Comparing Different Nonsmooth Minimization Methods and Software

N. Karmitsa[1] A. Bagirov[2] and M.M. Mäkelä [3]

Abstract: Most of nonsmooth optimization methods can be divided into two main groups: subgradient methods and bundle methods. In this paper, we test and compare different methods from both groups as well as some methods which may be considered as hybrid 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 applied to solve almost all nonsmooth optimization problems. The test set includes a large number of unconstrained nonsmooth convex and nonconvex problems of different size. In particular, it includes piecewise linear and quadratic problems. The aim of this work is not to foreground some method over the others but to get some insight on which method to select for certain types of problems.

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

1 Introduction

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

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

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 [46], mechanics [44], engineering [43], control theory [12], optimal shape design [22], data mining and machine learning [1, 8, 13, 25].

Most of methods for solving problems (P) can be divided into two main groups: subgradient [5, 6, 50, 51] and bundle methods [17, 23, 28, 35, 38, 41, 48, 49]. Both of these methods have their own supporters. Usually, when developing new methods, researchers compare them with similar methods. That is, bundle methods are compared with bundle methods and subgradient methods are compared with subgradient methods. Moreover, it is quite common that the test set used is rather concise (sometimes a very few problems), which naturally does not give a complete picture of how the algorithm would perform on different kind of problems.

1. Department of Mathematics, University of Turku, FI-20014 Turku, Finland. E-mail: napsu@karmitsa.fi
2. 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. E-mail: a.bagirov@ballarat.edu.au
3. Department of Mathematics, University of Turku, FI-20014 Turku, Finland. E-mail: makela@utu.fi

In this paper, we compare different subgradient and bundle methods, as well as some methods that lie between these two. A broad set of nonsmooth optimization test problems are used for this purpose. The methods included in our tests are the following:

- *Subgradient method*:
  - Shor's r-algorithm [24, 30, 50],
- *Bundle methods*:
  - proximal bundle method [41],
  - bundle-Newton method [34],
- *Hybrid methods*:
  - limited memory bundle method [19, 20],
  - discrete gradient method [4] and
  - quasi-secant method [2].

All the solvers tested are so-called general black box methods and, naturally, cannot beat the codes designed specifically for a particular class of problems (say e.g. for piecewise linear, min-max, or partially separable problems). Losing to codes designed for specific problems is a weakness, but only if the problem is known to be of that type. The strength of the general methods is that they require minimal information on the objective function for their implementation. Namely, the value of the objective function and, possibly, one arbitrary subgradient (generalized gradient [11]) are required at each point. The aim of our research is not to foreground some method over the others but to get some insight on which method to select for certain types of problems.

The paper is organized as follows. Section 2 describes the NSO methods tested and compared. The results of the numerical experiments are presented and discussed in Section 3. Section 4 concludes the paper 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 function $f : \mathbb{R}^n \to \mathbb{R}$ at a 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. An optimization method is said to be *globally convergent* if starting from any arbitrary point $\boldsymbol{x}_1$ it generates a sequence $\{\boldsymbol{x}_k\}$ that converges to a substationary point $\boldsymbol{x}^*$, that is, $\{\boldsymbol{x}_k\} \to \boldsymbol{x}^*$ whenever $k \to \infty$.

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 $f(\boldsymbol{x})$ of the objective function f and an arbitrary subgradient $\boldsymbol{\xi}$ from the subdifferential $\partial f(\boldsymbol{x})$.

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

The simplest methods for solving problem (P) are the *standard subgradient methods* [50]. The idea behind these methods (Kiev methods) is to generalize gradient methods (e.g. the steepest descent method) by replacing the gradient with an arbitrary subgradient. Thus, 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.

There are many results on convergence of subgradient methods [50]. For constant step sizes ($t_k = t$, for all k) the subgradient methods are guaranteed to converge to within some range of the optimal value. That is, we have

$$\lim_{k\to\infty} f_k^{\text{best}} - f^* \le \varepsilon,$$

where $f^* = \inf_{\boldsymbol{x}} f(\boldsymbol{x})$ denotes the optimal value of the problem, $f_k^{\text{best}} = \min\{f(\boldsymbol{x}_1), \dots f(\boldsymbol{x}_k)\}$ denotes the best objective value found in k iterations and $\varepsilon > 0$ depends on the step size.

For diminishing step sizes, the methods are guaranteed to converge to the optimal value. That is, we have

$$\lim_{k\to\infty} f(\boldsymbol{x}_k) = f^*,$$

if the objective function is convex and step sizes satisfy

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

Due to their simple structure and low storage requirements, subgradient methods are widely used 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) [31]. Naturally, some of these difficulties may be prevented with a good implementation of the method.

Next we shortly describe the 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 the conjugate gradient methods.

The r-algorithm is given as follows:

```
PROGRAM Shor's $r$-algorithm
  INITIALIZE $\boldsymbol{x}_0 \in \mathbb{R}^n$, $\beta \in (0,1)$ , $B_1 = I$, and $t_1 > 0$;
  Compute $\boldsymbol{\xi}_0 \in \partial f(\boldsymbol{x}_0)$ and $\boldsymbol{x}_1 = \boldsymbol{x}_0 - t_1\boldsymbol{\xi}_0$;
  Set $\bar{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_0$ and $k = 1$;
  WHILE the termination condition is not met
    Compute $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$ and $\boldsymbol{\xi}_k^* = B_k^T\boldsymbol{\xi}_k$;
    Calculate $\boldsymbol{r}_k = \boldsymbol{\xi}_k^* - \bar{\boldsymbol{\xi}}_k$ and $\boldsymbol{s}_k = \boldsymbol{r}_k/\|\boldsymbol{r}_k\|$;
    Compute $B_{k+1} = B_k R_\beta(\boldsymbol{s}_k)$, where $R_\beta(\boldsymbol{s}_k) = I + (\beta - 1)\boldsymbol{s}_k\boldsymbol{s}_k^T$
      is the space dilation matrix;
    Calculate $\bar{\boldsymbol{\xi}}_{k+1} = B_{k+1}^T\boldsymbol{\xi}_k$;
    Choose a step size $t_{k+1}$;
    Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - t_{k+1}B_{k+1}\bar{\boldsymbol{\xi}}_{k+1}$ and $k = k + 1$;
  END WHILE
  RETURN final solution $\boldsymbol{x}_k$;
END Shor's $r$-algorithm
```

In order to turn the above algorithm into an efficient optimization routine, one still has to find solutions 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 the r-algorithm [50]. If the objective function is piecewise smooth and coercive, it has only isolated local minimizers, and the r-algorithm is convergent to one of these minimizers [50].

2.2 Proximal Bundle Method (PBM)

In this subsection we briefly describe the proximal bundle method (PBM). For more details we refer to [29, 41, 49]. 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 $\emptyset \neq J_k \subset \{1, \ldots, k\}$. We approximate the objective function f below by a piecewise linear function, that is, f is replaced by the so-called *cutting-plane model*

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

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 *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 underestimate 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. The most common way to deal with the difficulties caused by nonconvexity is to replace the linearization error α_j^k by the *subgradient locality measure* [28]

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

where $\gamma \geq 0$ is a *distance measure parameter* ($\gamma = 0$ if f is convex). Then obviously $\beta_j^k \geq 0$ for all $j \in J_k$ and $\min_{\boldsymbol{x} \in K} \hat{f}_k(\boldsymbol{x}) \leq f(\boldsymbol{x}_k)$. Lately, other approaches have been introduced, for instance, in [15, 16, 21, 45].

The descent direction is then 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 $\boldsymbol{x}_k$ [29, 41, 49].

In order to determine the step size into the search direction $\boldsymbol{d}_k$, 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

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

where v_k is the predicted amount of descent, i.e. $v_k = \hat{f}_k(\boldsymbol{x}_k + \boldsymbol{d}_k) - f(\boldsymbol{x}_k) < 0$. 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 the gradient of f may be discontinuous. 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 a general bundle method is the following:

```
PROGRAM bundle method
  INITIALIZE $\boldsymbol{x}_1 \in \mathbb{R}^n$, $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{x}_1)$, $J_1$, $v_1$, and $\varepsilon_s > 0$;
  Set $k = 1$;
  WHILE the termination condition $|v_k| \leq \varepsilon_s$ is not met
    Generate the search direction $\boldsymbol{d}_k$;
    Find step sizes $t_L^k$ and $t_R^k$;
    Update $\boldsymbol{x}_k$, $v_k$ and $J_k$;
    Set $k = k + 1$;
    Evaluate $f(\boldsymbol{x}_k)$ and $\boldsymbol{\xi}_k \in \partial f(\boldsymbol{x}_k)$;
  END WHILE
  RETURN final solution $\boldsymbol{x}_k$;
END bundle method
```

Under the upper semi-smoothness assumption [7] the PBM can be proved to be globally convergent for locally Lipschitz continuous objective functions, which are not necessarily differentiable or convex [29, 41]. 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 [28], which accumulates information from the previous iterations. The convergence rate of the PBM is linear for convex functions under the assumption of the inverse growth condition [47] and for piecewise linear problems the PBM achieves a finite convergence [49].

2.3 Bundle Newton Method (BNEW)

Next we briefly describe the second order bundle-Newton method (BNEW) [34]. 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. Now, instead of the piecewise linear cutting-plane model (1), we introduce a piecewise quadratic model of the form

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

where $G_j = G(\boldsymbol{y}_j)$ and $\varrho_j \in [0, 1]$ is some damping parameter 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 $\min_{\boldsymbol{x} \in \mathbb{R}^n} \tilde{f}_k(\boldsymbol{x}) \le f(\boldsymbol{x}_k)$ [34]. The search direction $\boldsymbol{d}_k \in \mathbb{R}^n$ is now calculated as a solution of

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

The line search procedure of the BNEW is similar to that in the PBM (see Section 2.2). The only remarkable difference occurs in the termination condition for short and null steps. In other words, (7) is replaced by two conditions

$$-\beta_{k+1}^{k+1} + (\boldsymbol{\xi}_{k+1}^{k+1})^T d_k \ge m_R v_k \qquad \text{and} \qquad \|\boldsymbol{x}_{k+1} - \boldsymbol{y}_{k+1}\| \le C_S,$$

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

The pseudo-code for the BNEW is the same as that for the PBM (see Section 2.2). Under the upper semi-smoothness assumption [7] 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 [34].

2.4 Limited Memory Bundle Method (LMBM)

In this subsection, we describe the limited memory bundle algorithm (LMBM) [18, 19, 20, 27] for solving large-scale NSO problems. The method is a hybrid of the variable metric bundle methods [35, 52] and the limited memory variable metric methods [10], where the first ones have been developed for small- and medium-scale nonsmooth optimization and the latter ones for smooth large-scale optimization.

The 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.

The 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 the LMBM is rather similar to that used in the PBM (see Section 2.2). 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 the 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 the PBM).

In the 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 [18, 19, 20]).

Under the upper semi-smoothness assumption [7] the LMBM can be proved to be globally convergent for locally Lipschitz continuous objective functions [18, 20].

The pseudo-code of the LMBM is the following:

```
PROGRAM LMBM
  INITIALIZE $\boldsymbol{x}_1 \in \mathbb{R}^n$, $\boldsymbol{\xi}_1 \in \partial f(\boldsymbol{x}_1)$, and $\varepsilon_s > 0$;
  Set $k = 1$ and $\boldsymbol{d}_1 = -\boldsymbol{\xi}_1$;
  WHILE the termination condition $w_k \le \varepsilon_s$ is not met
    Find step sizes $t_L^k$ and $t_R^k$;
    Update $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$;
    Evaluate $f(\boldsymbol{x}_{k+1})$ and $\boldsymbol{\xi}_{k+1} \in \partial f(\boldsymbol{x}_k + t_R^k \boldsymbol{d}_k)$;
    IF $t_L^k > 0$ THEN
      Compute the search direction $\boldsymbol{d}_k$ using $\boldsymbol{\xi}_{k+1}$ and L-BFGS
        update;
    ELSE
      Compute the aggregate subgradient $\tilde{\boldsymbol{\xi}}_{k+1}$;
      Compute the search direction $\boldsymbol{d}_k$ using $\tilde{\boldsymbol{\xi}}_{k+1}$ and L-SR1
        update;
    END IF
    Set $k = k + 1$;
  END WHILE
  RETURN final solution $\boldsymbol{x}_k$;
END LMBM
```

2.5 Discrete Gradient Method (DGM)

Next we briefly describe the discrete gradient method (DGM). More details can be found in [4]. The DGM is the derivative free version of bundle methods. In contrast with bundle methods, which require the computation of a single subgradient of the objective function at each trial point, 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 $S_1 = \{\boldsymbol{g} \in \mathbb{R}^n \mid \|\boldsymbol{g}\| = 1\}$ 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 compute $i = \operatorname{argmax}\{|g_j|,\ j = 1, \dots, n\}$. For $\boldsymbol{e} \in G$ we define the sequence of n vectors $\boldsymbol{e}^j(\alpha) = (\alpha e_1, \alpha^2 e_2, \dots, \alpha^j e_j, 0, \dots, 0)$ $j = 1, \dots, n$ and for $\boldsymbol{x} \in \mathbb{R}^n$ and $\lambda > 0$, we consider the points

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

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

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

It has been proved in [3] that the closed convex set of discrete gradients is an approximation to the subdifferential $\partial f(\boldsymbol{x})$ for sufficiently small $\lambda > 0$ and it can be used to compute the descent direction for the objective. However, the computation of this set 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. 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

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

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

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

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

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

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

In [4] 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.

The pseudo-code of the DGM is the following:

```
PROGRAM DGM
  INITIALIZE $\boldsymbol{x}_1 \in \mathbb{R}^n$ and $k = 1$;
  OUTER ITERATION
    Set $s = 1$ and $\boldsymbol{x}_{k_s} = \boldsymbol{x}_k$;
    WHILE the termination condition is not met
      INNER ITERATION
        Compute the vector $\boldsymbol{v}_{k_s} = \text{argmin}_{\boldsymbol{v} \in \bar{D}(\boldsymbol{x}_{k_s})} \|\boldsymbol{v}\|^2$,
          where $\bar{D}(\boldsymbol{x}_{k_s})$ is a set of discrete gradients.
        IF $\|\boldsymbol{v}_{k_s}\| \leq \delta_k$ with $\delta_k > 0$ s.t. $\delta_k \searrow 0$ when $k \to \infty$ THEN
          Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_{k_s}$ and $k = k + 1$;
          Go to the next OUTER ITERATION;
        ELSE
          Compute the descent direction $\boldsymbol{d}_{k_s} = -\boldsymbol{v}_{k_s}/\|\boldsymbol{v}_{k_s}\|$;
          Find a step size $t_{k_s}$;
          Construct the following iteration $\boldsymbol{x}_{k_{s+1}} = \boldsymbol{x}_{k_s} + t_{k_s}\boldsymbol{d}_{k_s}$;
          Set $s = s + 1$ and go to the next INNER ITERATION;
        END IF
      END INNER ITERATION
    END WHILE
  END OUTER ITERATION
  RETURN final solution $\boldsymbol{x}_k$;
END DGM
```

2.6 Quasi-Secant Method (QSM)

Next we briefly describe the quasi-secant method (QSM). More details can be found in [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 [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 the given neighborhood of the current iteration point, which makes the method similar to the gradient sampling method.

We first define a quasi-secant for locally Lipschitz continuous functions.

DEFINITION 2.2. A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called a *quasi-secant* 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 quasi-secant by $\boldsymbol{v}(\boldsymbol{x}, \boldsymbol{g}, h)$.

For a given $h > 0$ let us consider the set of quasi-secants 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 quasi-secants as $h \searrow 0$:

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

A mapping $\boldsymbol{x} \mapsto QSec(\boldsymbol{x}, h)$ is called a *subgradient-related (SR)-quasi-secant 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-quasi-secants.*

The procedures used in the QSM for finding descent directions are pretty similar to those in the DGM but we use here the quasi-secant instead of the discrete gradient. 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 quasi-secants 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 quasi-secant $\boldsymbol{v}_{l+1}(\boldsymbol{x}, \boldsymbol{g}_{l+1}, h)$, update the bundle of quasi-secants $\bar{V}_{l+1}(\boldsymbol{x}) = \operatorname{conv}\{\bar{V}_l(\boldsymbol{x}) \cup \{\boldsymbol{v}_{l+1}(\boldsymbol{x}, \boldsymbol{g}_{l+1}, h)\}\}$ 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 the DGM.

The QSM is globally convergent for locally Lipschitz continuous functions under the assumption that the set $QSec(\boldsymbol{x}, h)$ is a SR-quasi-secant mapping [2]. The pseudocode of the QSM is the same as that for the DGM (see Section 2.5) when replacing discrete gradients $\boldsymbol{v}_{k_s}$ with quasi-secants $\boldsymbol{v}_{k_s}(\boldsymbol{x}_{k_s}, \boldsymbol{d}_{k_s}, h)$ and the set of discrete gradients $\bar{D}(\boldsymbol{x}_{k_s})$ with the set of quasi-secants $\bar{V}(\boldsymbol{x}_{k_s})$.

3 Numerical Experiments

In this section we compare the implementations of the methods described in the previous section. The more detailed description of the test results can be found in the technical report [26].

3.1 Solvers

The benchmarking results are dependent on the quality of the method and the quality of the implementation. Most methods have a variety of implementations. We have chosen the tested implementations due to their accessibility. The tested optimization codes with references to their more detailed descriptions are presented in Table 1. Then, we say a few words on each implementation and give the references from where the code can be downloaded. In Table 2 we recall the basic assumptions needed for the solvers.

Table 1: Tested pieces of software

Software	Author(s)	Method	Reference
`SolvOpt`	Kuntsevich & Kappel	Shor's r-algorithm	[24, 30, 50]
`PBNCGC`	Mäkelä	Proximal bundle	[39, 41]
`PNEW`	Lukšan & Vlček	Bundle-Newton	[34]
`LMBM`	Karmitsa	Limited memory bundle	[19, 20]
`DGM`	Bagirov et al.	Discrete Gradient	[4]
`QSM`	Bagirov & Ganjehlou	Quasi-Secant	[2]

`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 [24]). 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 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. 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 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 the dual projected gradient method proposed in [32]. `PBNCGC` can be used (free for academic purposes) via WWW-NIMBUS-system (http://nimbus.mit.jyu.fi/) [42] and it is available for downloading from http://napsu.karmitsa.fi/proxbundle/.

`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 (10) is solved by the subroutine `PLQDF1` [32]. `PNEW` is available for downloading from http://www.cs.cas.cz/luksan/subroutines.html.

`LMBM` is an implementation of the limited memory bundle method specifically developed for large-scale NSO. 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 quasi-secant solver for nonsmooth possibly nonconvex minimization. We have used both analytically calculated 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. 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. The experiments were performed on an Intel® Core™ 2 CPU 1.80GHz.

Table 2: Assumptions needed for software

Software	Assumptions to objective	Needed information
`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

The test set used in our experiments consists of classical academic nonsmooth minimization problems from the literature and their extensions. This includes, for instance, minmax, piecewise linear, piecewise quadratic and sparse problems as well as highly nonlinear nonsmooth problems. The test set was grouped into ten subclasses according to problems convexity and size:

XSC: Extra-small convex problems, $n \leq 20$ (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 [37]);

XSNC: Extra-small nonconvex problems (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 [37]);

SC: Small-scale convex problems, $n = 50$ (Problems 1 – 5 in [19], and 2 and 5 in `TEST29` [33] and six maximum of quadratic functions in report [26]);

SNC: Small-scale nonconvex problems (Problems 6 – 10 in [19], and 13, 17 and 22 in `TEST29` [33] 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 SC and SNC problems);

XLC and XLNC: Extra-large-scale convex and nonconvex problems, $n = 4000$ (see SC and SNC problems but only two maximum of quadratic functions with diagonal matrix);

Problems 2, 5, 13, 17, and 22 in `TEST29` are from the software package UFO (Universal Functional Optimization) [33]. They may also be found in [18].

All the solvers tested are so-called local methods. That is they do not attempt to find the global minimum of the nonconvex objective function. To enable the fair comparison between different solvers, the nonconvex test problems were selected such that all the solvers converged to the same local minimum of a problem. However, it is worth of mention that, when solvers converged to different local minima (i.e. in some nonconvex problems omitted from the test set), solvers `LMBM` and `SolvOpt` usually converged to the one local minimum, while `PBNCGC`, `DGM`, and `QSM` converged to another. Solver PNEW converged sometimes with the first group and some other times with the second. In addition, 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 [37] all the other solvers converged to the minimum reported in [37] but DGM and QSM "converged" to minus infinity.

3.3 Termination, parameters, and acceptance of the results.

Each implementation of an optimization method involves a variety of tunable parameters. The improper selection of these parameters can skew any benchmarking result. In our experiments, we have used the default settings of the codes as far as possible. However, some parameters naturally have to be tuned, in order to get any reasonable result.

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

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

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 problems from three different classes: problems 2.4 in [37] (XSC), 3.15 in [37] (XSNC), and 3 in [19] 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, we terminated the experiments if the elapsed CPU time exceeded half an hour for XS, S, M, and L problems and an hour for XL problems. With XS, S, and convex M problems this never happened. In addition, solvers LMBM and QSMA never needed the whole time limit. For other problems and solvers the appropriate discussion is given with the results of that problem class.

We have accepted the results for XS and X 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 one 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, we use $\gamma = 10^{-10}$ with it. 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 – 8 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 CPU time and number of function and subgradient evaluations (evaluations for short). In the following graphs $\rho_s(\tau)$ denotes the logarithmic performance profile

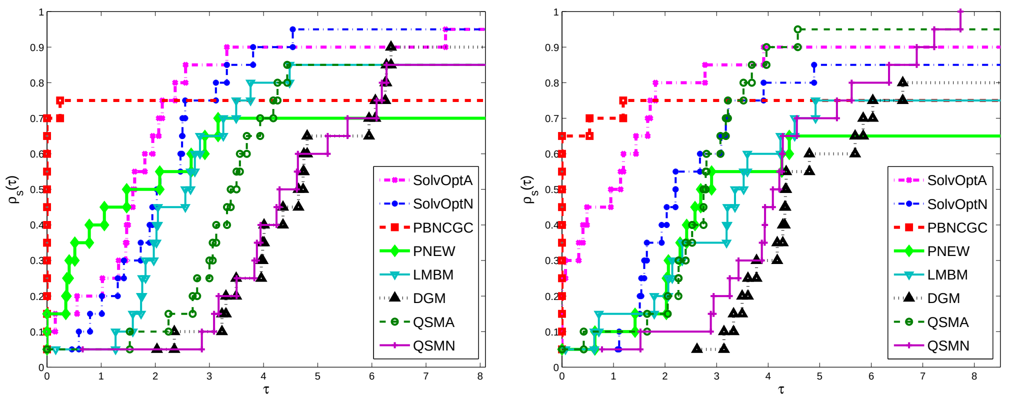
$$\rho_s(\tau) = \frac{\text{no. of problems where } \log_2(r_{p,s}) \leq \tau}{\text{total no. of problems}} \tag{11}$$

with $\tau \geq 0$, where $r_{p,s}$ is the performance ratio between the time (number of evaluations) to solve problem p by solver s over the lowest time (number of evaluations) required by any of the solvers. The ratio $r_{p,s}$ is set to infinity (or some sufficiently large number) if solver s fails to solve problem p.

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 Extra-small problems

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



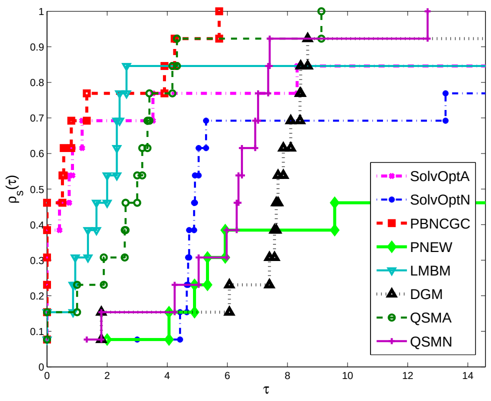
(a) Convex (b) Nonconvex

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

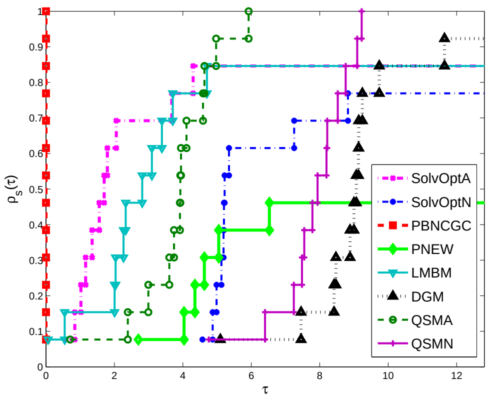
All the solvers succeed in solving approximately the same number of XSC and XSNC problems. However, it is noteworthy that `QSM` succeed in solving more nonconvex than convex problems. Most of the failures reported here (see Figure 1) are, in fact, inaccurate results: all the solvers but `PNEW` succeed in solving at least 95% of XSC problems with respect to the relaxed tolerance $\varepsilon = 10^{-2}$. The corresponding percentage for XSNC problems was 85%. The reason for the 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 [36]. If we, instead of only one (default) value, used a selected value for this parameter, also the solver `PNEW` solved 85% of XSNC problems.

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 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 SC problems (see Figure 2(a)) and `LMBM` was the most efficient solver in SNC problems (see Figure 3(a)).

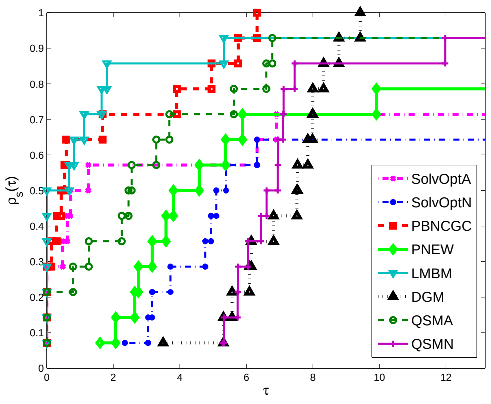


(a) CPU-time

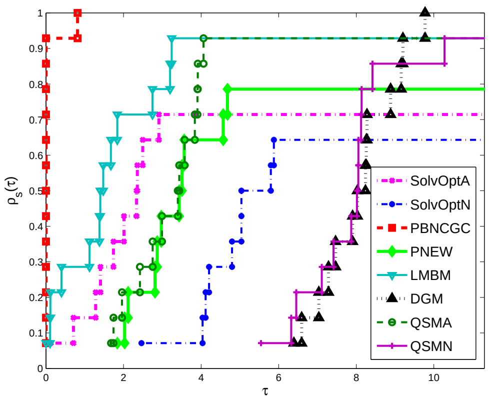


(b) Evaluations

Figure 2: Results for SC problems (13 problems with $n = 50$, $\varepsilon = 5 \cdot 10^{-4}$).



(a) CPU-time



(b) Evaluations

Figure 3: Results for SNC problems (14 problems with $n = 50$, $\varepsilon = 5 \cdot 10^{-4}$).

As with XS problems, the failures here (see Figures 2 and 3) are mostly inaccurate results: all the solvers but `PNEW` succeed in solving at least 90% of SC problems and 85% of SNC problems with respect to the relaxed tolerance. Especially the subgradient solvers `SolvOptA` and `SolvOptN` had difficulties with the accuracy in the nonconvex case: `SolvOptN` solved only about 64% of SNC problems with respect to tolerance $\varepsilon = 5 \cdot 10^{-4}$ and 92% with $\varepsilon = 10^{-2}$. For `SolvOptA` the corresponding values were 71% and 86%. Note, however that with XS problems `SolvOpt` was one of the most accurate solvers also in the nonconvex settings.

With the other solvers there was not a big difference in solving convex or nonconvex problems but with `PNEW`: It failed to solve all but one of the convex piecewise quadratic problems and succeed in solving all but one non-quadratic problems. In the nonconvex case PNEW succeed in solving all the piecewise quadratic problems but it had some difficulties with the other problems. Again, the reason for these failures is in its sensitivity to internal parameter XMAX.

3.4.3 Medium-scale and large problems

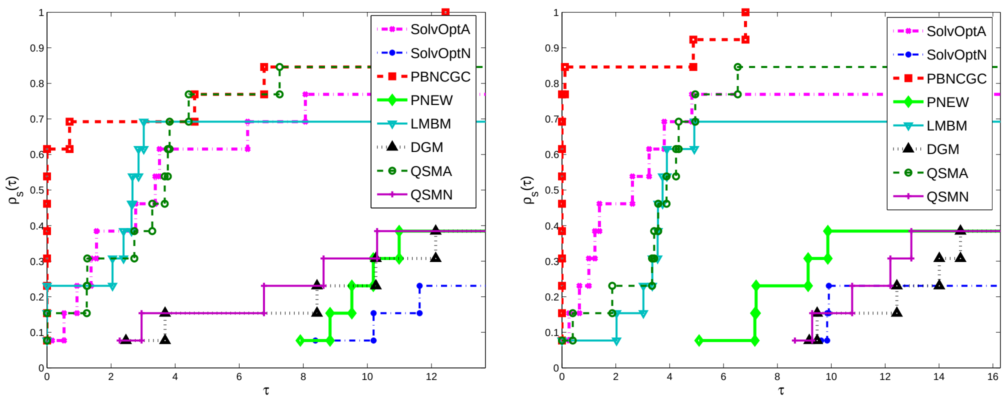
The results for medium- and large-scale problems reveal similar trends. Thus, we show here only the results for large problems in Figures 4 – 6. More illustrated results also for medium-scale problems can be found in the technical report [26].

When solving M and L problems, the solvers divided into two groups in terms of the efficiency: 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. In the nonconvex case (see Figure 5), the inaccuracy of SolvOptA made it slide to the group of less efficient solvers. In Figure 6, that illustrates the results with the relaxed tolerance, SolvOptA is again among the more efficient solvers. At the same time, the successfully solved piecewise quadratic problems almost rose PNEW to the group of more efficient solvers in MNC settings (especially, when comparing the numbers of evaluations, see [26]). The similar trend can not been seen here, since in LNC problems the time limit was exceeded in all the piecewise quadratic problems with PNEW.

Although PBNCGC was usually the most efficient solver tested (on 60% of both M and L problems when comparing the computational times), it was also the one who needed the longest time to compute problem 3 in [19] (both in M and L settings). Indeed, an average time used to solve a MC (LC) problem with PBNCGC 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 succeed in solving). The same trend can be seen in the nonconvex case. The efficiency of PBNCGC is mostly due to its efficiency in piecewise quadratic problems: it was the most efficient solver in almost all piecewise quadratic problems when comparing the computational times and superior when comparing the numbers of evaluations.

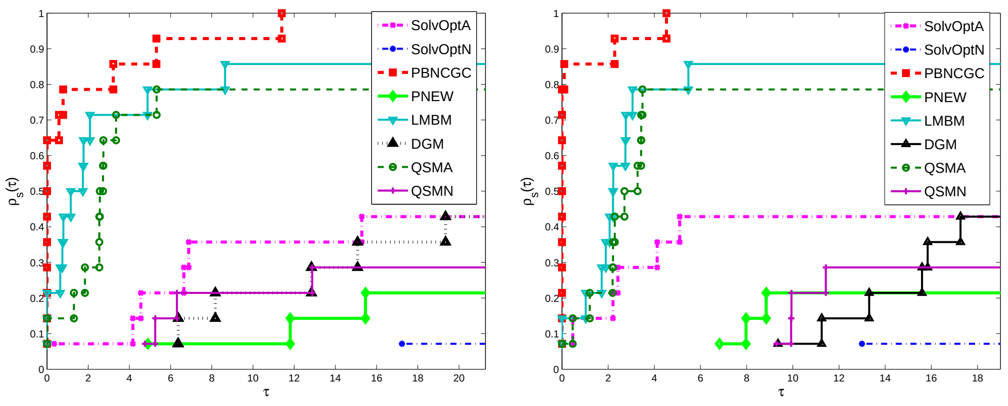
PBNCGC was also the most reliable solver tested in medium-scale settings while SolvOptN had some serious difficulties with the accuracy, especially in nonconvex cases: for instance, with the relaxed tolerance SolvOptN solved almost 80% of MNC problems while with the tolerance $\varepsilon = 10^{-3}$ less than 30%. Similar effect could be seen with SolvOptA, although not as pronounced. Naturally, with LNC problems the difficulties with the accuracy degenerated (see Figures 5 and 6). Also LMBM and QSMA had some difficulties with the accuracy in LNC case. With the relaxed tolerance, they solved all LNC problems. With this tolerance LMBM was clearly the most efficient solver in the non-quadratic problems and the computational times of both LMBM and QSMA were comparable with those of PBNCGC in the whole test set (see Figure 6).

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 [19] and 2 in TEST29 [33]). With the other methods, there were some difficulties already with $n = 50$ and some more with $n = 200$ (note that for S, M and L settings, the problems are the same, only the number of variables is changing). Further, solvers DGM, LMBM, and QSMN failed to solve (possible in addition to the two above mentioned



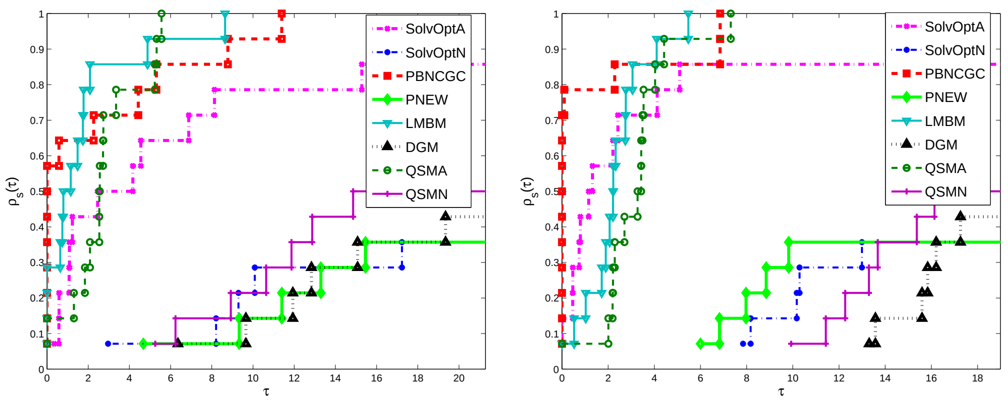
(a) CPU-time (b) Evaluations

Figure 4: Results for LC problems (13 problems with $n = 1000$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

Figure 5: Results for LNC problems (14 problems with $n = 1000$, $\varepsilon = 10^{-3}$).



(a) CPU-time (b) Evaluations

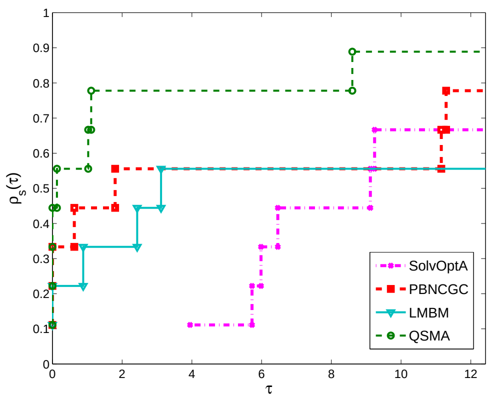
Figure 6: Results for LNC problems, low accuracy (14 problems with $n = 1000$, $\varepsilon = 10^{-2}$).

problems) two piecewise linear problems (Problem 2 in [19] and 5 in `TEST29` [33]) and also `QSMA` failed to solve one of them. 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 [19] 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.

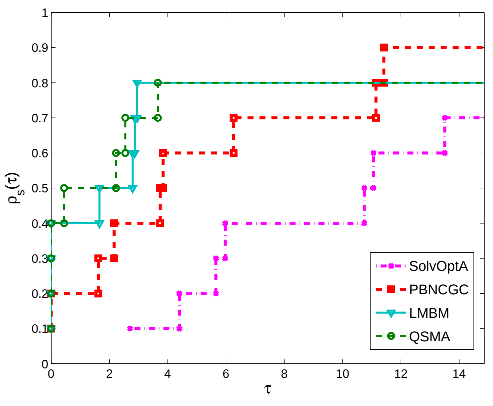
Naturally, for the solvers using difference approximations or discrete gradients, the number of evaluations (and thus also the computational time) grows enormously when the number of variables increases. Particularly, in L problems the time limit was exceeded with all these solvers in all the piecewise quadratic problems. Thus, the number of failures with these solvers is probably larger than it would be without the time limit. However, in almost all the cases, the results obtained were still very far from the minima, indicating very long computational times. In the non-quadratic L problems the time limit was exceeded once with `DGM` (accurate result), twice with `SolvOptN` (accurate results) and `QSMN` (rather far from the minima) and three times with `PBNCGC` (accurate results) and `PNEW` (both accurate result and results far from the minima). In MNC problems the time limit was exceeded couple of times with `DGM` and `QSMN`. In all these cases the results were acceptable at least with the relaxed tolerance.

3.4.4 Extra large problems

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



(a) Convex



(b) Nonconvex

Figure 7: CPU-times for XLC (9 ps.) and XLNC (10 ps.) problems ($n = 4000$, $\varepsilon = 10^{-3}$).

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 (for more details, see the technical report [26]). `QSMA` was also the most reliable of the solvers tested (see Figure 7(a)).

In the nonconvex case `LMBM` and `QSMA` were approximately as good in computational times, evaluations and reliability (see Figure 7(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.

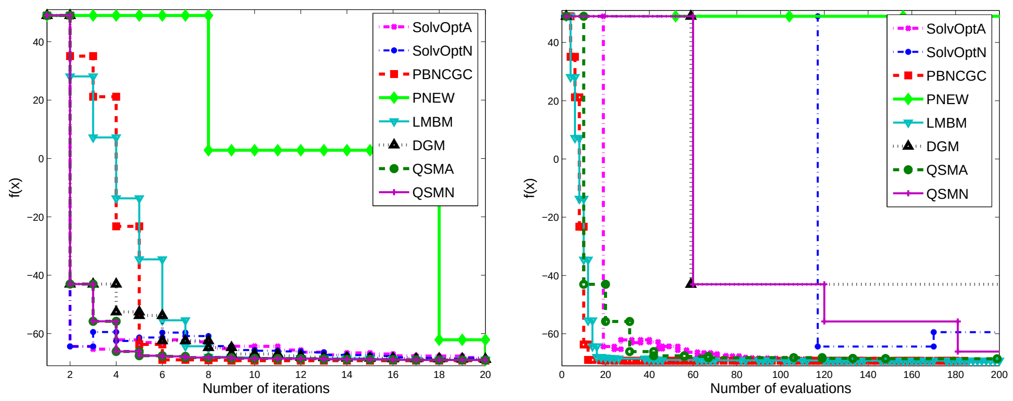
As before, LMBM solved all the problems it could solve in a relatively short time while with all the other solvers there were a wide variation in the computational times elapsed for different problems. However, in the convex case, the efficiency of LMBM was again ruined by its unreliability.

In XL problems, SolvOpt stopped twice because of the time limit (one accurate and one very far away from the optimum result) and PBNCGC stopped six times (19 problems total). Four of these terminations were within the desired accuracy, indicating that PBNCGC could be more efficient in terms of used computational time if it knew when to stop.

3.4.5 Convergence speed

In this subsection we study (experimentally) the convergence speed of the algorithms using one small-scale convex problem (Problem 3 in [19]). 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 convergence speed of the solvers tested (see Figure 8(b)), although, theoretically, proximal bundle method is at most linearly convergent.



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

Figure 8: Function values versus iterations (a), and function values versus number of function and subgradient evaluations (b).

With PNEW a large amount of subgradient evaluations is needed to compute the approximate Hessian. Although PNEW finally found the minimum, it did not decrease the value of the function in 200 first evaluations. Solvers SolvOptA, SolvOptN, DGM, QSMA, and QSMN took a very big step downwards already in iteration two (see Figure 8(a)). However, they took quite many function evaluations per iteration. In Figure 8 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 was SolvOptN, which never reached the desired accuracy (the final value obtained after 42 iterations and 2342 evaluations was -68.915).

4 Conclusions

We have tested the performance of different nonsmooth optimization solvers in solving 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 might be ambiguous what is the best, we give credentials both in the cases when the most efficient (in terms of used CPU time) or the most reliable solver is sought out. When there is more than one recommendation in Table 3, 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 possible solver to be reckoned for the problem class.

Although 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 extra-small problems, great variations in computational time in larger problems and the earlier results make us believe that our test set favored this solver over the others a little bit. PBNCGC is especially efficient for the piecewise quadratic and piecewise linear problems. Moreover, PBNCGC usually used the least number of evaluations for problems of any size. Thus, it should be a good choice for a solver when the objective function value and/or the subgradient are expensive to compute.

The limited memory bundle solver LMBM suffered from ill-fitting test problems in convex S, M, L and XL cases. In addition, LMBM lost out to PBNCGC in all but one piecewise quadratic problems. LMBM was quite reliable in the nonconvex case in all number of variables tested and it solved all the problems — even the largest ones — in relatively short time. LMBM works best for (highly) nonlinear functions.

The bundle-Newton solver PNEW suffers greatly from the fact that it is very sensitive to the choice of the internal parameter XMAX. A light tuning of this parameter (e.g. using a default value XMAX $= 1000$ and subsequently the smallest recommended value XMAX $= 2$ and then choosing the better result) would have yielded better results and, especially, the degree of success would have been much higher. In [40], PNEW is reported to be very efficient in quadratic problems. Also in our experiments, it solved the (nonconvex) piecewise quadratic problems faster that the non-quadratic. However, except for some XS problems, it did not beat the other solvers in these problems due to a large approximation of the Hessian matrix required.

The implementations of Shor's r-algorithm SolvOptA and SolvOptN did their best in XS problems both in convex and nonconvex settings. Nevertheless, SolvOptA solved also S, M, L and even XL problems (convex) rather efficiently. In larger nonconvex problems both of these methods suffered from inaccuracy.

Thus, when comparing the reliability in M, L and XL 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 problems. However, with QSMA there were some variations on the computational time 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, namely the discrete gradient solver DGM and quasi-secant solver with discrete gradients QSMN, usually lost out in efficiency to the

Table 3: 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	XS, S	PBNCGC, SolvOptA	PBNCGC, SolvOptA
or sparse	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	XS	PBNCGC, (PNEW$^{(1)}$, SolvOptA)	QSMA, SolvOptA
are expensive	S, M, L, XL	PBNCGC, (LMBM$^{(4)}$, SolvOptA)	PBNCGC, (LMBM$^{(4)}$, QSMA)
Subgradients are not	XS	SolvOptN	QSMN, SolvOptN$^{(5)}$, (DGM)
available	S, M	SolvOptN, QSMN	DGM, QSMN
	L	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.

solvers using analytical subgradients. However, in XS and S problems the reliability of `DGM` and `QSMN` seems to be very good with both convex and nonconvex problems. In larger cases, the usage of a method employing subgradients is, naturally, recommended. Nevertheless, if one needs 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. Moreover, in the case of highly nonconvex functions (supposing that one is seeking 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 are thankful to the anonymous referees for their valuable comments. We would also 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), the University of Ballarat (Australia) and the Australian Research Council.

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 (2010), 3–18.

[3] Bagirov, A. M., Continuous subdifferential approximations and their applications, *Journal of Mathematical Sciences,* 115, 5 (2003), 2567–2609.

[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] Fuduli, A., Gaudioso, M., and Giallombardo, G. A DC piecewise affine model and a bundling technique in nonconvex nonsmooth minimization. *Optimization Methods and Software 19*, 1 (2004), 89–102.

[16] Fuduli, A., Gaudioso, M., and Giallombardo, G. Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. *SIAM Journal on Optimization 14*, 3 (2004), 743–756.

[17] Gaudioso, M., and Monaco, M. F. Variants to the cutting plane approach for convex nondifferentiable optimization. *Optimization 25* (1992), 65–75.

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

[19] 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.

[20] 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.

[21] Hare, W., and Sagastizábal, C. A redistributed proximal bundle method for nonconvex optimization. Available in web page <URL: `http://www.optimization-online.org/DB_HTML/2009/04/2273.html`>, 2009. (April 14th, 2010).

[22] Haslinger, J., and Neittaanmäki, P. *Finite Element Approximation for Optimal Shape, Material and Topology Design*, 2nd ed. John Wiley & Sons, Chichester, 1996.

[23] Hiriart-Urruty, J.-B., and Lemaréchal, C. *Convex Analysis and Minimization Algorithms II*. Springer-Verlag, Berlin, 1993.

[24] Kappel, F., and Kuntsevich, A. An implementation of Shor's r-algorithm. *Computational Optimization and Applications 15* (2000), 193–205.

[25] Kärkkäinen, T., and Heikkola, E. Robust formulations for training multilayer perceptrons. *Neural Computation 16* (2004), 837–862.

[26] Karmitsa, N., Bagirov, A., and Mäkelä, M. M. Empirical and theoretical comparisons of several nonsmooth minimization methods and software. TUCS Technical Report, No. 959, Turku Centre for Computer Science, Turku, 2009. The report is available online at http://tucs.fi/publications/insight.php?id=tKaBaMa09a.

[27] 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.

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

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

[30] Kuntsevich, A., and Kappel, F. SolvOpt — the solver for local nonlinear optimization problems. Karl-Franzens University of Graz: Graz, Austria, 1997.

[31] 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.

[32] 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.

[33] 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.

[34] Lukšan, L., and Vlček, J. A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming 83* (1998), 373–391.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] Mäkelä, M. M., and Neittaanmäki, P. *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control.* World Scientific Publishing Co., Singapore, 1992.

[42] Miettinen, K., and Mäkelä, M. M. Synchronous approach in interactive multiobjective optimization. *European Journal of Operational Research 170*, 3 (2006), 909–922.

[43] Mistakidis, E. S., and Stavroulakis, G. E. *Nonconvex Optimization in Mechanics. Smooth and Nonsmooth Algorithms, Heuristics and Engineering Applications by the F.E.M.* Kluwer Academic Publishers, Dordrecht, 1998.

[44] Moreau, J. J., Panagiotopoulos, P. D., and Strang, G., Eds. *Topics in Nonsmooth Mechanics.* Birkhäuser Verlag, Basel, 1988.

[45] Noll, D., Prot, O., and Rondepierre, A. A proximity control algorithm to minimize nonsmooth and nonconvex functions. *Pacific Journal of Optimization 4*, 3 (2008), 571–604.

[46] 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.

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

[48] 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.

[49] 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.

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

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

[52] 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.