$$

in Step 6 (or equivalently $\boldsymbol{u}_k^T \boldsymbol{d}_k > t_R^k \boldsymbol{d}_k^T D_k^{-1} \boldsymbol{d}_k$), which implies that $\boldsymbol{u}_k^T \boldsymbol{v}_k > 0$, ensures the positive definiteness of the matrices D_{k+1} obtained by the SR1 update (see, e.g., [16]). Similarly, the condition

$$\boldsymbol{u}_k^T \boldsymbol{d}_k > 0$$

in Step 7 ensures the positive definiteness of the matrices D_{k+1} obtained by the BFGS update (note that $\boldsymbol{u}_k^T \boldsymbol{d}_k > 0$ holds whenever f is convex). Therefore, all the matrices generated by Algorithm 4.2 are positive definite [78].

The initial step size $t_I^k \in [t_{min}, t_{max})$ (see Step 4 in Algorithm 4.2) is calculated by using a bundle containing auxiliary points and corresponding function values

and subgradients [78]. The possibility of using step sizes greater than 1 is useful here, because information about the objective function included in matrix D_k is not sufficient for a proper step size determination in the nonsmooth case [78]. As mentioned before, the aggregation procedure (see Step 5 in Algorithm 4.2) uses only three subgradients and two locality measures to calculate the new aggregate values. Thus, the minimum size of the bundle is only 2 and a larger bundle (if it is employed) is used solely for the selection of the initial step size, which does not require time-consuming operations [54, 78].

Under mild assumptions, it can be proved that in Algorithm 4.2 every accumulation point of the sequence $(\boldsymbol{x}_k)$ is a stationary point of the objective function [54, 78].

Finally, we present a line search algorithm, which is used to determine the step sizes t_L^k and t_R^k in the variable metric bundle method for nonconvex locally Lipschitz continuous functions [78]. The algorithm given below can also be used with standard bundle methods described in Section 4.1 with two small changes: the initial step size t_I^k is fixed to 1, and for the line search parameter we have $\varepsilon_R \in (\varepsilon_L, 1)$.

ALGORITHM 4.3. *(Line Search).*

Data: Suppose that we have the current iteration point $\boldsymbol{x}_k$, the current search direction $\boldsymbol{d}_k$, 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 $\varepsilon_T \in (\varepsilon_L, \varepsilon_R - \varepsilon_A)$. 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 $\gamma \geq 0$ ($\gamma = 0$ if f is convex), the locality measure parameter $\omega \geq 1$, the desirable amount of descent w_k, and an interpolation parameter $\kappa \in (0, 1/2)$.

Step 0: (*Initialization.*) Set $t_A = 0$ and $t = t_U = t_I^k$.

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

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

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

Step 2: (*Serious step.*) If

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

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: (*Null step.*) If

$$-\beta + \boldsymbol{d}_k^T\boldsymbol{\xi} \geq -\varepsilon_R w_k,$$

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

Step 4: (*Interpolation.*) Choose

$$t \in [t_A + \kappa(t_U - t_A), t_U - \kappa(t_U - t_A)]$$

and go to Step 1.

It can be proved under some semi-smoothness assumptions (see, e.g., [78]) that the Algorithm 4.3 terminates in a finite number of iterations, finding step sizes t_L^k and t_R^k satisfying the condition $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \varepsilon_L t_L^k w_k$ and, in the case of $t_L^k = 0$ (null step), also condition (4.13).

5 LIMITED MEMORY BUNDLE METHOD

None of the general nonsmooth methods presented in Chapter 4 is very efficient for large-scale optimization. This assertion is supported, for example, by the numerical tests concerning proximal bundle methods, variable metric bundle methods, and the bundle-Newton method [53] presented in [27, 62]. Standard variable metric methods and variable metric bundle method cannot be used for large-scale problems, since they utilize dense matrices, whereas standard bundle methods are not applicable, because they need to solve a rather expensive quadratic direction finding problem (4.6) at every iteration, which is a time-consuming procedure. We have not found any general bundle-based solver for large-scale nonsmooth optimization problems in the literature and, as mentioned in the introduction, the subgradient methods (see, e.g., [76]) suffer from some serious disadvantages. Thus, there is an evident need of reliable and efficient solvers for nonsmooth large-scale optimization problems.

In addition to the nonsmoothness, the nonconvexity of the objective function brings along many difficulties. First, a nonconvex objective function may have many local minima. In this thesis, we attempt to find only one local minimum of the objective function. However, the optimality condition for locally Lipschitz continuous functions (see Theorem 2.3.3) is not sufficient without some convexity assumptions and, thus, we can not guarantee even a local optimality of the solution but only some candidates called stationary points of the objective function are to be looked for. Second, in convex case, the piecewise linear approximation is an underestimate for the objective function and the nonnegative linearization error (see (4.2)) is used to measure the accuracy of this approximation. In the nonconvex case, these properties are not valid anymore and the linearization error may have tiny or even negative value although the piecewise linear approximation is not a good model for the original problem. Furthermore, adding more cutting planes (more subgradients to the bundle) might make the approximation even worse due to nonconvexity.

The most common way to deal with the difficulties caused by nonconvexity is to use subgradient locality measures (see (4.3)) instead of linearization errors (see,

e.g., [33, 53, 63, 73, 78]). In addition, some other changes, for example, in the line search procedure have to be made in order to guarantee the global convergence of the methods in nonconvex case (see, e.g., [33]). Lately, a different approach was introduced in [18, 19], where the difference of two piecewise linear convex functions is used to construct a model for a nonconvex objective function.

In this chapter, we introduce a new limited memory bundle algorithm for solving large-scale nonsmooth and possible nonconvex unconstrained optimization problems. The method to be described is a hybrid of the variable metric bundle method [54, 78] (see Section 4.2) and the limited memory variable metric methods [8] (see Section 3.3), where the latter have been developed for smooth large-scale optimization. The new method exploits the ideas of the variable metric bundle method, namely the utilization of null steps, simple aggregation of subgradients, and the subgradient locality measures, but the search direction is calculated using a limited memory approach. Therefore, the time-consuming quadratic direction finding problem (4.6) does not need to be solved and the number of stored subgradients does not depend on the dimension of the problem. Furthermore, the method 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.

Besides using the idea of limited memory updating, there exist some other possibilities to deal with the variable metric approximation of the Hessian matrix in smooth large-scale settings as mentioned at the beginning of Section 3.2. However, we chose this limited memory approach to be adopted for nonsmooth problems because it does not need any information about the structure of the problem or its approximated Hessian. Thus, the only assumptions are that the objective function f is locally Lipschitz continuous and that we can evaluate the value of the objective function $f(\boldsymbol{x})$ and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ at every point $\boldsymbol{x} \in \mathbb{R}^n$. Note that the objective function is not supposed to be differentiable or convex.

This chapter is organized as follows. In the following section, we introduce the basic limited memory bundle algorithm together with a special line search procedure and a limited memory matrix updating. Then, in Section 5.2, we prove the global convergence of the method for locally Lipschitz continuous objective functions. In Sections 5.3, 5.4, and 5.5, we give some modifications to the basic limited memory bundle method and, when necessary, show how to prove their global convergence.

5.1 Basic Method

In this section, we introduce a limited memory bundle algorithm that generates a sequence of basic points $(\boldsymbol{x}_k) \in \mathbb{R}^n$ that, in the convex case, converges to a global minimum of an objective function $f : \mathbb{R}^n \to \mathbb{R}$. In the nonconvex case, since the optimality condition of Theorem 2.3.3 is not sufficient, the algorithm is only guaranteed to find a stationary point of the objective function.

We now go through the algorithm step by step and describe both its theoretical

properties and some details of the implementation. In what follows, we use the following notations already employed in earlier chapters: The current iteration point at iteration k is denoted by $\boldsymbol{x}_k$ and the current auxiliary point is denoted by $\boldsymbol{y}_k$. The subgradient of the objective function is denoted by $\boldsymbol{\xi}_k$ and an aggregate subgradient is denoted by $\tilde{\boldsymbol{\xi}}_k$. Moreover, we denote by D_k the limited memory variable metric update that, in smooth case, represents the approximation of the inverse of the Hessian matrix. Note that we recall many formulae and procedures given before to make this section more self-contained.

5.1.1 Direction Finding

We first describe how to find the search direction $\boldsymbol{d}_k$ by using the limited memory bundle method. The basic idea of finding the search direction is the same as with the limited memory variable metric methods (see, e.g., [8]) and the updates D_k are formed implicitly by using the compact representations of the limited memory variable metric matrices (see Section 3.3). However, due to the usage of null steps, some modifications similar to the variable metric bundle method [54, 78] (see Section 4.2) have to be made.

After a null step, the matrix D_k is formed by using the limited memory SR1 update (3.14). This update formula gives us the possibility to preserve the boundedness and some other properties of generated matrices that are required in the proof of global convergence (see Section 5.2). In addition, we use an aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ to calculate the search direction

$$\boldsymbol{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k. \tag{5.1}$$

Because the boundedness of the generated matrices is not required after a serious step, we then employ the more efficient limited memory BFGS update formula (3.11) to compute the matrix D_k and the search direction $\boldsymbol{d}_k$ is calculated by using the original subgradient $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$. Thus, after a serious step, the search direction is defined by

$$\boldsymbol{d}_k = -D_k\boldsymbol{\xi}_k. \tag{5.2}$$

Note that the matrix D_k is not formed explicitly but the search direction $\boldsymbol{d}_k$ is calculated using the limited memory approach to be described in Subsection 5.1.6.

5.1.2 Line Search

Next, we consider how to calculate a new iteration point $\boldsymbol{x}_{k+1}$ when the search direction $\boldsymbol{d}_k$ has been calculated. Similarly to the standard bundle methods and the variable metric bundle method, we use a special line search procedure that generates two points

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + t_L^k\boldsymbol{d}_k \qquad \text{and} \\ \boldsymbol{y}_{k+1} &= \boldsymbol{x}_k + t_R^k\boldsymbol{d}_k, \qquad \text{for } k \geq 1 \end{aligned} \tag{5.3}$$

with $\boldsymbol{y}_1 = \boldsymbol{x}_1$, where $t_R^k \in (0, t_I^k]$, $t_L^k \in [0, t_R^k]$ are step sizes, $t_I^k \in [t_{min}, t_{max})$ is the initial step size, and $t_{min} \in (0,1)$ and $t_{max} > 1$ are the lower and the upper bounds for the initial step size t_I^k, respectively. Similarly to the original variable metric bundle method, we have the possibility to use step sizes greater than 1 here, since information about the objective function included in the matrix D_k, is not sufficient for a proper step size determination in the nonsmooth case [78]. Again, the initial step size t_I^k is selected by using a bundle containing auxiliary points and the corresponding function values and subgradients. The procedure used is exactly the same as in the original variable metric bundle method [78].

As before, a necessary condition for a serious step to be taken is to have

$$t_R^k = t_L^k > 0 \qquad \text{and} \qquad f(\boldsymbol{y}_{k+1}) \leq f(\boldsymbol{x}_k) - \varepsilon_L^k t_R^k w_k, \tag{5.4}$$

where $\varepsilon_L^k \in (0, 1/2)$ is a line search parameter and $w_k > 0$ (see (5.11)) represents the desirable amount of descent of f at $\boldsymbol{x}_k$. If condition (5.4) is satisfied, we set $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ and a serious step is taken.

On the other hand, a null step is taken, if

$$t_R^k > t_L^k = 0 \qquad \text{and} \qquad -\beta_{k+1} + \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R^k w_k, \tag{5.5}$$

where $\varepsilon_R^k \in (\varepsilon_L^k, 1/2)$ is a line search parameter, $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, and β_{k+1} is the subgradient locality measure similar to bundle methods and variable metric bundle method, that is,

$$\beta_{k+1} = \max\{\, |f(\boldsymbol{x}_k) - f(\boldsymbol{y}_{k+1}) + \boldsymbol{\xi}_{k+1}^T (\boldsymbol{y}_{k+1} - \boldsymbol{x}_k)|,\, \gamma \|\boldsymbol{y}_{k+1} - \boldsymbol{x}_k\|^\omega \,\}, \tag{5.6}$$

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

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 $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$.

Next we present a line search algorithm, which is used to determine the step sizes t_L^k and t_R^k in the limited memory bundle method. The line search procedure used is quite similar to that in the original variable metric bundle method (see Algorithm 4.3). However, in order to guarantee the global convergence of the method, we use scaled line search parameters ε_L^k, ε_R^k, ε_A^k, and ε_T^k instead of fixed ones. Furthermore, in order to avoid many consecutive null steps, we have added an additional interpolation step (at Step 3 in Algorithm 5.1). In other words, we look for more suitable step sizes t_L^k and t_R^k by using an extra interpolation loop if necessary. The role of this additional step is that if we have already taken a null step at the previous iteration (that is, $i_{null} = k - m \geq 1$, where m is the index of the iteration after the latest serious step), we rather try to find a step size suitable for a serious step (that is, to make condition (5.4) valid) even if condition (5.5) required for a null step was satisfied. This additional interpolation step has no influence on the convergence properties but it has a significant effect on the efficiency of the method. The choice of the interpolation procedure (see Step 5

in Algorithm 5.1) has no effect on the convergence properties, either. Similarly to the original variable metric bundle method [78] we combine quadratic interpolation with the bisection procedure (see, Step 5 in Algorithm 5.1).

ALGORITHM 5.1. *(Modified Line Search).*

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 positive initial line search parameters $\varepsilon_L^I \in (0, 1/2)$, $\varepsilon_R^I \in (\varepsilon_L^I, 1/2)$, $\varepsilon_A^I \in (0, \varepsilon_R^I - \varepsilon_L^I)$, and $\varepsilon_T^I \in (\varepsilon_L^I, \varepsilon_R^I - \varepsilon_A^I)$. 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 $\gamma \geq 0$ (with $\gamma = 0$ if f is convex), the locality measure parameter $\omega \geq 1$, the desirable amount of descent w_k, and the maximum number of additional interpolations i_{max} available. In addition, suppose that we have the number of consecutive null steps $i_{null} \geq 0$.

Step 0: (*Initialization.*) Set $t_A = 0$, $t = t_U = t_I^k$, and $i_I = 0$. Calculate the scaled line search parameters

$$\varepsilon_L^k = \theta_k \varepsilon_L^I, \;\; \varepsilon_R^k = \theta_k \varepsilon_R^I, \;\; \varepsilon_A^k = \theta_k \varepsilon_A^I, \;\; \text{and} \;\; \varepsilon_T^k = \theta_k \varepsilon_T^I.$$

and the interpolation parameter

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

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

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

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

Step 2: (*Serious step.*) If

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

and either

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

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

$$-\beta + \theta_k \boldsymbol{d}_k^T \boldsymbol{\xi} \geq -\varepsilon_R^k w_k,$$

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.

It can be proved that Algorithm 5.1 terminates in a finite number of iterations (the proof is similar to that given in [78]) if the objective function f satisfies the following semi-smoothness assumptions: For any $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{d} \in \mathbb{R}^n$, and sequences $(\hat{\boldsymbol{\xi}}_i) \subset \mathbb{R}^n$ and $(\hat{t}_i) \subset \mathbb{R}_+$ satisfying $\hat{\boldsymbol{\xi}}_i \in \partial f(\boldsymbol{x} + \hat{t}_i \boldsymbol{d})$ and $\hat{t}_i \downarrow 0$, we have

$$\limsup_{i\to\infty} \hat{\boldsymbol{\xi}}_i^T \boldsymbol{d} \geq \liminf_{i\to\infty} \left(f(\boldsymbol{x} + \hat{t}_i \boldsymbol{d}) - f(\boldsymbol{x})\right)/\hat{t}_i.$$

In addition, on the output of Algorithm 5.1 (see Steps 2 and 4), the step sizes t_L^k and t_R^k satisfy the *serious descent criterion*

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \leq -\varepsilon_L^k t_L^k w_k \tag{5.7}$$

and, in the case of $t_L^k = 0$ (i.e., a null step), also condition (5.5).

5.1.3 Subgradient Aggregation

In principle, the aggregation procedure used with the limited memory bundle method is the same as that with the original variable metric bundle method [54, 78] (see Section 4.2). However, since the matrix D_k is not here formed explicitly, the practical implementation of the aggregation procedure differs from that of the original method.

As before, the aggregation procedure uses three subgradients and two locality measures. We denote by m the lowest index j satisfying $\boldsymbol{x}_j = \boldsymbol{x}_k$ (that is, m is the index of the iteration after the latest serious step). Suppose that we have the current subgradient $\boldsymbol{\xi}_m \in \partial f(\boldsymbol{x}_k)$, the auxiliary subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$, and the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ (note that $\tilde{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1$) available. In addition, suppose that we have the current locality measure β_{k+1} (see (5.6)) and the current aggregate locality measure $\tilde{\beta}_k$ from the previous iteration (note that $\tilde{\beta}_1 = 0$). We minimize the function

$$\begin{aligned}\varphi(\lambda_1, \lambda_2, \lambda_3) = {} & (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k)^T D_k (\lambda_1 \boldsymbol{\xi}_m + \lambda_2 \boldsymbol{\xi}_{k+1} + \lambda_3 \tilde{\boldsymbol{\xi}}_k) \\ & + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k),\end{aligned} \tag{5.8}$$

where $\lambda_i \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i = 1$, and the matrix D_k is formed by the limited memory BFGS update if $k = m$ (that is, the first null step after any serious step) and by the limited memory SR1 update, otherwise. The optimal values λ_i^k for all $i \in \{1, 2, 3\}$ can be calculated by using simple formulae similar to those given in [78].

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

$$\tilde{\boldsymbol{\xi}}_{k+1} = \lambda_1^k \boldsymbol{\xi}_m + \lambda_2^k \boldsymbol{\xi}_{k+1} + \lambda_3^k \tilde{\boldsymbol{\xi}}_k \tag{5.9}$$

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

$$\tilde{\beta}_{k+1} = \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k. \tag{5.10}$$

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

5.1.4 Stopping Criterion

For smooth functions, a necessary condition for a local minimum is that the gradient has to be zero and by continuity it gets small when we are close to an optimal point. This is no longer true when we replace the gradient by an arbitrary subgradient. However, due to the subgradient aggregation, we have quite a useful approximation to the gradient, namely the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$. Unfortunately, as a stopping criterion, the direct test $\|\tilde{\boldsymbol{\xi}}_k\| \leq \varepsilon$, for some positive ε, is too uncertain, if the current piecewise linear approximation of the objective function (see (4.1)) is too rough. Therefore, we use the term $\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k = -\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d}_k$ and the aggregate subgradient locality measure $\tilde{\beta}_k$ to improve the accuracy of the norm of the aggregate subgradient. The aggregate subgradient locality measure $\tilde{\beta}_k$ approximates the accuracy of the current linearization: If the value of this locality measure is large, then the linearization is rough. On the other hand, if the value is near zero, then the linearization is quite accurate and, thus, we can stop the algorithm if the term $\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k$ is small enough. Hence, the stopping parameter w_k at iteration k is defined by

$$w_k = -\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d}_k + 2\tilde{\beta}_k. \tag{5.11}$$

Similarly to bundle methods (see Section 4.1) the parameter w_k is also used during the line search procedure (see (5.4)) to represent the desirable amount of descent of the objective function f at $\boldsymbol{x}_k$.

This first part of our stopping criterion, that is, $w_k \leq \varepsilon$ is similar to that of the original variable metric bundle method. However, in practice the limited memory matrix D_k is not very accurate and computational experiments have shown that some accidental terminations may occur (that is, the optimization may terminate before the minimum point has actually been found). Thus, we add a second stopping parameter q_k, which does not depend on the matrix D_k. The second stopping parameter is similar to that in proximal bundle methods (see, e.g., [63]), that is,

$$q_k = \frac{1}{2}\tilde{\boldsymbol{\xi}}_k^T \tilde{\boldsymbol{\xi}}_k + \tilde{\beta}_k. \tag{5.12}$$

Now, the complete stopping criterion is given by:

If $w_k \leq \varepsilon$ and $q_k \leq \varepsilon$, for a given $\varepsilon > 0$, then stop.

5.1.5 Algorithm

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

ALGORITHM 5.2. *(Limited Memory Bundle Method).*

Data: Choose the final accuracy tolerance $\varepsilon > 0$, the positive initial line search parameters $\varepsilon_L^I \in (0, 1/2)$ and $\varepsilon_R^I \in (\varepsilon_L^I, 1/2)$, the distance measure parameter $\gamma \geq 0$ (with $\gamma = 0$ if f is convex), and the locality measure parameter $\omega \geq 1$. Select the lower and the upper bounds $t_{min} \in (0, 1)$ and $t_{max} > 1$ for serious steps. Select a control parameter $C > 0$ for the length of the direction vector, a correction parameter $\varrho \in (0, 1/2)$, and a restart parameter $\mu \in (0, 1)$.

Step 0: (*Initialization.*) Choose a starting point $\boldsymbol{x}_1 \in \mathbb{R}^n$. 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 correction indicator $i_C = 0$ and the number of current correction pairs $m_1 = 0$. 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: (*Direction finding.*) Compute

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

by a limited memory BFGS update if $m = k$ (using Algorithm 5.3) and by a limited memory SR1 update, otherwise (using Algorithm 5.4). Note that $\boldsymbol{d}_1 = -\tilde{\boldsymbol{\xi}}_1$.

Step 3: (*Correction.*) If $-\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d}_k < \varrho \tilde{\boldsymbol{\xi}}_k^T \tilde{\boldsymbol{\xi}}_k$ or $i_{CN} = 1$, then set

$$\boldsymbol{d}_k = \boldsymbol{d}_k - \varrho \tilde{\boldsymbol{\xi}}_k, \tag{5.13}$$

(i.e., $D_k = D_k + \varrho I$) and $i_C = 1$. Otherwise, set $i_C = 0$. If $i_C = 1$ and $m < k$, then set $i_{CN} = 1$.

Step 4: (*Restart.*) If

$$\tilde{\boldsymbol{\xi}}_k^T \boldsymbol{d}_k \leq -\mu \|\tilde{\boldsymbol{\xi}}_k\| \|\boldsymbol{d}_k\|,$$

then go to Step 5. Otherwise, reset $m_k = 0$, $i_C = 0$, $i_{CN} = 0$, $\beta_k = 0$, and $\tilde{\beta}_k = 0$. If $m \neq k$, then set $\boldsymbol{\xi}_k = \boldsymbol{\xi}_m$, $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_m$, and $m = k$. Set $\boldsymbol{d}_k = -\tilde{\boldsymbol{\xi}}_k$.

Step 5: (*Stopping criterion.*) Calculate w_k and q_k by (5.11) and (5.12), respectively. If $w_k \leq \varepsilon$ and $q_k \leq \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution.

Step 6: (*Line search.*) Set the scaling parameter for the length of the direction vector and for line search

$$\theta_k = \min\{\, 1, C/\|\boldsymbol{d}_k\|\,\}.$$

Calculate the initial step size $t_I^k \in [t_{min}, t_{max})$. Determine the step sizes $t_R^k \in (0, t_I^k]$ and $t_L^k \in [0, t_R^k]$ by the line search Algorithm 5.1. Set the corresponding values

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

Set $\boldsymbol{u}_k = \boldsymbol{\xi}_{k+1} - \boldsymbol{\xi}_m$ and $\boldsymbol{s}_k = \boldsymbol{y}_{k+1} - \boldsymbol{x}_k = t_R^k \theta_k \boldsymbol{d}_k$. If condition (5.4) is valid (i.e., we take a serious step), then set $\beta_{k+1} = 0$, $k = k + 1$, and go to Step 1. Otherwise (i.e., condition (5.5) is valid), calculate the locality measure β_{k+1} by (5.6).

Step 7: (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function (5.8), where D_k is calculated by the same updating formula as in Step 2 and $D_k = D_k + \varrho I$, if $i_C = 1$. Set

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

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

In order to guarantee the global convergence of the method, the boundedness of both the length of the direction vector (see Step 6 in Algorithm 5.2) and the matrices $B_i = D_i^{-1}$ (see Step 3 in Algorithm 5.2) are required. The utilization of correction (5.13) is equivalent to adding a positive definite matrix ϱI to the matrix D_k. Note that in Steps 2 and 7, the matrices D_k are not formed explicitly but the limited memory expressions (5.14) and (5.16) are used instead. Since the limited memory representations of matrix D_k do not contain information of the correction (5.13) that may have been added to the matrix in Step 3, the correction ϱI is added to the matrix D_k in Step 7 if $i_C = 1$.

In the case the direction vector $\boldsymbol{d}_k$ is almost orthogonal to the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ (see Step 4 in Algorithm 5.2), we reset all the values. That is, we restart the algorithm by using the current iteration point $\boldsymbol{x}_k$ as a starting point.

As mentioned before, the initial step size $t_I^k \in [t_{min}, t_{max})$ (see Step 6 in Algorithm 5.2) is calculated exactly in the same way as in the original variable metric bundle method for nonconvex objective functions [78]. Since the aggregation procedure (see Subsection 5.1.3) uses only three subgradients and two locality measures to calculate the new aggregate values, the minimum size of the bundle is two and a larger bundle (if it is employed) is used only for the selection of the initial step size.

5.1.6 Matrix Updating

Finally, we need to consider how to update the matrix D_k and, at the same time, how to find the search direction $\boldsymbol{d}_k$. Note that until this point, the procedures described are rather similar to those of the original variable metric bundle method [78] (see Section 4.2).

We use compact representations of limited memory matrices (see Section 3.3), since in addition to the BFGS updating formula, we need the SR1 updating formula and, for SR1 updates, there exists no recursive updating formula analogous to that given in Algorithm 3.2. Moreover, the compact representations of limited memory matrices facilitates the possibility to generalize the method for constrained optimization.

Let us recall that the basic idea of the limited memory matrix updating is that instead of storing the matrices D_k, we use information of the last few iterations to implicitly define the variable metric update. This is done by storing a certain number of correction pairs $(\boldsymbol{s}_i, \boldsymbol{u}_i)$, $(i < k)$, obtained in Step 6 of Algorithm 5.2. When the storage space available is used up, the oldest correction vectors are deleted to make room for new ones. All the subsequent iterations are of this form: one correction pair is deleted and a new one is inserted.

As in Section 3.2 we denote by m_c the maximum number of stored correction pairs supplied by the user $(3 \leq m_c)$ and by $m_k = \min\{k-1, m_c\}$ the current number of stored correction pairs. We assume that the maximum number of stored correction pairs m_c is constant, although it is possible to adapt all the formulae of this subsection to the case where m_c varies at every iteration (see, e.g., [37]).

Corresponding to Section 3.3, the $n \times m_k$ dimensional correction matrices S_k and U_k are defined by

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

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

As in (3.11), we define the limited memory BFGS update by the formula

$$D_k = \vartheta_k I + \begin{bmatrix} S_k & \vartheta_k U_k \end{bmatrix} \begin{bmatrix} (R_k^{-1})^T (C_k + \vartheta_k U_k^T U_k) R_k^{-1} & -(R_k^{-1})^T \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^T \\ \vartheta_k U_k^T \end{bmatrix}. \quad (5.14)$$

Where, as before, R_k is an upper triangular matrix of order m_k given by the form

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

C_k is a diagonal matrix of order m_k such that

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

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

$$\vartheta_k = \frac{\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1}}{\boldsymbol{u}_{k-1}^T \boldsymbol{u}_{k-1}}. \tag{5.15}$$

In addition, we define the limited memory SR1 update similarly to (3.14). In other words,

$$D_k = \vartheta_k I - (\vartheta_k U_k - S_k)(\vartheta_k U_k^T U_k - R_k - R_k^T + C_k)^{-1}(\vartheta_k U_k - S_k)^T, \tag{5.16}$$

where instead of (5.15), which might make the update formulae undefined, we use the value $\vartheta_k = 1$ for every k.

The limited memory BFGS and SR1 matrices are updated corresponding to what was discussed in Section 3.3. Thus, in addition to the two $n \times m_k$ matrices S_k and U_k, we store the $m_k \times m_k$ matrices R_k, $U_k^T U_k$, and C_k. As mentioned earlier, in practice m_k is clearly smaller than n and, therefore, the storage space required by these three auxiliary matrices is insignificant but the savings in computational costs are considerable.

At the kth iteration, we have to update the limited memory representation of D_{k-1} to get D_k. Thus, we delete a column from S_{k-1} and U_{k-1} if $m_k = m_{k-1}$ and add a new column to each of these matrices. Then we make the corresponding updates to R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}. The new triangular matrix R_k is formed of R_{k-1} by deleting the first row and the first column if $m_k = m_{k-1}$ and by adding a new column to the right and a new row to the bottom of the matrix. The new column is given by $S_k^T \boldsymbol{u}_{k-1}$ and the new row has the value zero in its first $m_k - 1$ components. For matrix $U_k^T U_k$ both the new column and the new row are given by $U_k^T \boldsymbol{u}_{k-1}$. As far as C_k is concerned, we delete the first element of C_{k-1} and add $\boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1}$ as the last element (note that C_k is stored as a vector).

We first give an efficient algorithm for updating the limited memory BFGS matrix D_k and for computing the search direction $\boldsymbol{d}_k = -D_k \boldsymbol{\xi}_k$. This algorithm is used whenever the previous step was a serious step. Note that, after a serious step, the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$ and the correction vectors obtained at the previous iteration in Step 6 of Algorithm 5.2 can be equally expressed as $\boldsymbol{s}_{k-1} = \boldsymbol{x}_k - \boldsymbol{x}_{k-1}$ and $\boldsymbol{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1}$. Therefore, the calculations used are very similar to those given in Section 3.3. In fact, all the calculations in Section 3.3 can be done by replacing the gradient $\nabla f(\boldsymbol{x})$ by an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$. This means that updating R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1} is possible within $O(m_k^2)$ operations by storing a small amount of additional information, namely the m_{k-1}-vectors $S_{k-1}^T \boldsymbol{\xi}_{k-1}$ and $U_{k-1}^T \boldsymbol{\xi}_{k-1}$ from the previous iteration. Then the products $S_k^T \boldsymbol{u}_{k-1} = S_k^T(\boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1})$ and $U_k^T \boldsymbol{u}_{k-1} = U_k^T(\boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1})$ can be computed efficiently, since we already know $m_k - 1$ components of $S_k^T \boldsymbol{\xi}_{k-1}$ and $U_k^T \boldsymbol{\xi}_{k-1}$ from $S_{k-1}^T \boldsymbol{\xi}_{k-1}$ and $S_{k-1}^T \boldsymbol{\xi}_{k-1}$, respectively. We only need to calculate $\boldsymbol{s}_{k-1}^T \boldsymbol{\xi}_{k-1}$ and $\boldsymbol{u}_{k-1}^T \boldsymbol{\xi}_{k-1}$ and carry out the subtractions.

ALGORITHM 5.3. *(BFGS Updating and Direction Finding).*

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

Step 1: (*Positive definiteness.*) If

$$\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1} > 0,$$

then set $m_k = \min\{\, m_{k-1} + 1, m_c \,\}$ and update the matrices; that is, go to Step 2. Otherwise, skip the updates (see the note below), set $m_k = m_{k-1}$ and $\vartheta_k = \vartheta_{k-1}$, compute $S_k^T \boldsymbol{\xi}_k$ and $U_k^T \boldsymbol{\xi}_k$, and go to Step 7.

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

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

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

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

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

Step 6: If $\boldsymbol{u}_{k-1}^T \boldsymbol{u}_{k-1} > 0$, compute

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

Otherwise, set $\vartheta_k = 1.0$.

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

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

Step 8: (*Search direction.*) Compute

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

The condition (see Algorithm 5.3, Step 1)

$$\boldsymbol{u}_i^T \boldsymbol{s}_i > 0 \qquad \text{for all } i = 1, \ldots, k-1, \tag{5.17}$$

assures the positive definiteness of the matrices obtained by the limited memory BFGS update (see, e.g., [8]). Skipping the updates (for example, if condition (5.17) is not satisfied) means that we simple set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$, and $C_k = C_{k-1}$.

In Step 6 of Algorithm 5.3 we have already calculated both $\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1}$ and $\boldsymbol{u}_{k-1}^T \boldsymbol{u}_{k-1}$ (during the calculation of condition (5.17) and as a last component of vector $U_k^T \boldsymbol{u}_{k-1}$) and, thus, we only need to do one division to obtain scaling parameter ϑ_k.

Note that even if the matrices R_k, $U_k^T U_k$, and C_k can be updated within $O(m_k^2)$ operations by the limited memory BFGS update, the calculation of the search direction still requires $O(nm_k)$ operations.

If the previous step was a null step, then $\tilde{\boldsymbol{\xi}}_k \neq \boldsymbol{\xi}_k$ and, thus, there is no reason to utilize the difference $\boldsymbol{u}_{k-1} = \boldsymbol{\xi}_k - \boldsymbol{\xi}_m$ with $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{y}_k)$ and $\boldsymbol{\xi}_m \in \partial f(\boldsymbol{x}_k)$ in the calculations of $S_k^T \boldsymbol{u}_{k-1}$ and $U_k^T \boldsymbol{u}_{k-1}$ (compare Step 4 in Algorithm 5.3). Furthermore, in order to guarantee the global convergence of the method, the sequence (w_k) has to be nonincreasing in consecutive null steps (that is, $w_k \leq w_{k-1}$). Thus, the condition (compare with condition (4.11))

$$\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k \leq \tilde{\boldsymbol{\xi}}_k^T D_{k-1} \tilde{\boldsymbol{\xi}}_k \tag{5.18}$$

has to be satisfied each time there occurs more than one consecutive null step. In practice, these two facts mean that there are some more calculations required in the case of null steps than in the case of serious steps. However, also in this case the search direction can be calculated by using $O(nm_k)$ operations.

We now give an efficient algorithm for updating the limited memory SR1 matrix D_k and for computing the search direction $\boldsymbol{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k$. Together with Step 3 of Algorithm 5.2, this procedure guarantees that condition (5.18) is valid even when the correction ϱI is added to the new matrix D_k (see Lemma 5.2.8). This algorithm is used whenever the previous step was a null step.

ALGORITHM 5.4. *(SR1 Updating and Direction Finding).*

Data: Suppose that the number of current correction pairs is m_{k-1} and the maximum number of stored correction pairs is m_c. Suppose that we have the most recent vectors $\boldsymbol{s}_{k-1}$ and $\boldsymbol{u}_{k-1}$ (from the previous iteration), the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$, the previous aggregate subgradient $\tilde{\boldsymbol{\xi}}_{k-1}$, the previous search direction $\boldsymbol{d}_{k-1}$, the $n \times m_{k-1}$ matrices S_{k-1} and U_{k-1}, the $m_{k-1} \times m_{k-1}$ matrices R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}, and the previous scaling parameter ϑ_{k-1} available. In addition, suppose that we have the number of consecutive null steps $i_{null} = k - m \geq 1$.

Step 1: (*Initial vectors and initialization.*) Compute m_{k-1}-vectors $S_{k-1}^T \tilde{\boldsymbol{\xi}}_k$ and $U_{k-1}^T \tilde{\boldsymbol{\xi}}_k$. Set $\vartheta_k = 1.0$ and $i_{up} = 0$.

Step 2: (*Positive definiteness.*) If

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

then set $m_k = \min\{ m_{k-1} + 1, m_c \}$ and calculate $\boldsymbol{s}_{k-1}^T \tilde{\boldsymbol{\xi}}_k$ and $\boldsymbol{u}_{k-1}^T \tilde{\boldsymbol{\xi}}_k$. Otherwise, skip the updates, set $S_k^T \tilde{\boldsymbol{\xi}}_k = S_{k-1}^T \tilde{\boldsymbol{\xi}}_k$, $U_k^T \tilde{\boldsymbol{\xi}}_k = U_{k-1}^T \tilde{\boldsymbol{\xi}}_k$, $m_k = m_{k-1}$, and go to Step 8.

Step 3: (*Update conditions.*) If either

$$\begin{aligned} & i_{null} = 1, \qquad \text{or} \\ & m_k < m_c, \end{aligned}$$

then update the matrices, that is, go to Step 4. Otherwise, solve the system of linear equations

$$(\vartheta_{k-1} U_{k-1}^T U_{k-1} - R_{k-1} - R_{k-1}^T + C_{k-1}) \boldsymbol{p} = \vartheta_{k-1} U_{k-1}^T \tilde{\boldsymbol{\xi}}_k - S_{k-1}^T \tilde{\boldsymbol{\xi}}_k.$$

to obtain $\boldsymbol{p} \in \mathbb{R}^{m_{k-1}}$. Calculate the vector $\boldsymbol{z} \in \mathbb{R}^n$ as

$$\boldsymbol{z} = \vartheta_{k-1} \tilde{\boldsymbol{\xi}}_k - (\vartheta_{k-1} U_{k-1} - S_{k-1}) \boldsymbol{p},$$

and the scalar

$$a = \tilde{\boldsymbol{\xi}}_k^T \boldsymbol{z}.$$

Set $i_{up} = 1$.

Step 4: Obtain S_k and U_k by updating S_{k-1} and U_{k-1}.

Step 5: Compute m_k-vectors $S_k^T \boldsymbol{u}_{k-1}$ and $U_k^T \boldsymbol{u}_{k-1}$.

Step 6: Update $m_k \times m_k$ matrices R_k, $U_k^T U_k$, and C_k.

Step 7: Construct m_k-vectors $S_k^T \tilde{\boldsymbol{\xi}}_k$ and $U_k^T \tilde{\boldsymbol{\xi}}_k$ using $S_{k-1}^T \tilde{\boldsymbol{\xi}}_k$, $U_{k-1}^T \tilde{\boldsymbol{\xi}}_k$, $\boldsymbol{s}_{k-1}^T \tilde{\boldsymbol{\xi}}_k$, and $\boldsymbol{u}_{k-1}^T \tilde{\boldsymbol{\xi}}_k$.

Step 8: (*Intermediate value.*) Solve $\boldsymbol{p} \in \mathbb{R}^{m_k}$ satisfying the system of linear equations

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

Step 9: (*Search direction.*) Compute

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

Step 10: (*Update conditions II.*) If $i_{up} = 1$, then calculate

$$b = \tilde{\boldsymbol{\xi}}_k \boldsymbol{d}_k,$$

and in case of

$$b + a < 0,$$

(i.e., condition (5.18) is not valid) set $m_k = m_{k-1}$, $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$, $C_k = C_{k-1}$, and

$$\boldsymbol{d}_k = -\boldsymbol{z}.$$

LEMMA 5.1.1. *The condition (see Algorithm 5.4, Step 2)*

$$-\boldsymbol{d}_i^T \boldsymbol{u}_i - \tilde{\boldsymbol{\xi}}_i^T \boldsymbol{s}_i < 0 \qquad \textit{for all } i = 1, \ldots, k-1, \tag{5.19}$$

assures the positive definiteness of the matrices obtained by the limited memory SR1 update.

PROOF. Let us denote $B_i = D_i^{-1}$ for all $i = 1, \ldots, k$. We now prove that each matrix B_k, $k \geq 1$ is positive definite when condition (5.19) is valid. Note that if B_k is positive definite, then also its inverse D_k is positive definite.

By applying the Sherman-Morrison-Woodbury formula (see, e.g., [16]) to (5.16) we obtain

$$B_k = B_1 + (U_k - B_1 S_k)(L_k + L_k^T + C_k - S_k^T B_1 S_k)^{-1}(U_k - B_1 S_k)^T, \tag{5.20}$$

where matrices S_k, U_k, and C_k are defined as before, $B_1 = D_1^{-1} = I$ is a positive definite initial matrix, and

$$(L_k)_{ij} = \begin{cases} (\boldsymbol{s}_{k-m_k-1+i})^T (\boldsymbol{u}_{k-m_k-1+j}) & \text{if } i > j \\ 0 & \text{otherwise.} \end{cases}$$

We denote

$$Q_k = \begin{bmatrix} \boldsymbol{q}_{k-m_k} & \cdots & \boldsymbol{q}_{k-1} \end{bmatrix} = U_k - B_1 S_k$$

and

$$N_k = L_k + L_k^T + C_k - S_k^T B_1 S_k.$$

Let us assume that B_{k-1} is positive definite for some $k > 1$ and that condition (5.19) is valid for $i = 1, \ldots, k-1$. We prove that also the matrix B_k is positive definite. To simplify the notation, we, from now on, omit the index $(k-1)$ and replace the index k by "+". Thus, equation (5.20) can be rewritten in the form

$$\begin{aligned} B_+ &= B_1 + Q_+ N_+^{-1} Q_+^T \\ &= B' + \frac{(\boldsymbol{u} - B'\boldsymbol{s})\,(\boldsymbol{u} - B'\boldsymbol{s})^T}{\boldsymbol{s}^T\,(\boldsymbol{u} - B'\boldsymbol{s})}, \end{aligned} \tag{5.21}$$

where $B' = B_1 + Q' N'^{-1} Q'^T$, N' is formed by deleting the first row and the first column from N if $k > m_c + 1$, and Q' is formed by deleting the first column from Q. If $k \leq m_c + 1$, then $N' = N$ and $Q' = Q$.

It can be easily seen from (5.21) that the new matrix B_+ is positive definite if B' is positive definite and the denominator $\boldsymbol{s}^T\,(\boldsymbol{u} - B'\boldsymbol{s})$ is greater than zero (note that the matrix $A = \boldsymbol{z}\boldsymbol{z}^T$ is positive semidefinite for all nonzero $\boldsymbol{z} \in \mathbb{R}^n$).

Next we prove that B' is positive definite. The current matrix B is positive definite by assumption and, due to condition (5.19), also $QN^{-1}Q^T$, N^{-1}, and N are positive definite. The positive definiteness of N implies the positive definiteness

of N' as a submatrix of matrix N. Now, since N' is positive definite also N'^{-1}, $Q'N'^{-1}Q'^T$, and, thus, $B' = B_1 + Q'N'^{-1}Q'^T$ are positive definite.

Using condition (5.19) and the fact that we have $\boldsymbol{s}_i = t_R^i\theta_i\boldsymbol{d}_i$, $t_R^i > 0$, $\theta_i \in (0,1]$, and $\boldsymbol{d}_i = -D_i\tilde{\boldsymbol{\xi}}_i$ for all $i = 1,\ldots,k-1$, we obtain

$$\boldsymbol{d}_i^T\boldsymbol{u}_i > t_R^i\theta_i\tilde{\boldsymbol{\xi}}_i^T D_i\tilde{\boldsymbol{\xi}}_i = t_R^i\theta_i\tilde{\boldsymbol{\xi}}_i^T D_iB_iD_i\tilde{\boldsymbol{\xi}}_i = t_R^i\theta_i\boldsymbol{d}_i^T B_i\boldsymbol{d}_i \geq t_R^i\theta_i\boldsymbol{d}_i^T B_i'\boldsymbol{d}_i \qquad (5.22)$$

for all $i = 1,\ldots,k-1$. The last inequality in formula (5.22) follows from the fact that for $i \leq m_c+1$ the positive definite matrix B_i is equal to B_i' and for $i > m_c+1$ it can be given in the form

$$\begin{aligned}
B_i &= B_1 + Q_iN_i^{-1}Q_i^T \\
&= B_1 + \begin{bmatrix}\boldsymbol{q}_- & Q_i'\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_-^T\boldsymbol{q}_- & \boldsymbol{q}_-^TS_i' \\ S_i'^T\boldsymbol{q}_- & N_i'\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{q}_-^T \\ Q_i'^T\end{bmatrix} \\
&= B_1 + \begin{bmatrix}\boldsymbol{q}_- & Q_i'\end{bmatrix}\begin{bmatrix}1/\delta & -\boldsymbol{q}_-^TS_i'N_i'^{-1}/\delta \\ -N_i'^{-1}S_i'^T\boldsymbol{q}_-/\delta & N_i'^{-1} + N_i'^{-1}S_i'^T\boldsymbol{q}_-\boldsymbol{q}_-^TS_i'N_i'^{-1}/\delta\end{bmatrix}\begin{bmatrix}\boldsymbol{q}_-^T \\ Q_i'^T\end{bmatrix} \\
&= B_i' + (\boldsymbol{q}_- - Q_i'N_i'^{-1}S_i'^T\boldsymbol{q}_-)(\boldsymbol{q}_- - Q_i'N_i'^{-1}S_i'^T\boldsymbol{q}_-)^T/\delta.
\end{aligned}$$

Here the subscript "$-$" denotes the index $i - m_i - 1$, the matrix S_i' is formed by deleting the first column from S_i, and the denominator

$$\delta = \boldsymbol{s}_-^T\boldsymbol{q}_- - \boldsymbol{q}_-^TS_i'N_i'^{-1}S_i'^T\boldsymbol{q}_-$$

is greater than zero, since the matrix N_i^{-1} is positive definite (that is, the upper left term $1/\delta$ has to be greater than zero).

From (5.22) and by using again the fact that $\boldsymbol{s}_i = t_R^i\theta_i\boldsymbol{d}_i$ we obtain

$$\boldsymbol{s}_i^T(\boldsymbol{u}_i - B_i'\boldsymbol{s}_i) > 0$$

for all $i = 1,\ldots,k-1$. Thus, the denominator in formula (5.21) is greater than zero and the new matrix B_+ and its inverse D_+ are positive definite. □

In order to use both Algorithms 5.3 and 5.4 with the same stored information, some modifications have to be made. First, during the limited memory SR1 update, we have to update the m_k-vectors $S_k^T\boldsymbol{\xi}_m$ and $U_k^T\boldsymbol{\xi}_m$ (that is, $S_{k-1}^T\boldsymbol{\xi}_{k-1}$ and $U_{k-1}^T\boldsymbol{\xi}_{k-1}$ to the next limited memory BFGS update). Furthermore, since we use the same correction matrices S_k and U_k in both of the BFGS and the SR1 updates, both the positive definiteness conditions (5.17) and (5.19) have to be valid in each case before we update the matrices. Nevertheless, it can be seen from (5.22) provided by the positive definiteness of B_i that condition (5.19) implies (5.17) and, thus, we only have to check the validity of (5.19). However, numerical experiments have shown that the simple skipping of the BFGS update (see Algorithm 5.3, Step 1) if condition (5.19) required for the SR1 update is not satisfied, makes the method inefficient. Therefore, in the case of the BFGS update, the new search direction $\boldsymbol{d}_k$ is calculated conventionally by using the most recent correction vectors $\boldsymbol{s}_{k-1}$ and $\boldsymbol{u}_{k-1}$ whenever the required positive definiteness condition (5.17) is valid but

the matrices are not updated, that is, the correction vectors are not stored, unless both the conditions (5.17) and (5.19) are satisfied. In practice, this means that the correction matrices S_k and U_k may actually include some indices smaller than $k - m_k$ due to skipping of updates and that, in the case of the BFGS update, the number of the current correction pairs used may be $m_k = m_c + 1$.

The new limited memory bundle method uses the limited memory approach to calculate the search direction and it requires only three subgradients and two locality measures to calculate the new aggregate values; thus, it avoids solving the time-consuming quadratic direction finding problem (4.6) appearing in standard bundle methods and the size of the bundle is independent of the dimension of the problem. Furthermore, both the search direction $\boldsymbol{d}_k$ and the aggregate values $\tilde{\boldsymbol{\xi}}_{k+1}$ and $\tilde{\beta}_{k+1}$ can be computed implicitly using at most $O(nm_c)$ operations. When $m_c \ll n$, this is much less than $O(n^2)$ operations needed in the original variable metric bundle method, which stores and manipulates the whole matrix D_k. These properties make the limited memory bundle method suitable for large-scale problems. This assertion is supported by numerical tests presented in [26, 27] and to be given in Chapter 6.

5.2 Convergence Analysis

In this section, we prove the global convergence of Algorithm 5.2. In addition to assuming that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous, the level set $\{ \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_1) \}$ is supposed to be bounded. Furthermore, we assume that each execution of the line search procedure is finite. Since the optimality condition of Theorem 2.3.3 is not sufficient without some convexity assumptions, and the objective function f is not supposed to be convex, we can only prove that Algorithm 5.2 either terminates at a stationary point or generates an infinite sequence $(\boldsymbol{x}_k)$ for which accumulation points are stationary for f. In order to do this, we assume that the final accuracy tolerance ε is equal to zero.

We start the convergence analysis by giving three technical results (in Lemmas 5.2.1, 5.2.2, and 5.2.3). After that, we prove (in Theorem 5.2.4) that having the values $w_k = 0$ and $q_k = 0$ implies that the corresponding point $\boldsymbol{x}_k$ is a stationary point for the objective function. For an infinite sequence $(\boldsymbol{x}_k)$, we first show (in Lemma 5.2.5) that the conditions $(q_k) \to 0$ and $(w_k) \to 0$ are equivalent due to correction (5.13) and, thus, we can restrict our consideration to the stopping parameter w_k. Then we show (in Lemma 5.2.6) that if $(\boldsymbol{x}_k)_{k \in \mathcal{K}} \to \bar{\boldsymbol{x}}$ and $(w_k)_{k \in \mathcal{K}} \to 0$ for some subset $\mathcal{K} \subset \{1, 2, \ldots\}$, then the accumulation point $\bar{\boldsymbol{x}}$ is a stationary point of the objective function. This assertion requires the uniformly positive definiteness of D_k, which also is guaranteed by correction (5.13). Furthermore, using the technical Lemma 5.2.7 and the fact that the sequence (w_k) is nonincreasing in the consecutive null steps due to additional testing procedure during the limited memory SR1 update (see Lemma 5.2.8), we prove that the indefinite sequence of consecutive null steps with $\boldsymbol{x}_k = \boldsymbol{x}_m$ implies $\mathbf{0} \in \partial f(\boldsymbol{x}_m)$ (in Lemma 5.2.9). Finally, in Theorem 5.2.10 we combine all the results obtained and show that every

accumulation point of $(\boldsymbol{x}_k)$ is stationary for the objective function.

The convergence analysis of the limited memory bundle method is very similar to that of the original variable metric bundle method for nonconvex objective functions [78]. In fact, Lemmas 5.2.2, 5.2.3, and 5.2.7 and their proofs are exactly the same as those given in [78] (note that the proof of Lemma 5.2.7 can be found from [54]) and also Lemmas 5.2.1, 5.2.6, 5.2.9, and Theorems 5.2.4, and 5.2.10 are very similar to the corresponding ones in the original variable metric bundle method. However, we give those results (with their proofs) here to make the convergence analysis of the limited memory bundle method self-contained.

There are two main differences between the convergence analysis of the original variable metric bundle method and the limited memory bundle method. The first one is that in the limited memory bundle method, we have two different stopping parameters w_k and q_k (see Step 5 in Algorithm 5.2) instead of only one [78]. However, Lemma 5.2.5 shows that $(q_k) \to 0$ implies $(w_k) \to 0$ and vice versa and, thus, it is enough to examine only the stopping parameter w_k, which is similar to that of the original variable metric bundle method. Furthermore, in the limited memory bundle method we are not able to guarantee that the sequence (w_k) is nonincreasing in the consecutive null steps without an additional testing procedure during the limited memory SR1 update (see Algorithm 5.4). Lemma 5.2.8 shows that together with Step 3 of Algorithm 5.2, the procedure used in Algorithm 5.4 guarantees that condition (5.18) (that is, $\tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k \leq \tilde{\boldsymbol{\xi}}_k^T D_{k-1} \tilde{\boldsymbol{\xi}}_k$) is valid even when the correction ϱI is added to the new matrix D_k.

For the convenience of the reader, we from now on give also the page references of the equations and algorithms used.

LEMMA 5.2.1. *At the kth iteration of Algorithm 5.2 [p. 58], we have*

$$\begin{aligned} w_k &= \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k, \qquad w_k \geq 2\tilde{\beta}_k, \qquad w_k \geq \varrho \|\tilde{\boldsymbol{\xi}}_k\|^2, \\ q_k &= \frac{1}{2}\|\tilde{\boldsymbol{\xi}}_k\|^2 + \tilde{\beta}_k, \qquad q_k \geq \tilde{\beta}_k, \qquad q_k \geq \frac{1}{2}\|\tilde{\boldsymbol{\xi}}_k\|^2, \end{aligned}$$

and

$$\beta_{k+1} \geq \gamma \|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\|^{\omega}.$$

Furthermore, if condition (5.19) [p. 65] is valid for $k = k + 1$, then

$$\boldsymbol{u}_k^T (D_k \boldsymbol{u}_k - \boldsymbol{s}_k) > 0. \tag{5.23}$$

PROOF. We point out first that $\tilde{\beta}_k \geq 0$ for all k by (5.6) [p. 54], (5.10) [p. 57], and Step 1 in Algorithm 5.2. The relations

$$\begin{aligned} w_k &= \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k, \qquad w_k \geq 2\tilde{\beta}_k, \qquad w_k \geq \varrho \|\tilde{\boldsymbol{\xi}}_k\|^2, \\ q_k &= \frac{1}{2}\|\tilde{\boldsymbol{\xi}}_k\|^2 + \tilde{\beta}_k, \qquad q_k \geq \tilde{\beta}_k, \qquad q_k \geq \frac{1}{2}\|\tilde{\boldsymbol{\xi}}_k\|^2 \end{aligned}$$

follow immediately from (5.1) [p. 53], (5.2) [p. 53], (5.11) [p. 57], (5.12) [p. 57], and (5.13) [p. 58], and from the fact that $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ in the case of a serious step (see Step 1 in Algorithm 5.2). Note that if correction (5.13) [p. 58] is used, we have $D_k = D_k + \varrho I$ and, thus, these results are valid also in this case.

By (5.6) [p. 54] and since we have $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ for null steps, and since we, on the other hand, have $\beta_{k+1} = 0$, and $\|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\| = 0$ for serious steps, we always have

$$\beta_{k+1} \geq \gamma \|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\|^{\omega}$$

for some $\gamma \geq 0$ and $\omega \geq 1$.

Now, we prove that condition (5.19) [p. 65] implies (5.23). If condition (5.19) is valid for k replaced with $k+1$, then $\tilde{\boldsymbol{\xi}}_k \neq \mathbf{0}$ (otherwise, we would have $-\boldsymbol{d}_k^T \boldsymbol{u}_k - \tilde{\boldsymbol{\xi}}_k^T \boldsymbol{s}_k = 0$). Using the positiveness of $\boldsymbol{u}_k^T \boldsymbol{s}_k$ (provided by the positive definiteness of D_k, in (5.22) [p. 66]), Cauchy's inequality, and the fact that we have $\boldsymbol{s}_k = t_R^k \theta_k \boldsymbol{d}_k$, $t_R^k > 0$, and $\theta_k \in (0, 1]$, we obtain

$$\begin{aligned}
(\boldsymbol{u}_k^T \boldsymbol{s}_k)^2 &= (t_R^k \theta_k \tilde{\boldsymbol{\xi}}_k^T D_k \boldsymbol{u}_k)^2 \\
&\leq (t_R^k \theta_k)^2 \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k \boldsymbol{u}_k^T D_k \boldsymbol{u}_k \\
&= t_R^k \theta_k \boldsymbol{u}_k^T D_k \boldsymbol{u}_k (-\boldsymbol{s}_k^T \tilde{\boldsymbol{\xi}}_k) \\
&< t_R^k \theta_k \boldsymbol{u}_k^T D_k \boldsymbol{u}_k \boldsymbol{d}_k^T \boldsymbol{u}_k = \boldsymbol{u}_k^T D_k \boldsymbol{u}_k \boldsymbol{u}_k^T \boldsymbol{s}_k.
\end{aligned}$$

Therefore, we have $\boldsymbol{u}_k^T \boldsymbol{s}_k < \boldsymbol{u}_k^T D_k \boldsymbol{u}_k$. □

LEMMA 5.2.2. *Suppose that Algorithm 5.2 [p. 58] is not terminated before the kth iteration. Then, there exist numbers $\lambda^{k,j} \geq 0$ for $j = 1, \ldots, k$ and $\tilde{\sigma}_k \geq 0$ such that*

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

PROOF. Let m be an index of the iteration after the latest serious step defined at Step 1 of Algorithm 5.2 (that is, $\boldsymbol{x}_j = \boldsymbol{x}_m$ for all $j = m, \ldots, k$). First we prove that there exist numbers $\lambda^{k,j} \geq 0$ for $j = m, \ldots, k$, such that

$$(\tilde{\boldsymbol{\xi}}_k, \tilde{\beta}_k) = \sum_{j=m}^{k} \lambda^{k,j} (\boldsymbol{\xi}_j, \beta_j), \qquad \sum_{j=m}^{k} \lambda^{k,j} = 1. \tag{5.24}$$

We prove this via induction. Suppose that $k = m$. Then we set $\lambda^{m,m} = 1$, since $\tilde{\boldsymbol{\xi}}_m = \boldsymbol{\xi}_m$ and $\tilde{\beta}_m = 0$ at Step 1 of Algorithm 5.2 and we have set $\beta_m = 0$ at Step 6 at the previous iteration ($\beta_1 = 0$ due to initialization). Thus, the base case is valid. Now, suppose that $k > m$, let $i \in \{m, \ldots, k-1\}$, and assume that (5.24) is valid for k replaced with i. We define

$$\begin{aligned}
&\lambda^{i+1,m} = \lambda_1^i + \lambda_3^i \lambda^{i,m}, \\
&\lambda^{i+1,j} = \lambda_3^i \lambda^{i,j} \quad \text{for } j = m+1, \ldots, i, \qquad \text{and} \\
&\lambda^{i+1,i+1} = \lambda_2^i,
\end{aligned}$$

where $\lambda_l^i \geq 0$ for all $l \in \{1, 2, 3\}$ are obtained at Step 7 of Algorithm 5.2.

Now, we have $\lambda^{i+1,j} \geq 0$ for all $j = m, \ldots, i+1$, and

$$\sum_{j=m}^{i+1} \lambda^{i+1,j} = \lambda_1^i + \lambda_3^i \left(\lambda^{i,m} + \sum_{j=m+1}^{i} \lambda^{i,j} \right) + \lambda_2^i = 1,$$

since $\sum_{j=m}^{i} \lambda^{i,j} = 1$ due to assumption and $\sum_{l=1}^{3} \lambda_l^i = 1$ (see Algorithm 5.2, Step 7).

Using relations (5.9) [p. 57], (5.10) [p. 57], and the result given above, we obtain

$$\begin{aligned}(\tilde{\boldsymbol{\xi}}_{i+1}, \tilde{\beta}_{i+1}) &= \lambda_1^i(\boldsymbol{\xi}_m, 0) + \lambda_2^i(\boldsymbol{\xi}_{i+1}, \beta_{i+1}) + \sum_{j=m}^{i} \lambda_3^i \lambda^{i,j}(\boldsymbol{\xi}_j, \beta_j) \\ &= \sum_{j=m}^{i+1} \lambda^{i+1,j}(\boldsymbol{\xi}_j, \beta_j),\end{aligned}$$

due to $\beta_m = 0$ and, thus, condition (5.24) is valid for $i+1$.

Now, we define

$$\begin{aligned}&\lambda^{k,j} = 0 \quad \text{for } j = 1, \ldots, m-1, \qquad \text{and} \\ &\tilde{\sigma}_k = \sum_{j=1}^{k} \lambda^{k,j} \|\boldsymbol{y}_j - \boldsymbol{x}_k\|.\end{aligned}$$

Since $\boldsymbol{x}_j = \boldsymbol{x}_k$ for $j = m, \ldots, k$, we obtain

$$\tilde{\sigma}_k = \sum_{j=m}^{k} \lambda^{k,j} \|\boldsymbol{y}_j - \boldsymbol{x}_j\|,$$

and, thus, by (5.24), Lemma 5.2.1, and the convexity of the function $g \to \gamma g^\omega$ on $\mathbb{R}_+$ for $\gamma \geq 0$ and $\omega \geq 1$ we have

$$\begin{aligned}\gamma \tilde{\sigma}_k^\omega &= \gamma \left(\sum_{j=m}^{k} \lambda^{k,j} \|\boldsymbol{y}_j - \boldsymbol{x}_j\| \right)^\omega \\ &\leq \sum_{j=m}^{k} \lambda^{k,j} \gamma \|\boldsymbol{y}_j - \boldsymbol{x}_j\|^\omega \\ &\leq \sum_{j=m}^{k} \lambda^{k,j} \beta_j \\ &= \tilde{\beta}_k.\end{aligned}$$

□

LEMMA 5.2.3. *Let $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ be given and suppose that there exist vectors $\bar{\boldsymbol{g}}$, $\bar{\boldsymbol{\xi}}_i$, $\bar{\boldsymbol{y}}_i$, and numbers $\bar{\lambda}_i \geq 0$ for $i = 1, \ldots, l$, $l \geq 1$, such that*

$$\begin{aligned} &(\bar{\boldsymbol{g}}, 0) = \sum_{i=1}^{l} \bar{\lambda}_i (\bar{\boldsymbol{\xi}}_i, \|\bar{\boldsymbol{y}}_i - \bar{\boldsymbol{x}}\|), \\ &\bar{\boldsymbol{\xi}}_i \in \partial f(\bar{\boldsymbol{y}}_i), \quad i = 1, \ldots, l, \qquad \textit{and} \\ &\sum_{i=1}^{l} \bar{\lambda}_i = 1. \end{aligned} \tag{5.25}$$

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

PROOF. Let $\mathcal{I} = \{\, i \mid 1 \leq i \leq l,\ \bar{\lambda}_i > 0 \,\}$. By (5.25) we have

$$\begin{aligned} &\bar{\boldsymbol{y}}_i = \bar{\boldsymbol{x}} \qquad \text{and} \\ &\bar{\boldsymbol{\xi}}_i \in \partial f(\bar{\boldsymbol{x}}) \end{aligned}$$

for all $i \in \mathcal{I}$. Thus, we have also

$$\begin{aligned} &\bar{\boldsymbol{g}} = \sum_{i \in \mathcal{I}} \bar{\lambda}_i \bar{\boldsymbol{\xi}}_i, \\ &\bar{\lambda}_i > 0, \qquad \text{for } i \in \mathcal{I}, \qquad \text{and} \\ &\sum_{i \in \mathcal{I}} \bar{\lambda}_i = 1, \end{aligned}$$

and $\bar{\boldsymbol{g}} \in \partial f(\bar{\boldsymbol{x}})$ by the convexity of $\partial f(\bar{\boldsymbol{x}})$ (see Theorem 2.2.6, p. 22). □

THEOREM 5.2.4. *If Algorithm 5.2 [p. 58] terminates at the kth iteration, then the point $\boldsymbol{x}_k$ is stationary for f.*

PROOF. If Algorithm 5.2 terminates at Step 5, then the fact $\varepsilon = 0$ implies that $w_k = 0$ and $q_k = 0$. Thus, $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{0}$ and $\tilde{\beta}_k = \tilde{\sigma}_k = 0$ by Lemma 5.2.1 and Lemma 5.2.2.

Now, by Lemma 5.2.2 and by using Lemma 5.2.3 with

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

we obtain $\boldsymbol{0} = \tilde{\boldsymbol{\xi}}_k \in \partial f(\boldsymbol{x}_k)$ and, thus, $\boldsymbol{x}_k$ is stationary for f. □

From now on, we suppose that Algorithm 5.2 does not terminate, that is, $w_k > 0$ and $q_k > 0$ for all k.

LEMMA 5.2.5. *Let the stopping parameters w_k and q_k of Algorithm 5.2 [p. 58] be defined by (5.11) [p. 57] and (5.12) [p. 57], respectively. Then*

$$(q_k) \to 0 \qquad \textit{if and only if} \qquad (w_k) \to 0.$$

PROOF. The condition $(q_k) \to 0$ implies $(\tilde{\boldsymbol{\xi}}_k) \to \mathbf{0}$ and $(\tilde{\beta}_k) \to 0$ by Lemma 5.2.1 and, thus, we have $(w_k) \to 0$.

On the other hand, by Lemma 5.2.1 we have $w_k \geq 2\tilde{\beta}_k \geq 0$ and $w_k \geq \varrho \|\tilde{\boldsymbol{\xi}}_k\|^2$ for some correction parameter $\varrho \in (0, 1/2)$. Therefore, $(w_k) \to 0$ implies $(\tilde{\beta}_k) \to 0$ and $(\tilde{\boldsymbol{\xi}}_k) \to \mathbf{0}$ and, thus, also $(q_k) \to 0$. □

In view of Lemma 5.2.5 we can, from now on, restrict our consideration to the stopping parameter w_k.

LEMMA 5.2.6. *Suppose that the level set* $\{ \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_1) \}$ *is bounded. Then, the sequences* $(\boldsymbol{y}_k)$ *and* $(\boldsymbol{\xi}_k)$ *are also bounded. If, in addition, there exist a point* $\bar{\boldsymbol{x}} \in \mathbb{R}^n$ *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* $\mathbf{0} \in \partial f(\bar{\boldsymbol{x}})$.

PROOF. The sequence $(\boldsymbol{x}_k)$ is bounded by assumption and the monotonicity of the sequence (f_k) obtained due to the serious descent criterion (5.7) [p. 56]. Since $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1}$ for serious steps and $\|\boldsymbol{y}_{k+1} - \boldsymbol{x}_{k+1}\| \leq t_{max} C$ for null steps (by (5.3) [p. 53] and due to the fact that 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 semicontinuity of ∂f (see Theorem 2.2.6, p. 22), we obtain the boundedness of the sequence $(\boldsymbol{\xi}_k)$.

Let

$$\mathcal{I} = \{1, \ldots, n+2\}.$$

Using the fact that $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{y}_k)$ for all $k \geq 1$, Lemma 5.2.2, and Carathéodory's theorem (see, e.g., [28], p. 98), we deduce that there exist vectors $\boldsymbol{y}^{k,i}$, $\boldsymbol{\xi}^{k,i}$, and numbers $\lambda^{k,i} \geq 0$ and $\tilde{\sigma}_k$ for $i \in \mathcal{I}$ and $k \geq 1$, such that

$$\begin{aligned} &(\tilde{\boldsymbol{\xi}}_k, \tilde{\sigma}_k) = \sum_{i \in \mathcal{I}} \lambda^{k,i} (\boldsymbol{\xi}^{k,i}, \|\boldsymbol{y}^{k,i} - \boldsymbol{x}_k\|), \\ &\boldsymbol{\xi}^{k,i} \in \partial f(\boldsymbol{y}^{k,i}), \qquad \text{and} \\ &\sum_{i \in \mathcal{I}} \lambda^{k,i} = 1, \end{aligned} \tag{5.26}$$

with

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

From the boundedness of $(\boldsymbol{y}_k)$, we obtain the existence of points $\boldsymbol{y}_i^*$ $(i \in \mathcal{I})$, and an infinite set $\mathcal{K}_0 \subset \mathcal{K}$ satisfying $(\boldsymbol{y}^{k,i})_{k \in \mathcal{K}_0} \to \boldsymbol{y}_i^*$ for $i \in \mathcal{I}$. The boundedness of $(\boldsymbol{\xi}_k)$ and $(\lambda^{k,i})$ gives us the existence of vectors $\boldsymbol{\xi}_i^* \in \partial f(\boldsymbol{y}_i^*)$, numbers λ_i^* for $i \in \mathcal{I}$, and an infinite set $\mathcal{K}_1 \subset \mathcal{K}_0$ satisfying $(\boldsymbol{\xi}^{k,i})_{k \in \mathcal{K}_1} \to \boldsymbol{\xi}_i^*$ and $(\lambda^{k,i})_{k \in \mathcal{K}_1} \to \lambda_i^*$ for $i \in \mathcal{I}$.

It can be seen from (5.26) that

$$\lambda_i^* \geq 0 \quad \text{for } i \in \mathcal{I}, \qquad \text{and} \qquad \sum_{i \in \mathcal{I}} \lambda_i^* = 1.$$

From the fact $(w_k)_{k\in\mathcal{K}} \to 0$, Lemma 5.2.1, and Lemma 5.2.2, we obtain

$$(\tilde{\boldsymbol{\xi}}_k)_{k\in\mathcal{K}} \to \mathbf{0}, \qquad (\tilde{\beta}_k)_{k\in\mathcal{K}} \to 0, \qquad \text{and} \qquad (\tilde{\sigma}_k)_{k\in\mathcal{K}} \to 0.$$

By letting $k \in \mathcal{K}_1$ approach infinity in (5.26), and by using Lemma 5.2.3 with

$$\begin{aligned} \bar{\boldsymbol{x}} &= \boldsymbol{x}_k, \qquad & l &= n+2, \qquad & \bar{\boldsymbol{g}} &= \mathbf{0}, \\ \bar{\boldsymbol{\xi}}_i &= \boldsymbol{\xi}_i^*, \qquad & \bar{\boldsymbol{y}}_i &= \boldsymbol{y}_i^*, \qquad & \bar{\lambda}_i &= \lambda_i^* \end{aligned}$$

we obtain $\mathbf{0} \in \partial f(\bar{\boldsymbol{x}})$. □

LEMMA 5.2.7. *Suppose that there exist vectors $\boldsymbol{p}$ and $\boldsymbol{g}$ together with numbers $w \geq 0$, $\alpha \geq 0$, $\beta \geq 0$, $M \geq 0$, and $c \in (0, 1/2)$ such that*

$$w = \|\boldsymbol{p}\|^2 + 2\alpha, \quad \beta + \boldsymbol{p}^T\boldsymbol{g} \leq cw, \quad \textit{and} \quad \max\{\|\boldsymbol{p}\|, \|\boldsymbol{g}\|, \sqrt{\alpha}\} \leq M.$$

Let $Q : [0,1] \to \mathbb{R}$ be such that

$$\begin{aligned} &Q(\lambda) = \|\lambda\boldsymbol{g} + (1-\lambda)\boldsymbol{p}\|^2 + 2(\lambda\beta + (1-\lambda)\alpha) \qquad \textit{and} \\ &b = (1-2c)/4M. \end{aligned}$$

Then

$$\min\{Q(\lambda) \mid \lambda \in [0,1]\} \leq w - w^2b^2.$$

PROOF. See the proof of Lemma 3.5 in [54]. □

LEMMA 5.2.8. *Suppose that the level set $\{\, \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_1) \,\}$ is bounded, the number of serious steps is finite, and the last serious step occurred at the iteration $m-1$. Then there exists a number $k^* \geq m$, such that*

$$\tilde{\boldsymbol{\xi}}_{k+1}^T D_{k+1} \tilde{\boldsymbol{\xi}}_{k+1} \leq \tilde{\boldsymbol{\xi}}_{k+1}^T D_k \tilde{\boldsymbol{\xi}}_{k+1} \qquad \textit{and} \tag{5.27}$$

$$\operatorname{tr}(D_k) < \frac{3}{2}n \tag{5.28}$$

for all $k \geq k^*$.

PROOF. Suppose first that $i_{CN} = 0$ for all $k \geq m$, that is, the correction ϱI (see Algorithm 5.2, Step 3, p. 58) is not added to any matrix D_k with $k \geq m$. If the limited memory SR1 update is not used, we have $D_{k+1} = D_k$, and condition (5.27) is valid with equalities. Otherwise, if $m_k < m_c$, the limited memory SR1 update is equal to the standard SR1 update (see (3.7), p. 30) and we have

$$D_{k+1} = D_k - \frac{(D_k\boldsymbol{u}_k - \boldsymbol{s}_k)(D_k\boldsymbol{u}_k - \boldsymbol{s}_k)^T}{\boldsymbol{u}_k^T(D_k\boldsymbol{u}_k - \boldsymbol{s}_k)},$$

where for the denominator we have $\boldsymbol{u}_k^T(D_k\boldsymbol{u}_k - \boldsymbol{s}_k) > 0$ by Lemma 5.2.1 and the numerator is positive (semi)definite matrix. Thus, condition (5.27) is valid in the

case $m_k < m_c$. Finally, if $m_k = m_c$, we update the matrix only if $\tilde{\boldsymbol{\xi}}_{k+1}^T(D_{k+1} - D_k)\tilde{\boldsymbol{\xi}}_{k+1} \leq 0$ (see Algorithm 5.4, Steps 3 and 10, p. 63). Since $i_{CN} = 0$ for all $k \geq m$, the correction ϱI is not added to the new matrix D_{k+1} and, thus, condition (5.27) is valid also in this case. Furthermore, in each case we have

$$\operatorname{tr}(D_k) - \frac{3}{2}n = \operatorname{tr}(D_k) - \operatorname{tr}(I) - \frac{1}{2}n = \operatorname{tr}(D_k - I) - \frac{1}{2}n < 0$$

for $k \geq m$, since the matrix $D_k - I$ is negative (semi)definite. Therefore, if $i_{CN} = 0$ for all $k \geq m$, conditions (5.27) and (5.28) are valid if we set $k^* = m$.

If $i_{CN} = 0$ does not hold for all $k \geq m$, then the correction ϱI, with $\varrho \in (0, 1/2)$, is added to all the matrices D_k with $k \geq \bar{k}$ (see Algorithm 5.2, Step 3). Here $\bar{k}$ denotes the index $k \geq m$ of the iteration when $i_{CN} = 1$ occurred for the first time. The limited memory matrices S_k, U_k, R_k, and C_k do not contain information of the correction ϱI that may have been added to the matrix D_k at the previous iteration. Let us now denote by $\hat{D}_k$ the matrix formed with the limited memory matrices and by D_k the corrected matrix, that is, $D_k = \hat{D}_k + \varrho I$ for all $k \geq \bar{k}$ (since we suppose $i_{CN} = 1$).

Now, all the results given above are valid for $\hat{D}_k$ and $\hat{D}_{k+1}$. Since for all $k \geq \bar{k}$, we have $D_k = \hat{D}_k + \varrho I$ and we set $D_{k+1} = \hat{D}_{k+1} + \varrho I$, condition (5.27) is valid for all $k \geq k^* = \bar{k}$. Now, in each case we have

$$\begin{aligned}
\operatorname{tr}(D_k) - \frac{3}{2}n &= \operatorname{tr}(\hat{D}_k + \varrho I) - \operatorname{tr}(I) - \frac{1}{2}n \\
&= \operatorname{tr}(\hat{D}_k - I) + \operatorname{tr}(\varrho I) - \frac{1}{2}n \\
&< \operatorname{tr}(\hat{D}_k - I) + \frac{1}{2}n - \frac{1}{2}n \\
&\leq 0
\end{aligned}$$

for $k \geq \bar{k}$, since the matrix $\hat{D}_k - I$ is negative (semi)definite and $\varrho \in (0, 1/2)$. Therefore, conditions (5.27) and (5.28) are valid for all $k \geq k^*$ with

$$k^* = \max\{\bar{k}, m\},$$

where $\bar{k} = 1$ if $i_{CN} = 0$. □

LEMMA 5.2.9. *Suppose that the level set* $\{\, \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_1) \,\}$ *is bounded, the number of serious steps is finite, and the last serious step occurred at the iteration* $m - 1$. *Then, the point* $\boldsymbol{x}_m$ *is stationary for* f.

PROOF. From (5.8) [p. 56], (5.9) [p. 57], (5.10) [p. 57], Lemma 5.2.1, and

Lemma 5.2.8 we obtain

$$\begin{aligned} w_{k+1} &= \tilde{\boldsymbol{\xi}}_{k+1}^T D_{k+1} \tilde{\boldsymbol{\xi}}_{k+1} + 2\tilde{\beta}_{k+1} \\ &\leq \tilde{\boldsymbol{\xi}}_{k+1}^T D_k \tilde{\boldsymbol{\xi}}_{k+1} + 2\tilde{\beta}_{k+1} \\ &= \varphi(\lambda_1^k, \lambda_2^k, \lambda_3^k) \\ &\leq \varphi(0,0,1) \\ &= \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k \\ &= w_k \end{aligned} \tag{5.29}$$

for $k \geq k^*$ with k^* defined in Lemma 5.2.8.

Let us, for a while, denote $D_k = W_k^T W_k$. Thus, function φ (see (5.8), p. 56) can be given in the form

$$\varphi(\lambda_1^k, \lambda_2^k, \lambda_3^k) = \|\lambda_1^k W_k \boldsymbol{\xi}_m + \lambda_2^k W_k \boldsymbol{\xi}_{k+1} + \lambda_3^k W_k \tilde{\boldsymbol{\xi}}_k\|^2 + 2(\lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k).$$

From (5.29) we obtain the boundedness of the sequences (w_k), $(W_k\tilde{\boldsymbol{\xi}}_k)$, and $(\tilde{\beta}_k)$. Furthermore, Lemma 5.2.8 assures the boundedness of (D_k) and (W_k) and, by Lemma 5.2.6, we obtain the boundedness of $(\boldsymbol{y}_k)$, $(\boldsymbol{\xi}_k)$, and $(W_k\boldsymbol{\xi}_{k+1})$. Let us denote

$$\begin{aligned} &M = \sup\{\|W_k\boldsymbol{\xi}_{k+1}\|, \|W_k\tilde{\boldsymbol{\xi}}_k\|, \sqrt{\tilde{\beta}_k} \mid k \geq k^*\}, \quad \text{and} \\ &b = (1 - 2\varepsilon_R^I)/4M, \end{aligned}$$

and let us first assume that $w_k > \delta > 0$ for all $k \geq k^*$. From the fact that

$$\begin{aligned} &\min\{\varphi(\lambda_1,\lambda_2,\lambda_3) \mid \lambda_i \geq 0,\ i = 1,2,3,\ \sum_{i=1}^{3} \lambda_i = 1\} \\ &\leq \min\{\varphi(0,\lambda,(1-\lambda)) \mid \lambda \in [0,1]\} \end{aligned}$$

and (5.29) we obtain

$$w_{k+1} \leq \min\{\|\lambda W_k \boldsymbol{\xi}_{k+1} + (1-\lambda) W_k \tilde{\boldsymbol{\xi}}_k)\|^2 + 2(\lambda\beta_{k+1} + (1-\lambda)\tilde{\beta}_k) \mid \lambda \in [0,1]\}.$$

Since $w_k = \tilde{\boldsymbol{\xi}}_k^T D_k \tilde{\boldsymbol{\xi}}_k + 2\tilde{\beta}_k$ by Lemma 5.2.1, $\boldsymbol{d}_k = -D_k\tilde{\boldsymbol{\xi}}_k$ (see Algorithm 5.2, Step 2, p. 58), and

$$-\beta_{k+1} + \theta_k \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \geq -\varepsilon_R^k w_k$$

by (5.5) [p. 54] and due to the fact that we use a scaled direction vector $\theta_k\boldsymbol{d}_k$ in the line search (see Algorithm 5.2, Step 6), we have

$$\theta_k\beta_{k+1} - \theta_k \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \leq \beta_{k+1} - \theta_k \boldsymbol{d}_k^T \boldsymbol{\xi}_{k+1} \leq \varepsilon_R^k w_k = \theta_k \varepsilon_R^I w_k.$$

Now, we can use Lemma 5.2.7 with

$$\begin{aligned} &\boldsymbol{p} = W_k\tilde{\boldsymbol{\xi}}_k, \qquad \boldsymbol{g} = W_k\boldsymbol{\xi}_{k+1}, \qquad w = w_k, \\ &\alpha = \tilde{\beta}_k, \qquad \beta = \beta_{k+1}, \qquad c = \varepsilon_R^I \end{aligned}$$

to obtain

$$w_{k+1} \le w_k - (w_k b)^2 < w_k - (\delta b)^2$$

for $k \ge k^*$ and, thus, for a sufficiently large index k, we have a contradiction with the assumption $w_k > \delta$. Therefore, due to monotonicity of w_k for $k \ge k^*$, we obtain $w_k \to 0$, $\boldsymbol{x}_k \to \boldsymbol{x}_m$, and by Lemma 5.2.6 we have $\mathbf{0} \in \partial f(\boldsymbol{x}_m)$. □

THEOREM 5.2.10. *Suppose that the level set* $\{\, \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \le f(\boldsymbol{x}_1) \,\}$ *is bounded. Then, every accumulation point of the sequence* $(\boldsymbol{x}_k)$ *is stationary for* f.

PROOF. Let $\bar{\boldsymbol{x}}$ be a 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 5.2.9, we can restrict our consideration to the case where the number of serious steps (with $t_L^k > 0$) is infinite. We denote

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

Obviously, $\mathcal{K}'$ is infinite and $(\boldsymbol{x}_k)_{k\in\mathcal{K}'} \to \bar{\boldsymbol{x}}$. 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 the serious descent criterion (5.7) [p. 56]. Using the fact that $t_L^k \ge 0$ for all $k \ge 1$ and condition (5.7), we obtain

$$0 \le \varepsilon_L^k t_L^k w_k \le f_k - f_{k+1} \to 0 \qquad \text{for } k \ge 1. \tag{5.30}$$

If the set $\mathcal{K}_1 = \{\, k \in \mathcal{K}' \mid t_L^k \ge t_{min} \,\}$ 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 (5.30) and, thus, by Lemma 5.2.6 we have $\mathbf{0} \in \partial f(\bar{\boldsymbol{x}})$.

If the set $\mathcal{K}_1$ is finite, then the set $\mathcal{K}_2 = \{\, k \in \mathcal{K}' \mid \beta_{k+1} > \varepsilon_A^k w_k \,\}$ has to be infinite (see Algorithm 5.1, Step 2, p. 55). To the contrary, let us assume that

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

From (5.30), we have $(t_L^k)_{k\in\mathcal{K}_2} \to 0$ and Step 6 in Algorithm 5.2 [p. 58] implies

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

for all $k \ge 1$. Thus, we have $(\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|)_{k\in\mathcal{K}_2} \to 0$. By (5.6) [p. 54], (5.30), and the boundedness of $(\boldsymbol{\xi}_k)$ by Lemma 5.2.6, and since $\boldsymbol{y}_{k+1} = \boldsymbol{x}_{k+1}$ for serious steps, we obtain $(\beta_{k+1})_{k\in\mathcal{K}_2} \to 0$, which is in contradiction to

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

Therefore, there exists an infinite set $\mathcal{K}_3 \subset \mathcal{K}_2$ such that $(w_k)_{k\in\mathcal{K}_3} \to 0$, $(\boldsymbol{x}_k)_{k\in\mathcal{K}_3} \to \bar{\boldsymbol{x}}$, and $\mathbf{0} \in \partial f(\bar{\boldsymbol{x}})$ by Lemma 5.2.6. □

Note that, if we choose $\varepsilon > 0$, Algorithm 5.2 terminates in a finite number of steps, since $(w_k) \to 0$ and $(q_k) \to 0$ in the case the number of serious steps is finite (see the proof of Lemma 5.2.9), and either $(w_k)_{k\in\mathcal{K}_1} \to 0$ and $(q_k)_{k\in\mathcal{K}_1} \to 0$ or $(w_k)_{k\in\mathcal{K}_3} \to 0$ and $(q_k)_{k\in\mathcal{K}_3} \to 0$ in the case the number of serious steps is infinite (see the proof of Theorem 5.2.10).

5.3 Adaptive Version

In this section, we describe an adaptive modification of the limited memory bundle method. The idea of this adaptive version is that the maximum number of stored correction pairs may change during the computation. This means that we can start the optimization with a small number of stored correction pairs and then, when we are closer to the optimal point, the number of stored correction pairs may be increased until some upper limit 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.

The procedures used in the adaptive version are almost the same as in the basic version. In fact, we only have to change Step 5 of Algorithm 5.2. Let us denote by m_u the *upper limit for the maximum number of stored correction pairs* supplied by user ($3 \leq m_u$) and by m_c the current maximum number of stored correction pairs ($3 \leq m_c \leq m_u$). Now, Step 5 of Algorithm 5.2 is given in the form

Step 5': (*Stopping criterion.*) Calculate w_k and q_k by (5.11) and (5.12), respectively. If $w_k \leq \varepsilon$ and $q_k \leq \varepsilon$, then stop with $\boldsymbol{x}_k$ as the final solution. Otherwise, if $w_k \leq 10^3\varepsilon$ and $m_c < m_u$, set $m_c = m_c + 1$.

The scaling factor 10^3 above has been chosen experimentally. Note that with $m_c = m_u$ this adaptive version reduces to the basic version.

The convergence analysis of the adaptive limited memory bundle method is the same as that given in Section 5.2 with m_c replaced by m_u.

5.4 Simple Scaling of Updates

In the basic version we always use the scaling parameter ϑ_k calculated by (5.15) if the previous step was a serious step, and the value $\vartheta_k = 1$, otherwise. In this section, we describe some other possibilities to scale the limited memory variable metric updates. As with the basic version, we concentrate on the simple scaling strategies, where the scaling matrix $D_k^{(0)}$ is supposed to be a scalar multiple of the identity matrix (that is, $D_k^{(0)} = \vartheta_k I$). The scaling strategies described in this section are similar to those developed for standard variable metric updates (see, e.g., [52, 68, 75]).

We use two different scaling parameters ϑ_k with the limited memory BFGS update (5.14). The first is that defined by the formula (5.15), that is,

$$\vartheta_k = \frac{\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1}}{\boldsymbol{u}_{k-1}^T \boldsymbol{u}_{k-1}},$$

and the second, which has been derived from the heuristic scaling parameter for the standard BFGS update (see, e.g., [48]), is given by

$$\vartheta_k = \frac{\boldsymbol{s}_{k-1}^T \boldsymbol{s}_{k-1}}{\boldsymbol{u}_{k-1}^T \boldsymbol{s}_{k-1}}. \tag{5.31}$$

Note that formulae (5.15) and (5.31) are unsuitable for scaling the limited memory SR1 update (5.16), since they might make the update formula undefined.

We have also studied some different scaling parameters for the limited memory SR1 update. However, our numerical experiments have shown that the scaling of the SR1 update in the limited memory bundle method either has no significant effect on the results or, in the worst case, makes the method numerically unstable. Thus, similarly to the basic version, we use the value $\vartheta_k = 1$ for every k if the limited memory SR1 update is used.

The scaling parameters (5.15) and (5.31) are combined with the value $\vartheta_k = 1$ using some scaling strategy. The following simple scaling strategies have been used with the limited memory bundle method:

- *No scaling* (NS). We set $\vartheta_k = 1$ at every iteration.
- *Preliminary scaling* (PS) [75]. The scaling parameter ϑ_k is calculated by (5.15) or (5.31) after the number of current correction pairs m_k is reset to zero. Otherwise, we set $\vartheta_k = 1$. More precisely, if $m_k = 0$, we set $\vartheta_k = \Gamma_l$ or $\vartheta_k = \Gamma_u$ if the scaling parameter ϑ_k does not lie in the interval $[\Gamma_l, \Gamma_u]$ for given $0 < \Gamma_l < \Gamma_u$. The number of stored correction pairs is reset to zero at the first iteration (that is, we scale at iteration two) and in the case the direction vector $\boldsymbol{d}_k$ is almost orthogonal to the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ (see Algorithm 5.2, Step 4).
- *Scaling at every iteration* (AS). The scaling parameter ϑ_k is calculated by (5.15) or (5.31) at every iteration. Similarly to (PS), we set $\vartheta_k = \Gamma_l$ or $\vartheta_k = \Gamma_u$ if $m_k = 0$ and the scaling parameter ϑ_k does not lie in the interval $[\Gamma_l, \Gamma_u]$.
- *Interval scaling* (IS) [52]. The scaling parameter ϑ_k is calculated by (5.15) or (5.31) at every iteration. If $m_k = 0$, we proceed as in (PS) or (AS). In any other iteration (that is, an iteration with $m_k \neq 0$), if the scaling parameter ϑ_k does not lie in the interval $[\vartheta_l, \vartheta_u]$, we set $\vartheta_k = 1$. Here, $0 < \vartheta_l < \vartheta_u$ and $\vartheta_u > 2$ are the given lower and upper bounds for the scaling parameter ϑ_k.

The values Γ_l and Γ_u in (PS), (AS), and (IS) serve as a safeguard. Similarly to [52], we have used the values $\Gamma_l = 0.01$ and $\Gamma_u = 100$. In interval scaling (IS), the values $\vartheta_l = 0.6$ and $\vartheta_u = 6.0$ for the scaling parameter (5.15) and $\vartheta_l = 0.5$ and $\vartheta_u = 5.0$ for the scaling parameter (5.31) have been chosen experimentally.

The simple scaling of updates has no influence on the global convergence of the limited memory bundle method (especially, since we do not scale the SR1 update).

5.5 L-BFGS Bundle Method

In practice, the limited memory SR1 update is quite frequently skipped due to condition (5.19) required for the positive definiteness of the update and the additional

testing procedure, which is used to guarantee that condition (5.18) is valid in the consecutive null steps. Therefore, we have developed a version where SR1 update is not used at all. In this version, we use the limited memory BFGS update (given in Algorithm 5.3) after a serious step if the positive definite condition (5.17) is valid and the update is simply skipped (that is, $D_k = D_{k-1}$) both if condition (5.17) is not satisfied and after each null step.

The procedures used in the L-BFGS bundle method are very similar to the basic Algorithm 5.2. The only differences are in Steps 2 and 7, where the limited memory matrix D_k is used. Moreover, we use the interval scaling strategy (IS) (see Section 5.4) in the calculation of the scaling parameter ϑ_k, since here this approach has been found to be more advantageous than scaling at every iteration.

Now, the L-BFGS bundle method can be obtained from Algorithm 5.2 by replacing Steps 2 and 7 with the following.

Step 2': (*Direction finding.*) If $m = k$ compute

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

by the limited memory BFGS update (using Algorithm 5.3). Otherwise, skip the updates and calculate the search direction $\boldsymbol{d}_k = -D_{k-1}\tilde{\boldsymbol{\xi}}_k$ using Algorithm 5.5. Note that $\boldsymbol{d}_1 = \boldsymbol{\xi}_1$.

Step 7': (*Aggregation.*) Determine multipliers $\lambda_i^k \geq 0$ for all $i \in \{1, 2, 3\}$, $\sum_{i=1}^{3} \lambda_i^k = 1$ that minimize the function (5.8), where $D_k = D_k + \varrho I$, if $i_C = 1$. Set

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

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

Since we use the limited memory approach, we have to recalculate the matrix $D_k = D_{k-1}$ also when the update is skipped. However, in that case, we do not store the new correction pairs but the same correction matrices are used as in the previous iteration. Moreover, after any null step, the search direction $\boldsymbol{d}_k$ has to be calculated using the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$, that is, $\boldsymbol{d}_k = -D_k \tilde{\boldsymbol{\xi}}_k$. Thus, we can not simply utilize Algorithm 5.3 and skip the updates but we have to use a special procedure to calculate the search direction. We now give an algorithm for calculation of the matrix D_k and for direction finding using the limited memory BFGS update. This algorithm is used with the L-BFGS bundle method if the previous step was a null step.

ALGORITHM 5.5. *(BFGS Skipping and Direction Finding).*

Data: Suppose that the number of current correction pairs is m_{k-1}. Suppose that we have the current aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$, the $n \times m_{k-1}$ matrices S_{k-1} and U_{k-1}, the $m_{k-1} \times m_{k-1}$ matrices R_{k-1}, $U_{k-1}^T U_{k-1}$, and C_{k-1}, and the previous scaling parameter ϑ_{k-1} available.

Step 1: (*Skipping the updates.*) Set $S_k = S_{k-1}$, $U_k = U_{k-1}$, $R_k = R_{k-1}$, $U_k^T U_k = U_{k-1}^T U_{k-1}$, $C_k = C_{k-1}$, $\vartheta_k = \vartheta_{k-1}$, and $m_k = m_{k-1}$.

Step 2: Compute m_k-vectors $S_k^T \tilde{\boldsymbol{\xi}}_k$ and $U_k^T \tilde{\boldsymbol{\xi}}_k$.

Step 3: (*Intermediate values.*) Solve the vectors $\boldsymbol{p}_1 \in \mathbb{R}^{m_k}$ and $\boldsymbol{p}_2 \in \mathbb{R}^{m_k}$ from the linear equations

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

Step 4: (*Search direction.*) Compute

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

The convergence analysis of the L-BFGS bundle method is very similar to that of the basic version of the method (see Section 5.2). In fact, all the results in Lemmas and Theorems 5.2.1 – 5.2.7 are valid also for the L-BFGS bundle method expect that, for the L-BFGS bundle method, we do not have property (5.23) given in Lemma 5.2.1. With the basic version this property is used to guarantee that condition (5.18) is valid in the case of consecutive null steps (see Lemma 5.2.8). However, we next prove that condition (5.18) is valid with the L-BFGS bundle method due to skipping of updates and, thus, property (5.23) is not required.

LEMMA 5.5.1. *Suppose that the level set* $\{ \boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_1) \}$ *is bounded, the number of serious steps is finite, and the last serious step occurred at the iteration* $m-1$. *Then there exists a number* $k^* \geq m$, *such that*

$$\tilde{\boldsymbol{\xi}}_{k+1}^T D_{k+1} \tilde{\boldsymbol{\xi}}_{k+1} = \tilde{\boldsymbol{\xi}}_{k+1}^T D_k \tilde{\boldsymbol{\xi}}_{k+1} \qquad \textit{and} \tag{5.32}$$

$$\operatorname{tr}(D_{k+1}) = \operatorname{tr}(D_k) \tag{5.33}$$

for all $k \geq k^*$.

PROOF. Suppose first that $i_{CN} = 0$ for all $k \geq m$, that is, the correction ϱI (see Algorithm 5.2, Step 3) is not added to any matrix D_k with $k \geq m$. Now, due to skipping of updates we have $D_{k+1} = D_k$ and, obviously, conditions (5.32) and (5.33) are valid if we set $k^* = m$.

If equality $i_{CN} = 0$ does not hold for all $k \geq m$, then the correction ϱI, with $\varrho \in (0, 1/2)$, is added to all the matrices D_k with $k \geq \bar{k}$ (see Algorithm 5.2, Step 3). Here $\bar{k}$ denotes the index $k \geq m$ of the iteration when $i_{CN} = 1$ occurred for the first time. Now, for all $k \geq \bar{k}$ we have $D_k = D_{\bar{k}} = D_{\bar{k}-1} + \varrho I$. Therefore, conditions (5.32) and (5.33) are valid for all $k \geq k^*$ with

$$k^* = \max \{\bar{k}, m\},$$

where $\bar{k} = 1$ if $i_{CN} = 0$. □

Now, if we replace Lemma 5.2.8 by Lemma 5.5.1, we see that the sequence (w_k) is nonincreasing in the consecutive null steps and all the remaining results (that is, Lemma 5.2.9 and Theorem 5.2.10) of Section 5.2 are valid also for the L-BFGS bundle method. Therefore, similarly to the basic version, the L-BFGS bundle method either terminates at a stationary point of the objective function f or generates an infinite sequence $(\boldsymbol{x}_k)$ for which accumulation points are stationary for f. Moreover, if we choose $\varepsilon > 0$, the L-BFGS bundle method terminates in a finite number of steps.

6 NUMERICAL EXPERIMENTS

In order to get some information of how the different versions of the limited memory bundle method work in practice we compare them to the proximal bundle method and to the limited memory BFGS method. A more extensive numerical analysis concerning the performance of the limited memory bundle method (the basic version) and some other existing bundle methods can be found in [27]. In this chapter, we first introduce the software tested and the testing environment. Then, we give the results of the numerical experiments and draw some conclusions.

6.1 Software Tested and Testing Environment

The experiments were performed in a SGI Origin 2000/128 supercomputer (MIPS R12000, 600 Mflop/s/processor). The algorithms were implemented in Fortran77 with double-precision arithmetic. The solvers used in our experiments are presented in Table 1 with abbreviated names and references.

TABLE 1: Tested pieces of software.

Software	Author(s)	Method	Reference
`PBNCGC`	Mäkelä	Proximal bundle	[61, 63]
`L-BFGS`	Nocedal	Limited memory BFGS	[46, 65]
`LMBM`	Haarala	Limited memory bundle	
`LBB`	Haarala	L-BFGS bundle	
`ALMBM`	Haarala	Adaptive limited memory bundle	

Proximal bundle method. The tested proximal bundle algorithm `PBNCGC` (version 2.0) is from the software package `NSOLIB` (NonSmooth Optimization LIBrary, see [59]). The solver utilizes the subgradient aggregation strategy of [33] and the quadratic direction finding problem (4.6) is solved by the `PLQDF1` subroutine, which is a realization of the dual range space quadratic programming method for minmax

approximation with linear constraints [47]. For a detailed description of the solver see [61, 63].

Limited memory BFGS Method. `L-BFGS` is a limited memory quasi-Newton algorithm for smooth large-scale unconstrained optimization. The solver has been developed at the Optimization Technology Center as a joint venture of the Argonne National Laboratory and the Northwestern University. The basic ideas of the smooth limited memory BFGS method are given in Section 3.2 and for a detailed description of the solver we refer to [46, 65].

Limited Memory Bundle Methods. The limited memory bundle algorithm `LMBM`, the L-BFGS bundle algorithm `LBB`, and the adaptive limited memory bundle algorithm `ALMBM` are our new solvers for nonsmooth large-scale unconstrained optimization. The methods are described in detail in Chapter 5. The implementations of the solvers are our own expect the selection of the initial step size, where we have used the original pieces by Lukšan and Vlček [51].

In this chapter we first compare the limited memory bundle method `LMBM` to the proximal bundle method `PBNCGC` and to the limited memory BFGS method `L-BFGS`. We use the solver `PBNCGC` as a benchmark since the proximal bundle method is the most frequently used bundle method in nonsmooth optimization (see, e.g., [63]). On the other hand, `PBNCGC` has not been developed for large-scale optimization and, therefore, we compare the new method also to the smooth large-scale optimization solver `L-BFGS`. Thus, the solvers were first tested with 22 smooth minimization problems given in [55]. However, we removed three problems (problems 1, 2, and 10 in [55]) where the solvers converged to different local minima.

Next the solvers `LMBM` and `PBNCGC` were tested with 10 academic nonsmooth minimization problems described in [27] (for details of the problems, see Appendix A). We tried to solve these nonsmooth problems also with the smooth solver `L-BFGS` but it failed in almost all the cases.

Then, we compared the different scaling strategies of the basic version `LMBM` (see Section 5.4) and the different versions `LMBM`, `ALMBM`, and `LBB` (see Sections 5.1, 5.3, and 5.5) by using 20 academic nonsmooth minimization problems (for details of the problems, see Appendix A).

Finally, we tested the different versions of the limited memory bundle method and the proximal bundle solver `PBNCGC` with two practical nonsmooth image restoration problems [30, 31].

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

- the problem was solved with the desired accuracy. That is, $w_k \leq \varepsilon$, where w_k is defined by (4.8) for `PBNCGC`, by (5.11) for `LMBM` and its modifications (note that also $q_k \leq \varepsilon$, see (5.12)), and $w_k = \|\nabla f(\boldsymbol{x}_k)\| / \max\{1, \|\boldsymbol{x}_k\|\}$ for `L-BFGS`.

In addition, for `LMBM`, `ALMBM`, and `LBB` we say that the optimization terminated successfully if

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

In proportion, we accepted the result for `PBNCGC` if

- $|w_{k+1} - w_k| \leq 1.0 \cdot 10^{-12}$ in 10 subsequent iterations and the result obtained was less than two significant digits greater than the desired accuracy of the solution.

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

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

Otherwise, we say that the optimization failed.

The test results are analyzed using the performance profiles introduced in [14]. As performance measures, we use computation times and numbers of function evaluations. In the following graphs $\rho_s(\tau)$ denotes the logarithmic performance profile

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

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

There are two important facts to be kept in mind to have a good interpretation of the performance profiles. The value of $\rho_s(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the corresponding solver s is the best and the value of $\rho_s(\tau)$ at the rightmost abscissa is the percentage of test problems that the corresponding solver s can solve (this does not depend on the measured performance). Finally, the relative efficiency and reliability of each solver can be directly seen from the performance profiles: the higher is the particular curve the better is the corresponding solver. For more information of performance profiles, see [14].

6.2 Numerical Results

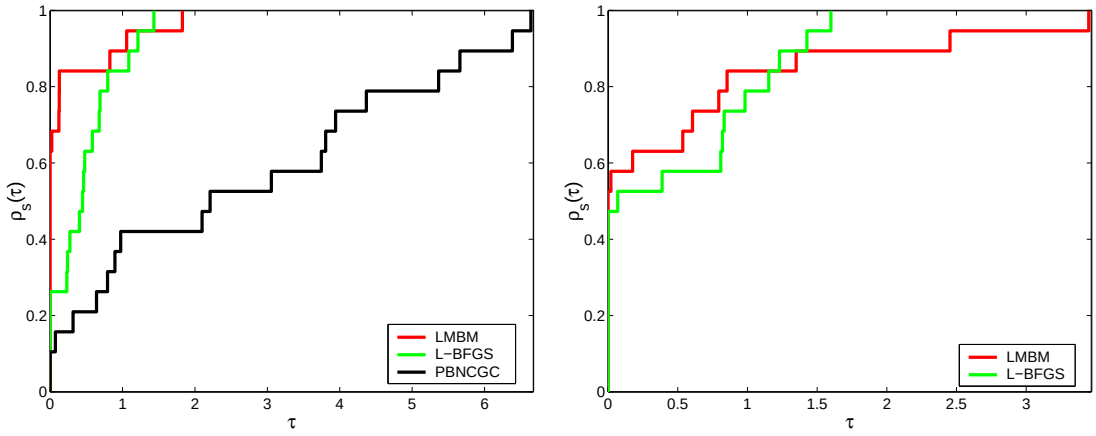
In this section, we analyze the results obtained in the numerical experiments. First, in Subsection 6.2.1, we compare the limited memory bundle solver `LMBM` to the proximal bundle solver `PBNCGC` and to the limited memory BFGS solver `L-BFGS` using some smooth test problems. After that, in Subsection 6.2.2, we first compare `LMBM` to `PBNCGC` and then, we bring into comparison the different scaling strategies and the different versions of the basic method by using some academic nonsmooth minimization problems. Finally, in Subsection 6.2.3, we analyze the performance of the new solvers when applied to nonsmooth practical applications.

6.2.1 Smooth Test Problems

The solvers PBNCGC, L-BFGS, and the basic version of the limited memory bundle solver LMBM were first tested with 19 smooth problems with 1000 variables and, in case of the limited memory solvers L-BFGS and LMBM, also with 10 000 variables (for details of the problems, see [55]).

We tested the bundle solvers PBNCGC and LMBM with a relatively small size of the bundle, that is, $m_\xi = 10$. For the limited memory solvers L-BFGS and LMBM, the maximum number of stored correction pairs (m_c) was set to 7 and for LMBM the maximum number of additional interpolations used at each iteration was set to 200 (default value). As a stopping parameter, we used $\varepsilon = 10^{-5}$ in all the cases. For the bundle solvers PBNCGC and LMBM the value of the distance measure parameter γ (see (4.3) and (5.6)), which depends on the convexity of the objective function, was chosen experimentally individual to the each problem. That is, we minimized all the problems with four different values of the distance measure parameter and selected the best results to be reported (we first checked the accuracy of the result and then the computation time elapsed). The tested values of the distance measure parameter were $\gamma = 0.0$, 0.25, 0.5, and 0.9. Correspondingly, for L-BFGS we chose the line search parameter ε_L (see (3.2)). The tested values for ε_L were 10^{-4}, 10^{-3}, 10^{-2}, and 10^{-1}. Otherwise, the default parameters of the solvers were used.

The results of the smooth experiments are summarized in Figure 2 where we compare the computation times of the solvers. In Figure 2(a) we give the performance profile for each of the solvers with 1000 variables and in Figure 2(b) for the limited memory solvers LMBM and L-BFGS with 10 000 variables.



(a) $n = 1000$ (b) $n = 10\ 000$

FIGURE 2: Results for smooth problems.

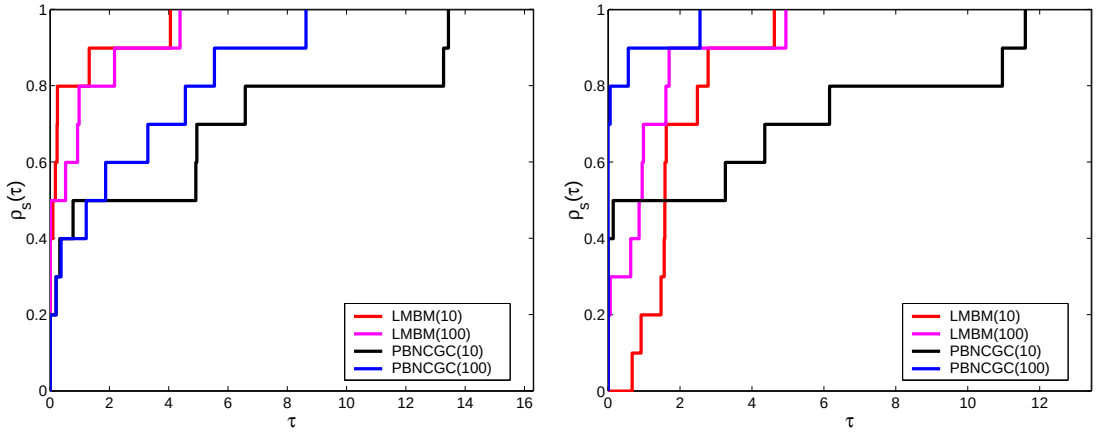
To sum up, the new limited memory bundle solver LMBM was usually the most efficient method tested. To be more precise, it was the most efficient on 63% of the problems with $n = 1000$ and on 53% of the problems with $n = 10\ 000$

(see Figure 2). Moreover, the new solver `LMBM` was almost twice as fast as the limited memory BFGS solver `L-BFGS`, which has been developed for smooth large-scale minimization. Nevertheless, the limited memory BFGS solver `L-BFGS` was very efficient with these large-scale smooth problems as well. However, it usually needed more iterations and function evaluations and, thus, also computation time than `LMBM`. The proximal bundle solver `PBNCGC` was computationally inefficient with these large-scale problems (see Figure 2(a)).

6.2.2 Nonsmooth Test Problems

The solvers `LMBM` and `PBNCGC` were tested with 10 nonsmooth academic minimization problems described in [27] (for details of the problems, see the first 10 problems in Appendix A). Half of these problems were convex and the other half were nonconvex. The numbers of variables used were 50, 200, and 1000, and the solvers were tested with relatively small sizes of the bundles, that is, $m_\xi = 10$ and $m_\xi = 100$. In what follows, we denote these different modifications by `LMBM(10)`, `LMBM(100)`, `PBNCGC(10)`, and `PBNCGC(100)`. Otherwise, the parameter values were chosen similarly to the smooth problems.

The results of the nonsmooth experiments are summarized in Figures 3 – 5. In these figures, we give the performance profiles of the solvers with different numbers of variables. In Figures (a) we compare the computation times of the solvers and in Figures (b) the numbers of function evaluations required.



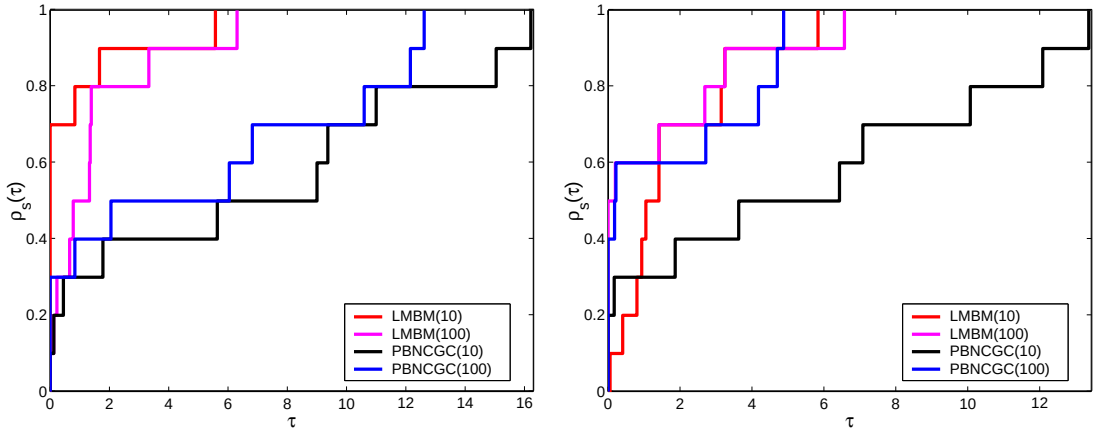
(a) CPU time

(b) Function evaluations

FIGURE 3: Results for nonsmooth problems with 50 variables.

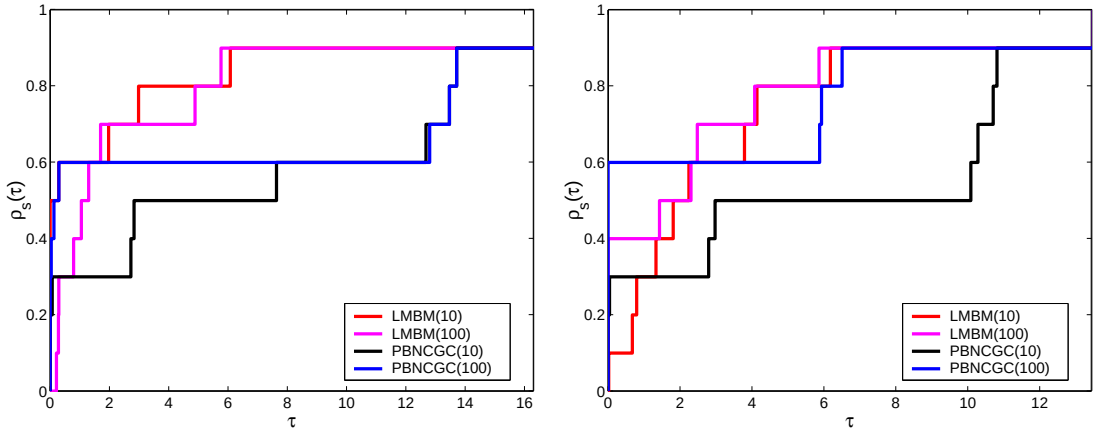
In Figures (a) we see that `LMBM` with the small size of the bundle was usually the most efficient solver with all the tested numbers of variables, although, the differences in the average computation times were not substantial in the small-scale cases (with $n = 50$).

With larger problems `LMBM(10)` was the most efficient solver on 70% of the problems with $n = 200$, and on 50% of the problems with $n = 1000$ (see Figures 4(a) and 5(a)). Notice that with 1000 variables all the tested solvers failed to solve 10% of the problems (see Figure 5(a)).



(a) CPU time (b) Function evaluations

FIGURE 4: Results for nonsmooth problems with 200 variables.



(a) CPU time (b) Function evaluations

FIGURE 5: Results for nonsmooth problems with 1000 variables.

Besides being the most efficient solver tested on the generality of the problems, LMBM(10) also solved the rest of the problems really fast while, with the proximal bundle solver PBNCGC (with both sizes of bundles), there was a great variation in the computation times of different problems, especially, in problems with 1000 variables. Also LMBM(100) was very efficient with these large-scale problems, although, it usually needed slightly more computation time than LMBM(10) and, thus, with 1000 variables it was never the most efficient solver tested (see Figure 5(a)). Note, however, that LMBM(100) was usually faster than the proximal bundle solver PBNCGC but, since this happened in the same problems as with the solver LMBM(10) and the solver LMBM(10) was faster than LMBM(100), this can not be seen directly in Figure 5(a).

In most cases, the solver PBNCGC(100) used the smallest numbers of iterations and function evaluations (see Figures (b)). However, as with the computation times, there was a great variation in the number of function evaluations required for different problems and, indeed, on the average, the solver LMBM(100) required the least function evaluations per problem (with $n = 1000$, the average number of function evaluations used with LMBM(100) was 3243 while, with PBNCGC(100), it was 5288). With the solver PBNCGC the required number of function evaluations was significantly smaller when the size of the bundle was larger. This was usually true also for the solver LMBM but with LMBM the difference was not so substantial. With LMBM the decrease in the number of function evaluations is due to the fact that the selection of the initial step size is more accurate when a larger bundle is used. On the other hand, each individual iteration was more costly when the size of the bundle was increased. In practice, this means that for problems with expensive objective function and subgradient evaluations, it is better to use larger bundles and, thus, fewer iterations and function evaluations.

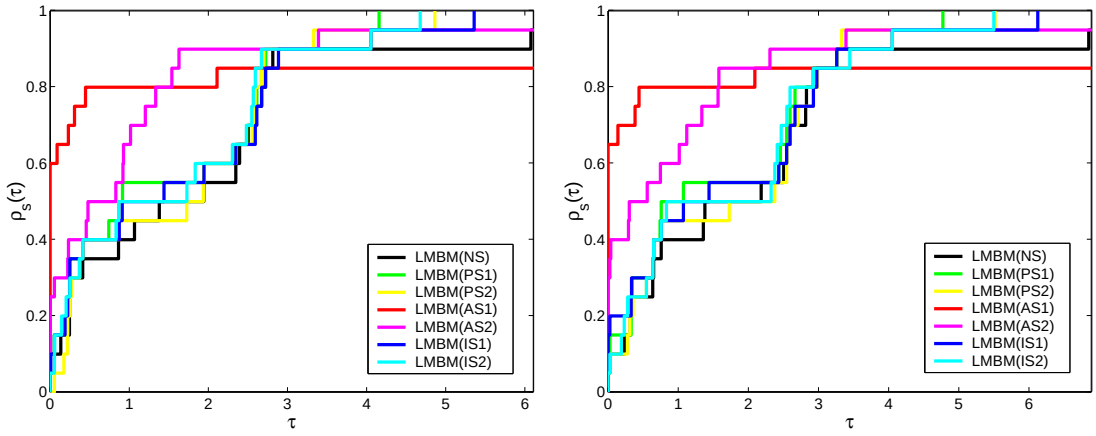
We conclude from these experiments that the limited memory bundle method was very efficient for large-scale nonsmooth optimization. For problems with 1000 variables, LMBM was on the average about 40 times faster than the proximal bundle solver PBNCGC. Moreover, LMBM found the (local) minimum in quite a reliable way for both convex and nonconvex optimization problems (see Figures 3 – 5).

Different scaling of updates. Next, we compared the different scaling strategies and parameters of the limited memory variable metric updates. In order to do this, we combined the basic limited memory bundle solver LMBM with the different scaling strategies (see Section 5.4). The parameter values of the solver LMBM were chosen similarly to those used with the smooth problems. That is, the size of the bundle (m_ξ) was set to 10, the maximum number of stored correction pairs (m_c) was set to 7, the distance measure parameter γ was chosen experimentally between four values, the stopping parameter ε was equal to 10^{-5}, and, otherwise, the default parameters of the solver were used. The experiment to be described was done by using a set of 20 nonsmooth academic minimization problems with 1000 variables (for details of the problems, see Appendix A).

We have used the scaling strategies described in Section 5.4 and the two different scaling parameters ϑ_k defined in (5.15) and (5.31). In what follows, we

use the abbreviations (NS) for no scaling, (PS) for preliminary scaling, (AS) for scaling at every iteration, and (IS) for interval scaling. In addition, we denote by "1" the scaling parameter defined in (5.15) and by "2" the scaling parameter defined in (5.31). Note that with this notation, the basic limited memory bundle solver is to be denoted by `LMBM(AS1)`.

The results of the experiments are summarized in Figure 6. In this figure, we give the performance profiles of the solver `LMBM` with the different scaling strategies. In Figure 6(a) we compare the computation times and in Figure 6(b) the numbers of function evaluations required.



(a) CPU time

(b) Function evaluations

FIGURE 6: Results with different scaling of updates.

Scaling at every iteration with the scaling parameter (5.15) (that is, `LMBM(AS1)`) was clearly the most efficient scaling strategy tested (on 60% of problems, see Figure 6(a)). Unfortunately, it was also the most unreliably, and the solver failed to solve three of the problems when this scaling strategy was used. Note, however, that all these failures can be prevented by a suitable choice of the parameters.

Scaling at every iteration with the scaling parameter (5.31) (that is, `LMBM(AS2)`) was the second best choice of the different scaling strategies when comparing the computation times (see Figure 6(a)). Moreover, the solver `LMBM(AS2)` failed to solve only one problem in the test set. However, the average computation time elapsed with the scaling strategy (AS2) was about five times longer than that with the scaling strategy (AS1).

All the other scaling strategies tested behaved quite similarly when compared to each other. With the (NS) strategy the solver failed to solve one problem but in all the other cases, the solver succeeded to solve all the problems in the test set. Unfortunately, the average computation times elapsed with these scaling strategies increased approximately sixfold to that used with (AS1). Note, however, that if we ignore the problem (12) (see Appendix A), which was the most time-consuming of the problems tested with all the scaling strategies but (AS1), the computation times elapsed with the different scaling strategies (NS), (PS1), (PS2), (AS2), (IS1),

and (IS2) are all only about twice as long as the computation time elapsed with the scaling strategy (AS1).

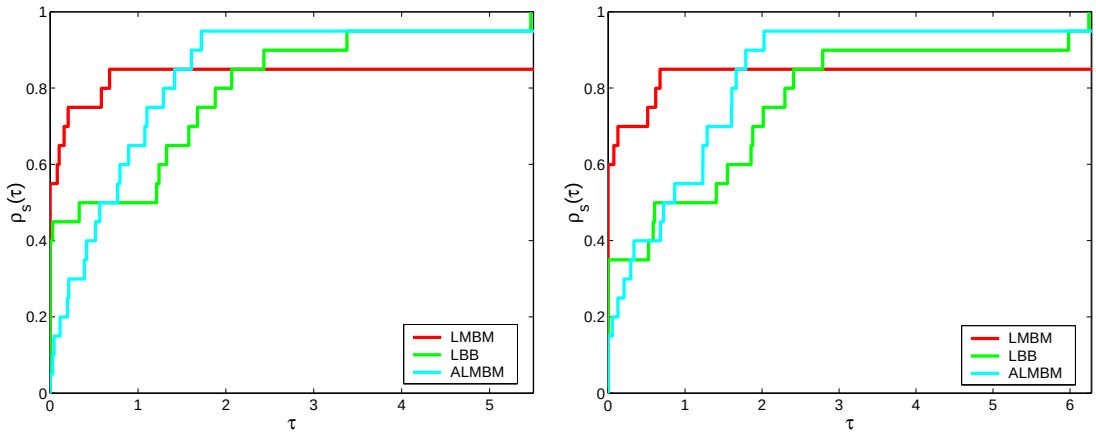
Naturally, the differences between the computation times elapsed with the different scaling strategies correspond to the differences between the numbers of function evaluations required (see Figure 6).

We conclude from these experiments that the different scaling of the limited memory variable metric updates may, in some cases, lead to more accurate results but it does not, in general, improve the behavior of the method. Moreover, the same kind of improvement on the accuracy properties may be obtained with a careful choice of the parameters. On the other hand, the internal parameters used in these experiments have been originally chosen for the basic version of the solver (that is, `LMBM(AS1)`). Thus, the efficiency of the solver with the different scaling strategies might increase if we chose these parameters individually for each scaling strategies.

Different versions. Finally, we compared the different versions of the limited memory bundle method. As before, the experiment was done by using a set of 20 nonsmooth minimization problems with 1000 variables (for details of the problems, see Appendix A).

Let us first recall that `LMBM` is the basic limited memory bundle method, `LBB` is the limited memory BFGS bundle method and `ALMBM` is the adaptive version of the limited memory bundle method. With the adaptive version `ALMBM`, we have used the initial maximum number of stored correction pairs (m_c) equal to 7 (the same as that with the basic version `LMBM`) and the upper limit for the maximum number of stored correction pairs (m_u) equal to 50. Otherwise, the parameter values for the solvers were chosen similarly to those used with different scaling of updates.

The results of the experiments are summarized in Figure 7, where we, as before, first compare the computation times of the solvers (in Figure 7(a)) and then the numbers of function evaluations required (in Figure 7(b)).



(a) CPU time (b) Function evaluations

FIGURE 7: Results with different versions.

Again the basic version LMBM was usually the most efficient one (that is, on 55% of the problems) but also the most unreliable of the solvers tested (see Figure 7(a)). The adaptive version ALMBM succeed to solve two of the three problems where the basic version failed but, then, it lost slightly on the efficiency. The L-BFGS bundle solver LBB succeeded to solve all the problems. However, with LBB, there was quite a big variation in the computation times of different problems and, on the average, LBB was the most time-consuming of the different versions.

Similarly to the different scaling strategies tested, the differences between the computation times of the versions LMBM, LBB, and ALMBM correspond to the differences between the required number of function evaluations (see Figure 7). For example, the L-BFGS bundle solver LBB usually needed about twice as much function evaluations than the basic version LMBM and, thus, it also needed approximately twice the time to solve the problems. With the solver LBB this increased number of function evaluations is probably caused by the update skipping after every null step. This means that the advantage of the most recent correction pairs is not taken into account in the best possible way and, thus, the search direction obtained may not be as good as that with the basic version.

Based on this part of the experiments we can conclude that the different versions of the limited memory variable metric method are quite comparable to each other. The usage of the adaptive version ALMBM or the L-BFGS bundle version LBB may increase the computation time needed when compared to the basic version LMBM but, on the other hand, it may also lead to more accurate results.

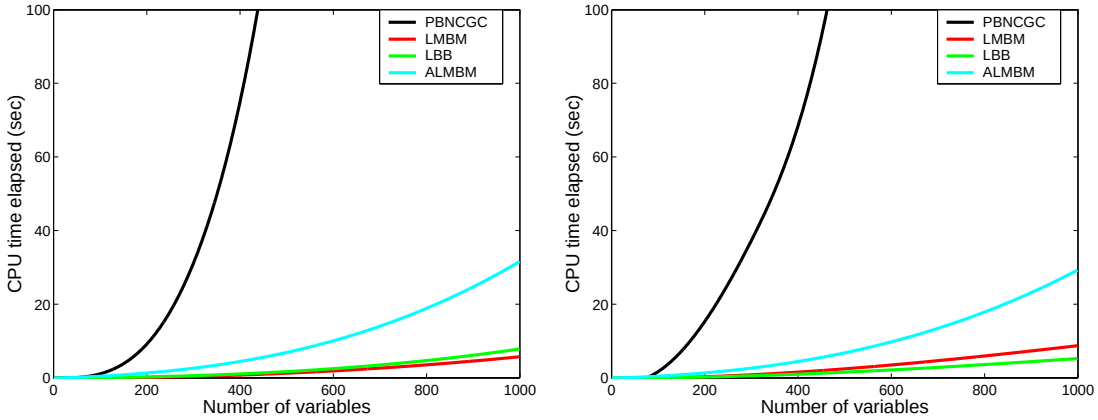
6.2.3 Image Restoration Problems

Finally, we tested the different versions of the limited memory bundle method and the proximal bundle solver PBNCGC with two convex image restoration problems [30, 31], which are typical nonsmooth large-scale optimization problems arising in optimal control applications. In what follows, we denote by (1) the problem described in [31] and by (2) the problem described in [30]. Both of the problems are so-called noise reduction problems. That is, it is assumed that the observed images are degraded by a random noise. The noise reduction problems are formulated as minimization problems consisting of a least squares fit and a nonsmooth bounded variational type regularization term (for more details of the formulations, see [30, 31]).

The solvers were tested with relatively small sizes of the bundles, that is, $m_\xi = 10$ for the different versions of the limited memory bundle method and $m_\xi = 100$ for the proximal bundle solver PBNCGC (since the experiments in the previous subsection have shown that a larger bundle works better with PBNCGC). The stopping parameter $\varepsilon = 10^{-4}$ was used with all the solvers and the value of the distance measure parameter γ was set to zero (since the objective functions are convex). Otherwise, the default parameters of the solvers were used.

In Figure 8, we give the computation times elapsed for the problems with different number of variables. Furthermore, we give some more specified results for the problems with 100, 300, and 1000 variables in Tables 2 and 3, where Ni and Nf denote the numbers of iterations and function evaluations used, respectively, and

f denotes the value of the objective function at termination. In addition to the results obtained in our experiment, we give (in Table 2) the reported final values of the objective function for problem (1) obtained with the special active set method (ACS) [31]. The active set method has not been applied for problem (2) in [30] and, thus, in Table 3, we only give the results obtained in our experiment.



(a) Problem (1)

(b) Problem (2)

FIGURE 8: CPU times elapsed for the image restoration problems.

TABLE 2: Results for the image restoration problem (1).

Solver/n	100		300		1000	
	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
ACS	–	0.7981	–	2.7586	–	9.8950
PBNCGC	186/187	0.7976	1342/1343	2.7581	15421/15422	9.7617
LMBM	442/669	0.7979	1707/1919	2.7666	6343/6373	9.8152
LBB	863/3714	0.7981	1819/6492	2.7682	6848/18721	9.9196
ALMBM	736/4953	0.7976	2601/5762	2.7602	11812/13326	9.7694

TABLE 3: Results for the image restoration problem (2).

Solver/n	100		300		1000	
	Ni/Nf	f	Ni/Nf	f	Ni/Nf	f
PBNCGC	249/250	0.5973	1430/1431	2.2517	16172/16173	7.9571
LMBM	764/1062	0.5974	1476/1597	2.2603	10028/10073	7.9958
LBB	639/2833	0.5976	1713/6205	2.2635	4692/12305	8.0823
ALMBM	772/4721	0.5972	4814/10531	2.2536	10981/12388	7.9771

Based on the numerical results, we can conclude the superiority of the limited memory bundle solver LMBM and its different versions LBB and ALMBM when comparing the computation times (see Figure 8), because in all the cases, they used significantly less CPU time than the proximal bundle solver PBNCGC. With large number of variables, they also usually needed less iterations and function evaluations than PBNCGC (see Tables 2 and 3). However, the accuracies of the new solvers were somewhat disappointing: The minima of the objective function found with these solvers were usually a little bit greater than those obtained with the proximal bundle solver PBNCGC (especially, in case of LBB, see Tables 2 and 3). Nevertheless, in most cases, the results obtained for problem (1) with LMBM and ALMBM were smaller than those obtained with the active-set method ACS used in [31] (see Table 2), and both visually and with respect to the reconstruction error [30, 31] all the results of LMBM, ALMBM, and LBB were comparable to the results of the proximal bundle method PBNCGC (for both the problems). In fact, in all large-scale cases the reconstruction errors obtained with the different versions of the limited memory bundle method were smaller than those obtained with the proximal bundle method PBNCGC.

Note that the behavior of the methods with this image restoration problem differs from the academic problems. Now, the L-BFGS bundle solver LBB was the most unreliable of the different versions while, with the academic problems it succeeded to solve all the problems with the desired accuracy. Furthermore, with 1000 variables, the adaptive version ALMBM used about four times more computation time than the basic version LMBM while, before, this difference was only 1.6. On the other hand, with 1000 variables even this adaptive version ALMBM, which was the most time-consuming but also the most reliable of the different versions, was about 50 times faster than the proximal bundle solver PBNCGC (see Figure 8).

7 CONCLUSIONS

In this thesis, we have considered the optimization of nonsmooth but locally Lipschitz continuous objective functions with a special emphasis on large-scale problems. The existing bundle methods are reliable and efficient methods for small- and medium-scale problems. However, in large-scale cases their computation demand expands. On the other hand, the subgradient methods suffer from some serious disadvantages and, thus, they are not necessarily the best choices for large-scale optimization. Furthermore, we have not been able to find any general solver for nonsmooth large-scale problems in the literature. Thus, there is an evident need for a reliable and efficient solver for nonsmooth large-scale optimization problems.

In this thesis, we have introduced a new limited memory bundle method for nonsmooth large-scale optimization. We have tested the performance of the new method with different minimization problems. The numerical experiments confirm that the new method is efficient and reliable for both smooth and nonsmooth optimization problems.

We have proved the global convergence of the method for locally Lipschitz continuous objective functions, which are not necessarily differentiable or convex. In addition, we have introduced some different modifications to the basic algorithm and proved their global convergence when necessary.

Our numerical experiments showed that the limited memory bundle method, as well as its different modifications, can successfully solve both convex and nonconvex minimization problems (locally). With large numbers of variables they usually used significantly less CPU time than the other solvers tested: With the smooth problems, the new limited memory bundle solver `LMBM` was almost twice as fast as the limited memory BFGS solver `L-BFGS`, which has been developed for smooth large-scale minimization. In addition, with 1000 variables, `LMBM` was on the average about 40 times faster than the proximal bundle solver `PBNCGC`. For nonsmooth problems these differences were even more perceptible. For example, for the image restoration problems, the basic limited memory bundle solver `LMBM` was about 80 times faster than `PBNCGC` already with 300 variables.

With smooth problems and with academic nonsmooth problems, the accuracy

of the new solvers was comparable to that of the other solvers tested. However, with the nonsmooth image restoration problems, the minima found with the limited memory bundle solvers could be slightly greater than those of the proximal bundle solver and with a large number of variables some inaccurate results occurred. Nevertheless, in most cases, the results obtained with the basic version `LMBM` and with the adaptive version `ALMBM` were smaller than those obtained with the active-set method, which has been developed for this kind of problems. Moreover, visually all the results of the different versions of the limited memory bundle method were as good as the results of the proximal bundle method `PBNCGC` and, when comparing the computation times, the new limited memory bundle solvers were considerably more efficient.

In summary, we have introduced a new limited memory bundle method for large-scale nonsmooth unconstrained optimization. This method fills the gap, which exists in the field of nonsmooth optimization with large numbers of variables. Although the new method has already proven useful, there is some further work required before the idea is complete. Possible areas of future development include the following: more practical nonsmooth numerical experiments, implementation with better accuracy properties and more extensive search for internal parameters, constraint handling, and parallelization.

A LARGE-SCALE NONSMOOTH TEST PROBLEMS

Many practical optimization applications involve nonsmooth functions of many variables. However, there exist only few large-scale academic test problems for the nonsmooth case and there is no established practice for testing solvers for large-scale nonsmooth optimization. For this reason, we now introduce the nonsmooth test problems used in our numerical experiments.

We first present 10 nonsmooth minimization problems introduced in [27]. The problems have been constructed either by chaining and extending small existing nonsmooth problems or by "nonsmoothing" large smooth problems (that is, for example, by replacing the term x_i^2 by $|x_i|$). All these problems can be formulated with any number of variables. We first give the formulation of the objective function f and the starting point $\boldsymbol{x}_1 = (x_1^{(1)}, \ldots, x_n^{(1)})^T$ for each problem. Then, we collect some details of the problems as well as the references to the original problems in Table 4.

After that, we present 10 nonsmooth minimization problems from the collection `TEST29` of the software package `UFO` (*Universal Functional Optimization*) [50]. Also these problems can be formulated with any number of variables. We do not give any details of these problems but only the formulation of the objective function f, the starting point $\boldsymbol{x}_1 = (x_1^{(1)}, \ldots, x_n^{(1)})^T$, and the original number of the problem in the collection `TEST29`.

In what follows, we denote by $\mathrm{div}(i, j)$ the integer division for positive integers i and j, that is, the maximum integer not greater than i/j, and by $\mathrm{mod}(i, j)$ the remainder after integer division, that is, $\mathrm{mod}(i, j) = j(i/j - \mathrm{div}(i, j))$.

1. Generalization of MAXQ

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

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

2. Generalization of MXHILB

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

$x_i^{(1)} = 1.0, \quad$ for all $i = 1, \ldots, n.$

3. Chained LQ

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

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

4. Chained CB3 I

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

$x_i^{(1)} = 2.0, \quad$ for all $i = 1, \ldots, n.$

5. Chained CB3 II

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

$x_i^{(1)} = 2.0, \quad$ for all $i = 1, \ldots, n.$

6. Number of Active Faces

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

where $g(y) = \ln(|y| + 1).$

$x_i^{(1)} = 1.0, \quad$ for all $i = 1, \ldots, n.$

7. Nonsmooth generalization of Brown function 2

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

$x_i^{(1)} = 1.0, \quad$ when $\mathrm{mod}(i, 2) = 0$ and
$x_i^{(1)} = -1.0, \quad$ when $\mathrm{mod}(i, 2) = 1$, $(i = 1, \ldots, n).$

8. Chained Mifflin 2

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

$x_i^{(1)} = -1.0, \quad$ for all $i = 1, \ldots, n.$

9. Chained Crescent I

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

$x_i^{(1)} = 2.0, \quad$ when $\mathrm{mod}(i, 2) = 0$ and
$x_i^{(1)} = -1.5, \quad$ when $\mathrm{mod}(i, 2) = 1$, $(i = 1, \ldots, n).$

10. Chained Crescent II

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

$$x_i^{(1)} = 2.0, \quad \text{when } \mathrm{mod}(i, 2) = 0 \text{ and}$$
$$x_i^{(1)} = -1.5, \quad \text{when } \mathrm{mod}(i, 2) = 1, \ (i = 1, \ldots, n).$$

The details of the problems 1 – 10 are given in Table 4, where p denotes the problem number, $f(\boldsymbol{x}^*)$ is the minimum value of the objective function, and the symbols "−" (nonconvex) and "+" (convex) denote the convexity of the problems. In addition, the references to the original problems in each case are given in Table 4.

TABLE 4: Test problems.

p	$f(\boldsymbol{x}^*)$	Convex	Original problem and reference
1	0.0	+	MAXQ, $n = 20$, see [72]
2	0.0	+	MXHILB, $n = 50$, see [34]
3	$-(n-1)2^{1/2}$	+	LQ, $n = 2$, see [79]
4	$2(n-1)$	+	CB3, $n = 2$, see [10]
5	$2(n-1)$	+	CB3, $n = 2$, see [10]
6	0.0	−	See [24]
7	0.0	−	Generalization of Brown function, see [12]
8	varies*	−	Mifflin 2, $n = 2$, see [25]
9	0.0	−	Crescent, $n = 2$, see [33]
10	0.0	−	Crescent, $n = 2$, see [33]

* $f(\boldsymbol{x}^*) \approx -34.795$ for $n = 50$, $f(\boldsymbol{x}^*) \approx -140.86$ for $n = 200$, and $f(\boldsymbol{x}^*) \approx -706.55$ for $n = 1000$.

Next, we present 10 nonsmooth minimization problems from the collection `TEST29` of the software package `UFO` (*Universal Functional Optimization*) [50].

11. Problem 2 in `TEST29`

$$f(\boldsymbol{x}) = \max_{1 \le i \le n} |x_i|.$$

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

12. Problem 5 in `TEST29`

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

$x_i^{(1)} = 1.0, \quad$ for all $i = 1, \ldots, n$.

13. Problem 6 in `TEST29`

$f(\boldsymbol{x}) = \max_{1 \le i \le n} |(3 - 2x_i)x_i + 1 - x_{i-1} - x_{i+1}|$,

where $x_0 = x_{n+1} = 0$.

$x_i^{(1)} = -1.0,$ for all $i = 1, \ldots, n$.

14. Problem 11 in `TEST29`

$f(\boldsymbol{x}) = \sum_{k=1}^{2(n-1)} |f_k(\boldsymbol{x})|$,

where

$f_k(\boldsymbol{x}) = x_i + x_{i+1}((1 + x_{i+1})x_{i+1} - 14) - 29$, when $\mathrm{mod}(k, 2) = 0$,
$f_k(\boldsymbol{x}) = x_i + x_{i+1}((5 - x_{i+1})x_{i+1} - 2) - 13$, when $\mathrm{mod}(k, 2) = 1$,

and $i = (k + 1)/2$.

$x_i^{(1)} = 0.5, \quad$ for $i = 1, \ldots, n - 1$, and
$x_n^{(1)} = -2.0$.

15. Problem 13 in `TEST29`

$f(\boldsymbol{x}) = \sum_{k=1}^{2(n-2)} \left| y_l + \sum_{h=1}^{3} \frac{h^2}{l} \prod_{j=1}^{4} \frac{x_{i+j}}{|x_{i+j}|} |x_{i+j}^{j/hl}| \right|$,

where $i = 2\,\mathrm{div}(k + 3, 4) - 2$, $l = \mathrm{mod}(k - 1, 4) + 1$, and
$y_1 = -14.4$, $y_2 = -6.8$, $y_3 = -4.2$, and $y_4 = -3.2$.

$x_i^{(1)} = 0.8, \quad$ when $\mathrm{mod}(i, 4) = 0$,
$x_i^{(1)} = -0.8,$ when $\mathrm{mod}(i, 4) = 1$,
$x_i^{(1)} = 1.2, \quad$ when $\mathrm{mod}(i, 4) = 2$, and
$x_i^{(1)} = -1.2,$ when $\mathrm{mod}(i, 4) = 3$, $(i = 1, \ldots, n)$.

16. Problem 17 in TEST29

$f(\boldsymbol{x}) = \max_{1\le i\le n} |5 - (j+1)(1-\cos x_i) - \sin x_i - \sum_{k=5j+1}^{5j+5} \cos x_k|,$

where $j = \mathrm{div}(i-1, 5)$.

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

17. Problem 19 in TEST29

$f(\boldsymbol{x}) = \max_{1\le i\le n}((3-2x_i)x_i - x_{i-1} - 2x_{i+1} + 1)^2,$

where $x_0 = x_{n+1} = 0$.

$x_i^{(1)} = -1.0, \text{ for all } i = 1, \ldots, n.$

18. Problem 20 in TEST29

$f(\boldsymbol{x}) = \max_{1\le i\le n} |(0.5x_i - 3)x_i - 1 + x_{i-1} + 2x_{i+1}|,$

where $x_0 = x_{n+1} = 0$.

$x_i^{(1)} = -1.0, \text{ for all } i = 1, \ldots, n.$

19. Problem 22 in TEST29

$f(\boldsymbol{x}) = \max_{1\le i\le n} |2x_i + \frac{1}{2(n+1)^2}(x_i + \frac{i}{n+1} + 1)^3 - x_{i-1} - x_{i+1}|,$

where $x_0 = x_{n+1} = 0$.

$x_i^{(1)} = \frac{i}{n+1}\left(\frac{i}{n+1} - 1\right), \text{ for all } i = 1, \ldots, n$

20. Problem 24 in TEST29

$f(\boldsymbol{x}) = \max_{1\le i\le n} |2x_i + \frac{10}{(n+1)^2}\sinh(10x_i) - x_{i-1} - x_{i+1}|,$

where $x_0 = 0$ and $x_{n+1} = 1$.

$x_i^{(1)} = 1.0, \quad \text{for all } i = 1, \ldots, n.$

YHTEENVETO (FINNISH SUMMARY)

Tutkimuksen tavoitteena on kehittää tehokkaita ja luotettavia numeerisia menetelmiä suurten epäsileiden (ei välttämättä differentioituvien) optimointitehtävien ratkaisemiseksi. Tällaisia optimointitehtäviä esiintyy monissa käytännön sovelluksissa. Esimerkkeinä mainittakoon optimaalisen muodon ongelmat, taloustieteen veromallit, kemian prosessiteollisuus, teräksen jatkuvavalu sekä kuvankäsittely. Epäsileiden tehtävien ratkaiseminen perinteisillä gradienttipohjaisilla menetelmillä (ns. sileillä menetelmillä) ei yleensä onnistu, vaan niiden ratkaisemiseksi tarvitaan epäsileitä optimointimenetelmiä. Nykyiset epäsileän optimoinnin menetelmät eivät kuitenkaan ole tehokkaita, jos muuttujien määrä on suuri (yli 500), kuten käytännön sovelluksissa usein on. Näin ollen tarve luotettaville ja tehokkaille suurten epäsileiden optimointitehtävien ratkaisijoille on ilmeinen.

Epäsileän optimoinnin menetelmät jaetaan perusajatustensa mukaan kahteen luokkaan: aligradienttimenetelmiin (*subgradient methods*) ja kimppumenetelmiin (*bundle methods*). Tässä työssä rajoitutaan tarkastelemaan kimppumenetelmiä, koska näitä pidetään tämän hetken tehokkaimpina ja luotettavimpina epäsileän optimoinnin menetelminä.

Tutkimuksessa kehitetään uusi ns. rajoitetun muistin kimppumenetelmä (*limited memory bundle method*) suurten epäsileiden optimointitehtävien ratkaisemiseksi. Uusi menetelmä yhdistää hyvät ominaisuudet muuttuvan metriikan kimppumenetelmästä ja rajoitetun muistin muuttuvan metriikan menetelmästä, joista ensiksi mainittu on kehitetty pienten epäsileiden tehtävien ratkaisemiseen ja viimeksi mainittu on kehitetty suurten sileiden tehtävien ratkaisemiseen. Uusi menetelmä toimii erittäin tehokkaasti, kun muuttujien määrä on suuri ja monipuoliset numeeriset testit kertovat menetelmän käyttökelpoisuudesta ja hyödyllisyydestä. Kyseessä on uraauurtava työ, sillä kirjallisuudesta ei toistaiseksi löydy tehokasta menetelmää, joka soveltuisi suurten epäsileiden optimointitehtävien ratkaisemiseen.

Rajoitetun muistin kimppumenetelmän globaali konvergenssi todistetaan lokaalisti Lipschitz-jatkuville objektifunktioille. Objektifunktion ei siis tarvitse olla differentioituva eikä konveksi. Lisäksi työssä esitetään joitakin muunnelmia uudesta rajoitetun muistin kimppumenetelmästä. Näiden muunnelmien tarkoituksena on parantaa perusmenetelmän tarkkuutta ilman, että menetelmän tehokkuus merkittävästi heikkenee. Kehitettyjen menetelmien tehokkuutta testataan ja havainnollistetaan numeeristen esimerkkien avulla.

Avainsanat: Epäsileä optimointi, suuret tehtävät, kimppumenetelmät, muuttuvan metriikan menetelmät, rajoitetun muistin muuttuvan metriikan menetelmät.

REFERENCES

[1] Auslender, A. Numerical methods for nondifferentiable convex optimization. *Mathematical Programming Study 30* (1987), 102–126.

[2] Beck, A., and Teboulle, M. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operetions Research Letters 31*, 3 (2003), 167–175.

[3] Bonnans, J. F., Gilbert, J. C., Lemaréchal, C., and Sagastizábal, C. A family of variable metric proximal methods. *Mathematical Programming 68* (1995), 15–47.

[4] Bränndlund, U., Kiwiel, K. C., and Lindberg, P. O. A descent proximal level bundle method for convex nondifferentiable optimization. *Operations Research Letters 17* (1995).

[5] Broyden, C. G. The convergence of a class of double-rank minimization algorithms, Part I — General considerations, Part II — The new algorithm. *Journal of the Institute of Mathematics and Its Applications 6* (1970), 76–90, 222–231.

[6] Buckley, A. G., and LeNir, A. QN-like variable storage conjugate gradients. *Mathematical Programming 27* (1983), 155–175.

[7] 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.

[8] 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.

[9] Byrd, R. H., Nocedal, J., and Yuan, Y. Global convergence of a class of quasi-Newton methods on convex problems. *SIAM Journal on Numerical Analysis 24*, 3 (1987), 1171–1189.

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

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

[12] 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.

[13] Davidon, W. C. Variable metric method for minimization. Technical Report ANL-5990 (Rev.), Argonne National Laboratory, Research and Development, 1959.

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

[15] Fletcher, R. A new approach to variable metric algorithms. *Computer Journal 13* (1970), 317–322.

[16] Fletcher, R. *Practical Methods of Optimization*, second ed. John Wiley and Sons, Chichester, 1987.

[17] Fletcher, R., and Powell, M. J. D. A rapidly convergent descent method for minimization. *Computer Journal 6* (1963), 163–168.

[18] Fuduli, A., Gaudioso, M., and Giallombardo, G. A DC piecewise affine model and a bundling technique in nonconvex nonsmooth minimization. *Optimization Methods and Software 19*, 1 (2004), 89–102.

[19] Fuduli, A., Gaudioso, M., and Giallombardo, G. Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. *SIAM Journal on Optimization 14*, 3 (2004), 743–756.

[20] Gaudioso, M., and Monaco, M. F. Variants to the cutting plane approach for convex nondifferentiable optimization. *Optimization 25* (1992), 65–75.

[21] Gilbert, J.-C., and Lemaréchal, C. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming 45* (1989), 407–435.

[22] Goldfarb, D. A family of variable metric methods derived by variational means. *Mathematics of Computation 24* (1970), 23–26.

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

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

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

[26] Haarala, M., Miettinen, K., and Mäkelä, M. M. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. Submitted.

[27] 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.

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

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

[30] Kärkkäinen, T., Majava, K., and Mäkelä, M. M. Comparison of formulations and solution methods for image restoration problems. Reports of the Department of Mathematical Information Technology, Series B. Scientific Computing, B 14/2000 University of Jyväskylä, Jyväskylä, 2000.

[31] 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.

[32] Kelley, J. E. The cutting plane method for solving convex programs. *Journal of the SIAM 8* (1960), 703–712.

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

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

[35] Kiwiel, K. C. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming 46* (1990), 105–122.

[36] Kiwiel, K. C. Approximations in decomposition of large-scale convex programs via a nondifferentiable optimization method. In *Proceedings of the 11th Triennial IFAC World Congress, Tallin, Estonia, 1990* (1991), Ü. Jaaksoo and V. I. Utkin, Eds., vol. 1, Pergamon Press, Oxford, England, pp. 161–166.

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

[38] Lalee, M., Nocedal, J., and Todd, P. On the implementation of an algorithm for large-scale equality constrained optimization. *SIAM Journal on Optimization 8*, 3 (1998), 682–706.

[39] Lemaréchal, C. An extension of Davidon methods to nondifferentiable problems. In *Nondifferentiable Optimization, Mathematical Programming Study 3*, M. L. Balinski and P. Wolfe, Eds. 1975, pp. 95–109.

[40] Lemaréchal, C. Combining Kelley's and conjugate gradient methods. In *Abstracts of IX International Symposium on Mathematical Programming* (Budapest, Hungary, 1976).

[41] Lemaréchal, C. Nonsmooth optimization and descent methods. Technical Report 78/4, IIASA, Laxemburg, Austria, 1978.

[42] Lemaréchal, C. Numerical experiments in nonsmooth optimization. In *Proceedings of the IIASA workshop on Progress in Nondifferentiable Optimization* (Laxemburg, Austria, 1982), E. A. Nurminski, Ed., pp. 61–84.

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

[44] Lemaréchal, C., Nemirovskii, A., and Nesterov, Y. New variants of bundle methods. *Mathematical Programming 69* (1995), 111–147.

[45] Li, D.-H., and Fukushima, M. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization 11*, 4 (2001), 1054–1064.

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

[47] Lukšan, L. Dual method for solving a special problem of quadratic programming as a subproblem at linearly constrained nonlinear minmax approximation. *Kybernetika 20* (1984), 445–457.

[48] Lukšan, L. Computational experience with known variable metric updates. Technical Report 534, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1992.

[49] Lukšan, L., and Spedicato, E. Variable metric methods for unconstrained optimization and nonlinear least squares. *Journal of Computational and Applied Mathematics 124* (2000), 61–95.

[50] Lukšan, L., Tůma, M., Šiška, M., Vlček, J., and Ramešová, N. UFO 2002. Interactive system for universal functional optimization. Technical Report 883, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2002.

[51] Lukšan, L., and Vlček, J. PVAR — Variable metric methods for unconstrained and linearly constrained nonsmooth optimization. Available in web page <URL: `http://www.uivt.cas.cz/~luksan/subroutines.html`>. (2nd August, 2004).

[52] Lukšan, L., and Vlček, J. Simple scaling for variable metric updates. Technical Report 611, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1995.

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

[54] 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.

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

[56] Lukšan, L., and Vlček, J. Introduction to nonsmooth analysis. Theory and algorithms. Technical Report DMSIA 1/2000, University of Bergamo, 2000.

[57] Lukšan, L., and Vlček, J. Variable metric methods for nonsmooth optimization. Technical Report 837, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2001.

[58] Mäkelä, M. M. Methods and algorithms for nonsmooth optimization. Reports on Applied Mathematics and Computing 2, Department of Mathematics, University of Jyväskylä, 1989.

[59] Mäkelä, M. M. Issues of implementing a Fortran subroutine package NSOLIB for nonsmooth optimization. Technical Report 5/1993, Department of Mathematics, Laboratory of Scientific Computing, University of Jyväskylä, 1993.

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

[61] 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.

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

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

[64] Miettinen, K., Mäkelä, M. M., and Männikkö, T. Optimal control of continuous casting by nondifferentiable multiobjective optimization. *Computational Optimization and Applications 11* (1998), 177–194.

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

[66] Nocedal, J. Theory of algorithms for unconstrained optimization. *Acta Numerica 1* (1992), 199–242.

[67] Nocedal, J. Large scale unconstrained optimization. In *The State of the Art in Numerical Analysis*, A. Watson and I. Duff, Eds. Oxford University Press, 1997, pp. 311–338.

[68] Nocedal, J., and Yuan, Y. Analysis of a self-scaling quasi-Newton method. *Mathematical Programming 61* (1993), 19–37.

[69] Overton, M. L. Large-scale optimization of eigenvalues. *SIAM Journal on Optimization 2* (1992), 88–120.

[70] Rockafellar, R. T. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[71] Rockafellar, R. T. Monotone operators and the proximal point algorithm. *SIAM Journal on Optimal Control and Optimization 14* (1976), 877–898.

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

[73] 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* (1992), 121–152.

[74] Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation 24* (1970), 647–657.

[75] Shanno, D. F., and Phua, K. J. Matrix conditioning and nonlinear optimization. *Mathematical Programming 14* (1978), 144–160.

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

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

[78] 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.

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