Napsu Karmitsa | Adil Bagirov | Marko M. Mäkelä

Empirical and Theoretical Comparisons of Several Nonsmooth Minimization Methods and Software

Turku Centre for Computer Science

TUCS Technical Report
No 959, October 2009

Empirical and Theoretical Comparisons of Several Nonsmooth Minimization Methods and Software

Napsu Karmitsa
Department of Mathematics
University of Turku
FI-20014 Turku, Finland
napsu@karmitsa.fi
Adil Bagirov
Centre for Informatics and Applied Optimization
School of Information Technology and Mathematical Sciences
University of Ballarat
University Drive, Mount Helen, PO Box 663
Ballarat, VIC 3353, Australia.
a.bagirov@ballarat.edu.au
Marko M. Mäkelä
Department of Mathematics
University of Turku
FI-20014 Turku, Finland
makela@utu.fi

TUCS Technical Report
No 959, October 2009

Abstract

The most of nonsmooth optimization methods may be divided in two main groups: subgradient methods and bundle methods. Usually, when developing new algorithms and testing them, the comparison is made between similar kinds of methods. In this report we test and compare both different bundle methods and different subgradient methods as well as some methods which may be considered as hybrids of these two and/or some others. All the solvers tested are so-called general black box methods which, at least in theory, can be used to solve almost all kinds of problems. The test set included large amount of different unconstrained nonsmooth minimization problems. That is, for instance, convex and nonconvex problems, piecewise linear and quadratic problems and problems with different sizes. The aim of this work is not to foreground some method over the others but to get some kind of insight which kind of method to select for certain types of problems.

Keywords: Nondifferentiable programming, bundle methods, subgradient methods, numerical performance.

TUCS Laboratory
Applied Mathematics

1 Introduction

We compare different nonsmooth optimization (NSO) methods for solving unconstrained optimization problems of the form

$$\begin{cases} \text{minimize} & f(\boldsymbol{x}) \\ \text{such that} & \boldsymbol{x} \in \mathbb{R}^n, \end{cases} \tag{P}$$

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

Problems of type (P) are encountered in many application areas: for instance, in economics [41], mechanics [40], engineering [39], control theory [12], optimal shape design [19], data mining [1, 8] and in particular cluster analysis [13], and machine learning [22].

The most of methods for solving these problems may be divided in two main groups: subgradient (see e.g. [5, 6, 45, 46]) and bundle methods (see e.g. [15, 20, 24, 31, 34, 37, 43, 44]). Both of these methods have their own supporters. Usually, when developing new methods and doing some numerical experiments with them, the researchers compare the new method with similar kinds of methods. That is, bundle methods are compared with bundle methods and subgradient methods are compared with other subgradient methods. Moreover, it is quite common that the test set used is rather concise (sometimes only a couple of problems), which naturally does not give a good impression of how the algorithm would perform in different kinds of problems.

In this report we compare both different subgradient methods and different bundle methods, as well as some methods that lie between these two. Moreover, we use a broad test settings including different classes of nonsmooth problems. The methods included in our tests are the following:

- *Subgradient methods*:
  - standard subgradient method [45],
  - Shor's r-algorithm [21, 26, 45],
- *Bundle methods*:
  - proximal bundle method [37],
  - bundle-Newton method [30],
- *Hybrid methods*:
  - limited memory bundle method [17, 18],
  - discrete gradient method [4] and
  - quasi-secant method [2, 3].

All the solvers tested are so-called general black box methods and, naturally, can not 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, possible, one arbitrary subgradient (generalized gradient [11]) 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 which kind of method to select for certain types of problems. Suppose, for instance, that you want to minimize a problem known to be nonconvex and nonsmooth with 200 variables. In this work, we are going to analyze which is the best method to use.

The report is organized as follows. Section 2 introduces the NSO methods tested and compared. The results of the numerical experiments are presented and discussed in Section 3 and Section 4 concludes the report and gives our credentials for good-performing algorithms for different problem classes.

In what follows we denote by $\|\cdot\|$ the Euclidean norm in $\mathbb{R}^n$ and by $\boldsymbol{a}^T\boldsymbol{b}$ the inner product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ (bolded symbols are used for vectors). The *subdifferential* $\partial f(\boldsymbol{x})$ [11] of a locally Lipschitz continuous function $f : \mathbb{R}^n \to \mathbb{R}$ at any point $\boldsymbol{x} \in \mathbb{R}^n$ is given by

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

where "conv" denotes the convex hull of a set. Each vector $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$ is called a *subgradient*. The point $\boldsymbol{x}^* \in \mathbb{R}^n$ is called *substationary* if $\boldsymbol{0} \in \partial f(\boldsymbol{x}^*)$. Substationarity is a necessary condition for local optimality and, in the convex case, it is also sufficient for global optimality.

2 Methods

In this section we give short descriptions of the methods to be compared. The implementational details are given in Section 3 and in the references. In what follows (if not stated otherwise), we assume that at every point $\boldsymbol{x}$ we can evaluate the value of the objective function $f(\boldsymbol{x})$ and an arbitrary subgradient $\boldsymbol{\xi}$ from the subdifferential $\partial f(\boldsymbol{x})$.

2.1 Standard Subgradient Method

The first method to be considered here is the cornerstone of NSO: standard subgradient method [45].

The idea behind subgradient methods (Kiev methods) is to generalize smooth methods (e.g. steepest descent method) by replacing the gradient with an arbitrary subgradient. Therefore, the iteration formula for these methods is

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

where $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$ 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. [27]).

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

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

An extensive overview of various subgradient methods can be found in [45].

2.2 Shor's r-algorithm (Space Dilation Method)

Next we shortly describe the ideas of 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 steepest descent and conjugate gradient method.

Let f be a convex function on $\mathbb{R}^n$. Shor's r-algorithm is given as follows:

```
PROGRAM Shor's r-algorithm
  INITIALIZE x_0 ∈ R^n, β ∈ (0,1) , B_1 = I, and t_1 > 0;
  Compute ξ_0 ∈ ∂f(x_0) and x_1 = x_0 − t_1 ξ_0;
  Set ξ̄_1 = ξ_0 and k = 1;
  WHILE the termination condition is not met
    Compute ξ_k ∈ ∂f(x_k) and ξ*_k = B_k^T ξ_k;
    Calculate r_k = ξ*_k − ξ̄_k and s_k = r_k/‖r_k‖;
    Compute B_{k+1} = B_k R_β(s_k), where R_β(s_k) = I + (β − 1)s_k s_k^T
      is the space dilation matrix;
    Calculate ξ̄_{k+1} = B_{k+1}^T ξ_k;
    Choose a step size t_{k+1};
    Set x_{k+1} = x_k − t_k B_{k+1} ξ̄_{k+1} and k = k + 1;
  END WHILE
  RETURN final solution x_k;
END Shor's r-algorithm
```

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 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 Shor's r-algorithm [45]. In the nonconvex case, if the objective function is coercive under some additional assumptions, the r-algorithm is convergent to isolated local minimizers [45].

2.3 Proximal Bundle Method (PBM)

In this subsection we describe the ideas of the proximal bundle method (PBM) for nonsmooth and nonconvex minimization. For more details we refer to [25], [37] and [44].

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 $\boldsymbol{x}_k$ and some trial points $\boldsymbol{y}_j \in \mathbb{R}^n$ (from past iterations) and subgradients $\boldsymbol{\xi}_j \in \partial f(\boldsymbol{y}_j)$ for $j \in J_k$, where the index set J_k is a nonempty subset of $\{1, \dots, k\}$.

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

$$\hat{f}_k(\boldsymbol{x}) = \max_{j \in J_k} \{f(\boldsymbol{y}_j) + \boldsymbol{\xi}_j^T(\boldsymbol{x} - \boldsymbol{y}_j)\}. \tag{1}$$

This model can be written in an equivalent form

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

where

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

is a so-called *linearization error*. If f is convex, then $\alpha_j^k \geq 0$ for all $j \in J_k$ and $\hat{f}_k(\boldsymbol{x}) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^n$. In other words, the cutting-plane model $\hat{f}_k$ is an under estimate for f and the nonnegative linearization error α_j^k measures how good an approximation the model is to the original problem. In the nonconvex case, these facts are not valid anymore and thus the linearization error α_j^k can be replaced by so called *subgradient locality measure* (cf. [24])

$$\beta_j^k = \max\{|\alpha_j^k|, \gamma\|\boldsymbol{x}_k - \boldsymbol{y}_j\|^2\}, \tag{2}$$

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 $\min_{\boldsymbol{x} \in K} \hat{f}_k(\boldsymbol{x}) \leq f(\boldsymbol{x}_k)$.

The descent direction is calculated by

$$\boldsymbol{d}_k = \operatorname{argmin}_{\boldsymbol{d}\in\mathbb{R}^n}\{\hat{f}_k(\boldsymbol{x}_k+\boldsymbol{d}) + \frac{1}{2}u_k\boldsymbol{d}^T\boldsymbol{d}\}, \tag{3}$$

where the stabilizing term $\frac{1}{2}u_k\boldsymbol{d}^T\boldsymbol{d}$ guarantees the existence of the solution $\boldsymbol{d}_k$ 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 x_k (see e.g. [25, 37, 44]).

In order to determine the step size into the search direction $\boldsymbol{d}_k$, PBM uses so-called *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

$$f(\boldsymbol{x}_k + t_L^k\boldsymbol{d}_k) \leq f(\boldsymbol{x}_k) + m_L t_L^k v_k, \tag{4}$$

where v_k is the predicted amount of descent

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

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

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_L^k\boldsymbol{d}_k \qquad \text{and} \qquad \boldsymbol{y}_{k+1} = \boldsymbol{x}_{k+1}. \tag{5}$$

Otherwise, if (4) holds but $0 < t_L^k < \bar{t}$, we take a *short serious step*

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

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

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k \qquad \text{and} \qquad \boldsymbol{y}_{k+1} = \boldsymbol{x}_k + t_R^k\boldsymbol{d}_k, \tag{6}$$

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

$$-\beta_{k+1}^{k+1} + \boldsymbol{\xi}_{k+1}^T\boldsymbol{d}_k \geq m_R v_k. \tag{7}$$

In short serious steps and null steps there exists discontinuity in the gradient of f. Then the requirement (7) ensures that $\boldsymbol{x}_k$ and $\boldsymbol{y}_{k+1}$ lie on the opposite sides of this discontinuity and the new subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$ 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.

The pseudo-code of general bundle method is the following:

```
PROGRAM bundle method
  INITIALIZE x_1 ∈ R^n, ξ_1 ∈ ∂f(x_1), J_1, v_1, and ε_s > 0;
  Set k = 1;
  WHILE the termination condition |v_k| ≤ ε_s is not met
    Generate the search direction d_k;
    Find step sizes t_L^k and t_R^k;
    Update x_k, v_k and J_k;
    Set k = k + 1;
    Evaluate f(x_k) and ξ_k ∈ ∂f(x_k);
  END WHILE
  RETURN final solution x_k;
END bundle method
```

Under the upper semi-smoothness assumption [7] PBM can be proved to be globally convergent for locally Lipschitz continuous objective functions, which are not necessarily differentiable or convex (see e.g. [25, 37]). 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 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 [24], which accumulates information from the previous iterations. The convergence rate of PBM is linear for convex functions [42] and for piecewise linear problems PBM achieves a finite convergence [44].

2.4 Bundle Newton Method (BNEW)

Next we describe the main ideas of the second order bundle-Newton method (BNEW) [30]. We suppose that at each $\boldsymbol{x} \in \mathbb{R}^n$ we can evaluate, in addition to the function value and an arbitrary subgradient $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$, also an $n \times n$ symmetric matrix $G(\boldsymbol{x})$ approximating the Hessian matrix $\nabla^2 f(\boldsymbol{x})$. Now, instead of piecewise linear cutting-pane model (1) we introduce a piecewise quadratic model of the form

$$\tilde{f}_k(\boldsymbol{x}) = \max_{j \in J_k} \{ f(\boldsymbol{y}_j) + \boldsymbol{\xi}_j^T(\boldsymbol{x} - \boldsymbol{y}_j) + \frac{1}{2}\varrho_j(\boldsymbol{x} - \boldsymbol{y}_j)^T G_j(\boldsymbol{x} - \boldsymbol{y}_j)\}, \tag{8}$$

where $G_j = G(\boldsymbol{y}_j)$ and $\varrho_j \in [0, 1]$ is some damping parameter. The model (8) can be written equivalently as

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

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

$$\alpha_j^k = f(\boldsymbol{x}_k) - f(\boldsymbol{y}_j) - \boldsymbol{\xi}_j^T(\boldsymbol{x}_k - \boldsymbol{y}_j) - \frac{1}{2}\varrho_j(\boldsymbol{x}_k - \boldsymbol{y}_j)^T G_j(\boldsymbol{x}_k - \boldsymbol{y}_j). \tag{9}$$

Note that now, even in the convex case, α_j^k might be negative. Therefore we replace the linearization error (9) by the subgradient locality measure (2) and we remain the property $\min_{\boldsymbol{x}\in\mathbb{R}^n} \tilde{f}_k(\boldsymbol{x}) \leq f(\boldsymbol{x}_k)$ (see [30]).

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

$$\boldsymbol{d}_k = \text{argmin}_{\boldsymbol{d}\in\mathbb{R}^n}\{\tilde{f}_k(\boldsymbol{x}_k + \boldsymbol{d})\}. \tag{10}$$

The line search procedure of BNEW follows the same principles than in PBM (see Section 2.3). The only remarkable difference occurs in the termination condition for short and null steps. In other words, (7) 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

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

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

The pseudo-code for the method is the same as for PBM (see Section 2.3). Under the upper semi-smoothness assumption [7] BNEW can be proved to be globally convergent for locally Lipschitz continuous objective functions. For strongly convex functions, the convergence rate of BNEW is superlinear [30].

2.5 Limited Memory Bundle Method (LMBM)

In this subsection, we very shortly describe the limited memory bundle algorithm (LMBM) [16, 17, 18, 23] for solving general, possibly nonconvex, large-scale NSO problems. The method is a hybrid of the variable metric bundle methods [31, 47] and the limited memory variable metric methods (see e.g. [10]), 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 $\boldsymbol{d}_k$ is calculated using a limited memory approach. That is,

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

where $\tilde{\boldsymbol{\xi}}_k$ 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 $\boldsymbol{d}_k$ is calculated using the limited memory approach.

LMBM uses the original subgradient $\boldsymbol{\xi}_k$ after the serious step (cf. (5)) and the aggregate subgradient $\tilde{\boldsymbol{\xi}}_k$ after the null step (cf. (6)) for direction finding (i.e. we set $\tilde{\boldsymbol{\xi}}_k = \boldsymbol{\xi}_k$ 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

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

Here $\boldsymbol{\xi}_m \in \partial f(\boldsymbol{x}_k)$ is the current subgradient (m denotes the index of the iteration after the latest serious step, i.e. $\boldsymbol{x}_k = \boldsymbol{x}_m$), $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$ is the auxiliary subgradient, and $\tilde{\boldsymbol{\xi}}_k$ is the current aggregate subgradient from the previous iteration ($\tilde{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1$). In addition, β_{k+1} is the current subgradient locality measure (cf. (2)) and $\tilde{\beta}_k$ is the current aggregate subgradient locality measure ($\tilde{\beta}_1 = 0$). The resulting aggregate subgradient $\tilde{\boldsymbol{\xi}}_{k+1}$ and aggregate subgradient locality measure $\tilde{\beta}_{k+1}$ are computed from the formulae

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

The line search procedure used in LMBM is rather similar to that used in PBM (see Section 2.3). However, due to the simple aggregation procedure above only one trial point $\boldsymbol{y}_{k+1} = \boldsymbol{x}_k + t_R^k \boldsymbol{d}_k$ and a corresponding subgradient $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{y}_{k+1})$ need to be stored.

As a stopping parameter, we use the value

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

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 PBM).

In LMBM both the limited memory BFGS (L-BFGS) and the limited memory SR1 (L-SR1) update formulae [10] 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 [16, 17, 18]).

The pseudo-code of LMBM is the following:

```
PROGRAM LMBM
  INITIALIZE x_1 ∈ R^n, ξ_1 ∈ ∂f(x_1), and ε_s > 0;
  Set k = 1 and d_1 = −ξ_1;
  WHILE the termination condition w_k ≤ ε_s is not met
    Find step sizes t_L^k and t_R^k;
    Update x_k to x_{k+1};
    Evaluate f(x_{k+1}) and ξ_{k+1} ∈ ∂f(x_k + t_R^k d_k);
    IF t_L^k > 0 THEN
      Compute the search direction d_k using ξ_{k+1} and L-BFGS
        update;
    ELSE
      Compute the aggregate subgradient ξ̃_{k+1};
      Compute the search direction d_k using ξ̃_{k+1} and L-SR1
        update;
    END IF
    Set k = k + 1;
  END WHILE
  RETURN final solution x_k;
END LMBM
```

Under the upper semi-smoothness assumption [7] LMBM can be proved to be globally convergent for locally Lipschitz continuous objective functions [16, 18].

2.6 Discrete Gradient Method (DGM)

Next we briefly describe the discrete gradient method (DGM). More details can be found in [4]. The idea of DGM is to hybridize derivative free methods with bundle methods. In contrast with bundle methods, which require the computation of a single subgradient of the objective function at each trial point, 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

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

the sphere of the unit ball and by

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

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

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

be a set of all vertices of the unit hypercube in $\mathbb{R}^n$. We take any $\boldsymbol{g} \in S_1$, $\boldsymbol{e} \in G$, $z \in P$, a positive number $\alpha \in (0, 1]$, and we compute $i =$

argmax $\{|g_j|,\ j = 1, \ldots, n\}$. For $\boldsymbol{e} \in G$ we define the sequence of n vectors $\boldsymbol{e}^j(\alpha) = (\alpha e_1, \alpha^2 e_2, \ldots, \alpha^j e_j, 0, \ldots, 0)$ $j = 1, \ldots, n$ and for $\boldsymbol{x} \in \mathbb{R}^n$ and $\lambda > 0$, we consider the points

$$\boldsymbol{x}_0 = \boldsymbol{x} + \lambda \boldsymbol{g}, \qquad \boldsymbol{x}_j = \boldsymbol{x}_0 + z(\lambda)\boldsymbol{e}^j(\alpha), \qquad j = 1, \ldots, n.$$

DEFINITION 2.1. The *discrete gradient* of the function f at the point $\boldsymbol{x} \in \mathbb{R}^n$ is the vector $\Gamma^i(\boldsymbol{x}, \boldsymbol{g}, \boldsymbol{e}, z, \lambda, \alpha) = (\Gamma^i_1, \ldots, \Gamma^i_n) \in \mathbb{R}^n$ with the following coordinates:

$$\Gamma^i_j = \left[z(\lambda)\alpha^j e_j)\right]^{-1}\left[f(\boldsymbol{x}_j) - f(\boldsymbol{x}_{j-1})\right], \qquad j = 1, \ldots, n,\ j \neq i,$$
$$\Gamma^i_i = (\lambda g_i)^{-1}\left[f(\boldsymbol{x} + \lambda \boldsymbol{g}) - f(\boldsymbol{x}) - \lambda \sum_{j=1, j\neq i}^{n} \Gamma^i_j g_j\right].$$

It has been proved in [4] that the closed convex set of discrete gradients

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

is an approximation to the subdifferential $\partial f(\boldsymbol{x})$ 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 $D_0(\boldsymbol{x}, \lambda)$ is not easy, and therefore, in 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 inner iteration. We start by describing the direction finding procedure. 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 $\boldsymbol{v}_1 = \Gamma^i(\boldsymbol{x}, \boldsymbol{g}_1, \boldsymbol{e}, z, \lambda, \alpha)$ with respect to any initial direction $\boldsymbol{g}_1 \in S_1$ and we set the initial bundle of discrete gradients $\bar{D}_1(\boldsymbol{x}) = \{\boldsymbol{v}_1\}$. Then we compute the vector

$$\boldsymbol{w}_l = \mathrm{argmin}_{\,\boldsymbol{w} \in \bar{D}_l(\boldsymbol{x})} \|\boldsymbol{w}\|^2,$$

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

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

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

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

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

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

and continue the direction finding procedure with $l = l+1$. Note that, at each subiteration the approximation of the subdifferential $\partial f(\boldsymbol{x})$ is improved. It has been proved in [4] that the direction finding procedure is terminating.

When the descent direction $\boldsymbol{d}_{k_s}$ has been found, we need to compute the next (inner) iteration point $\boldsymbol{x}_{k_{s+1}} = \boldsymbol{x}_{k_s} + t_{k_s}\boldsymbol{d}_{k_s}$, where the step size t_{k_s} is defined as

$$t_{k_s} = \text{argmax}\ \{t \geq 0 \mid f(\boldsymbol{x}_{k_s} + t\boldsymbol{d}_{k_s}) - f(\boldsymbol{x}_{k_s}) \leq -c_2 t \|\boldsymbol{v}_{k_s}\|\},$$

with given $c_2 \in (0, c_1]$.

The pseudo-code of DGM is the following:

```
PROGRAM DGM
  INITIALIZE x_1 ∈ R^n and k = 1;
  OUTER ITERATION
    Set s = 1 and x_{k_s} = x_k;
    WHILE the termination condition is not met
      INNER ITERATION
        Compute the vector v_{k_s} = argmin_{v∈D̄(x_{k_s})} ||v||^2,
          where D̄(x_{k_s}) is a set of discrete subgradients.
        IF ||v_{k_s}|| ≤ δ_k with δ_k > 0 s.t. δ_k ↘ 0 when k → ∞ THEN
          Set x_{k+1} = x_{k_s} and k = k + 1;
          Go to the next OUTER ITERATION;
        ELSE
          Compute the descent direction d_k = −v_{k_s}/||v_{k_s}||;
          Find a step size t^k;
          Construct the following iteration x_{k_{s+1}} = x_{k_s} + t^k d_k;
          Set s = s + 1 and go to the next INNER ITERATION;
        END IF
      END INNER ITERATION
    END WHILE
  END OUTER ITERATION
  RETURN final solution x_k;
END DGM
```

In [4] it is proved that DGM is globally convergent for locally Lipschitz continuous functions under assumption that the set of discrete gradients uniformly approximates the subdifferential.

2.7 Quasisecant method (QSM)

In this subsection we briefly describe the quasisecant method (QSM). More details can be found in [2, 3]. Here, it is again assumed that one can compute both the function value and one subgradient at any point.

QSM can be considered as a hybrid of bundle methods and gradient sampling method [9]. The method builds up information about the approximation of the subdifferential using 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 gradient sampling method.

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

DEFINITION 2.2. A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called a *quasisecant* of the function f at the point $\boldsymbol{x} \in \mathbb{R}^n$ in the direction $\boldsymbol{g} \in S_1$ with the length $h > 0$ if

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

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

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

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

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

$$\begin{aligned} QSL(\boldsymbol{x}) = \{\boldsymbol{w} \in \mathbb{R}^n \mid & \exists\, \boldsymbol{g} \in S_1,\ h_k > 0,\ h_k \searrow 0 \text{ when } k \to \infty \\ & \text{s.t. } \boldsymbol{w} = \lim_{k\to\infty} \boldsymbol{v}(\boldsymbol{x}, \boldsymbol{g}, h_k)\}. \end{aligned}$$

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

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

$$W_0(\boldsymbol{x}, h) = \operatorname{cl}\operatorname{conv} QSec(\boldsymbol{x}, h)$$

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 $W_0(\boldsymbol{x}, h)$, and therefore we use only a few quasisecants from the set to calculate the descent direction in QSM.

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

$$\boldsymbol{w}_l = \operatorname{argmin}_{\boldsymbol{w}\in\bar{V}_l(\boldsymbol{x})} \|\boldsymbol{w}\|^2,$$

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

QSM is globally convergent for locally Lipschitz continuous functions under the assumption that the set $QSec(\boldsymbol{x}, h)$ is a SR-quasisecant mapping, that is, quasisecants can be computed using subgradients [2, 3]. The pseudo-code of QSM is the same as that for DGM (see Section 2.6) when replacing discrete gradients $\boldsymbol{v}_{k_s}$ with quasisecants $\boldsymbol{v}_{k_s}(\boldsymbol{x}_{k_s}, \boldsymbol{d}_{k_s}, h)$ and the set of discrete gradients $\bar{D}(\boldsymbol{x}_{k_s})$ with the set of quasisecants $\bar{V}(\boldsymbol{x}_{k_s})$.

3 Numerical Experiments

In this section we compare the implementations of the methods described in the previous section. First we introduce the solvers and problems used, then we say few words about the parameters and termination conditions of the codes and, finally, we report the numerical results obtained and draw some conclusions.

3.1 Solvers

The tested optimization codes are presented in Table 1. The experiments were performed on an Intel® Core™ 2 CPU 1.80GHz.

Table 1: Tested pieces of software

Software	Author(s)	Method	Reference
`SUBG`	Karmitsa	Subgradient	[45]
`SolvOpt`	Kuntsevich & Kappel	Shor's r-algorithm	[21, 26, 45]
`PBNCGC`	Mäkelä	Proximal bundle	[35, 37]
`PNEW`	Lukšan & Vlček	Bundle-Newton	[30]
`LMBM`	Karmitsa	Limited memory bundle	[17, 18]
`DGM`	Bagirov et al.	Discrete Gradient	[4]
`QSM`	Bagirov & Ganjehlou	QuasiSecant	[2, 3]

`SUBG` is a crude implementation of the basic subgradient algorithm. The step length is chosen to be in some extent constant. Let us denote by l the largest integer, smaller than or equal to it_{max}/c, where it_{max} is the maximum number of iterations and $c > 0$ is the user-specified maximum number of different step sizes. We take $t_k = t_{init}$ in the first l iterations and

$$t_k = \frac{t_{j\times l}}{10(j+1)} \qquad \text{for } k = j \times l + 1, \ldots, (j+1) \times l \text{ and } j = 1, \ldots, c.$$

We use the following three criteria as a stopping rule for `SUBG`: the number of function evaluations (and iterations) is restricted by parameter it_{max} and also the algorithm stops if either it cannot decrease the value of the objective function within m_1 successive iterations (i.e. $f(\boldsymbol{x}_l) > f_{best}$ for all $l = k, \ldots, k + m_1$, where f_{best} is the smallest value of the objective function obtained so far and $k \geq 1$), or it can not find a descent direction within m_2 successive iterations (i.e. $f(\boldsymbol{x}_{l+1}) > f(\boldsymbol{x}_l)$ for all $l = k, \ldots, k + m_2$, $k \geq 1$). Since a subgradient method is not a descent method we store the best value f_{best} of the objective function and the corresponding point $\boldsymbol{x}_{best}$ and return them as a solution if any of the stopping rule above is met.

`SUBG` is available for downloading from http://napsu.karmitsa.fi/subgra/.

`SolvOpt` (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 [21]).

In `SolvOpt` one can select to use either original subgradients or difference approximations of them (i.e. the user does not have to code difference approximations but to select one parameter to do this automaticly). In our experiments we have used both analytically and numerically calculated subgradients. In what follows, we denote `SolvOptA` and `SolvOptN`, respectively, the corresponding solvers.

The MatLab, C and Fortran source codes for `SolvOpt` are available for downloading from http://www.kfunigraz.ac.at/imawww/kuntsevich/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 (3) is solved by the `PLQDF1` subroutine implementing dual projected gradient gradient method proposed in [28].

`PBNCGC` can be used (free for academic purposes) via WWW-NIMBUS -system (http://nimbus.mit.jyu.fi/) [38].

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 automaticly). The quadratic direction finding problem (10) is solved by the subroutine PLQDF1 [28]. PNEW is available for downloading from http://www.cs.cas.cz/luksan/subroutines.html.

LMBM is an implementation of a limited memory bundle method specially 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. The Fortran 77 source code and the mex-driver (for MatLab users) are available for downloading from http://napsu.karmitsa.fi/lmbm/.

DGM is a discrete gradient solver for derivative free optimization. To apply DGM, one only needs to be able to compute at every point $\boldsymbol{x}$ the value of the objective function and the subgradient will be approximated. The source code of DGM is available on request: a.bagirov@ballarat.edu.au.

QSM is a quasisecant solver for nonsmooth possible nonconvex minimization. We have used both analytically calculated subgradients and approximated subgradients in our experiments (this can be done automaticly by selecting one parameter). In what follows, we denote QSMA and QSMN, respectively, the corresponding solvers. The source code of QSM is available on request: a.bagirov@ballarat.edu.au.

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.

In Table 2 we recall the basic assumptions needed for the solvers.

Table 2: Assumptions needed for softwares

Software	Assumptions to objective	Needed information
SUBG	convex	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
SolvOptA	convex	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
SolvOptN	convex	$f(\boldsymbol{x})$
PBNCGC	semi-smooth	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
PNEW	semi-smooth	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$, (approximated Hessian)
LMBM	semi-smooth	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
DGM	quasi-differentiable, semi-smooth	$f(\boldsymbol{x})$
QSMA	quasi-differentiable, semi-smooth	$f(\boldsymbol{x})$, arbitrary $\boldsymbol{\xi} \in \partial f(\boldsymbol{x})$
QSMN	quasi-differentiable, semi-smooth	$f(\boldsymbol{x})$

3.2 Problems

We consider ten types of problems:

SC: Small-scale ($n \leq 20$) problems with nonsmooth convex objective function (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 in [33]);

SNC: Small-scale problems with nonsmooth nonconvex objective function (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 in [33]);

MC: Medium-scale ($n = 50$) problems with nonsmooth convex objective function (Problems 1 – 5 in [17], and 2 and 5 in `TEST29` [29] and six maximum of quadratic functions, see Appendix);

MNC: Medium-scale problems with nonsmooth nonconvex objective function (Problems 6 – 10 in [17], and 13, 17 and 22 in `TEST29` [29] and six maximum of quadratic functions);

LC: Large-scale ($n = 200$) problems with nonsmooth convex objective function (see MC problems);

LNC: Large-scale problems with nonsmooth nonconvex objective function (see MNC problems);

XLC: Extra-large-scale ($n = 1000$) problems with nonsmooth convex objective function (see MC problems);

XLNC: Extra-large-scale problems with nonsmooth nonconvex objective function (see MNC problems);

XXLC: Extra-extra-large-scale ($n = 4000$) problems with nonsmooth convex objective function (see MC problems but only two maximum of quadratics with diagonal matrix);

XXLNC: Extra-extra-large-scale problems with nonsmooth nonconvex objective function (see MNC problems but only two maximum of quadratics with diagonal matrix).

Problems 2, 5, 13, 17, and 22 in `TEST29` are from the software package UFO (Universal Functional Optimization) [29]. They may also be found in [16]. The problems were selected such that in all cases all the solvers converged to the same local minimum. However, it is worth of mention 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 the different local minimum. 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 [33] all the other solvers converged to the minimum reported in [33] but DGM and QSM "converged" to minus infinity.

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: problems 2.4 in [33] (SC), 3.15 in [33] (SNC), and 3 in [17] with $n = 50$ (MC). 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 small- and medium-scale problems ($n \leq 50$) with respect to the tolerance $\varepsilon = 5 \cdot 10^{-4}$. 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 possible be prevented with a more tight stopping parameter).

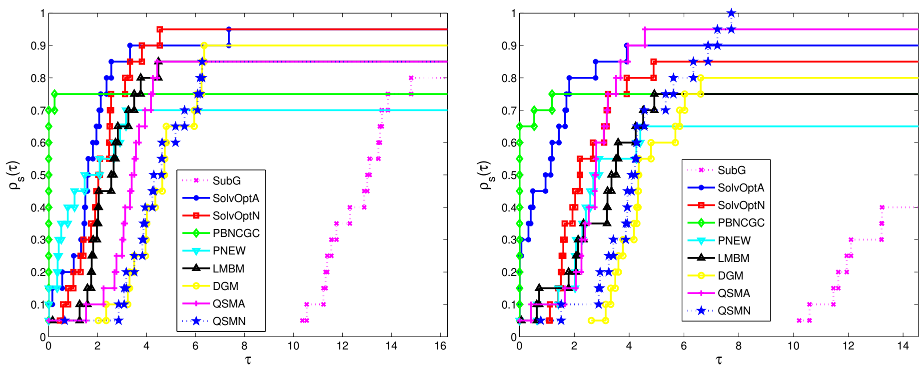
With all the bundle based solvers the distance measure parameter value $\gamma = 0.5$ was used with nonconvex problems. With PBNCGC and LMBM the value $\gamma = 0$ was used with convex problems and, since with PNEW γ has to be positive, $\gamma = 10^{-10}$ was used with PNEW. For those solvers storing 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.

3.4 Results

The results are summarized in Figures 1 – 13 and in Table 3. The results are analyzed using the performance profiles introduced in [14]. 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 [14].

3.4.1 Small problems

There was not a big difference in the computational times of the different solvers when solving the small-scale problems. Thus, only the numbers of function and subgradient evaluations are reported in Figure 1.



(a) Convex (b) Nonconvex

Figure 1: Evaluations for small problems (20 problems with $n \leq 20$, $\varepsilon = 5 \cdot 10^{-4}$).

`PBNCGC` was usually the most efficient solver when comparing the numbers of function and subgradient evaluations. This is, in fact, true for all sizes of problems. Thus, `PBNCGC` should be a good choice as a solver in the 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 small-scale problems (both SC and SNC) which means that it had almost the worst degree of success in solving these problems.

`SUBG` is not at all suitable for nonconvex problems: it failed in 60% of the problems ($\varepsilon = 5 \cdot 10^{-4}$, see Figure 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 nonconvex problems. `SolvOptA` was also the most efficient method expect for `PBNCGC` (especially in the nonconvex case) and, when comparing to `PBNCGC`, it was more reliable.

Except for SUBG, the solvers had not a big difference in the numbers of success in solving SC or SNC problems. However, it is noteworthy that QSM computed nonconvex problems more reliable than convex ones.

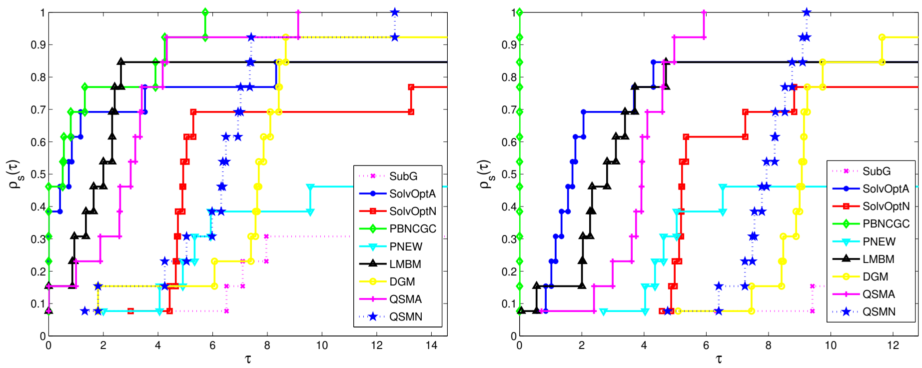
Most of the failures reported here are, in fact, inaccurate results: all the solvers but PNEW succeed in solving equal or more that 95% of SC problems with respect to the relaxed tolerance $\varepsilon = 10^{-2}$. The corresponding percentage for SNC problems was 85%, although here also SUBG failed to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

In SC problems PNEW was the second most efficient solver (see Figure 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 internal parameter XMAX (RPAR(9) in the code) which is noted also in [32]. If we, instead of only one (default) value, used a selected value for this parameter, also the solver PNEW solved 85% of SNC problems.

The derivative free solvers DGM and QSMN performed similar 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 SC and SNC problems. However, in the nonconvex case, also SolvOptN lost out to QSMN in reliability.

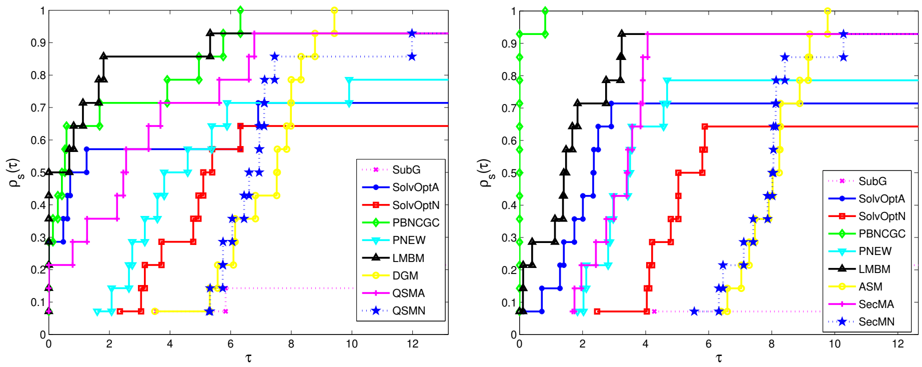
3.4.2 Medium-scale problems

Already with medium-scale problems, there was a wide diversity on the computational times of different codes. Moreover, the numbers of function and subgradient evaluations used with solvers were not anymore directly comparable with the elapsed computational times. For instance, PBNCGC was clearly the winner when comparing the numbers of evaluations (see Figures 2(b) and 3(b)). However, when comparing computational times, SolvOptA was equally efficient with PBNCGC in MC problems (see Figure 2(a)) and LMBM was the most efficient solver in MNC problems (see Figure 3(a)).



(a) CPU-time (b) Evaluations

Figure 2: CPU-time and evaluations for MC problems (13 problems with $n = 50$, $\varepsilon = 5 \cdot 10^{-4}$).



(a) CPU-time (b) Evaluations

Figure 3: CPU-time and evaluations for MNC problems (14 problems with $n = 50$, $\varepsilon = 5 \cdot 10^{-4}$).

SUBG was clearly the worst solver with respect to both computational times and evaluations in both MC and MNC 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 $\varepsilon = 5 \cdot 10^{-4}$ 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 benefit most if we instead of tolerance $\varepsilon = 5 \cdot 10^{-4}$ would have 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. Thus, SolvOpt solved convex problems much better than nonconvex with respect to tolerance $\varepsilon = 5 \cdot 10^{-4}$ (see Figures 2 and 3).

With the other solvers there was not a big difference in solving convex or nonconvex problems but with PNEW: PNEW solved about 79% of the nonconvex problems and only 46% of the convex problems. Also LMBM succeed in solving little bit more nonconvex than convex problems. In the convex case, PBNCGC, QSMA and QSMN succeed to solve 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 succeed in solving more than 90% of the problems. In the nonconvex case, PBNCGC and DGM solved all the problems successfully. With relaxed parameter QSMA and QSMN succeed as well and all the solvers but PNEW and SUBG managed to solve more than 85% of the problems.

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.

PNEW failed to solve all but one of the convex quadratic problems and succeed in solving all but one non-quadratic problems. In the nonconvex case PNEW succeed 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 internal parameter XMAX. If we, instead of only one (default) value XMAX=1000, used also the value XMAX=2 for this parameter and select the better result, PNEW solved all the convex quadratic problems (7 pc.) successfully. The computations with XMAX=2 failed only in one problem, where XMAX=1000 succeed. In the nonconvex case, the solver succeed in solving all quadratic problems with both parameter XMAX=1000 and XMAX=2. However, with XMAX=2 the number of used function and subgradient calls was almost fourfold when compared to that used with parameter XMAX=1000. The usage of actual Hessian instead of the approximation helped a little bit in the convex quadratic case. However the computational times were enormously longer (when compared to those obtained with XMAX = 2). In the nonconvex case and with larger problems (LC, LNC, XLC, XLNC), the usage of actual Hessian made the results worse.

In [36], 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, expect for some small problems, it was not in any case the most efficient method.

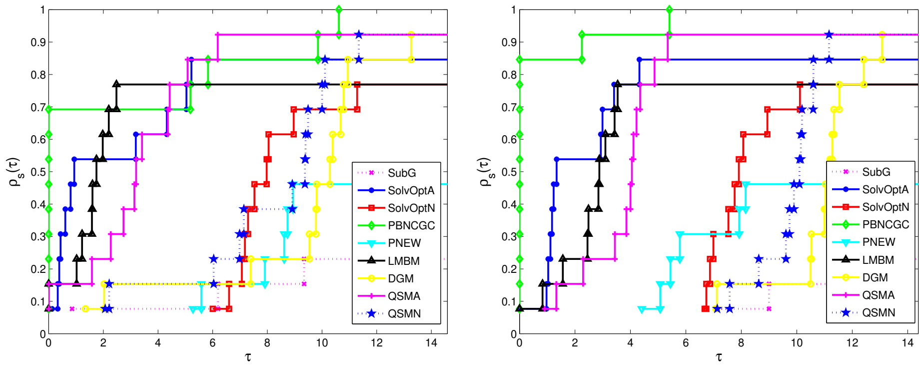
3.4.3 Large problems

When solving large-scale problems, the solvers divided into two groups (more clearly in the convex case, see Figure 4): 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 Figure 6), the inaccuracy of SolvOptA made it slide to the group of less efficient solvers. On the other hand, successfully solved quadratic problems almost rose PNEW to the group of more efficient solvers (especially, when comparing the numbers of function evaluations, see Figure 6(b)).

Although PBNCGC was usually (on 70%) the most efficient solver tested in the convex case (see Figure 4(a)), it was also the one who needed the longest time to compute problem 3 in [17]. Indeed, an average time used to solve a LC problem with PBNCGC was 15.7 seconds while with SolvOptA and LMBM they were 1.3 and 1.6 seconds, respectively (the average times are calculated using 9 problems that all the solvers above succeed in solving).

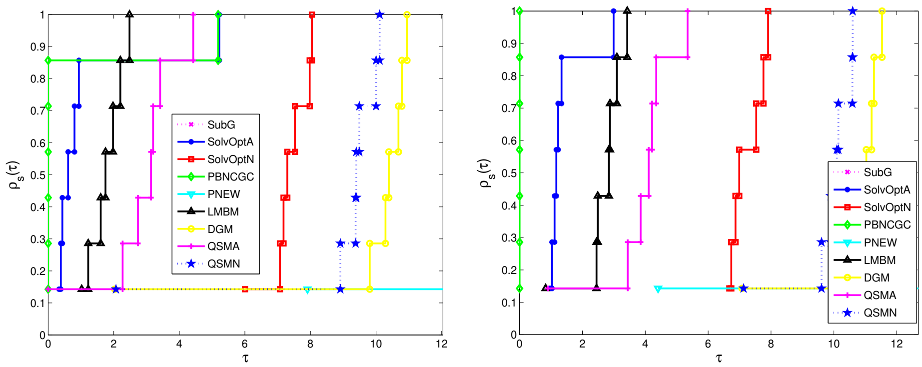
In the nonconvex case, LMBM and PBNCGC were the most efficient solvers. However, with PBNCGC, there was a big variation in the computational times for different problems while with LMBM all the problems were solved equally efficiently.

The efficiency of PBNCGC is mostly due to its efficiency in quadratic problems (i.e. problem 1 in [17] and six maxq-problems in the convex case and six maxq-problems in the nonconvex case, see Appendix): PBNCGC was the most efficient



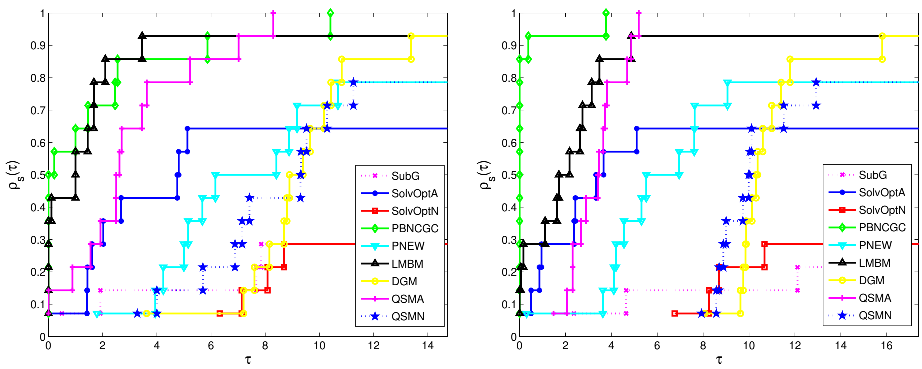
(a) CPU-time (b) Evaluations

Figure 4: CPU-time and evaluations for LC problems (13 problems with $n = 200$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

Figure 5: CPU-time and evaluations for LC maxq-problems (7 problems with $n = 200$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

Figure 6: CPU-time and evaluations for LNC problems (14 problems with $n = 200$, $\varepsilon = 10^{-3}$).

solver in all but one of these problems when comparing the computational times and superior in all cases when comparing the numbers of evaluations. Figure 5 illustrates the performance of the solvers with convex quadratic problems. As before PNEW failed in all but one of these problems.

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

SolvOptN had some serious difficulties with the accuracy: for instance, with the relaxed tolerance SolvOptN solved almost 80% of LNC problems (in Figure 6 less than 30%). Similar effect could been seen with SolvOptA, although not as pronounced.

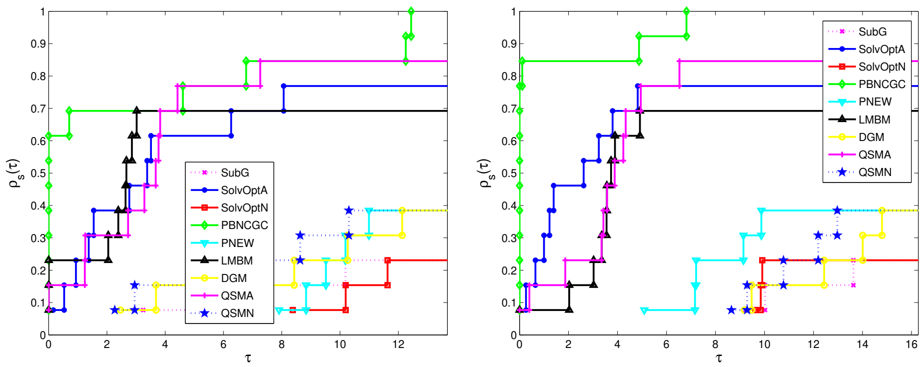
Naturally, for the solvers using difference approximation or some other approximation based on the calculation of the function or subgradient values, the number of evaluations grows enormously when the number of variables increases. However, if you need to solve a problem, where the subgradient is not available, the best solver would probably be SolvOptN (only in the convex case) due its efficiency or QSMN due its reliability.

3.4.4 Extra large problems

Likewise with large problems, there were two clear groups in extra large problems (see Figures 7 and 8). Again PBNCGC was clearly the most reliable and efficient solver tested and again the efficiency of PBNCGC is mostly due to its efficiency in quadratic problems. That is, while being clearly the most efficient method in almost all quadratic problems, the average time used to a problem (including all the problems that all the solvers below succeed in solving and in the nonconvex case ignoring the solver SolvOptA which solved only 43 % of the problems) was much larger with PBNCGC (266 sec for the convex and 325 sec for the nonconvex problems) than that with, for instance, LMBM (54.5 and 95.7), QSMA (98.6 and 190.8) or SolvOptA (22.0 for the convex problems). Indeed, in the convex case, LMBM was the most efficient solver in all the non-quadratic problems it could solve. Unfortunately, LMBM succeed in solving only three of them (from six).

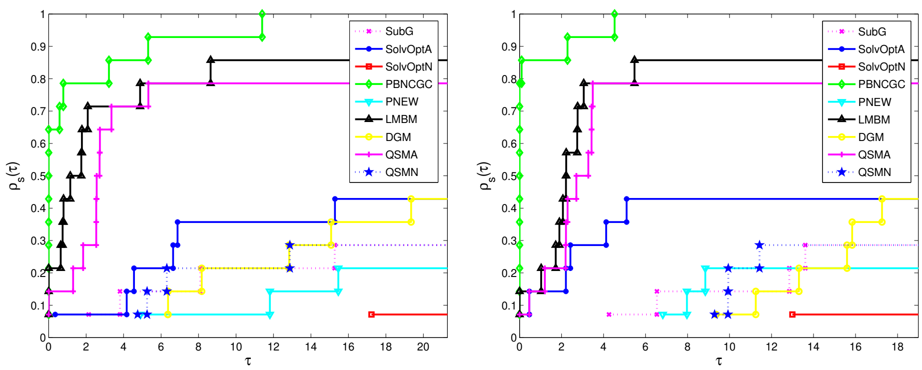
In the nonconvex case (see Figure 8), the inaccuracy of SolvOptA made it again slide to the group of less efficient solvers. Figure 9 illustrates the results with the relaxed tolerance $\varepsilon = 10^{-2}$. As can be seen, here SolvOptA is among more efficient solvers, although its accuracy is not as good as that of the others.

Also LMBM and QSMA had some difficulties with the accuracy in the nonconvex case (see Figure 8). With the relaxed tolerance, they solved all XLNC



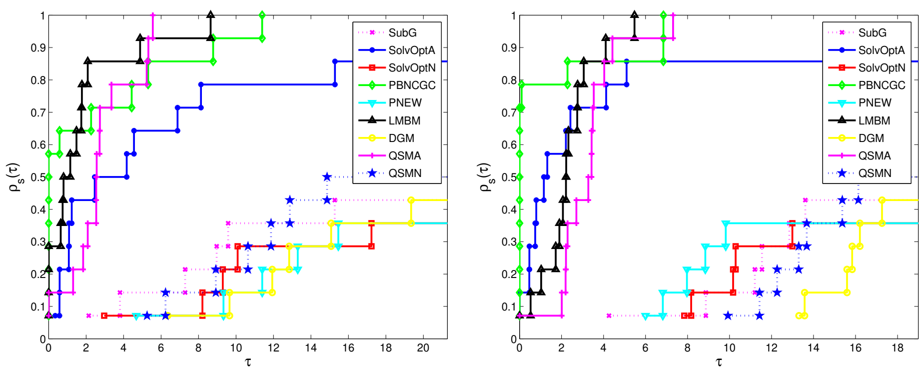
(a) CPU-time (b) Evaluations

Figure 7: CPU-time and evaluations for XLC problems (13 problems with $n = 1000$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

Figure 8: CPU-time and evaluations for XLNC problems (14 problems with $n = 1000$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

Figure 9: CPU-time and evaluations for XLNC problems, low accuracy (14 problems with $n = 1000$, $\varepsilon = 10^{-2}$).

problems (see Figure 9). 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.

Solvers PBNCGC, DGM and QSM were the only solvers which solved two extra large problems in which there is only one nonzero element in the subgradient vector (i.e. Problem 1 in [17] and 2 in TEST29 [29]). With the other methods, there were some difficulties already with $n = 50$ and some more with $n = 200$ (note that for M, L and XL 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 [17] that LMBM is best suited for the problems with dense subgradient vector whose component depend on the current iteration point. This result is in line with the noted result that LMBM solves nonconvex problems very efficiently.

In the convex case PNEW solved all but the above mentioned two problems and the maximum of quadratics problems. 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 [17] and 5 in TEST29 [29]) and also QSMA failed to solve one of them.

In all the maximum of quadratics problems, the time limit was exceeded with all the solvers using some kind of approximation for subgradients or Hessian. Thus, the number of failures with these solvers is probably larger than it should be.

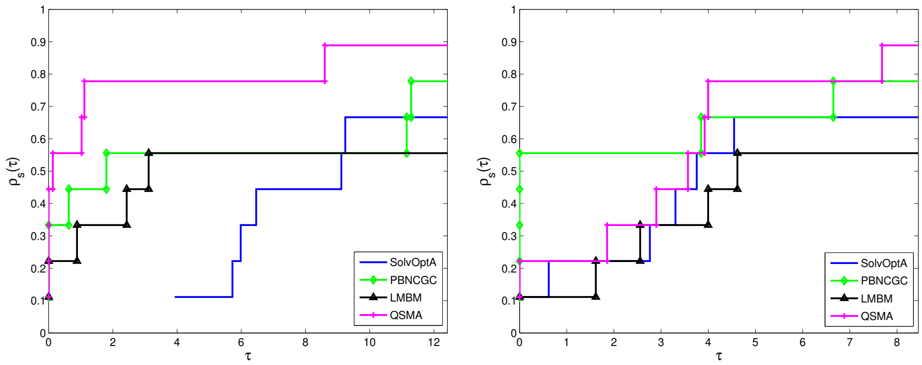
3.4.5 Extra extra large problems

Finally we tested the most efficient solvers so far, that is LMBM, PBNCGC, QSMA and SolvOptA, with the problems with $n = 4000$.

In the convex case, the solver QSMA, which has kept 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 Figure 10).

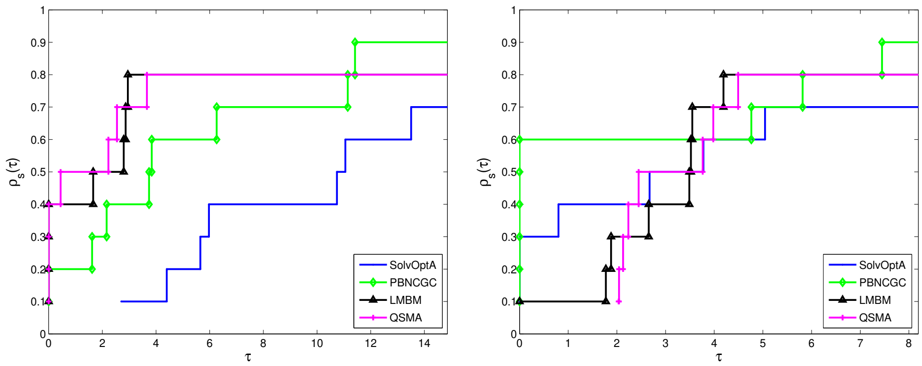
LMBM solved all the problems it could solve in a relatively short time while with all the other solvers there were a wide variation in the computational times elapsed for different problems. However, in the convex case, the efficiency of LMBM was again ruined by its unreliability.

In the nonconvex case LMBM and QSMA were approximately as good both in computational times, evaluations and reliability (see Figure 11). 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.



(a) CPU-time (b) Evaluations

Figure 10: CPU-time and evaluations for XXLC problems (9 problems with $n = 4000$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

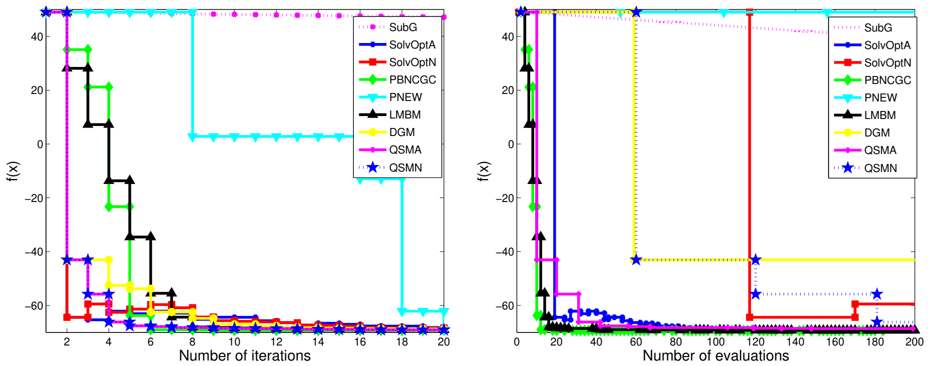
Figure 11: CPU-time and evaluations for XXLNC problems (10 problems with $n = 4000$, $\varepsilon = 10^{-3}$).

3.4.6 Convergence speed and number of success

In this subsection we first study (experimentally) the convergence speed of the algorithms using one medium-scale convex problem (Problem 3 in [17]). The exact minimum value for this function (with $n = 50$) is $-49 \cdot 2^{1/2} \approx -69.296$.

For the limited memory bundle method the rate of convergence has not been studied theoretically. However, at least in this particular problem, 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 Figure 12(b)), although, teoretically, proximal bundle method is only linearly convergent.

`SUBG` converged linearly but very, very slowly and `PNEW`, although finally found the minimum, did not decrease the value of the function in 200 first evaluations. Naturally, with `PNEW` a large amount of subgradient evaluations are needed to compute the approximative Hessian. Solvers `SolvOptA`, `SolvOptN`, `DGM`,



(a) First 20 iterations (b) First 200 evaluations

Figure 12: Function values versus iterations (a), and function values versus number of function and subgradient evaluations (b).

QSMA, and QSMN took a very big step downwards already in iteration two (see Figure 12(a)). However, they took quite many function evaluations per iteration. In Figure 12 it is easy to see that Shor's r-algorithm (i.e. solvers SolvOptA and SolvOptN) is not a descent method.

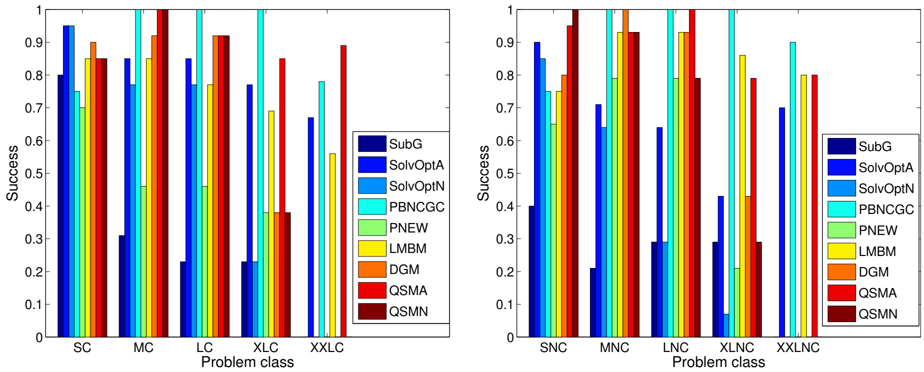
In order to see how quickly the solvers reach some specific level, we studied the value of the function equal to -69. With PBNCGC it took only 8 iterations to go below that level. The corresponding values for other solvers were 17 with QSMA and QSMN, 20 with LMBM and PNEW, and more than 20 with all the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

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 Figure 13 we give the proportions of the succesfully 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 Figure 13 to see if the convexity or the number of variables have any significant effect on the success rate of the solvers.

In the figure, we see that with both variants of SolvOpt the degree of success decrease clearly when the number of variables increases or the problem is nonconvex. 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



(a) Convex (b) Nonconvex

Figure 13: Proportions of successfully terminated runs within different problem classes: convex problems (a), and nonconvex problems (b).

exceeded in many cases with $n = 1000$, thus the exception. With PNEW the result could be different if tuned parameter XMAX was used. With LMBM the result is in harmony with the earlier claims [16, 17] that LMBM works better for more nonlinear functions.

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

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 3, where we give our recommendations for the "best" solver for different problem classes. Since it is not always unambiguous what is the best, we give credentials both in the cases when 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 3, the solvers are given in the 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 can not say that it is clearly the best method tested. The inaccuracy in small-scale 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 quadratics and piecewise linear problems.

Table 3: Summation of the results

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

(1) PNEW may require tuning of internal parameter XMAX. (2) LMBM, if not a piecewise linear or sparse problem. (3) PBNCGC in piecewise linear problems, QSMA in other sparse problems. (4) LMBM especially in the nonconvex case. (5) QSMN especially in the nonconvex case, SolvOptN only in the convex case.

On the other hand, the limited memory bundle solver LMBM suffered from ill-fitting test problems in convex M, L, XL and XXL cases. In the test set there were 4 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 — in relatively short time while, for example, with PBNCGC there were a 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 small-scale problems the bundle-Newton solver PNEW was the second most efficient solver tested. However, PNEW suffers very badly 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 [36] and, indeed, it solved (nonconvex) quadratic problems faster that non-quadratic. However, with $n \geq 50$ it did not beat the other solvers in these problems due to large approximation of the Hessian matrix required.

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

Thus, when comparing the reliability in large-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 large problems. However, with QSMA there were a 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 quasisecant solver with discrete gradients QSMN, usually lost out in efficiency to the solvers using analytical subgradients. However, in small and medium-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.

To answer the question asked in the Introduction, the best solver for a non-convex nonsmooth problem with 200 variables is the limited memory bundle method. But that is only if we knew nothing else about the objective. If we, for instance, knew that the objective has sparse subgradient vector or it is a maximum of quadratics problem the best solver would probably be proximal bundle method. On the other hand, if we are unable to identify subgradient vector, either Shor's r-algorithm with finite difference approximations or, since in some nonsmooth cases finite differences may cause serious misinterpretation [27], discrete gradient method or secant method with discrete gradients would be a good choice.

Acknowledgements

We would like to acknowledge professors Kuntsevich and Kappel for providing Shor's r-algorithm in their web-page as well as professors Lukšan and Vlček for providing the bundle-Newton algorithm.

The work was financially supported by the University of Turku (Finland) and the University of Ballarat (Australia).

Appendix

Maximum of quadratics. Maximum of quadratic functions are defined as the point-wise maximum of a finite collection of quadratics functions. That is

$$f(\boldsymbol{x}) = \max\{f_j(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A_j \boldsymbol{x} + \boldsymbol{b}_j^T \boldsymbol{x} + c_j \mid j = 1, \dots, n_f\},$$

where A_j are $n \times n$ symmetric matrices (in the convex case positive definite), $\boldsymbol{b}_j \in \mathbb{R}^n$ and $c_j \in \mathbb{R}$. With this definition, many different examples are easily created by choosing the values of n, n_f, and the sparsity parameter $0 \leq p_s \leq 1$ ($p_s = 0$ causes the diagonal matrix, $p_s = 1$ causes the dense matrix, and $0 < p_s < 1$ causes the sparse matrix with approximately $p_s n^2 + n$ nonzeros) and then randomly generating n_f objects A_j, $\boldsymbol{b}_j$ and c_j. Depending on the positive definiteness of matrices A_j both convex and nonconvex problems can be created (for more details of the procedure see the following algorithm). In our experiments, we used 6 different combinations of the values n_f, and p_s to create both convex and nonconvex problems. The values used were $n_f = 5$ and 10, and $p_s = 0$, 0.6 and 1. The numbers of variables used were $n = 50$, 200 and 1000.

In the following algorithm some more details of the data generation are given:

```
PROGRAM Create Data for Maximum of Quadratics
  INITIALIZE Select a lower and upper bound for random
    number generator L,U ∈ ℝ.  Fix the dimension n, the
    number of elemental functions n_f and the sparsity
    parameter p_s (0 ≤ p_s ≤ 1).  Set i_con = 1, if a convex
    problem is needed and i_con = 0, otherwise;
  IF i_con = 0 THEN
    Set c_1 = L and randomly generate c_j ∈ (L,U) for
    j = 2,...,n_f;
  ELSE
    Randomly generate c_j ∈ (L,U) for j = 1,...,n_f;
  END IF
  FOR ALL j = 1,...,n_f randomly generate vectors b_j ∈ (L,U)^n;
  FOR ALL j = 1,...,n_f randomly generate symmetric matrices
    A_j ∈ (L,U)^{n×n} such that there exist approximately
    p_s n^2 + n nonzero entries and all the diagonal elements
    are nonzero.
    IF i_con = 1 THEN
      Add the identity matrix multiplied by one plus the
      absolute value of the smallest eigenvalue to each
      matrix A_j to obtain positive definite matrices;
    ELSE
      Add the identity matrix multiplied by one plus the
      absolute value of the smallest eigenvalue to the
      first matrix A_1.  Otherwise, check that the minimum
      eigenvalue is negative for each matrix A_j (with
      j = 2,...,n_f).  Regenerate any A_j whose eigenvalue is
      non-negative;
    END IF
  Randomly generate the starting point x_1 ∈ (L,U)^n;
END Create Data for Maximum of Quadratics
```

Note: In the nonconvex case we enforce the first elemental function to be convex (i.e. the matrix A_1 to be positive definite) in order to obtain finite results. This does not restrict the overall nonconvexity of the problem (assuming $n_f > 1$) since at $\boldsymbol{x} = 0$ all the nonconvex elemental functions have a larger value than the convex one.

The MatLab-file `makeproblem.m` for generating the random data as well as the Fortran subroutine `maxq.f` that reads the data-file and calculates the value of the function and subgradient are available for downloading from http://napsu.karmitsa.fi/testproblems/.

References

[1] ÄYRÄMÖ, S. *Knowledge Mining Using Robust Clustering*. PhD thesis, University of Jyväskylä, Department of Mathematical Information Technology, 2006.

[2] BAGIROV, A. M., AND GANJEHLOU, A. N. A quasisecant method for minimizing nonsmooth functions. *Optimization Methods and Software 25*, 1 (2009), 3–18.

[3] BAGIROV, A. M., AND GANJEHLOU, A. N. A secant method for nonsmooth optimization. Submitted, 2009.

[4] BAGIROV, A. M., KARASOZEN, B., AND SEZER, M. Discrete gradient method: A derivative free method for nonsmooth optimization. *Journal of Optimization Theory and Applications 137* (2008), 317–334.

[5] BECK, A., AND TEBOULLE, M. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters 31*, 3 (2003), 167–175.

[6] BEN-TAL, A., AND NEMIROVSKI, A. Non-Euclidean restricted memory level method for large-scale convex optimization. *Mathematical Programming 102*, 3 (2005), 407–456.

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

[8] BRADLEY, P. S., FAYYAD, U. M., AND MANGASARIAN, O. L. Mathematical programming for data mining: Formulations and challenges. *INFORMS Journal on Computing 11* (1999), 217–238.

[9] BURKE, J. V., LEWIS, A. S., AND OVERTON, M. L. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization 15* (2005), 751–779.

[10] 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.

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

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

[13] DEMYANOV, V. F., BAGIROV, A. M., AND RUBINOV, A. M. A method of truncated codifferential with application to some problems of cluster analysis. *Journal of Global Optimization 23*, 1 (2002), 63–80.

[14] DOLAN, E. D., AND MORÉ, J. J. Benchmarking optimization software with performance profiles. *Mathematical Programming 91* (2002), 201–213.

[15] GAUDIOSO, M., AND MONACO, M. F. Variants to the cutting plane approach for convex nondifferentiable optimization. *Optimization 25* (1992), 65–75.

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

[17] 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.

[18] HAARALA, N., MIETTINEN, K., AND MÄKELÄ, M. M. Globally convergent limited memory bundle method for large-scale nonsmooth optimization. *Mathematical Programming 109*, 1 (2007), 181–205.

[19] HASLINGER, J., AND NEITTAANMÄKI, P. *Finite Element Approximation for Optimal Shape, Material and Topology Design*, 2nd ed. John Wiley & Sons, Chichester, 1996.

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

[21] KAPPEL, F., AND KUNTSEVICH, A. An implementation of Shor's r-algorithm. *Computational Optimization and Applications 15* (2000), 193–205.

[22] KÄRKKÄINEN, T., AND HEIKKOLA, E. Robust formulations for training multilayer perceptrons. *Neural Computation 16* (2004), 837–862.

[23] KARMITSA, N., MÄKELÄ, M. M., AND ALI, M. M. Limited memory interior point bundle method for large inequality constrained nonsmooth minimization. *Applied Mathematics and Computation 198*, 1 (2008), 382–400.

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

[25] KIWIEL, K. C. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming 46* (1990), 105–122.

[26] KUNTSEVICH, A., AND KAPPEL, F. SolvOpt — the solver for local nonlinear optimization problems. Karl-Franzens University of Graz: Graz, Austria, 1997.

[27] LEMARÉCHAL, C. Nondifferentiable optimization. In *Optimization*, G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, Eds. Elsevier North-Holland, Inc., New York, 1989, pp. 529–572.

[28] 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.

[29] 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.

[30] LUKŠAN, L., AND VLČEK, J. A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming 83* (1998), 373–391.

[31] LUKŠAN, L., AND VLČEK, J. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory and Applications 102*, 3 (1999), 593–613.

[32] LUKŠAN, L., AND VLČEK, J. NDA: Algorithms for nondifferentiable optimization. Technical Report 797, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2000.

[33] LUKŠAN, L., AND VLČEK, J. Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2000.

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

[35] 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.

[36] 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.

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

[38] MIETTINEN, K., AND MÄKELÄ, M. M. Synchronous approach in interactive multiobjective optimization. *European Journal of Operational Research 170*, 3 (2006), 909–922.

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

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

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

[42] ROBINSON, S. M. Linear convergence of epsilon-subgradient descent methods for a class of convex functions. *Mathematical Programming 86* (1999), 41–50.

[43] SAGASTIZÁBAL, C., AND SOLODOV, M. An infeasible bundle method for nonsmooth convex constrained optimization without a penalty function or a filter. *SIAM Journal on Optimization 16*, 1 (2005), 146–169.

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

[45] SHOR, N. Z. *Minimization Methods for Non-Differentiable Functions*. Springer-Verlag, Berlin, 1985.

[46] URYASEV, S. P. Algorithms for nondifferentiable optimization problems. *Journal of Optimization Theory and Applications 71* (1991), 359–388.

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

TURKU

CENTRE *for*

COMPUTER

SCIENCE

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

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-2367-9
ISSN 1239-1891