

Chapter 15

Subgradient and Bundle Methods for Nonsmooth Optimization

Marko M. Mäkelä, Napsu Karmitsa, and Adil Bagirov

Abstract The nonsmooth optimization methods can mainly be divided into two groups: subgradient and bundle methods. Usually, when developing new algorithms and testing them, the comparison is made between similar kinds of methods. The goal of this work is to test and compare different bundle and subgradient methods as well as some hybrids of these two and/or some others. The test set included a large amount of different unconstrained nonsmooth minimization problems, e.g., convex and nonconvex problems, piecewise linear and quadratic problems, and problems with different sizes. Rather than foreground some method over the others, our aim is to get some insight on which method is suitable for certain types of problems.

15.1 Introduction

We consider unconstrained nonsmooth optimization (NSO) problems of the form

$$\min_{x \in \mathbb{R}^n} f(x), \quad (15.1)$$

where the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is supposed to be locally Lipschitz continuous. Note that no differentiability or convexity assumptions are made.

NSO problems of type (15.1) arise in many application areas: in economics [38], mechanics [37], engineering [36], control theory [11], optimal shape design [17], data mining [1, 7] and in particular cluster analysis [12], and machine learning [20].

M.M. Mäkelä (✉) · N. Karmitsa
Department of Mathematics, University of Turku, 20014 Turku, Finland
e-mail: makela@utu.fi

N. Karmitsa
e-mail: napsu@karmitsa.fi

A. Bagirov
Centre for Informatics and Applied Optimization, School of Science, Information Technology and Engineering, University of Ballarat, University Drive, Mount Helen, PO Box 663, Ballarat, VIC 3353, Australia
e-mail: a.bagirov@ballarat.edu.au

Most of the methods for solving problems of type (15.1) can be divided into two main groups: subgradient (see, e.g., [4, 5, 42, 43]) and bundle methods (see, e.g., [14, 18, 23, 32, 35, 40, 41]). Both of these method groups have their own supporters. Usually, when developing new methods, researchers compare them with similar methods. Moreover, it is quite common that the test set used is rather concise.

In this work, we compare different subgradient and bundle methods, as well as some of the methods that lie between these two. The main criteria in numerical comparison are the efficiency and the reliability of the methods. Moreover, we use a broad test setting including different classes of nonsmooth problems. All the solvers tested are so-called general black box methods and, naturally, cannot beat the codes designed specially for a particular class of problems (say, e.g., for piecewise linear, min-max, or partially separable problems). However, rather than seeing this generality as a weakness, it should be seen as a strength due to the minimal information of the objective function required for the calculations. Namely, the value of the objective function and, possibly, one arbitrary subgradient (the generalized gradient [10]) at each point.

The aim of our research is not to foreground some method over the others—it is a well-known fact that different methods work well for different types of problems and none of them is good for all types of problems—but to get some kind of insight on which kind of method to select for certain types of problems.

This work is organized as follows. Section 15.2 introduces the NSO methods tested and compared. The results of the numerical experiments are presented and discussed in Sects. 15.3 and 15.4 concludes the work and gives our credentials for well-performing algorithms for different problem classes.

In what follows, we denote by $\|\cdot\|$ the Euclidean norm in \mathbb{R}^n and by $\mathbf{a}^T \mathbf{b}$ the inner product of the vectors \mathbf{a} and \mathbf{b} . The *subdifferential* $\partial f(\mathbf{x})$ [10] of a locally Lipschitz continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at any point $\mathbf{x} \in \mathbb{R}^n$ is given by

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

where “conv” denotes the convex hull of a set. Each vector $\boldsymbol{\xi} \in \partial f(\mathbf{x})$ is called a *subgradient*.

15.2 Methods

In this section, we give short descriptions of the methods to be compared. For more details we refer to [22] and to the original references. In what follows (if not stated otherwise), we assume that at every point \mathbf{x} we can evaluate the value of the objective function $f(\mathbf{x})$ and an arbitrary subgradient $\boldsymbol{\xi}$ from the subdifferential $\partial f(\mathbf{x})$.

15.2.1 Standard Subgradient Method

The first method to be considered here is the cornerstone of NSO: the standard subgradient method [42]. The idea behind subgradient methods (Kiev methods) is to generalize smooth methods (e.g., the steepest descent method) by replacing the gradient with an arbitrary subgradient. Therefore, the iteration formula for these methods is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k \frac{\boldsymbol{\xi}_k}{\|\boldsymbol{\xi}_k\|},$$

where $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$ is any subgradient and $t_k > 0$ is a predetermined step size.

Due to this simple structure and low storage requirements, subgradient methods are widely used methods in NSO. However, basic subgradient methods suffer from some serious disadvantages: a nondescent search direction may occur and thus, the selection of step size is difficult; there exists no implementable subgradient-based stopping criterion; and the convergence speed is poor (not even linear) (see, e.g., [26]).

The standard subgradient method is proved to be globally convergent if the objective function is convex and the step sizes satisfy

$$\lim_{k \rightarrow \infty} t_k = 0 \quad \text{and} \quad \sum_{j=1}^{\infty} t_j = \infty.$$

15.2.2 Shor's r -Algorithm (Space Dilation Method)

Next we shortly describe the ideas of a more sophisticated subgradient method, the well-known Shor's r -algorithm with space dilations along the difference of two successive subgradients. The basic idea of Shor's r -algorithm is to interpolate between the steepest descent and conjugate gradient method.

The iteration formula for Shor's r -algorithm is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - t_k B_{k+1} B_{k+1}^T \boldsymbol{\xi}_k,$$

where $\boldsymbol{\xi}_k \in \partial f(\mathbf{x}_k)$ and $t_k > 0$. The space dilation matrix B_{k+1} is initialized with $B_1 = I$ and it is updated by

$$B_{k+1} = B_k (I + (\beta - 1) \mathbf{s}_k \mathbf{s}_k^T),$$

where $\beta \in (0, 1)$, $\mathbf{s}_k = \mathbf{r}_k / \|\mathbf{r}_k\|$ and $\mathbf{r}_k = B_k^T (\boldsymbol{\xi}_k - \boldsymbol{\xi}_{k-1})$.

In order to turn the above r -algorithm into an efficient optimization routine, one has to find a solution to the following problems: how to choose the step sizes t_k (including the initial step size t_1) and how to design a stopping criterion which does not need information on subgradients.

If the objective function is convex and twice continuously differentiable, its Hessian is Lipschitz, and the starting point is chosen from some neighborhood of the optimal solution, then the n -step quadratic rate convergence can be proved for the r -algorithm. If the objective function is nonconvex and coercive under some additional assumptions, then the r -algorithm is convergent to isolated local minimizers [42].

15.2.3 Proximal Bundle Method (PBM)

In this subsection, we describe the ideas of the proximal bundle method (PBM) for nonsmooth and nonconvex minimization (see, e.g., [24, 35, 41]).

The basic idea of bundle methods is to approximate the whole subdifferential of the objective function instead of using only one arbitrary subgradient at each point. In practice, this is done by gathering subgradients from the previous iterations into a bundle. Suppose that at the k -th iteration of the algorithm we have the current iteration point \mathbf{x}_k and some trial points $\mathbf{y}_j \in \mathbb{R}^n$ (from past iterations) and subgradients $\xi_j \in \partial f(\mathbf{y}_j)$ for $j \in J_k$, where the index set $J_k \neq \emptyset$ and $J_k \subset \{1, \dots, k\}$.

The idea behind the PBM is to approximate the objective function f below by a piecewise linear function, that is, f is replaced by the so-called *cutting-plane model*

$$\hat{f}_k(\mathbf{x}) = \max_{j \in J_k} \{f(\mathbf{y}_j) + \xi_j^T(\mathbf{x} - \mathbf{y}_j)\}. \quad (15.2)$$

This model can be written in an equivalent form

$$\hat{f}_k(\mathbf{x}) = \max_{j \in J_k} \{f(\mathbf{x}_k) + \xi_j^T(\mathbf{x} - \mathbf{x}_k) - \alpha_j^k\},$$

where

$$\alpha_j^k = f(\mathbf{x}_k) - f(\mathbf{y}_j) - \xi_j^T(\mathbf{x}_k - \mathbf{y}_j) \quad \text{for all } j \in J_k$$

is a so-called *linearization error*. If f is convex, then \hat{f}_k is an underestimate for f and $\alpha_j^k \geq 0$ for all $j \in J_k$. In the nonconvex case, these facts are not valid anymore and thus the linearization error α_j^k can be replaced by the so-called *subgradient locality measure* (cf. [23])

$$\beta_j^k = \max\{|\alpha_j^k|, \gamma \|\mathbf{x}_k - \mathbf{y}_j\|^2\}, \quad (15.3)$$

where $\gamma \geq 0$ is the *distance measure parameter* ($\gamma = 0$ if f is convex). Then obviously $\beta_j^k \geq 0$ for all $j \in J_k$ and $\min_{\mathbf{x} \in K} \hat{f}_k(\mathbf{x}) \leq f(\mathbf{x}_k)$.

The descent direction is calculated by

$$\mathbf{d}_k = \arg \min_{\mathbf{d} \in \mathbb{R}^n} \left\{ \hat{f}_k(\mathbf{x}_k + \mathbf{d}) + \frac{1}{2} u_k \mathbf{d}^T \mathbf{d} \right\}, \quad (15.4)$$

where the stabilizing term $\frac{1}{2}u_k \mathbf{d}^T \mathbf{d}$ guarantees the existence of the solution \mathbf{d}_k and keeps the approximation local enough. The weighting parameter $u_k > 0$ improves the convergence rate and accumulates some second order information about the curvature of f around \mathbf{x}_k (see, e.g., [24, 35, 41]).

In order to determine the step size into the search direction \mathbf{d}_k , the PBM uses the following *line search procedure*: Assume that $m_L \in (0, \frac{1}{2})$, $m_R \in (m_L, 1)$ and $\bar{t} \in (0, 1]$ are some fixed line search parameters. We first search for the largest number $t_L^k \in [0, 1]$ such that $t_L^k \geq \bar{t}$ and

$$f(\mathbf{x}_k + t_L^k \mathbf{d}_k) \leq f(\mathbf{x}_k) + m_L t_L^k v_k, \quad (15.5)$$

where v_k is the predicted amount of descent

$$v_k = \hat{f}_k(\mathbf{x}_k + \mathbf{d}_k) - f(\mathbf{x}_k) < 0.$$

If such a parameter exists, we take a *long serious step*

$$\mathbf{x}_{k+1} = \mathbf{x}_k + t_L^k \mathbf{d}_k \quad \text{and} \quad \mathbf{y}_{k+1} = \mathbf{x}_{k+1}. \quad (15.6)$$

Otherwise, if (15.5) holds but $0 < t_L^k < \bar{t}$, we take a *short serious step*

$$\mathbf{x}_{k+1} = \mathbf{x}_k + t_L^k \mathbf{d}_k \quad \text{and} \quad \mathbf{y}_{k+1} = \mathbf{x}_k + t_R^k \mathbf{d}_k$$

and, if $t_L^k = 0$, we take a *null step*

$$\mathbf{x}_{k+1} = \mathbf{x}_k \quad \text{and} \quad \mathbf{y}_{k+1} = \mathbf{x}_k + t_R^k \mathbf{d}_k, \quad (15.7)$$

where $t_R^k > t_L^k$ is such that

$$-\beta_{k+1}^{k+1} + \xi_{k+1}^T \mathbf{d}_k \geq m_R v_k. \quad (15.8)$$

In short serious steps and null steps there exists discontinuity in the gradient of f . Then the requirement (15.8) ensures that \mathbf{x}_k and \mathbf{y}_{k+1} lie on the opposite sides of this discontinuity and the new subgradient $\xi_{k+1} \in \partial f(\mathbf{y}_{k+1})$ will force a remarkable modification of the next search direction finding problem. The iteration is terminated if $v_k \geq -\varepsilon_s$, where $\varepsilon_s > 0$ is a final accuracy tolerance supplied by the user.

Under the upper semi-smoothness assumption [6] the PBM can be proved to be globally convergent for locally Lipschitz continuous functions (see, e.g., [24, 35]). In addition, in order to implement the above algorithm one has to bound somehow the number of stored subgradient and trial points, that is, the cardinality of the index set J_k . The global convergence of bundle methods with a limited number of stored subgradients can be guaranteed by using a subgradient aggregation strategy [23], which accumulates information from the previous iterations. The convergence rate of the PBM is linear for convex functions [39] and for piecewise linear problems the PBM achieves a finite convergence [41].

15.2.4 Bundle Newton Method (BNEW)

Next we describe the main ideas of the second-order bundle-Newton method (BNEW) [29]. We suppose that at each $\mathbf{x} \in \mathbb{R}^n$ we can evaluate, in addition to the function value and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\mathbf{x})$, also an $n \times n$ symmetric matrix $G(\mathbf{x})$ approximating the Hessian matrix $\nabla^2 f(\mathbf{x})$. Now, instead of the piecewise linear cutting-plane model (15.2) we introduce a piecewise quadratic model of the form

$$\tilde{f}_k(\mathbf{x}) = \max_{j \in J_k} \left\{ f(\mathbf{y}_j) + \boldsymbol{\xi}_j^T (\mathbf{x} - \mathbf{y}_j) + \frac{1}{2} \varrho_j (\mathbf{x} - \mathbf{y}_j)^T G_j (\mathbf{x} - \mathbf{y}_j) \right\}, \quad (15.9)$$

where $G_j = G(\mathbf{y}_j)$ and $\varrho_j \in [0, 1]$ is some damping parameter. The model (15.9) can be written equivalently as

$$\tilde{f}_k(\mathbf{x}) = \max_{j \in J_k} \left\{ f(\mathbf{x}_k) + \boldsymbol{\xi}_j^T (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2} \varrho_j (\mathbf{x} - \mathbf{x}_k)^T G_j (\mathbf{x} - \mathbf{x}_k) - \alpha_j^k \right\}$$

and for all $j \in J_k$ the linearization error takes the form

$$\alpha_j^k = f(\mathbf{x}_k) - f(\mathbf{y}_j) - \boldsymbol{\xi}_j^T (\mathbf{x}_k - \mathbf{y}_j) - \frac{1}{2} \varrho_j (\mathbf{x}_k - \mathbf{y}_j)^T G_j (\mathbf{x}_k - \mathbf{y}_j). \quad (15.10)$$

Note that now, even in the convex case, α_j^k might be negative. Therefore we replace the linearization error (15.10) by the subgradient locality measure (15.3) and we remain the property $\min_{\mathbf{x} \in \mathbb{R}^n} \tilde{f}_k(\mathbf{x}) \leq f(\mathbf{x}_k)$ (see [29]).

The search direction $d_k \in \mathbb{R}^n$ is now calculated as the solution of

$$\mathbf{d}_k = \arg \min_{\mathbf{d} \in \mathbb{R}^n} \{ \tilde{f}_k(\mathbf{x}_k + \mathbf{d}) \}. \quad (15.11)$$

The line search procedure of the BNEW follows the same principles than in the PBM (see Sect. 15.2.3). The only remarkable difference occurs in the termination condition for short and null steps. In other words, (15.8) is replaced by two conditions

$$-\beta_{k+1}^{k+1} + (\boldsymbol{\xi}_{k+1}^{k+1})^T d_k \geq m_R v_k$$

and

$$\|\mathbf{x}_{k+1} - \mathbf{y}_{k+1}\| \leq C_S,$$

where $C_S > 0$ is a parameter supplied by the user.

Under the upper semi-smoothness assumption [6] the BNEW can be proved to be globally convergent for locally Lipschitz continuous objective functions. For strongly convex functions, the convergence rate of the BNEW is superlinear [29].

15.2.5 Limited Memory Bundle Method (LMBM)

In this subsection, we very shortly describe the limited memory bundle algorithm (LMBM) [15, 16] for solving general, possibly nonconvex, large-scale NSO problems. The method is a hybrid of the variable metric bundle methods [44] and the limited memory variable metric methods (see, e.g., [9]), where the first ones have been developed for small- and medium-scale nonsmooth optimization and the latter ones, on the contrary, for smooth large-scale optimization.

LMBM exploits the ideas of the variable metric bundle methods, namely the utilization of null steps, simple aggregation of subgradients, and the subgradient locality measures, but the search direction \mathbf{d}_k is calculated using a limited memory approach. That is,

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

where $\tilde{\xi}_k$ is an (aggregate) subgradient and D_k is the limited memory variable metric update that, in the smooth case, represents the approximation of the inverse of the Hessian matrix. Note that the matrix D_k is not formed explicitly but the search direction \mathbf{d}_k is calculated using the limited memory approach.

The LMBM uses the original subgradient ξ_k after the serious step (cf. (15.6)) and the aggregate subgradient $\tilde{\xi}_k$ after the null step (cf. (15.7)) for direction finding (i.e. we set $\tilde{\xi}_k = \xi_k$ if the previous step was a serious step). The aggregation procedure is carried out by determining multipliers λ_i^k satisfying $\lambda_i^k \geq 0$ for all $i \in \{1, 2, 3\}$, and $\sum_{i=1}^3 \lambda_i^k = 1$ that minimize the function

$$\begin{aligned} \varphi(\lambda_1, \lambda_2, \lambda_3) = & [\lambda_1 \xi_m + \lambda_2 \xi_{k+1} + \lambda_3 \tilde{\xi}_k]^T D_k [\lambda_1 \xi_m + \lambda_2 \xi_{k+1} + \lambda_3 \tilde{\xi}_k] \\ & + 2(\lambda_2 \beta_{k+1} + \lambda_3 \tilde{\beta}_k). \end{aligned}$$

Here $\xi_m \in \partial f(\mathbf{x}_k)$ is the current subgradient (m denotes the index of the iteration after the latest serious step, i.e. $\mathbf{x}_k = \mathbf{x}_m$), $\xi_{k+1} \in \partial f(\mathbf{y}_{k+1})$ is the auxiliary subgradient, and $\tilde{\xi}_k$ is the current aggregate subgradient from the previous iteration ($\tilde{\xi}_1 = \xi_1$). In addition, β_{k+1} is the current subgradient locality measure (cf. (15.3)) and $\tilde{\beta}_k$ is the current aggregate subgradient locality measure ($\tilde{\beta}_1 = 0$). The resulting aggregate subgradient $\tilde{\xi}_{k+1}$ and the aggregate subgradient locality measure $\tilde{\beta}_{k+1}$ are computed from the formulae

$$\tilde{\xi}_{k+1} = \lambda_1^k \xi_m + \lambda_2^k \xi_{k+1} + \lambda_3^k \tilde{\xi}_k \quad \text{and} \quad \tilde{\beta}_{k+1} = \lambda_2^k \beta_{k+1} + \lambda_3^k \tilde{\beta}_k.$$

The line search procedure used in the LMBM is rather similar to that used in the PBM (see Sect. 15.2.3). However, due to the simple aggregation procedure above only one trial point $\mathbf{y}_{k+1} = \mathbf{x}_k + t_R^k \mathbf{d}_k$ and a corresponding subgradient $\xi_{k+1} \in \partial f(\mathbf{y}_{k+1})$ need to be stored.

As a stopping parameter, we use the value $w_k = -\tilde{\xi}_k^T \mathbf{d}_k + 2\tilde{\beta}_k$ and we stop if $w_k \leq \varepsilon_s$ for some user specified $\varepsilon_s > 0$. The parameter w_k is also used during the

line search procedure to represent the desirable amount of descent (cf. v_k in the PBM).

In the LMBM both the limited memory BFGS (L-BFGS) and the limited memory SR1 (L-SR1) update formulae [9] are used in calculations of the search direction and the aggregate values. The idea of limited memory matrix updating is that instead of storing large $n \times n$ matrices D_k , one stores a certain (usually small) number of vectors obtained at the previous iterations of the algorithm, and uses these vectors to implicitly define the variable metric matrices. In the case of a null step, we use the L-SR1 update, since this update formula allows us to preserve the boundedness and some other properties of generated matrices which guarantee the global convergence of the method. Otherwise, since these properties are not required after a serious step, the more efficient L-BFGS update is employed (for more details, see [15, 16]).

Under the upper semi-smoothness assumption [6] the LMBM can be proved to be globally convergent for locally Lipschitz continuous objective functions [16].

15.2.6 Discrete Gradient Method (DGM)

Next we briefly describe the discrete gradient method (DGM) [3]. The idea of the DGM is to hybridize derivative free methods with bundle methods. That is, the DGM approximates subgradients by discrete gradients using function values only. Similarly to bundle methods, the previous values of discrete gradients are gathered into a bundle and the null step is used if the current search direction is not good enough.

We start with the definition of the discrete gradient. Let us denote by

$$S_1 = \{ \mathbf{g} \in \mathbb{R}^n \mid \|\mathbf{g}\| = 1 \}$$

the sphere of the unit ball and by

$$P = \{ z \mid z : \mathbb{R}_+ \rightarrow \mathbb{R}_+, \lambda > 0, \lambda^{-1}z(\lambda) \rightarrow 0, \lambda \rightarrow 0 \}$$

the set of univariate positive infinitesimal functions. In addition, let

$$G = \{ \mathbf{e} \in \mathbb{R}^n \mid \mathbf{e} = (e_1, \dots, e_n), |e_j| = 1, j = 1, \dots, n \}$$

be a set of all vertices of the unit hypercube in \mathbb{R}^n . We take any $\mathbf{g} \in S_1$, $\mathbf{e} \in G$, $z \in P$, a positive number $\alpha \in (0, 1]$, and we compute $i = \arg \max \{ |g_j|, j = 1, \dots, n \}$. For $\mathbf{e} \in G$ we define the sequence of n vectors $\mathbf{e}^j(\alpha) = (\alpha e_1, \alpha^2 e_2, \dots, \alpha^j e_j, 0, \dots, 0)$ $j = 1, \dots, n$ and for $\mathbf{x} \in \mathbb{R}^n$ and $\lambda > 0$, we consider the points

$$\mathbf{x}_0 = \mathbf{x} + \lambda \mathbf{g}, \quad \mathbf{x}_j = \mathbf{x}_0 + z(\lambda) \mathbf{e}^j(\alpha), \quad j = 1, \dots, n.$$

Definition 15.1 The *discrete gradient* of the function f at the point $\mathbf{x} \in \mathbb{R}^n$ is the vector $\Gamma^i(\mathbf{x}, \mathbf{g}, \mathbf{e}, z, \lambda, \alpha) = (\Gamma_1^i, \dots, \Gamma_n^i) \in \mathbb{R}^n$ with the following coordinates:

$$\Gamma_j^i = [z(\lambda) \alpha^j e_j]^{-1} [f(\mathbf{x}_j) - f(\mathbf{x}_{j-1})], \quad j = 1, \dots, n, j \neq i,$$

$$\Gamma_i^i = (\lambda g_i)^{-1} \left[f(\mathbf{x} + \lambda \mathbf{g}) - f(\mathbf{x}) - \lambda \sum_{j=1, j \neq i}^n \Gamma_j^i g_j \right].$$

It has been proved in [3] that the closed convex set of discrete gradients

$$D_0(\mathbf{x}, \lambda) = \text{cl conv} \{ \mathbf{v} \in \mathbb{R}^n \mid \exists \mathbf{g} \in S_1, \mathbf{e} \in G, z \in P \\ \text{such that } \mathbf{v} = \Gamma^i(\mathbf{x}, \mathbf{g}, \mathbf{e}, z, \lambda, \alpha) \}$$

is an approximation to the subdifferential $\partial f(\mathbf{x})$ for sufficiently small $\lambda > 0$. Thus, it can be used to compute the descent direction for the objective. However, the computation of the whole set $D_0(\mathbf{x}, \lambda)$ is not easy, and therefore, in the DGM we use only a few discrete gradients from the set to calculate the descent direction.

Let us denote by l the index of the subiteration in the direction-finding procedure, by k the index of the outer iteration, and by s the index of the inner iteration. In what follows we use only the iteration counter l whenever possible without confusion. At every iteration k_s we first compute the discrete gradient $\mathbf{v}_1 = \Gamma^i(\mathbf{x}, \mathbf{g}_1, \mathbf{e}, z, \lambda, \alpha)$ with respect to any initial direction $\mathbf{g}_1 \in S_1$ and we set the initial bundle of discrete gradients $\bar{D}_1(\mathbf{x}) = \{\mathbf{v}_1\}$. Then we compute the vector

$$\mathbf{w}_l = \arg \min_{\mathbf{w} \in \bar{D}_l(\mathbf{x})} \|\mathbf{w}\|^2,$$

that is the distance between the convex hull $\bar{D}_l(\mathbf{x})$ of all computed discrete gradients and the origin. If this distance is less than a given tolerance $\delta > 0$, we accept the point \mathbf{x} as an approximate stationary point and go to the next outer iteration. Otherwise, we compute another search direction

$$\mathbf{g}_{l+1} = -\frac{\mathbf{w}_l}{\|\mathbf{w}_l\|}$$

and we check whether this direction is descent. If it is, we have

$$f(\mathbf{x} + \lambda \mathbf{g}_{l+1}) - f(\mathbf{x}) \leq -c_1 \lambda \|\mathbf{w}_l\|,$$

with the given numbers $c_1 \in (0, 1)$ and $\lambda > 0$. Then we set $\mathbf{d}_{k_s} = \mathbf{g}_{l+1}$, $\mathbf{v}_{k_s} = \mathbf{w}_l$ and stop the direction finding procedure. Otherwise, we compute another discrete gradient $\mathbf{v}_{l+1} = \Gamma^i(\mathbf{x}, \mathbf{g}_{l+1}, \mathbf{e}, z, \lambda, \alpha)$ into the direction \mathbf{g}_{l+1} , update the bundle of discrete gradients

$$\bar{D}_{l+1}(\mathbf{x}) = \text{conv} \{ \bar{D}_l(\mathbf{x}) \cup \{\mathbf{v}_{l+1}\} \}$$

and continue the direction finding procedure with $l = l + 1$. Note that at each subiteration the approximation of the subdifferential $\partial f(\mathbf{x})$ is improved. It has been proved in [3] that the direction finding procedure is terminating.

In [3], it is proved that the DGM is globally convergent for locally Lipschitz continuous functions under the assumption that the set of discrete gradients uniformly approximates the subdifferential.

15.2.7 Quasisecant Method (QSM)

In this subsection, we briefly describe the quasisecant method (QSM) [2]. Here, it is again assumed that one can compute both the function value and one subgradient at any point.

The QSM can be considered as a hybrid of bundle methods and the gradient sampling method [8]. The method builds up information about the approximation of the subdifferential using a bundling idea, which makes it similar to bundle methods, while subgradients are computed from a given neighborhood of a current iteration point, which makes the method similar to the gradient sampling method.

We start this subsection with the definition of a quasisecant for locally Lipschitz continuous functions.

Definition 15.2 A vector $\mathbf{v} \in \mathbb{R}^n$ is called a *quasisecant* of the function f at the point $\mathbf{x} \in \mathbb{R}^n$ in the direction $\mathbf{g} \in S_1$ with the length $h > 0$ if

$$f(\mathbf{x} + h\mathbf{g}) - f(\mathbf{x}) \leq h\mathbf{v}^T \mathbf{g}.$$

We will denote this quasisecant by $\mathbf{v}(\mathbf{x}, \mathbf{g}, h)$.

For a given $h > 0$ let us consider the set of quasisecants at a point \mathbf{x}

$$QSec(\mathbf{x}, h) = \{\mathbf{w} \in \mathbb{R}^n \mid \exists \mathbf{g} \in S_1 \text{ s.t. } \mathbf{w} = \mathbf{v}(\mathbf{x}, \mathbf{g}, h)\}$$

and the set of limit points of quasisecants as $h \searrow 0$:

$$QSL(\mathbf{x}) = \left\{ \mathbf{w} \in \mathbb{R}^n \mid \exists \mathbf{g} \in S_1, h_k > 0, h_k \searrow 0 \text{ when } k \rightarrow \infty \right. \\ \left. \text{s.t. } \mathbf{w} = \lim_{k \rightarrow \infty} \mathbf{v}(\mathbf{x}, \mathbf{g}, h_k) \right\}.$$

A mapping $\mathbf{x} \mapsto QSec(\mathbf{x}, h)$ is called a *subgradient-related (SR)-quasisecant mapping* if the corresponding set $QSL(\mathbf{x}) \subseteq \partial f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$. In this case, the elements of $QSec(\mathbf{x}, h)$ are called *SR-quasisecants*. In the sequel, we will consider sets $QSec(\mathbf{x}, h)$ which contain only SR-quasisecants.

It has been shown in [2] that the closed convex set of quasisecants

$$W_0(\mathbf{x}, h) = \text{cl conv } QSec(\mathbf{x}, h)$$

can be used to find a descent direction for the objective with any $h > 0$. However, it is not easy to compute the entire set $W_0(\mathbf{x}, h)$, and therefore we use only a few quasisecants from the set to calculate the descent direction in the QSM.

The procedures used in the QSM are pretty similar to those in the DGM but instead of the discrete gradient $\mathbf{v}_l = \Gamma^i(\mathbf{x}, \mathbf{g}_l, \mathbf{e}, z, \lambda, \alpha)$ we use here the quasisecant $\mathbf{v}_l(\mathbf{x}, \mathbf{g}_l, h)$. Thus, at every iteration k_s we compute the vector

$$\mathbf{w}_l = \arg \min_{\mathbf{w} \in \bar{V}_l(\mathbf{x})} \|\mathbf{w}\|^2,$$

Table 15.1 Tested pieces of software

Software	Author(s)	Method	Reference
SUBG	Karmitsa	Subgradient	[42]
SolVOpt	Kuntsevich & Kappel	Shor's r -algorithm	[19, 25, 42]
PBNCGC	Mäkelä	Proximal bundle	[33, 35]
PNEW	Lukšan & Vlček	Bundle-Newton	[29]
LMBM	Karmitsa	Limited memory bundle	[15, 16]
DGM	Bagirov et al.	Discrete Gradient	[3]
QSM	Bagirov & Ganjehlou	QuasiSecant	[2]

where $\bar{V}_l(\mathbf{x})$ is a set of all quasisecants computed so far. If $\|\mathbf{w}_l\| < \delta$ with a given tolerance $\delta > 0$, we accept the point \mathbf{x} as an approximate stationary point, a so-called (h, δ) -stationary point [2], and we go to the next outer iteration. Otherwise, we compute another search direction $\mathbf{g}_{l+1} = -\mathbf{w}_l/\|\mathbf{w}_l\|$ and we check whether this direction is descent or not. If it is, we set $\mathbf{d}_{k_s} = \mathbf{g}_{l+1}$, $\mathbf{v}_{k_s} = \mathbf{w}_l$ and stop the direction-finding procedure. Otherwise, we compute another quasisecant $\mathbf{v}_{l+1}(\mathbf{x}, \mathbf{g}_{l+1}, h)$, update the bundle of quasisecants $\bar{V}_{l+1}(\mathbf{x}) = \text{conv}\{\bar{V}_l(\mathbf{x}) \cup \{\mathbf{v}_{l+1}(\mathbf{x}, \mathbf{g}_{l+1}, h)\}\}$ and continue the direction-finding procedure with $l = l + 1$. It has been proved in [2] that the direction-finding procedure is terminating. When the descent direction \mathbf{d}_{k_s} has been found, we need to compute the next (inner) iteration point similarly to that in the DGM.

The QSM is globally convergent for locally Lipschitz continuous functions under the assumption that the set $QSec(\mathbf{x}, h)$ is a SR-quasisecant mapping, that is, quasisecants can be computed using subgradients [2].

15.3 Numerical Experiments

In what follows, we compare the implementations of the methods described above. The more detailed description about the test results can be found in [21].

15.3.1 Solvers

The tested optimization codes are presented in Table 15.1. The codes or links for downloading the codes are available from <http://napsu.karmitsa.fi/nsossoftware/>. The experiments were performed on an Intel[®] Core[™] 2 CPU 1.80 GHz.

SUBG is a crude implementation of the basic subgradient algorithm. The step length is chosen to be to some extent constant. We use the following three criteria as a stopping rule for SUBG: the number of function evaluations (and iterations) is restricted by a parameter and also the algorithm stops if either it cannot decrease the

value of the objective function within some successive iterations, or it cannot find a descent direction within some successive iterations. Since a subgradient method is not a descent method, we store the best value f_{best} of the objective function and the corresponding point \mathbf{x}_{best} and return them as a solution if any of the stopping rules above is met.

`SolvOpt` (a solver for local nonlinear optimization problems) is an implementation of Shor's r -algorithm. The approaches used to handle the difficulties with step size selection and termination criteria in Shor's r -algorithm are heuristic (for details see [19]). In `SolvOpt` one can select to use either original subgradients or their difference approximations (i.e. the user does not have to code difference approximations but to select one parameter to do this automatically). In our experiments we have used both analytically and numerically calculated subgradients. In what follows, we denote `SolvOptA` and `SolvOptN`, respectively, the corresponding solvers. There exist MatLab, C, and Fortran source codes for `SolvOpt`. In our experiments we used `SolvOpt v.1.1 HP-UX FORTRAN-90` sources. To compile the code, we used `gfortran`, the GNU Fortran 95 compiler.

`PBNCGC` is an implementation of the most frequently used bundle method in NSO, that is, the proximal bundle method. The code includes the constraint handling (bound constraints, linear constraints, and nonlinear/nonsmooth constraints). The quadratic direction-finding problem (15.4) is solved by the subroutine `PLQDF1` implementing dual projected gradient method proposed in [27].

`PNEW` is a bundle-Newton solver for unconstrained and linearly constrained NSO. We used the numerical calculation of the Hessian matrix in our experiments (this can be done automatically). The quadratic direction-finding problem (15.11) is solved by the same subroutine `PLQDF1` [27] like in `PBNCGC`.

`LMBM` is an implementation of a limited memory bundle method specifically developed for large-scale nonsmooth problems. In our experiments we used the adaptive version of the code with the initial number of stored correction pairs (used to form the variable metric update) equal to 7 and the maximum number of stored correction pairs equal to 15. These values have been chosen according to the numerical experiments.

`DGM` is a discrete gradient solver for derivative free optimization. To apply `DGM`, one only needs to be able to compute at every point \mathbf{x} the value of the objective function and the subgradient will be approximated.

`QSM` is a quasisecant solver for nonsmooth, possibly nonconvex minimization. We have used both analytically calculated subgradients and approximated subgradients in our experiments (this can be done automatically by selecting one parameter). In what follows, we denote `QSMa` and `QSMN`, respectively, the corresponding solvers.

All the algorithms but `SolvOpt` were implemented in Fortran77 with double-precision arithmetic. To compile the codes, we used `g77`, the GNU Fortran 77 compiler.

15.3.2 Problems

We consider ten types of problems:

XSC: Extra-small convex problems, $n \leq 20$ ([31, Problems 2.1–2.7, 2.9, 2.22 and 2.23, and 3.4–3.8, 3.10, 3.12, 3.16, 3.19 and 3.20]);

XSNC: Extra-small nonconvex problems ([31, Problems 2.8, 2.10–2.12, 2.14–2.16, 2.18–2.21, 2.24 and 2.25, and 3.1, 3.2, 3.15, 3.17, 3.18 and 3.25]);

SC: Small-scale convex problems, $n = 50$ ([15, Problems 1–5], Problems 2 and 5 in TEST29 [28], and six maximum of quadratic functions [21]);

SNC: Small-scale nonconvex problems ([15, Problems 6–10], and Problems 13, 17 and 22 in TEST29 [28], and six maximum of quadratic functions);

MC and MNC: Medium-scale convex and nonconvex problems, $n = 200$ (see SC and SNC problems);

LC and LNC: Large-scale convex and nonconvex problems, $n = 1000$ (see MC and MNC problems);

XLC and XLNC: Extra-large-scale convex and nonconvex problems, $n = 4000$ (see MC and MNC problems but only two maximum of quadratics with a diagonal matrix);

Problems 2, 5, 13, 17, and 22 in TEST29 are from the software package UFO (Universal Functional Optimization) [28]. The problems were selected so that in all cases all the solvers converged to the same local minimum. However, it is worth mentioning that, in the case of different local minima (i.e. in some nonconvex problems omitted from the test set), solvers LMBM, SolvOpt, and SUBG usually converged to the same local minimum, while PBNCGC, DGM, and QSM converged to a different local minimum. The solver PNEW converged sometimes with the first group and some other times with the second. Moreover, DGM and QSM seem to have an aptitude for finding global or at least smaller local minima than the other solvers. For example, in Problems 3.13 and 3.14 in [31] all the other solvers converged to the minimum reported in [31] but DGM and QSM “converged” to minus infinity.

15.3.3 Termination, Parameters, and Acceptance of the Results

The determination of stopping criteria for different solvers, such that the comparison of different methods is fair, is not a trivial task.

We say that a solver finds the solution with respect to a tolerance $\varepsilon > 0$ if

$$\frac{f_{best} - f_{opt}}{1 + |f_{opt}|} \leq \varepsilon,$$

where f_{best} is a solution obtained with the solver and f_{opt} is the best-known (or optimal) solution.

We fixed the stopping criteria and parameters for the solvers using three different problems from three different problem classes: Problem 2.4 in [31] (XSC), Problem 3.15 in [31] (XSNC), and Problem 3 in [15] with $n = 50$ (SC). With all the solvers we sought the loosest termination parameters such that the results for all the three test problems were still acceptable with respect to the tolerance $\varepsilon = 10^{-4}$. In addition to the usual stopping criteria of the solvers, we terminated the experiments if the elapsed CPU time exceeded half an hour.

We have accepted the results for XS and S problems ($n \leq 50$) with respect to the tolerance $\varepsilon = 5 \times 10^{-4}$. With larger problems ($n \geq 200$), we have accepted the results with the tolerance $\varepsilon = 10^{-3}$. In what follows, we report also the results for all problem classes with respect to the relaxed tolerance $\varepsilon = 10^{-2}$ to have an insight into the reliability of the solvers (i.e. is a failure a real failure or is it just an inaccurate result which could possibly be prevented with a more tight stopping parameter).

With all the bundle-based solvers the distance measure parameter value $\gamma = 0.5$ was used with nonconvex problems. With PBNCGC and LMBM the value $\gamma = 0$ was used with convex problems and, since with PNEW γ has to be positive, $\gamma = 10^{-10}$ was used with PNEW. For those solvers storing subgradients (or approximations of subgradients)—that is, PBNCGC, PNEW, LMBM, DGM, and QSM—the maximum size of the bundle was set to $\min\{n + 3, 100\}$. For all other parameters we used the default settings of the codes.

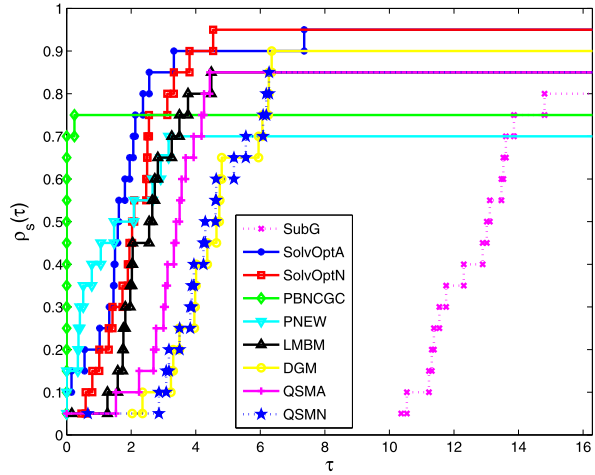
15.3.4 Results

The results are summarized in Figs. 15.1–15.8 and in Table 15.2. The results are analyzed using the performance profiles introduced in [13]. We compare the efficiency of the solvers both in terms of computational times and numbers of function and subgradient evaluations (evaluations for short). In the performance profiles, the value of $\rho_s(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the corresponding solver is the best (it uses least computational time or evaluations) and the value of $\rho_s(\tau)$ at the rightmost abscissa gives the percentage of test problems that the corresponding solver can solve, that is, the reliability of the solver (this does not depend on the measured performance). Moreover, the relative efficiency of each solver can be directly seen from the performance profiles: the higher the particular curve, the better the corresponding solver. For more information on performance profiles, see [13].

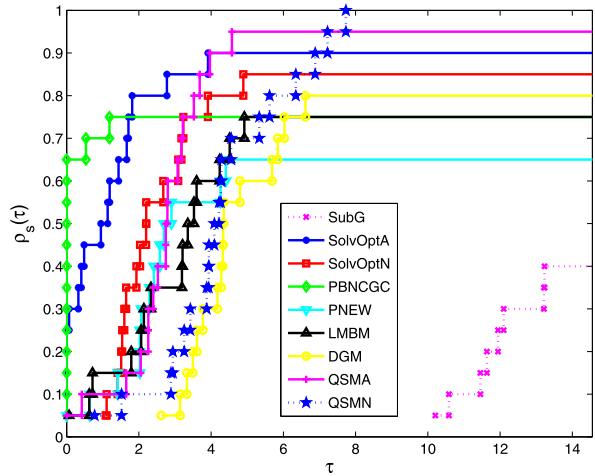
15.3.4.1 Extra-Small Problems

There was not a big difference in the computational times of the different solvers when solving the XS problems. Thus, only the numbers of function and subgradient evaluations are reported in Fig. 15.1.

Fig. 15.1 Evaluations for XS problems (20 problems with $n \leq 20$, $\varepsilon = 5 \times 10^{-4}$)



(a) Convex



(b) Nonconvex

PBNCGC was usually the most efficient solver when comparing the numbers of evaluations. This is, in fact, true for all sizes of problems. Thus, PBNCGC should be a good choice as a solver in case the objective function value and/or the subgradient are expensive to compute. However, PBNCGC failed to achieve the desired accuracy in 25 % of the extra-small problems (both XSC and XSNC) which means that it had almost the worst degree of success in solving these problems.

SUBG is highly unsuitable for nonconvex problems: it failed in 60 % of the problems ($\varepsilon = 5 \times 10^{-4}$, see Fig. 15.1(b)). On the other hand, SolvOpt was one of the most reliable solvers together with QSM in both convex and nonconvex settings although, theoretically, Shor’s r -algorithm is not supposed to solve noncon-

vex problems. `SolvOptA` was also the most efficient method except for `PBNCGC` (especially in the nonconvex case).

Except for `SUBG`, the solvers did not have big differences in the numbers of success in solving XSC or XSNC problems. However, it is noteworthy that the `QSM` computed nonconvex problems more reliably than convex ones.

Most of the failures reported here are, in fact, inaccurate results: all the solvers but `PNEW` succeed in solving equal or more than 95 % of XSC problems with respect to the relaxed tolerance $\varepsilon = 10^{-2}$. The corresponding percentage for XSNC problems was 85 %, although here also `SUBG` failed to solve so many problems.

In XSC problems `PNEW` was the second most efficient solver (see Fig. 15.1(a)). However, it failed to solve 30 % of the convex problems and 35 % of the nonconvex problems. The reason for this relatively large number of failures with `PNEW` is in its sensitivity to the internal parameter `XMAX` (`RPAR`(9) in the code) which is noted also in [30]. If we, instead of only one (default) value, used a selected value for this parameter, also the solver `PNEW` solved 85 % of XSNC problems.

The derivative-free solvers `DGM` and `QSMN` performed similarly in these small-scale problems but `QSMN` was clearly more reliable in the nonconvex case. `SolvOptN` usually used less evaluations than the derivative-free solvers both in XSC and XSNC problems. However, in the nonconvex case, also `SolvOptN` lost out to `QSMN` in reliability.

15.3.4.2 Small-Scale Problems

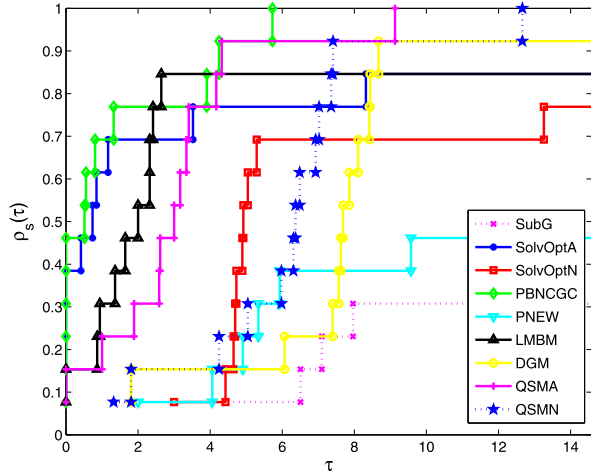
Already with small-scale problems, there was a wide diversity on the computational times of different codes. Moreover, the numbers of evaluations used with solvers were no longer directly comparable with the elapsed computational times. For instance, `PBNCGC` was clearly the winner when comparing the numbers of evaluations (see Figs. 15.2(b) and 15.3(b)). However, when comparing computational times, `SolvOptA` was equally efficient with `PBNCGC` in SC problems (see Fig. 15.2(a)) and `LMBM` was the most efficient solver in SNC problems (see Fig. 15.3(a)).

`SUBG` was clearly the worst solver with respect to both computational times and evaluations in both SC and SNC problems. It was also the most unreliable solver. It solved only about 30 % of the convex and 20 % of the nonconvex problems and it failed in all the quadratic problems.

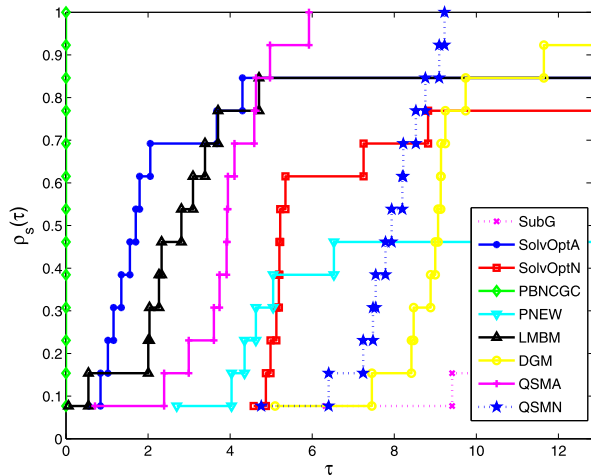
Also the other subgradient solver `SolvOpt` had some difficulties with the accuracy, especially in the nonconvex case. `SolvOptN` solved about 77 % of the convex problems with respect to tolerance $\varepsilon = 5 \times 10^{-4}$ and 92 % with $\varepsilon = 10^{-2}$. For `SolvOptA` the corresponding values were 85 % and 92 %. In the nonconvex case, the values were 64 % vs. 92 % for `SolvOptN` and 71 % vs. 86 % for `SolvOptA`. In other words, `SolvOpt` would have benefited most if we instead of tolerance $\varepsilon = 5 \times 10^{-4}$ had used the relaxed tolerance $\varepsilon = 10^{-2}$ to accept the results. Note, however, that with small-scale problems `SolvOpt` was one of the most reliable solvers.

With the other solvers there were no big differences in solving convex or nonconvex problems apart from `PNEW`: `PNEW` solved about 79 % of the nonconvex

Fig. 15.2 CPU-time and evaluations for SC problems (13 problems with $n = 50$, $\varepsilon = 5 \times 10^{-4}$)



(a) CPU-time

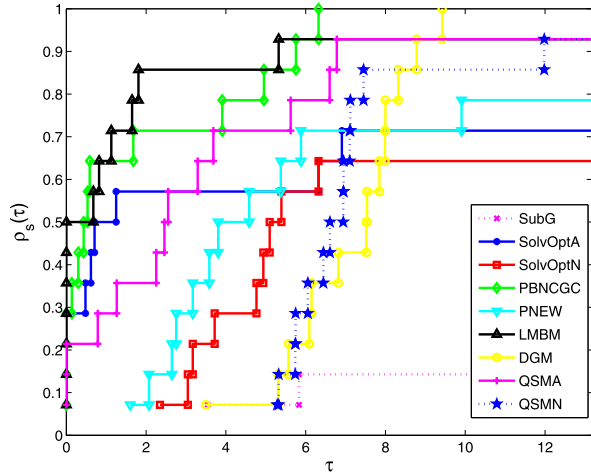


(b) Evaluations

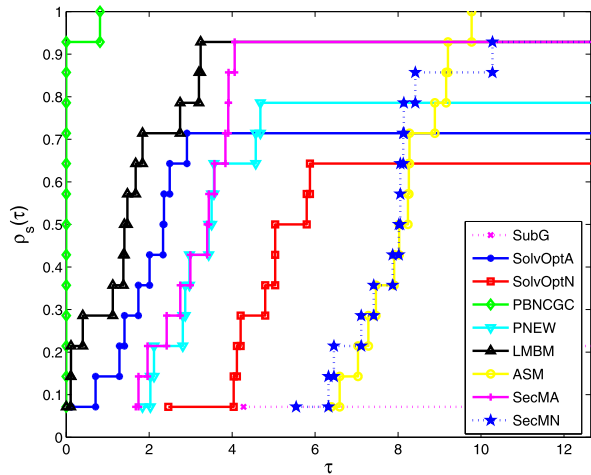
problems and only 46 % of the convex problems. Also LMBM succeeded in solving a little bit more nonconvex than convex problems. In the convex case, PBNCGC, QSMA, and QSMN succeeded in solving all the problems with the desired accuracy. With the relaxed tolerance $\varepsilon = 10^{-2}$ also DGM managed to solve all the problems and all the solvers but PNEW and SUBG succeeded in solving more than 90 % of the problems. In the nonconvex case, PBNCGC and DGM solved all the problems successfully. With a relaxed parameter QSMA and QSMN succeeded as well and all the solvers except PNEW and SUBG managed to solve more than 85 % of the problems.

The solvers DGM and QSMN behaved rather similarly but QSMN was a little bit more efficient both with respect to computational times and evaluations. SolvOptN outperformed these two methods in efficiency but lost clearly in reliability.

Fig. 15.3 CPU-time and evaluations for SNC problems (14 problems with $n = 50$, $\varepsilon = 5 \times 10^{-4}$)



(a) CPU-time

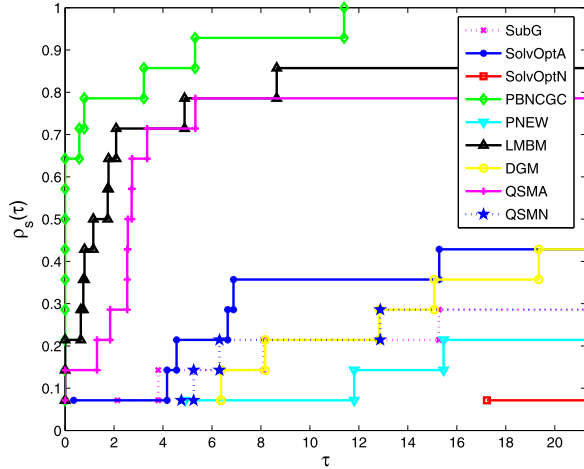


(b) Evaluations

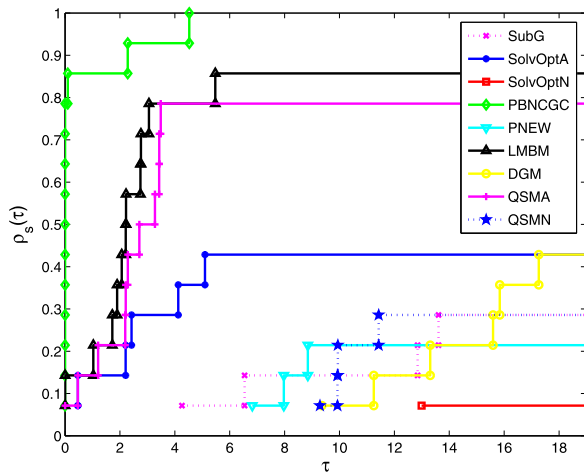
PNEW failed to solve all but one of the convex quadratic problems and succeeded in solving all but one non-quadratic problems. In the nonconvex case PNEW succeeded in solving all the quadratic problems but then it had some difficulties with the other problems. Again, the reason for these failures is in its sensitivity to the internal parameter XMAX.

In [34], PNEW is reported to be very efficient in quadratic problems. Also in our experiments, PNEW was clearly more efficient with the quadratic problems than with the non-quadratic. However, except for some small problems, it was not the most efficient method in any of the cases.

Fig. 15.4 CPU-time and evaluations for LNC problems (14 problems with $n = 1000$, $\varepsilon = 10^{-3}$)



(a) CPU-time



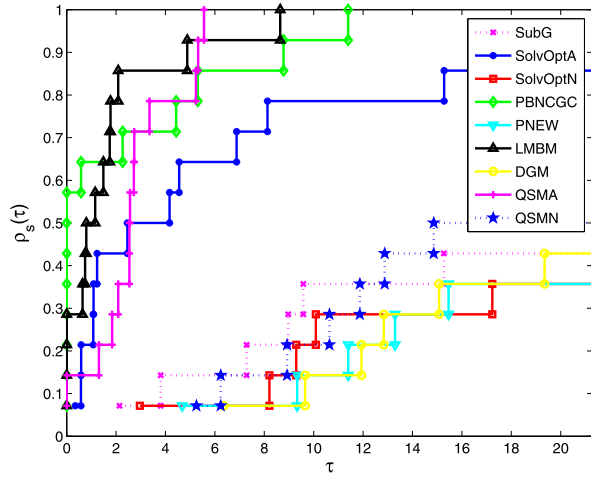
(b) Evaluations

15.3.4.3 Medium and Large-Scale Problems

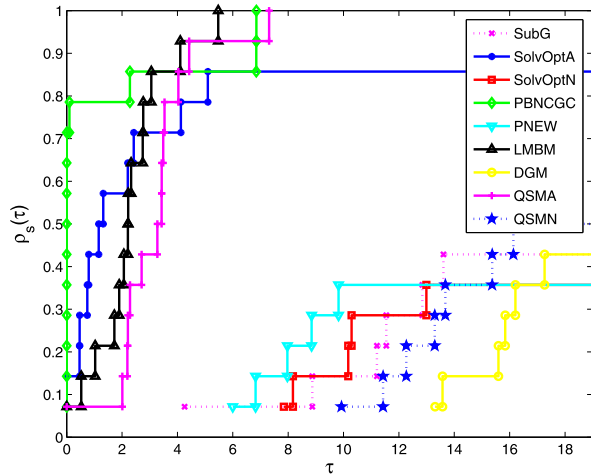
The results for medium and large-scale problems reveal similar trends. Thus, we show here only the results for large problems in Figs. 15.4 and 15.5. More illustrated results also for medium-scale problems can be found in [21].

When solving medium and large-scale problems, the solvers are divided into two groups: the first group consists of more efficient solvers: LMBM, PBNCGC, QSMA, and SolvOptA. The second group consists of solvers using some kind of approximation for subgradients or Hessian, and SUBG. In the nonconvex case (see Fig. 15.4), the inaccuracy of SolvOptA made it slide to the group of less efficient solvers. In Fig. 15.5 illustrating the results with the relaxed tolerance, SolvOptA

Fig. 15.5 CPU-time and evaluations for LNC problems, low accuracy (14 problems with $n = 1000$, $\varepsilon = 10^{-2}$)



(a) CPU-time



(b) Evaluations

is among the more efficient solvers. Nevertheless, its accuracy is not as good as that of the others. At the same time, the successfully solved quadratic problems almost lifted PNEW to the group of more efficient solvers in large-scale nonconvex settings (especially when comparing the numbers of evaluations, see [21]). The similar trend cannot have been seen here, since in LNC problems the time limit was exceeded in all the maximum of quadratic problems with PNEW.

Although PBNGCGC was usually (on 70 % of medium-sized and 60 % of large problems) the most efficient solver tested in the convex case, it was also the one that needed the longest time to compute Problem 3 in [15] (both in medium and large-scale settings). Indeed, an average time used to solve an MC (LC) problem with PBNGCGC was 15.7 (266.0) seconds while with *SolvOptA*, *LMBM*, and *QSMA* they

were 1.3 (22.0), 1.6 (54.5), and 4.9 (98.6) seconds, respectively (the average times are calculated using 9 (7) problems that all the solvers above managed to solve).

In the MNC case, LMBM and PBNCGC were the most efficient solvers. However, also here with PBNCGC there was notable variation in the computational times for different problems while with LMBM all the problems were solved equally efficiently. In LNC settings also the solver QSMA solved the problems quite efficiently (see Figs. 15.4 and 15.5).

The efficiency of PBNCGC is mostly due to its efficiency in quadratic problems: it was the most efficient solver in almost all quadratic problems when comparing the computational times, and superior when comparing the numbers of evaluations. As before, PNEW failed in all but one of the convex quadratic problems.

Besides usually being the most efficient solver, PBNCGC was also the most reliable solver tested in medium-scale settings. In the MC case it was the only solver that succeeded in solving all the problems with the desired accuracy. In the MNC case QSMA was successful as well. With the relaxed tolerance $\varepsilon = 10^{-2}$ also SolvOptA, QSMA, QSMN, and DGM managed to solve all the MC problems, while LMBM and SolvOptN succeeded in solving more than 84 % of the problems. In the MNC case, LMBM, PBNCGC QSMA, QSMN, and DGM solved all the problems with the relaxed tolerance.

SolvOptN had some serious difficulties with the accuracy, especially in non-convex cases. For instance, with the relaxed tolerance SolvOptN solved almost 80 % of the MNC problems while with the tolerance $\varepsilon = 10^{-3}$ less than 30 %. A similar effect could be seen with SolvOptA, although not as pronounced. Naturally, with the LNC problems the difficulties with the accuracy degenerated (see Figs. 15.4 and 15.5).

Also LMBM and QSMA had some difficulties with the accuracy in the LNC case (see Fig. 15.4). With the relaxed tolerance, they solved all LNC problems (see Fig. 15.5). With this tolerance LMBM was clearly the most efficient solver in non-quadratic problems and the computational times of both LMBM and QSMA were comparable with those of PBNCGC in the whole test set.

The solvers PBNCGC, DGM, and QSM were the only solvers which solved two LC problems in which there is only one nonzero element in the subgradient vector (i.e. Problem 1 in [15] and Problem 2 in TEST29 [28]). With the other methods, there were some difficulties already with $n = 50$ and some more with $n = 200$. (Note that for small, medium and large-scale settings, the problems are the same, only the number of variables is changing.) In the case of LMBM these difficulties are easy to explain: the approximation of the Hessian formed during the calculations is dense and, naturally, not even close to the real Hessian in sparse problems. It has been reported [15] that LMBM is best suited for the problems with a dense subgradient vector whose components depend on the current iteration point. This result is in line with the noted result that LMBM solves nonconvex problems very efficiently.

In the LC case PNEW solved all but the above mentioned two problems and the maximum of quadratics problems. The solvers DGM, LMBM, SUBG, and QSMN failed to solve (possible in addition to the two above-mentioned problems) two piecewise linear problems (Problem 2 in [15] and Problem 5 in TEST29 [28]) and QSMA also failed to solve one of them.

Naturally, for the solvers using difference approximation or some other approximation based on the calculation of the function or subgradient values, the number of evaluations (and thus also the computational time) grows enormously when the number of variables increases. Particularly, in large-scale problems the time limit was exceeded with all these solvers in all the maximum of quadratic problems. Thus, the number of failures with these solvers is probably larger than it should be. Nevertheless, if you need to solve a problem where the subgradient is not available, the best solver would probably be `SolvOptN` (only in the convex case) due to its efficiency or `QSMN` due to its reliability.

15.3.4.4 Extra-Large Problems

Finally, we tested the most efficient solvers so far, that is `LMBM`, `PBNCGC`, `QSMA` and `SolvOptA`, using the problems with $n = 4000$. In the convex case, the solver `QSMA`, which has kept a rather low profile until now, was clearly the most efficient method although `PBNCGC` still used the least evaluations. `QSMA` was also the most reliable of the solvers tested (see Fig. 15.6(a)).

In the nonconvex case, `LMBM` and `QSMA` were approximately equally good in computational times, evaluations, and reliability (see Fig. 15.6(b)). Here `PBNCGC` was the most reliable solver, although with the tolerance $\varepsilon = 10^{-2}$ `QSMA` was the only solver that solved all the nonconvex problems. `LMBM` and `PBNCGC` failed in one and `SolvOpt` in two problems.

As before, `LMBM` solved all the problems it could solve in a relatively short time while with all the other solvers there was notable variation in the computational times elapsed for different problems. However, in the convex case, the efficiency of `LMBM` was again ruined by its unreliability.

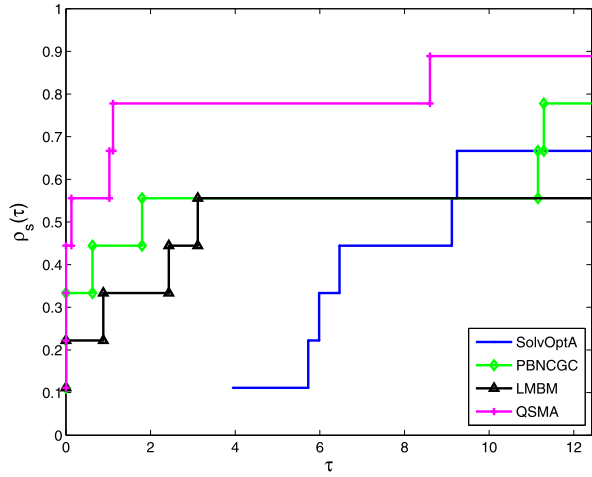
15.3.4.5 Convergence Speed and Number of Success

In this subsection, we first study (experimentally) the convergence speed of the algorithms using one small-scale convex problem (Problem 3 in [15]). The exact minimum value for this function (with $n = 50$) is $-49 \times 2^{1/2} \approx -69.296$.

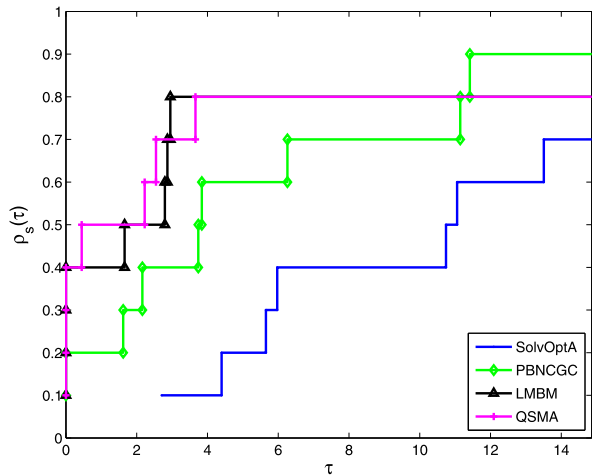
For the limited memory bundle method the rate of convergence has not been studied theoretically. However, at least in this particular problem, the solvers `LMBM` and `PBNCGC` converged at approximately the same rate. Moreover, if we study the number of evaluations, `PBNCGC` and `LMBM` seem to have the fastest converge speed of the solvers tested (see Fig. 15.7(b)) although, theoretically, the proximal bundle method is only linearly convergent.

`SUBG` converged linearly but extremely slowly and `PNEW`, although it finally found the minimum, did not decrease the value of the function in the first 200 evaluations. Naturally, with `PNEW` a large amount of subgradient evaluations are needed to compute the approximative Hessian. The solvers `SolvOptA`, `SolvOptN`, `DGM`, `QSMA`, and `QSMN` took a very big step downwards already in iteration two (see

Fig. 15.6 CPU-times for convex (9 pc.) and nonconvex (10 ps.) XL problems ($n = 4000, \varepsilon = 10^{-3}$)



(a) Convex

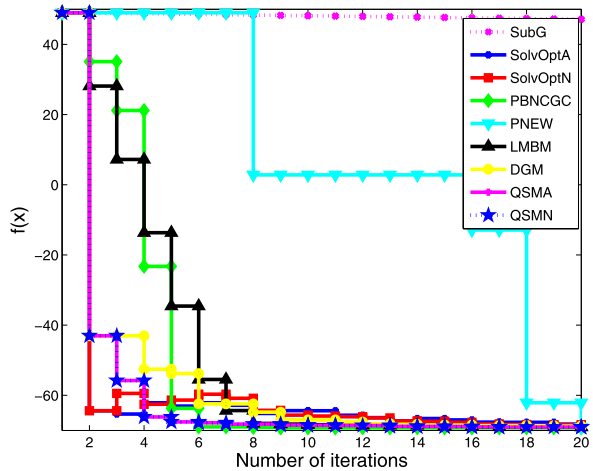


(b) Nonconvex

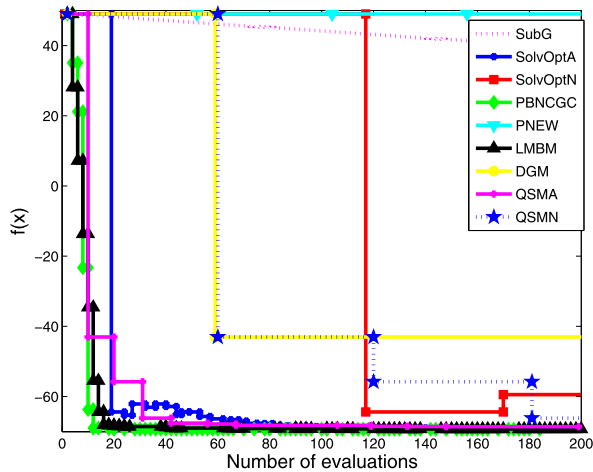
Fig. 15.7(a)). However, they took quite many function evaluations per iteration. In Fig. 15.7 it is easy to see that Shor's r -algorithm (i.e. solvers SolvOptA and SolvOptN) is not a descent method.

In order to see how quickly the solvers reach some specific level, we studied the value of the function equal to -69 . With PBNCGC it took only 8 iterations to go below that level. The corresponding values for other solvers were 17 with QSMA and QSMN, 20 with LMBM and PNEW, and more than 20 with all the other solvers. In terms of function and subgradient evaluations, the values were 18 with PBNCGC, 64 with LMBM, and 133 with SolvOptA. Other solvers needed more than 200 evaluations to go below -69 .

Fig. 15.7 Function values versus iterations (a), and function values versus the number of function and subgradient evaluations (b)



(a) First 20 iterations



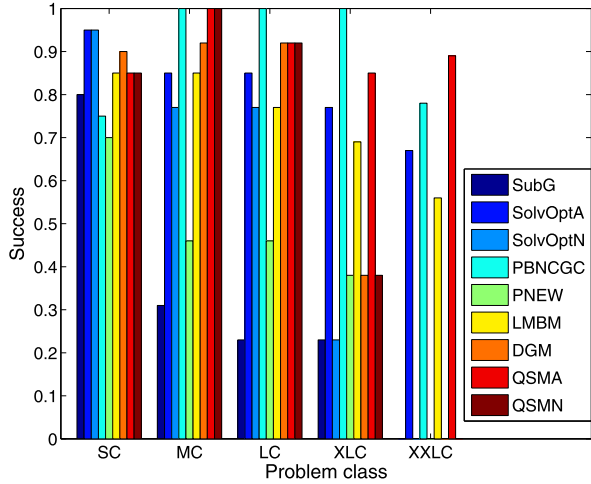
(b) First 200 evaluations

The worst of the solvers were SUBG which took 7382 iterations and 14764 evaluations to reach the desired accuracy and stop, and *SolvOptN* which never reached the desired accuracy (the final value obtained after 42 iterations and 2342 evaluations was -68.915).

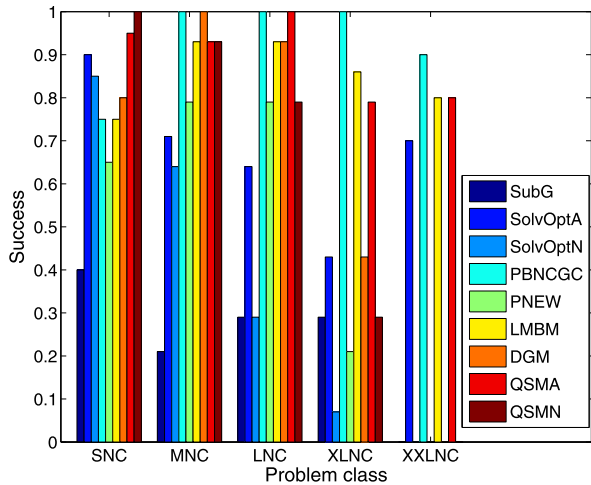
Finally, in Fig. 15.8 we give the proportions of the successfully terminated runs obtained with each solver within the different problem classes. Although we have already said something about the reliability of the solvers, we study the figure to see if the convexity or the number of variables have any significant effect on the success rate of the solvers.

In Fig. 15.8, we see that with both variants of *SolvOpt* the degree of success decreases clearly when the number of variables increases or the problem is noncon-

Fig. 15.8 Proportions of successfully terminated runs within different problem classes: convex problems (a) and nonconvex problems (b)



(a) Convex



(b) Nonconvex

vex. In addition, with the solvers that use approximations to subgradient or Hessian there is a clear drop-out when moving from 200 variables to 1000 variables. At least one reason for this is that with $n = 1000$ the solvers terminated because of the maximum time limit (thus failing to reach the desired accuracy).

DGM and QSMN were reliable methods both with convex and nonconvex problems up to 200 variables, while LMBM and PNEW solved the nonconvex problems more reliably than the convex ones. With PNEW the maximum time limit was exceeded in many cases with $n = 1000$, thus the exception. With PNEW the result could be different if the tuned parameter XMAX was used. With LMBM the result is in harmony with the earlier claims [15] that LMBM works better for more nonlinear functions.

PBNCGC solved small-scale and larger problems in a very reliable way but it was almost the worst solver in extra-small problems. This result has probably nothing to do with the problem's size but more with the different problem classes used.

15.4 Conclusions

We have tested the performance of different nonsmooth optimization solvers in the solution of different nonsmooth problems. The results are summarized in Table 15.2, where we give our recommendations for the “best” solver for different problem classes. Since it is not always unambiguous what the best option is, we give credentials both in the cases where the most efficient (in terms of used computer time) and the most reliable solver are sought out. If there is more than one solver recommended in Table 15.2, the solvers are given in alphabetical order. The parenthesis in the table mean that the solver is not exactly as good as the first one but still a solver to be reckoned with the problem class.

Although in our experiments we got extremely good results with the proximal bundle solver PBNCGC, we cannot say that it is clearly the best method tested. The inaccuracy in extra-small problems, great variations in the computational times occurred in larger problems, and the earlier results obtained make us believe that our test set favored this solver over the others a little bit. Even so, we can say that PBNCGC was one of the best solvers tested and it is especially efficient for the maximum of quadratic and piecewise linear problems.

On the other hand, the limited memory bundle solver LMBM suffered from ill-conditioned test problems in convex small, medium, large and extra-large cases. In the test set there were four problems (out of 13) in which LMBM was known to have difficulties. In addition, LMBM did not beat PBNCGC in any maximum of quadratics problems but in one with $n = 4000$. This, however, is not the inferiority of LMBM but rather the superiority of PBNCGC in these kinds of problems. LMBM was quite reliable in the nonconvex case in all numbers of variables tested and it solved all the problems—even the largest ones—a in relatively short time while, for example, with PBNCGC there was great variation on the computational times of different problems. LMBM works best for (highly) nonlinear functions while for piecewise linear functions it might be a good idea to find another solver.

In convex extra-small problems, the bundle-Newton solver PNEW was the second most efficient solver tested. However, PNEW suffers greatly from the fact that it is very sensitive to the internal parameter XMAX. Already using two values for this parameter (e.g., default value 1000 and the smallest recommended value 2), the results would have been much better and especially the degree of success would have been much higher. The solver has been reported to be best suited for quadratic problems [34] and, indeed, it solved (nonconvex) quadratic problems faster than non-quadratic. However, with $n \geq 50$ it did not beat the other solvers in these problems due to the large approximation of the Hessian matrix required.

Table 15.2 Summation of the results

Problem's type	Problem's size	Seeking for efficiency	Seeking for reliability
Convex	XS	PBNCGC, PNEW ⁽¹⁾ , (SolvOpt(A+N))	DGM, SolvOpt(A+N)
	S, M, L	LMBM ⁽²⁾ , PBNCGC, (QSMA, SolvOptA)	PBNCGC, QSMA
	XL	LMBM ⁽²⁾ , QSMA	QSMA, (PBNCGC)
Nonconvex	XS	PBNCGC, SolvOptA, (QSMA)	QSM(A+N), (SolvOptA)
	S, M, L	LMBM, PBNCGC, (QSMA)	DGM, LMBM, PBNCGC
	XL	LMBM, QSMA	PBNCGC, (LMBM, QSMA)
Piecewise linear or sparse	XS, S	PBNCGC, SolvOptA	PBNCGC, SolvOptA
	M, L, XL	PBNCGC, QSMA ⁽³⁾	DGM, PBNCGC, QSMA
Piecewise quadratic	XS	PBNCGC, PNEW ⁽¹⁾ , (LMBM, SolvOptA)	LMBM, PBNCGC, PNEW ⁽¹⁾ , SolvOptA
	S, M, L, XL	LMBM, PBNCGC, (SolvOptA)	DGM, LMBM, PBNCGC, QSMA
Highly nonlinear	XS	LMBM, PBNCGC, SolvOptA	LMBM, QSMA, SolvOptA
	S	LMBM, PBNCGC	LMBM, PBNCGC, QSMA
	M, L, XL	LMBM	LMBM, QSMA
Function evaluations are expensive	XS	PBNCGC, (PNEW ⁽¹⁾ , SolvOptA)	QSMA, SolvOptA
	S, M, L, XL	PBNCGC, (LMBM ⁽⁴⁾ , SolvOptA)	PBNCGC, (LMBM ⁽⁴⁾ , QSMA)
Subgradients are not available	XS	SolvOptN	QSMN, SolvOptN ⁽⁵⁾ , (DGM)
	S, M	SolvOptN, QSMN	DGM, QSMN
	L	QSMN, (DGM)	DGM, QSMN

The standard subgradient solver SUBG is usable only for extra-small convex problems: the degree of success was 80 % in XSC, otherwise it was less than 40 %. In addition, the implementations of Shor's r -algorithm SolvOptA and SolvOptN did their best in extra-small problems (also in the nonconvex case!). Nevertheless, SolvOptA solved also medium, large and even extra-large problems (convex) rather efficiently. In larger nonconvex problems these methods suffered from inaccuracy.

Thus, when comparing the reliability in medium-scale settings, it seems that one should select PBNCGC for convex problems while LMBM is good for nonconvex problems. On the other hand, the quasi-secant solver QSMA was reliable and efficient both in convex and nonconvex medium-sized problems. However, with QSMA there was some variation on the computational times of different problems (not as much as PBNCGC, though) while LMBM solved all the problems in a relatively short time.

The solvers using discrete gradients, that is, the discrete gradient solver DGM and the quasisecant solver with discrete gradients, QSMN, usually lost out in efficiency to the solvers using analytical subgradients. However, in extra-small and small-scale problems the differences were not significant and the reliability of DGM and QSMN seems to be very good both with convex and nonconvex problems. Moreover in the case of highly nonconvex functions (supposing that you seek for global optimum) DGM or QSM (either with or without subgradients) would be a good choice, since these methods tend to jump over the narrow local minima.

Acknowledgements We would like to acknowledge professors A. Kuntsevich and F. Kappel for providing Shor's r -algorithm in their web-page as well as professors L. Lukšan and J. Vlček for providing the bundle-Newton algorithm. The work was financially supported by the University of Turku (Finland) and the University of Ballarat (Australia) and the Australian Research Council.

References

1. Äyrämö S (2006) Knowledge mining using robust clustering. PhD thesis, University of Jyväskylä
2. Bagirov AM, Ganjehlou AN (2010) A quasisecant method for minimizing nonsmooth functions. *Optim Methods Softw* 25(1):3–18
3. Bagirov AM, Karasözen B, Sezer M (2008) Discrete gradient method: Derivative-free method for nonsmooth optimization. *J Optim Theory Appl* 137(2):317–334
4. Beck A, Teboulle M (2003) Mirror descent and nonlinear projected subgradient methods for convex optimization. *Oper Res Lett* 31(3):167–175
5. Ben-Tal A, Nemirovski A (2005) Non-Euclidean restricted memory level method for large-scale convex optimization. *Math Program* 102(3):407–456
6. Bihain A (1984) Optimization of upper semidifferentiable functions. *J Optim Theory Appl* 44(4):545–568
7. Bradley PS, Fayyad UM, Mangasarian OL (1999) Mathematical programming for data mining: Formulations and challenges. *INFORMS J Comput* 11(3):217–238
8. Burke JV, Lewis AS, Overton ML (2005) A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM J Optim* 15(3):751–779
9. Byrd RH, Nocedal J, Schnabel RB (1994) Representations of quasi-Newton matrices and their use in limited memory methods. *Math Program* 63(2):129–156
10. Clarke FH (1983) Optimization and nonsmooth analysis. Wiley, New York
11. Clarke FH, Ledyaev YS, Stern RJ, Wolenski PR (1998) Nonsmooth analysis and control theory. Springer, New York
12. Demyanov VF, Bagirov AM, Rubinov AM (2002) A method of truncated codifferential with application to some problems of cluster analysis. *J Glob Optim* 23(1):63–80
13. Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. *Math Program* 91(2):201–213
14. Gaudioso M, Monaco MF (1992) Variants to the cutting plane approach for convex nondifferentiable optimization. *Optimization* 25(1):65–75
15. Haarala M, Miettinen K, Mäkelä MM (2004) New limited memory bundle method for large-scale nonsmooth optimization. *Optim Methods Softw* 19(6):673–692
16. Haarala N, Miettinen K, Mäkelä MM (2007) Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Math Program* 109(1):181–205
17. Haslinger J, Neittaanmäki P (1996) Finite element approximation for optimal shape, material and topology design, 2nd edn. Wiley, Chichester

18. Hiriart-Urruty J-B, Lemaréchal C (1993) Convex analysis and minimization algorithms. II. Advanced theory and bundle methods. Springer, Berlin
19. Kappel F, Kuntsevich AV (2000) An implementation of Shor's r -algorithm. *Comput Optim Appl* 15(2):193–205
20. Kärkkäinen T, Heikkola E (2004) Robust formulations for training multilayer perceptrons. *Neural Comput* 16(4):837–862
21. Karmita N, Bagirov A, Mäkelä MM (2009) Empirical and theoretical comparisons of several nonsmooth minimization methods and software. TUCS Technical Report 959, Turku Centre for Computer Science, Turku. Available online <http://tucs.fi/publications/insight.php?id=tKaBaMa09a>
22. Karmita N, Bagirov A, Mäkelä MM (2012) Comparing different nonsmooth minimization methods and software. *Optim Methods Softw* 27(1):131–153. doi:10.1080/10556788.2010.526116
23. Kiwiel KC (1985) Methods of descent for nondifferentiable optimization. Lecture notes in mathematics, vol 1133. Springer, Berlin
24. Kiwiel KC (1990) Proximity control in bundle methods for convex nondifferentiable minimization. *Math Program* 46(1):105–122
25. Kuntsevich A, Kappel F (1997) SolvOpt – the solver for local nonlinear optimization problems. Graz. <http://www.uni-graz.at/imawww/kuntsevich/solvopt/>
26. Lemaréchal C (1989) Nondifferentiable optimization. In: Nemhauser GL, Rinnooy Kan AHG, Todd MJ (eds) Optimization. North-Holland, Amsterdam, pp 529–572
27. Lukšan L (1984) Dual method for solving a special problem of quadratic programming as a subproblem at linearly constrained nonlinear minimax approximation. *Kybernetika* 20(6):445–457
28. Lukšan L, Tůma M, Šiška M, Vlček J, Ramešová N (2002) Ufo 2002: Interactive system for universal functional optimization. Technical report V-883, Academy of Sciences of the Czech Republic, Prague
29. Lukšan L, Vlček J (1998) A bundle-Newton method for nonsmooth unconstrained minimization. *Math Program* 83(3):373–391
30. Lukšan L, Vlček J (2000) NDA: Algorithms for nondifferentiable optimization. Technical report V-797, Academy of Sciences of the Czech Republic, Prague
31. Lukšan L, Vlček J (2000) Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical report V-798, Academy of Sciences of the Czech Republic, Prague
32. Mäkelä MM (2002) Survey of bundle methods for nonsmooth optimization. *Optim Methods Softw* 17(1):1–29
33. Mäkelä MM (2003) 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 B13/2003, University of Jyväskylä, Jyväskylä
34. Mäkelä MM, Miettinen M, Lukšan L, Vlček J (1999) Comparing nonsmooth nonconvex bundle methods in solving hemivariational inequalities. *J Glob Optim* 14(2):117–135
35. Mäkelä MM, Neittaanmäki P (1992) Nonsmooth optimization: Analysis and algorithms with applications to optimal control. World Scientific, River Edge
36. Mistakidis ES, Stavroulakis GE (1998) Nonconvex optimization in mechanics. Algorithms, heuristics and engineering applications by the FEM. Kluwer, Dordrecht
37. Moreau JJ, Panagiotopoulos PD, Strang G (eds) (1988) Topics in nonsmooth mechanics. Birkhäuser, Basel
38. Outrata J, Kočvara M, Zowe J (1998) Nonsmooth approach to optimization problems with equilibrium constraints. Theory, applications and numerical results. Kluwer, Dordrecht
39. Robinson SM (1999) Linear convergence of epsilon-subgradient descent methods for a class of convex functions. *Math Program* 86(1):41–50
40. Sagastizábal C, Solodov M (2005) An infeasible bundle method for nonsmooth convex constrained optimization without a penalty function or a filter. *SIAM J Optim* 16(1):146–169

41. Schramm H, Zowe J (1992) A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM J Optim* 2(1):121–152
42. Shor NZ (1985) *Minimization methods for non-differentiable functions*. Springer, Berlin
43. Uryasev SP (1991) New variable-metric algorithms for nondifferentiable optimization problems. *J Optim Theory Appl* 71(2):359–388
44. Vlček J, Lukšan L (2001) Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. *J Optim Theory Appl* 111(2):407–430