Benchmark of Some Nonsmooth Optimization Solvers for Computing Nonconvex Proximal Points

Warren Hare* Claudia Sagastizábal†‡

February 28, 2006

Key words: nonconvex optimization, proximal point, algorithm benchmarking
AMS 2000 Subject Classification:
Primary: 90C26, 90C30, 68W40
Secondary: 49N99, 65Y20, 68W20

Abstract

The major focus of this work is to compare several methods for computing the proximal point of a nonconvex function via numerical testing. To do this, we introduce two techniques for randomly generating challenging nonconvex test functions, as well as two very specific test functions which should be of future interest to Nonconvex Optimization Benchmarking. We then compare the effectiveness of four algorithms ("CProx," "N1CV2," "N2FC1," and "RGS") in computing the proximal points of such test functions. We also examine two versions of the CProx code to investigate (numerically) if the removal of a "unique proximal point assurance" subroutine allows for improvement in performance when the proximal point is not unique.

1 Introduction

Benchmarking is an important tool in numerical optimization, not only to evaluate different solvers, but also to uncover solver defficiencies and to ensure the quality of optimization software. In particular, many researchers in the area of continuous optimization have devoted considerable work to collecting suitable test problems, and comparing different solvers performance both in Linear and Nonlinear Programming; we refer to `http://plato.asu.edu/bench.html` and `http://www-neos.mcs.anl.gov/` for an exhaustive list of different codes and related works.

Nonsmooth Optimization (NSO), by contrast, badly lacks the test problems and performance testing of specialized methods. Although the state of the art for convex nonsmooth optimization is quite advanced and reliable software is in use (see [BGLS03] and references therein), very few works in the literature include comparisons with some other method. In addition to the low availability of free NSO solvers, a possible reason is the lack of a significant library of NSO test functions.

The situation is even worse in Nonconvex Nonsmooth Optimization: there are not many test problems, and there are not many specially designed solvers either. Prior to the random gradient

*Dept. of Comp. and Software, McMaster University, Hamilton, ON, Canada, `whare@cecm.sfu.ca`
†IMPA, Estrada Dona Castorina 110, Jardim Botânico, Rio de Janeiro RJ 22460-320, Brazil. On leave from INRIA, France. Research supported by CNPq Grant No. 383066/2004-2, `sagastiz@impa.br`
‡Both authors would like to thank Michael Overton for his assistance in using the RGS code.

sampling algorithm introduced in [BLO02], NSO methods consisted essentially in "fixing" some convex bundle algorithm to adapt it to the nonconvex setting, [LSB81], [Mif82], [LV98]. The recent work [HS05] made a contribution in the area, by giving a new insight on how to pass from convex to nonconvex bundle methods. More precisely, in [HS05] the problem of computing a proximal point of a nonconvex nonsmooth function (cf. (1) below) is addressed. A full nonconvex nonsmooth algorithm based on the building blocks developed in [HS05] is very promising, because the more robust and efficient variants of convex bundle methods are of the proximal type, [Kiw90], [LS97].

In this work we provide a comprehensive suite of NSO functions and give a benchmark for the specific NSO problem of computing the proximal point of a nonconvex function, i.e.,

$$\text{find} \quad p \in P_R f(x) := \operatorname{argmin}_{w \in \mathbb{R}^N} \left\{ f(w) + R\frac{1}{2}|w - x|^2 \right\}, \tag{1}$$

where $R > 0$ and $x \in \mathbb{R}^N$ are given. Our purpose is twofold. First, we assemble a sufficient set of relevant NSO functions in order to be able to evaluate numerically different approaches. Second, by comparing different NSO solvers performance on problems of type (1), we start tuning the method in [HS05] in order to use it as part of a future nonconvex proximal bundle method.

The main concern in presenting benchmark results is in removing some of the ambiguity in interpreting results. For example, considering only arithmetic means may give excessive weight to single test cases. Initial performance comparisons were developed for complementarity problems in [BDF97]. In [DM02], the approach was expanded by the introduction of performance profiles, reporting (cumulative) distribution functions for a given performance metric. Performance profiles therefore present a descriptive measure providing a wealth of information such as solver efficiency, robustness, and probability of success in compact graphical form. For the numerical testing in this work, we give performance profiles in Sections 4, 5, and 6. All the statistical information on the results in reported in the form of tables in Appendix A.

For defining the NSO functions composing the battery, we consider two categories:

- randomly generated functions, either defined as the pointwise maximum of a finite collection of quadratic functions, or as sum of polynomial functions (operating on the absolute value of the point components); see Sections 4 and 5, respectively.
- two specific functions, the *spike* and the *waterfalls* functions; described in Section 6.

The maximum of quadratic functions family, in particular, is defined so that any of the generated problems (of the form (1)) are solved by $x^* = 0 \in \mathbb{R}^N$. This is an interesting feature, because it allows to measure the quality of solutions found by a given solver. For all the functions, an *oracle* is defined, which for any given $x \in \mathbb{R}^N$ computes the value $f(x)$ and *one* subgradient for the function, i.e., some $g_f \in \partial f(x)$. As usual in NSO, we do not assume that there is any control over which particular subgradients are computed by the oracle. In this setting, rather than measuring effectiveness of an algorithm in terms of time, the number of oracle calls made should be considered.

The solvers we test in this benchmark are CProx, N1CV2, N2FC1, and RGS, corresponding, respectively, to [HS05], [LS97], [LSB81], and [BLO02]. Due to time constraints, we did not test the bundle, Newton, and variable metric nonsmooth methods described in [LV00]. We will include these solvers in a future benchmarking work.

The remainder of this paper is organized as follows. In Section 2 we provide the theoretical background for this work, as well as a brief description of each the algorithms benchmarked in this

work. Sections 4, 5, and 6 contain the descriptions, performance profiles, and conclusions for our battery of test functions. Statistical tables for the tests are stored in Appendix A. We conclude with some general discussion in Section 7.

2 Theoretical background and Solvers

Before describing the test functions and giving the numerical results, we recall some important concepts related to Variational Analysis and the proximal point mapping. We also give a succinct description of the solvers used for our benchmark.

2.1 Brief review of Variational Analysis

For $f : \mathbb{R}^N \to \mathbb{R} \cup \{+\infty\}$ proper lower semicontinuous (lsc) function and point $\bar{x} \in \mathbb{R}^N$ where f is finite-valued, we use the Mordukhovich subdifferential denoted by $\partial f(\bar{x})$ in [RW98, Def 8.3]. For such a function we use the term *regular* to refer to subdifferential regularity as defined in [RW98, Def. 7.25], and the following definitions:

- The function f is *prox-regular* at $\bar{x}$ for $\bar{v} \in \partial f(\bar{x})$ if there exist $\epsilon > 0$ and $r > 0$ such that

$$f(x') \geq f(x) + \langle v, x' - x \rangle - \frac{r}{2}|x' - x|^2$$

  whenever $|x' - \bar{x}| < \epsilon$ and $|x - \bar{x}| < \epsilon$, with $x' \neq x$ and $|f(x) - f(\bar{x})| < \epsilon$, while $|v - \bar{v}| < \epsilon$ with $v \in \partial f(x)$. We call a function *prox-regular* at $\bar{x}$ if it is prox-regular at $\bar{x}$ for all $v \in \partial f(\bar{x})$.
- The function f is *lower-$\mathcal{C}^2$* on an open set V if at any point x in V, f appended with a quadratic term is a convex function on an open neighbourhood V' of x; see [RW98, Thm. 10.33, p. 450].
- The function f is *semi-convex* if f appended with some quadratic term is a convex function on $\mathbb{R}^N$.

For the definitions above, the following relations hold:

$$\begin{array}{rcl} f \text{ convex} & \Rightarrow & f \text{ semi-convex} \\ f \text{ semi-convex} & \Rightarrow & f \text{ lower-}\mathcal{C}^2 \text{ on any open set } V \\ f \text{ lower-}\mathcal{C}^2 \text{ on an open set } V & \Rightarrow & f \text{ prox-regular at all } x \in V \\ f \text{ prox-regular at } x & \Rightarrow & f \text{ regular at } x. \end{array} \tag{2}$$

The first two implications are obvious, while final two can be found in [RW98, Prop 13.33] and [RW98, p. 610]. Additional relations to semismooth functions [Mif77] can be found in [HU85].

- The *proximal point mapping* of the function f at the point x is defined by $P_R f(x)$ given in (1), where the minimizers can form a nonsingleton or empty set, [Mor65], [RW98, Def 1.22]. The *Moreau envelope* of f at x, also called proximal envelope, or Moreau-Yosida regularization, is given by the optimal value in (1), when considered a mapping of x.
- The function f is *prox-bounded* if for some prox-parameter R and prox-center x, the proximal point mapping is nonempty. In this case $P_R f$ will be nonempty for any $x \in \mathbb{R}$ [RW98, Ex 1.24]. The *threshold of prox-boundedness*, denoted by r_{pb}, is then the infimum of the set of all R such that $P_R f$ is nonempty for some $x \in \mathbb{R}^N$.

The following lemma reveals the importance of implications (2), more specifically of prox-regularity, when considering proximal points.

Lemma 2.1 *[HP06, Thm 2.3] Suppose f is a prox-bounded function, and the prox-parameter R is greater than the threshold of prox-boundedness. Then for any prox-center x the following holds:*

$$p = P_R f(x) \Rightarrow 0 \in \partial f(p) + R(p - x) \tag{3}$$

Furthermore, if f is prox-regular at $\bar{x}$ for $\bar{v} \in \partial f(\bar{x})$, then there exists a neighbourhood V of $\bar{x} + \frac{1}{R}\bar{v}$ such that, whenever R is sufficiently large, the inclusion in (3) *is necessary and sufficient for $P_R f(x)$ to be single-valued for all $x \in V$.*

When generating our collection of NSO functions, we make use of Lemma 2.1 to define, in addition to the test function, a prox-parameter R sufficiently large for the proximal point to exist; see the maximum of quadratic functions family in Section 4.

2.2 Algorithms composing the benchmark

As far as we can tell, CProx is currently the only method specifically designed for proximal point calculations on nonconvex functions. Nevertheless, any general purpose NSO algorithm could be used to compute a proximal point for a nonconvex function. More precisely, since (1) is a nonconvex nonsmooth unconstrained optimization problem, the computation of the proximal point could be done by minimizing the function

$$f_R(\cdot) := f(\cdot) + R\frac{1}{2}|\cdot - x^0|^2.$$

Note that, having an oracle for f (i.e., computing $f(x)$ and $g(x) \in \partial f(x)$), gives in a straightforward manner an oracle for f_R, since for any $x \in \mathbb{R}^N$

$$f_R(x) = f(x) + R\frac{1}{2}|x - x^0|^2 \quad \text{and} \quad g_R(x) = g(x) + R(x - x^0) \in \partial f_R(x).$$

For our benchmark, in addition to CProx, we numerically examine three NSO methods: N1CV2, N2FC1, and RGS. We now give a brief description of the main features of the solvers tested.

2.3 Bundle solvers N1CV2 and N2FC1

Bundle methods appear as a two steps stabilization of cutting-planes methods. First, they generate a sampling point y^{j+1} by solving a stabilized quadratic programming subproblem. Second, they *select* some iterates satisfying certain descent condition. These selected points, or *stability centers*, form a subsequence $\{x^k\} \subset \{y^j\}$ that aims at guaranteeing a sufficient decrease in f_R. Specifically, given the last generated stability center x^k and $\mu_k > 0$, y^{j+1} is the solution to

$$\min_{y \in \mathbb{R}^N} \check{f}_R(y) + \frac{1}{2}\mu_k |y - x^k|^2 ,$$

where $\check{f}_R$ is a cutting-planes model for the function f_R.

If for some $m_1 \in]0,1[$

$$f_R(y^{j+1}) \leq f_R(x^k) - m_1 \left(f_R(x^k) - \check{f}_R(y^{j+1}) - \frac{1}{2}\mu_k |y^{j+1} - x^k|^2 \right) ,$$

then y^{j+1} provides a sufficient decrease. A new stability center is thus defined, $x^{n+1} = y^{j+1}$, μ_k is updated, and both j and n are increased.

Otherwise, the model $\check{f}_R$ is considered not accurate enough. Then a *null step* is declared: there is no new stability center and only j is increased.

Note that in both cases the new oracle information $f_R(y^{j+1})$ and $g_R^{j+1} \in \partial f_R(y^{j+1})$ is incorporated to the cutting-planes model $\check{f}_R$.

As usual in nonlinear programming, the method can be globalized by merging the bundling technique above with a line-search. Since the line-search function is not differentiable, this may be a tricky issue. A possibility is to do a curve-search, like subalgorithm (CS) in [LS97], replacing μ_k in the quadratic programming subproblem by μ_k/t, where $t > 0$ is a trial step-size used to interpolate/extrapolate until either a null step or a new stabilized center is found. We omit entering into more technicalities here and refer to [LS97] for more details.

The first bundle algorithm we tested is N1CV2, the proximal bundle method from [LS97], designed along the lines described above for **convex** NSO. Parameters μ_k are updated with a "poor-man" rule that mimics a quasi-Newton method for the Moreau envelope of $\check{f}_R$. Although the curve-search CS can work also for semismooth functions, the method is designed for convex functions only, so poor performance is to be expected for highly nonconvex functions, such as the piecewise polynomials and the waterfalls families in Sections 5 and 6.2 below.

The second bundle method is N2FC1, an ε-steepest descent bundle method specially designed for nonconvex minimization with box constraints (we set $|x_i| \leq 10^5$ in our runs). The method is described in [LSB81]. It is essentially a dual method, following the lines given above for bundle methods. The most important difference with the convex case is that the cutting-planes model $\check{f}_R$ is modified to ensure that it remains a lower approximation of f_R even when the function is nonconvex. With this modification, there is no longer a Moreau envelope to refer to, so the parameter updating used in N2FC1 is mostly heuristic. For this reason, poor performance of the method is to be expected for "almost convex" functions, such as some in the maximum of quadratic functions family in Section 4. N2FC1 performs a linesearch that is shown to end at a finite number of steps for semismooth functions. Likewise, convergence of the method is shown for semismooth functions.

A Fortran 77 code for N1CV2 is available upon request at `http://www-rocq.inria.fr/estime/modulopt/optimization-routines/n1cv2.html`. N2FC1 can be requested from its author, Claude Lemaréchal, `http://www.inrialpes.fr/bipop/people/lemarechal/`.

2.4 Algorithm CProx

This algorithm, specifically designed to compute proximal points of nonconvex functions, was introduced in [HS05]. It is shown to be convergent for lower-$\mathcal{C}^2$ functions, whenever the prox-parameter R is sufficiently large.

Each iteration k of CProx proceeds by splitting the proximal point parameter R into two values η_k and μ_k such that $\eta_k + \mu_k = R$. Rewriting the proximal point problem (1) as

$$\operatorname{argmin} \left\{ f(y) + R\frac{1}{2}|y - x^0|^2 \right\} = \operatorname{argmin} \left\{ \left(f(y) + \eta_k \frac{1}{2}|y - x^0|^2 \right) + \mu_k \frac{1}{2}|y - x^0|^2 \right\},$$

the idea of CProx is, instead of working with the nonconvex function f, to work on the (hopefully more convex) function $f_{\eta_k} := f + \eta_k \frac{1}{2}|\cdot - x^0|^2$. Somewhat similarly to bundle methods, CProx creates a piecewise linear model of f_{η_k}, and calculates the proximal point for this model by solving a quadratic programming problem. A subroutine for dynamically selecting η_k and μ_k ensures that these parameters will eventually satisfy the properties required for convergence of the algorithm. An additional feature of CProx is that it checks that R is sufficiently large for (3) to hold as an equivalence.

A Matlab code for CProx is available upon request from the authors.

2.5 Random gradient sampling algorithm

The final algorithm we consider in this paper is the random gradient sampling algorithm RGS, introduced in [BLO02]. Unlike the other methods examined in this work, RGS is not a bundle-like algorithm. It is however designed for dealing with oracle based minimization. Specifically, this method is intended for minimizing functions that are continuous and for which the gradient exists and is readily computable almost everywhere on $\mathbb{R}^N$. Algorithm RGS is largely inspired by the classical Clarke subgradient formula

$$\partial f_R(\bar{x}) = \limsup_{\varepsilon \searrow 0} \operatorname{conv}\{\nabla f_R(x) : x \text{ such that } |x - \bar{x}| \leq \varepsilon\}.$$

Briefly, at iteration k, given x^k, the gradient of f_R is computed at x^k and at randomly generated points near x^k within a sampling diameter, and the convex combination of these gradients with smallest 2-norm, say d^k, is computed by solving a quadratic program. One should view $-d^k$ as a kind of stabilized steepest descent direction. A line search is then used to obtain the next iterate and reduce the function value. If the magnitude of d^k is below a prescribed tolerance, or a prescribed iteration limit is exceeded, the sampling diameter is reduced by a prescribed factor, and the process is repeated. Besides its simplicity and wide applicability, a particularly appealing feature of the gradient sampling algorithm is that it provides approximate "optimality certificates": if d^k is small for a small sampling diameter, one can be reasonably sure that a local minimizer has been approximated.

Another interesting feature of RGS is that, like bundle methods, it tends to detect nearby "ridges" and move parallel to them instead of towards them. Consider for example, the (convex) function $f(x_1, x_2) := \frac{1}{2}x_1^2 + 100|x_2|$ and the point $x^k := (1, 0.0001)$. The steepest descent at this point yields the unpromising search direction $-\nabla f(1, 0.0001) = (-1, -100)$. However, if the two points $(1, 0.0001)$ and $(1, -0.0001)$ are used to create a more robust local subgradient,

$$\tilde{\partial} f(1, 0.0001) = \operatorname{conv}\{\nabla f(1, 0.0001), \nabla f(1, -0.0001)\} = \operatorname{conv}\{(1, 100), (1, -100)\},$$

then the new descent direction is $d^k = -\text{Proj}(0, \tilde{\partial} f(1, 0.0001)) = (-1, 0)$, a vastly more promising search direction. In this sense, the "robust local subgradient" $\tilde{\partial} f(x^k)$ can be interpreted as the nonconvex analog of the ε-subdifferential $\partial_\varepsilon f(x^k)$, a fundamental concept for convex bundle methods.

A Matlab implementation of RGS is available at: `www.cs.nyu.edu/faculty/overton/papers/gradsamp/`.

3 Benchmark parameters and measures

All of the algorithms tested in this work have a variety of optional parameters which can be used to improve their performance. Accordingly,

> *in all of the tests performed, each algorithm's input parameters were adjusted towards best performance.*[1]

Completely fair data comparisons can only occur if all the solvers have a uniform stopping criterion. Since altering the actual solver source code is not practical (nor fair), it becomes difficult to meet such criterion. However, the codes for CProx, N1CV2, and N2FC1 all have the option of setting a maximum number of oracle calls. Since we are considering proximal point computations in the setting of oracle based optimization, considering efficiency in terms of the number of function evaluations is natural. Therefore, we set the maximum number of oracle calls each algorithm is allowed to make and compare the "best found point" for each algorithm.

> *In all tests performed, each algorithm was set to terminate after 100 oracle calls were detected.*

As mentioned, in the case of CProx, N1CV2, and N2FC1 setting the maximum number of oracle calls permitted is a simple matter. Since CProx makes exactly one oracle call per iteration, fixing the number of oracles is done by setting the maximum number of iterations to 100, while N1CV2 and N2FC1 have the option of setting a maximum number of oracle calls directly. In RGS, however, setting the maximum number of oracle calls permitted is not so simple. This is because RGS is equipped with various subroutines (such as random sampling and line searches) whose premature cessation may cause inaccurate test conclusions. Therefore, we allowed this algorithm to complete these subroutines before exiting and returning the final solution. In some cases this caused RGS to use more than 100 oracle calls.

A performance profile is essentially a condensed graphical form of statistical information, including solver robustness and efficiency. Our performance profiles use two ratios for measuring performance, depending on whether or not x^*, a *unique* solution to (1), is available.

If x^* is the known unique location of the proximal point and x^{best} is the best point found out of all the oracle calls made (i.e. x^{best} is such that, $f(x^{best}) = \min_{i=0,1,2,\ldots n} f(x^i) + R\frac{1}{2}|x^i - x^0|^2$), our first formula

$$-\log_{10}\left(\frac{|x^{best} - x^*|}{|x^0 - x^*|}\right) \qquad \text{(R.A.)}$$

measures the relative gain in accuracy in computing the actual proximal point. We report these results via the $\log_{10}$ scale, since the relative gain in accuracy can often be very large or very small. On this scale, a positive number (roughly) represents the number digits of accuracy obtained through the algorithm, a 0 means $x^{best} = x^0$, while a negative number means the algorithm actually moved away from the correct proximal point. It should be noted that movement away from the

[1]The authors would like to again acknowledge Michael Overton's assistance in setting the RGS parameters to optimal performance.

correct proximal point still requires $f(x^{best}) + R\frac{1}{2}|x^{best} - x^0| \leq f(x^0)$, so some improvement in the objective function must still have been achieved.

Performance profiles using the (R.A.) scale (Figure 1 in our benchmark) plot the (R.A.) value (on the x-axis) versus the ratio of tests which successfully achieved this value (on the y-axis). As such, the location where a profile first decreases from the y value 1 describes the gain in accuracy the algorithm achieved on every problem, while the location where a profile first obtains a y value of 0 yields the best gain in accuracy achieved using that algorithm. In general, algorithms whose profiles are "higher" have outperformed algorithms with "lower" profiles.

Formula (R.A.) provides information on quality of the solution returned by a solver, this is of particular interest for nonconvex problems. However, if x^* is unknown or not unique, this formula becomes unusable. In this case, we use a second formula, measuring the total decrease in the objective function:

$$f(x^0) - \left(f(x^{best}) + R\frac{1}{2}|x^{best} - x^0|^2\right). \tag{T.D.}$$

We no longer use a $\log_{10}$ scale, nor do we divide by some factor to create a "relative total decrease". The $\log_{10}$ scaling is avoided because in the second formula the total decrease in objective function value is often quite small, so a $\log_{10}$ scale would result in many negative numbers. This causes some conflict with our first formula where negative numbers are construed as bad. The reason we do not attempt to create a "relative total decrease" with the second formula is because the correct value to divide by would be $(f(x^0) - f(x^*) - R\frac{1}{2}|x^* - x^0|^2)$, but the (T.D.) scale is used in Section 5 where the actual proximal point is unknown and in Subsection 6.2 where the actual proximal point is very unlikely to be achieved.

Performance profiles using the (T.D.) scale plot, the (T.D.) value (on the x-axis) versus the ratio of tests which successfully achieved this value (on the y-axis). As such, the location where a profile first decreases from the y value 1 describes the minimal decrease the algorithm achieved on every problem, while the location where a profile first obtains a y value of 0 yields the maximum decrease achieved using that algorithm. As before, in general, algorithms whose profiles are "higher" have outperformed algorithms with "lower" profiles.

4 "Maximum of Quadratics" Functions

In this section we consider test functions defined as the point-wise maximum of a finite collection of quadratic functions:

$$f(x) := \max\left\{\langle x, A_i x\rangle + \langle B_i, x\rangle + C_i : i = 1, 2, ..., \texttt{nf}\right\}, \tag{4}$$

where A_i are $N \times N$ symmetrical matrices (no assumption on their positive definiteness is made), $B_i \in \mathbb{R}^N$, and $C_i \in \mathbb{R}$.

This class of functions has several practical advantages. First, many different examples are easily created by choosing values for N and `nf`, then randomly generating `nf` objects A_i, B_i and C_i. Second, oracles are easy to define, because for each given x any *active* index $j \leq$ `nf` (i.e., an index where the maximum in (4) is attained), yields a subgradient $A_j x + B_j \in \partial f(x)$. In particular, if there are $\texttt{nf}_{\texttt{act}}$ active indices at 0, the following relations hold (reordering indices if necessary):

$$f(0) = C_i = C \text{ for } i = 1, 2, ...\texttt{nf}_{\texttt{act}} \quad \text{and} \quad \partial f(0) = \text{conv}\{B_i : i = 1, \ldots, \texttt{nf}_{\texttt{act}}\}. \tag{5}$$

Furthermore, a function given by (4) is always prox-bounded with computable threshold:

$$r_{pb} = \max\{|A_i| : i = 1, 2, ...\mathtt{nf}\}.$$

Thus, a "large enough" prox-parameter R can be estimated a priori, by taking any value bigger than the norm of all matrices A_i. However, it should be noted that, the application of this prox-parameter results in the objective function for proximal point computation, $f + R\frac{1}{2}|\cdot - x^0|^2$, is actually a convex function. Therefore, one would expect tried and true algorithms for convex optimization to vastly outperform any nonconvex optimization algorithm.

To assess our method, in all our runs we fix the desired proximal point to be the zero vector, and choose starting points x^0 so that $P_R f(x^0) = 0$. This is done as follows:

0. Select a lower and upper bound for random number generation $L, U \in \mathbb{R}$, and a prox-parameter scaling $R_{sc} \geq 1$.
1. Fix a dimension N, the number of quadratic functions defining the maximum $\mathtt{nf}$, and the number of active subgradients $\mathtt{nf_{act}} \leq \mathtt{nf}$ at $P_R(x^0) = 0 \in \mathbb{R}^N$.
2. Set $C_1 = C_2 = \ldots = C_{\mathtt{nf_{act}}} = U$ and randomly generate $C_i \in (L, U)$ for $i = \mathtt{nf_{act}}+1, \ldots, \mathtt{nf}$.
3. For all $i \leq \mathtt{nf}$ randomly generate $B_i \in (L, U)^N$.
4. For all $i \leq \mathtt{nf}$ randomly generate symmetric matrices $A_i \in (L, U)^{N\times N}$ such that the minimum eigenvalue for each A_i is negative.
5. Set $R = R_{sc}(\max|A_i|) + 1$, take any $x^0 \in \frac{1}{R}\mathrm{conv}\{B_i : i = 1, \ldots, \mathtt{nf_{act}}\}$ as starting point.

The condition that the minimum eigenvalue for each A_i is negative in Step 4 ensures that the resulting function is nonconvex near 0. This step is easily accomplished by simply regenerating any A_i whose minimum eigenvalue is nonnegative (in practice, if $N > 1$, randomly generated symmetric matrices tend to result in at least one negative eigenvalue.) In Step 5 we assure that $P_R f(x^0) = 0$ as,

$$p = P_R(x^0) \iff R(x^0 - p) \in \partial f(p) \quad \text{so,} \quad p = 0 \iff Rx^0 \in \partial f(0).$$

For our purposes, in Step 0 we fix $L = -10$, $U = 10$ and $R_{sc} = 12$, then consider various combinations for N, $\mathtt{nf}$, and $\mathtt{nf_{act}}$.

For our benchmark, we use 8 different combinations of N, $\mathtt{nf}$, and $\mathtt{nf_{act}}$, then randomly generate 20 test functions for each combination. This provides a total of 160 test functions. We then ran each algorithm for a maximum of 100 oracle calls on each test function. Using formula (R.A.) we calculated the relative accuracy resulting for each test run. In Appendix A, Tables 1 and 2 report the worst accuracy, mean accuracy, and best accuracy for each test set and algorithm. Tables 1 and 2 also provide details on the combinations of N, $\mathtt{nf}$, and $\mathtt{nf_{act}}$ used, and the mean number of oracle calls required for each test set and algorithm. Figure 1 provides a performance profile for the tests.

In general it would appear that CProx outperformed the other three algorithms in all of these tests. It is worth noting however that in test sets with $\mathtt{nf_{act}} = 1$, N2FC1 generally used only a fifth of the possible oracle calls, and in the best case came up with nearly the same (R.A.) as N1CV2 and CProx. Algorithms CProx, and N1CV2 also performed better on test sets with one active gradient, but not to the same levels as N2FC1.

Conversely, RGS behaved quite poorly on tests with only one active subgradient, but better on tests with multiply active subgradients. As much of the inspiration for RGS is based on functions with sharp corners or ridges this is not surprising. In Sections 5 and 6 where the objective functions are less smooth we shall see RGS perform better.

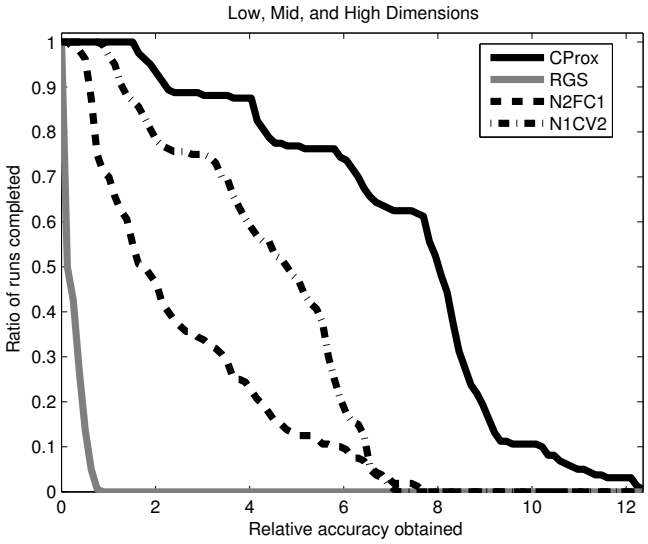

Figure 1: Performance Profile for maximum of quadratic functions family

5 "Piecewise Polynomials" family

In this section we consider test sets constructed by composition of polynomial with absolute value functions. More precisely, we define

$$f(x) = \sum_{i=1}^{N} p_i(|x_i|),$$

where N is the dimension of the problem, and for $i = 1, \ldots, N$, x_i is the i^{th}-coordinate of x, and p_i is a polynomial function.

As in Section 4, these function are easy to generate, and have a simple subgradient formula

$$\partial f(x) = \partial p_1(|x_1|) \times \partial p_2(|x_2|) \times ... \partial p_N(|x_N|).$$

Moreover, if we only use monic polynomials (polynomials where the coefficient of the highest power is equal to 1), we ensure that the function is bounded below and therefore prox-bounded. Finally, away from the origin, such functions can easily be rewritten as the maximum of polynomials, and therefore, are lower-$\mathcal{C}^2$.

However, considering the one dimensional example

$$f(x) = (|x| - 1)^2 - 1 = |x|^2 - 2|x|,$$

it is clear that functions of this form need not be regular at 0. Furthermore, whenever the degree of the polynomials used is greater than two, such functions are not semi-convex.

A disadvantage of such functions is the difficulty in computing the correct proximal point. Although for points away from the origin, with sufficiently large R, a closed form solution for the proximal point could be constructed, the solution would nontrivial. Moreover, due to the symmetry of the function, the proximal point at 0 is not unique. Indeed, suppose $p \in P_R f(0)$, so

$f(p)+R\frac{1}{2}|p-0|^2 = \min\{f(y)+R\frac{1}{2}|y-0|^2\}$. Since $f(p) = f(-p)$ and $R\frac{1}{2}|p-0|^2 = R\frac{1}{2}|-p-0|^2$ we must also have $-p \in P_R f(0)$. Due to these reasons, we chose to use the second of our comparison formulae, (T.D.).

We generate our test functions as follows:

0. Select a lower and upper bound for random number generation $L, U \in \mathbb{R}$.

1. Fix a dimension N, and the degree for the polynomials `Deg`.

2. For each $i = 1, 2, ...N$ generate `Deg` $- 1$ random numbers $a_1, a_2, ...a_{\texttt{Deg}-1} \in [L, U]$ and set

$$p_i(x) = x^{\texttt{Deg}} + \sum_{j=1}^{\texttt{Deg}-1} a_j x^j.$$

3. Set $f(x) = \sum_{i=1}^{N} p_i(|x_i|)$

For our purposes, in Step 0 we fix L and U to be -1 and 0. This causes a nonconvex movement in all polynomial powers of degree less than `Deg`. Notice that we have also set the constant coefficient for each polynomial, a_0, to be zero. Since the polynomials are summed, a positive constant term would have no effect, other than a vertical shifting of the function.

Having generated our test problems, we consider various prox-parameters and two different prox-centers for the calculations.

For our first prox-center, we select a location where the function should behave smoothly: $x_i^0 = 1$ for all $i = 1, \ldots, N$. Our second prox-center focuses on an area where the function is very poorly behaved: $x_i^0 = 0$ for all $i = 1, \ldots, N$. Since the generated test functions are not prox-regular (nor even regular) at $0 \in \mathbb{R}^N$, we anticipate a problem in the CProx code in this case. Specifically, CProx involves a subroutine which checks whether R is sufficiently large to ensure that the statement 3 holds as an if-and-only-if statement. Since the function is not prox-regular at our prox-center, we must expect this subroutine to fail. To examine the effect this subroutine has on convergence, we consider a second version of the algorithm denoted CPrFR, which is identical to CProx, except that it bypasses this subroutine (this is done by setting the parameter "P.CheckR" in the CProx code to off).

In Appendix A, in Tables 3 to 5 we compare the total decrease in objective function value for each algorithm, using the prox-center $x_i^0 = 1$. Tables 3 to 5 also provide details on the combinations of N, `Deg`, and R used, and the mean number of oracle calls required for each test set and algorithm. Likewise, Tables 6 to 8 provide the same data with the prox-center $x_i^0 = 0$ for $i = 1, \ldots, N$.

Figures 2 and 3 provide a performance profiles for the two different starting points.

Surprisingly, regardless of starting point, CProx and CPrFR caused nearly identical total decrease in objective function value. However, CProx did tend to terminate earlier. This suggests that the subroutine which determined if R is sufficiently large did not fail until CProx had already come quite close to a local minimum.

In general it would appear that when starting at $x_i^0 = 1$ CProx (and CPrFR) outperformed the remaining algorithms. Although N1CV2 and N2FC1 both performed very well when `Deg` was set to 3, when `Deg` ≥ 5 they failed to find any improvement in objective function value. N1CV2 failures are reported in the curve-search, a natural outcome, since N1CV2 is designed for convex functions only. N2FC1 fails mostly in the quadratic program solution, perhaps tighter box

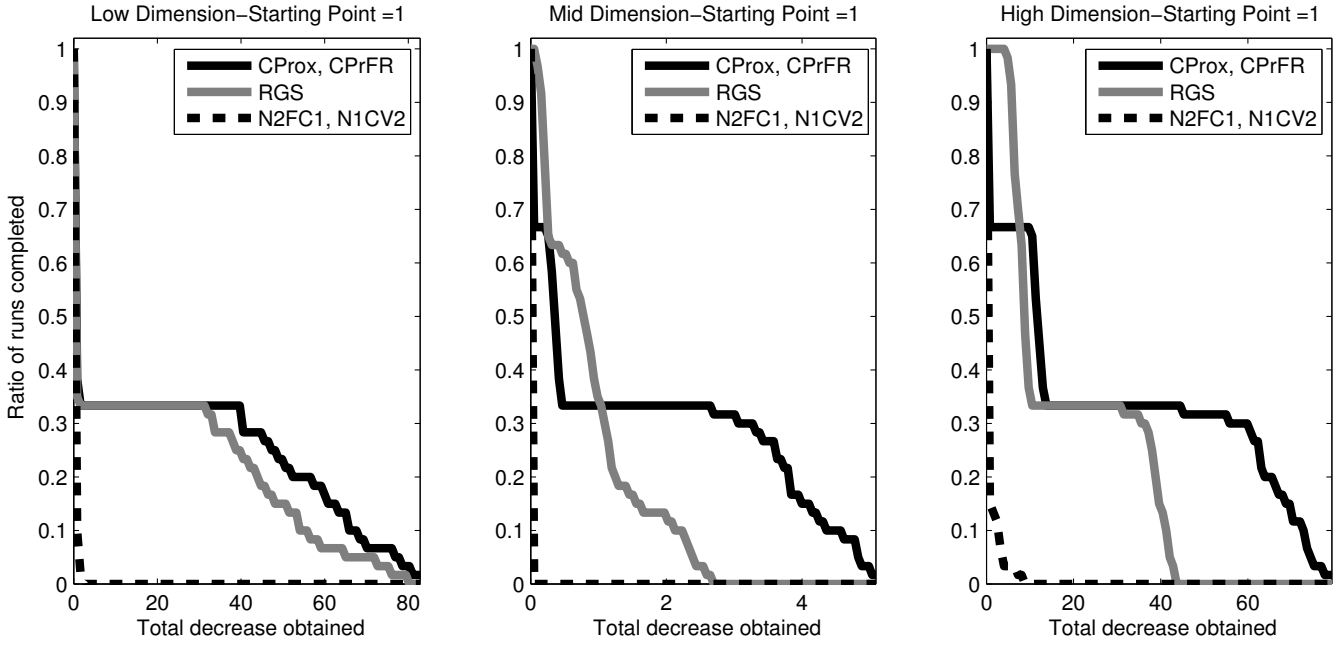

Figure 2: Performance Profiles for piecewise polynomials family ($x_i^0 = 1$)

constraints would improve N2FC1performance in these tests:. RGS performed quite well on all tests, coming a close second to CProx.

Shifting the prox-center to the nonregular point $0 \in \mathbb{R}^N$ seemed to have little effect on CProx, CPrFR, and RGS. Again we see these algorithms performing well, with RGS only slightly outperformed by CProx. However, the nonregular prox-center seemed to have a very positive effect on both N1CV2 and N2FC1. In both cases, the new prox-center allowed the algorithm to find a better value for the objective function, and in the case of N1CV2 this better objective value was considerably better than those found by the other algorithms.

6 Two Specific Test Functions

In this section we consider the following functions:

1. **The Spike:**
$$f := \sqrt{|x|},$$

2. **The Waterfalls**: for $x \in \mathbb{R}^N$ set
$$f_i(x) := \begin{cases} x_i^2 & : \quad x <= 0 \\ -2^{-N+1}(x_i - 2^N)^2 + 2^k & : \quad x \in [2^{N-1}, 2^N], \end{cases}$$
then
$$f(x) = \sum_i f_i(x).$$

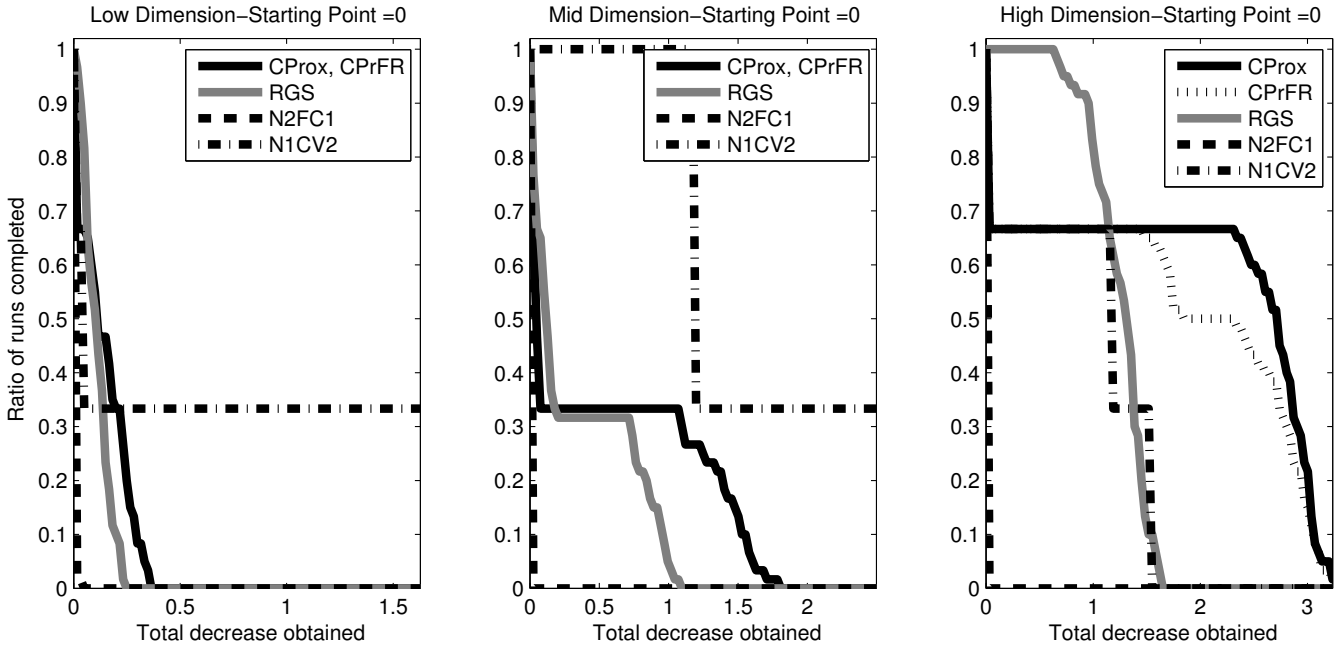

Figure 3: Performance Profiles for piecewise polynomials family ($x_i^0 = 0$)

6.1 The Spike

The function $f = \sqrt{|\cdot|}$ is the classical example of a function which is not prox-regular at 0. As such, f is not semi-convex, nor lower-$\mathcal{C}^2$ near 0. Therefore, one cannot rely on the classical subgradient formula from equation (3). Conversely, the function is bounded below, and therefore prox-bounded with threshold equal to 0. Moreover, as we shall see below, numerically calculating an exact proximal point is achievable for this problem.

Before we discuss the prox-parameters and prox-centers we test, it is worth nothing that, due to the radial symmetry this function exhibits, for CProx, N1CV2, and N2FC1 we need to consider this function only in one dimension. Indeed, consider a prox-center $x^0 \neq 0$, prox-parameter $R > 0$ and any point x^m on the line connecting $0 \in \mathbb{R}^N$ to x^0; i.e., $x^m = \tau x^0$ for some $\tau \in \mathbb{R} \setminus \{0\}$. The subdifferential of $f + R\frac{1}{2}|\cdot - x^0|^2$ is given by

$$\partial(\sqrt{|\cdot|} + R\frac{1}{2}|\cdot - x^0|^2)(x^m) = \frac{1}{2}|x^m|^{-3/2}x^m + R(x^m - x^0).$$

Since $x^m \neq 0$, we see that

$$\nabla f_R(x^m) = \left(\frac{1}{2}\tau^{-3/2}|x^0|^{-3/2} + R(\tau - 1)\right)x^0.$$

In other words, all the cutting planes created when calling the oracle will act only along the direction created by x^0. As such, all iterates created via CProx, N1CV2, and N2FC1 will lie on the line connecting x^0 and 0.

This analysis fails to hold for RGS, as the random sampling involved can move iterates away from the line segment. We therefore test CProx, N1CV2, and N2FC1 in one dimension only,

but test RGS in dimensions 1, 10, and 100. Furthermore, to examine the randomness invoked by RGS, for each test run of RGS we run the algorithm 20 times. Despite its random nature, RGS was remarkably consistent in its results (the standard deviations for the 20 runs for each test was always less than 10^{-4}). In the results below we always select the best resulting run for RGS.

For our tests we set the prox-center to be $x^0 = 1$ (in higher dimensions we take $x_1^0 = 1$, and $x_i^0 = 0$ for $i = 2, \ldots, N$), and consider various prox-parameters: $R = 0.1$, $R = 0.25$, $R = 1$, and $R = 2$. Even though the classical subgradient formula from equation (3) does not hold with equality for this function, we can nonetheless use it to derive the critical points for the function:

$$p = P_R f(x^0) \Rightarrow 0 \in \partial \left(\sqrt{|\cdot|} + R\frac{1}{2}|\cdot - x^0|^2 \right)(p),$$

$$p = P_R f(x^0) \Rightarrow p = 0 \text{ or } \frac{1}{2}|p|^{-3/2}p + R(p - x^0) = 0. \tag{6}$$

This equation can be solved symbolically by any number of mathematically solvers. Using Maple we found:

- for $R = 0.1, R = 0.25$, and $R = 1$ equation (6) yields two imaginary roots and the correct proximal point $p = 0$,
- for $R = 2$ equation (6) yields $p \in \{0, 0.0726811601, 0.7015158585\}$, the correct proximal point is $p = 0.7015158585$.

As for Section 4, we report the accuracy as obtained via equation (R.A.). Similarly to Section 5, since f is not semi-convex, we also include results of the algorithm CPrFR. Our results are listed in Appendix A, Table 9.

Our first result of note is the appearance of "∞" in the one dimensional RGS tests. This means that RGS exactly identified the correct proximal point of the function. Although this looks impressive, it is likely due to RGSs line search style, and the fact the prox-center is exactly 1 unit away from the proximal point. If other design factors were involved the "∞" would likely reappear in the higher dimension tests.

Our next observation is that the results of CProx, CPrFR, N1CV2, and N2FC1 appear to rely heavily on the prox-parameter. Conversely, RGS resulted in a consistent performance regardless of prox-parameter. This suggests that, although in general RGS may not be the most successful algorithm, it is the most robust. Since robustness was a major concern in the design of RGS, this is a very positive result for its authors.

In comparing CProx and CPrFR we note that the removal of the subroutine for determining if R is sufficiently large to guarantee convergence does not appear to have any positive effect on the algorithm. Indeed in the one case where CPrFR outperformed CProx ($R = 2$), CPrFR used 100 times more oracles calls than CProx, but only only improve (R.A.) by a factor of 7.22.

6.2 The Waterfalls

The waterfalls function is also poorly behaved in terms of proximal envelopes. Although the function is prox-bounded with threshold equal to 0, it is not semi-convex, nor lower-$\mathcal{C}^2$ near 0, nor prox-regular at $0 \in \mathbb{R}^N$. Therefore, one cannot rely on the classical subgradient formula from equation (3).

For example, consider a point $\bar{x} \in \mathbb{R}^N$ such that $\bar{x}_i = 2^{n_i}$ for each $i = 1, 2, ...N$. The subgradient of waterfalls function is easily seen to be

$$\partial f(x) = \partial f_1(x) \times \partial f_2(x) \times ...\partial f_N(x),$$

where

$$\partial f_i(x) = \begin{cases} [0,2] & : \quad x_i = 2^n \text{ for some } n \\ -2^{n+2}(x_i - 2^n) & : \quad x_i \in [2^{n-1}, 2^n]. \end{cases}$$

So at $\bar{x}$ we see $\partial f(\bar{x}) = [0,2] \times [0,2] \times ...[0,2]$. Hence if $\bar{x}$ is the prox-center, then for any $R > 0$ one has

$$p = \bar{x} \text{ satisfies } 0 \in \partial f(p) + R(p - \bar{x}).$$

However, unless R is large, $\bar{x} \neq P_R f(\bar{x})$.

Moveover, consider the point $\bar{x}/2^m$. Then

$$\partial f(\bar{x}/2^m) + R(\bar{x}/2^m - \bar{x}) = ([0,2] \times [0,2] \times ...[0,2]) - (1 - 2^{-m})R\bar{x}.$$

If $R\bar{x}_i < 2$ for each $i = 1, 2, ...N$, then this will result in $0 \in \partial f(\bar{x}/2^m) + R(\bar{x}/2^m - \bar{x})$ for all m sufficiently large. That is, equation (3) will be satisfied at an infinite series of points. In this case the correct proximal point is 0.

For our tests, we initially considered the prox-center $x_i^0 = 1$ for $i = 1, \ldots, N$, several values of R and several dimensions. However, in this case all algorithms tested stalled immediately. This is probably because all the algorithms tested contain some stopping criterion based on (a convex combination of) subgradients approaching zero.

We therefore considered what would happen if instead we randomly selected a prox-center nearby $x_i^0 = 1$, not exactly equal to this point. For each test, we randomly generated 20 prox-centers of the form

$$x_i^0 = \texttt{rand}_i \quad \text{for } i = 1, \ldots, N,$$

where $\texttt{rand}_i$ is a random number with values in $[1 - 0.0001, 1 + 0.0001]$. We then ran each algorithm starting from this prox-center, and compared the improvement in objective function decrease via formula (T.D.). Our results appear in Appendix A, Tables 10 to 12, and in the performance profile in Figure 4.

As in Subsection 6.1, drawing conclusions from these test results is tricky. The clearest conclusion is that, regardless of dimension, and as expected, N1CV2 performed very poorly on this problem. Indeed, in all tests N1CV2 terminated after one oracle call, and returning an insignificant (T.D.), reporting a failure (in the curve-search, or in the quadratic program solution).

In one dimension, N2FC1 appears to give the greatest (T.D.) followed by RGS and CProx. CProx again has the distinct advantage of using less oracle calls, while RGS and N2FC1 use approximately the same number of oracle calls.

In ten dimensions, we see a fairly even (T.D.) in CProx, CPrFR, N2FC1, and RGS. However, again we see CProx uses less oracle calls than the remaining solvers.

In 100 dimensions, we see CProx and CPrFR giving considerably better (T.D.) than N2FC1 and RGS, with CProx continuing to use less oracle calls than than the other algorithms. Nonetheless, we see RGS continues to perform adequately, reinforcing the robustness property of this algorithm.

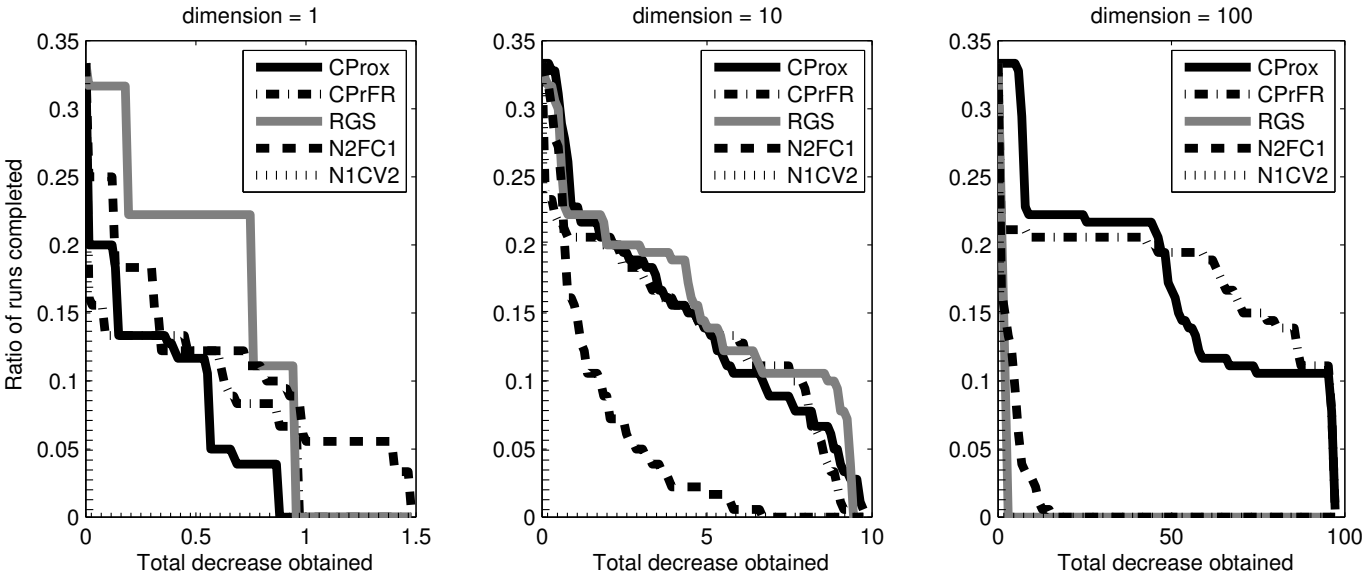

Figure 4: Performance Profiles for waterfalls family

7 Conclusions

In this paper we have presented several new benchmarking algorithmic techniques for computing proximal points of nonconvex functions. The choice of test problems for benchmarks is difficult and inherently subjective. Therefore, in order to reduce the risk of bias in benchmark results, it is sometimes helpful to perform benchmarks on various sets of models and observing solver performance trends over the whole rather than relying on a single benchmark test set. For this reason, in Sections 4 and 5 we give two methods to randomly generate large collections of challenging test functions for this problem, while in Section 6 we outlined two very specific test functions which should be of interest to nonconvex optimization benchmarking.

Using these problems, this paper compares four algorithms: CProx, N1CV2, N2FC1, and RGS. We further examined two versions of CProx, to determine (numerically) if a small modification of the code would improve performance.

One can draw cautiously the following conclusions concerning the codes. Overall we saw CProx in general outperformed the remaining algorithms *on this collection of test problems*. N1CV2 has proven its value for the "more convex" instances, i.e., the maximum of quadratic functions family. Finally, we saw RGS also perform very well, in the high dimension and nonconvex cases specially, reinforcing the robustness of that particular algorithm.

References

[BDF97] S. C. Billups, S. P. Dirkse, and M. C. Ferris. A comparison of large scale mixed complementarity problem solvers. *Comp. Optim. Applic.*, 7:3–25, 1997.

[BGLS03] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, and C. Sagastizábal. *Numerical Optimization. Theoretical and Practical Aspects*. Universitext. Springer-Verlag, Berlin, 2003. xiv+423 pp.

[BLO02] J. V. Burke, A. S. Lewis, and M. L. Overton. Approximating subdifferentials by random sampling of gradients. *Math. Oper. Res.*, 27(3):567–584, 2002.

[DM02] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2, Ser. A):201–213, 2002.

[HP06] W. L. Hare and R. A. Poliquin. Prox-regularity and stability of the proximal mapping. *submitted to: Journal of Convex Analysis*, pages 1–18, 2006.

[HS05] W. L. Hare and C. Sagastizábal. Computing proximal points of nonconvex functions. *submitted to: Math. Prog.*, 2005.

[HU85] J.-B. Hiriart-Urruty. Miscellanies on nonsmooth analysis and optimization. In *Nondifferentiable optimization: motivations and applications (Sopron, 1984)*, volume 255 of *Lecture Notes in Econom. and Math. Systems*, pages 8–24. Springer, Berlin, 1985.

[Kiw90] K.C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Math. Programming*, 46:105–122, 1990.

[LS97] C. Lemaréchal and C. Sagastizábal. Variable metric bundle methods: from conceptual to implementable forms. *Math. Programming*, 76:393–410, 1997.

[LSB81] C. Lemaréchal, J.-J. Strodiot, and A. Bihain. On a bundle method for nonsmooth optimization. In O.L. Mangasarian, R.R. Meyer, and S.M. Robinson, editors, *Nonlinear Programming 4*, pages 245–282. Academic Press, 1981.

[LV98] L. Lukšan and J. Vlček. A bundle-Newton method for nonsmooth unconstrained minimization. *Math. Programming*, 83(3, Ser. A):373–391, 1998.

[LV00] L. Lukšan and J. Vlček. NDA: Algorithms for nondifferentiable optimization. Research Report V-797, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 2000.

[Mif77] R. Mifflin. Semi-smooth and semi-convex functions in constrained optimization. *SIAM Journal on Control and Optimization*, 15:959–972, 1977.

[Mif82] R. Mifflin. A modification and extension of Lemarechal's algorithm for nonsmooth minimization. *Math. Programming Stud.*, 17:77–90, 1982.

[Mor65] J-J. Moreau. Proximité et dualité dans un espace hilbertien. *Bull. Soc. Math. France*, 93:273–299, 1965.

[RW98] R. T. Rockafellar and R. J-B. Wets. *Variational Analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998.

A Appendix: Tables.

N, nf, $\texttt{nf}_{\texttt{act}}$	Alg.	Worst (R.A.)	Mean (R.A.)	Best (R.A.)	Mean Oracle Calls
5, 5, 1	CProx	7.8	8.4	9.3	100
	N1CV2	5.1	5.9	7	100
	N2FC1	0.14	3.6	7.6	20
	RGS	0.0004	0.16	0.37	100
10, 5, 5	CProx	5.8	6.4	7	100
	N1CV2	3.2	3.6	4.2	100
	N2FC1	1.1	3.7	5.5	86
	RGS	-0.019	0.14	0.34	110
20, 30, 1	CProx	7.6	8.3	9.2	100
	N1CV2	4.0	5.5	7.0	100
	N2FC1	0.25	2.6	7	18
	RGS	-0.4	-0.14	0.045	110
20, 30, 30	CProx	8.9	11	12	100
	N1CV2	3.1	4.7	6.5	100
	N2FC1	0.88	1.6	2.6	100
	RGS	0.27	0.44	0.67	130

Table 1: Low Dimension Test Results for Maximum of Quadratics

N, nf, $\texttt{nf}_{\texttt{act}}$	Alg.	Worst (R.A.)	Mean (R.A.)	Best (R.A.)	Mean Oracle Calls
50, 30, 1	CPROX	7.7	8.4	9.2	95
	N1CV2	3.5	5.8	7.2	100
	N2FC1	1.1	3.7	7.6	23
	RGS	-0.48	-0.29	-0.084	100
50, 60, 30	CPROX	1.5	2	3.6	100
	N1CV2	0.82	1.2	1.4	100
	N2FC1	0.51	0.66	0.8	100
	RGS	0.35	0.55	0.77	100
100, 30, 1	CPROX	7.7	8.4	9.1	100
	N1CV2	4.4	5.9	6.9	100
	N2FC1	1.0	3.4	7.5	28
	RGS	-0.44	-0.3	-0.13	200
100, 30, 30	CPROX	4	4.5	5.9	100
	N1CV2	1.4	1.9	2.7	100
	N2FC1	0.47	0.64	0.83	100
	RGS	0.22	0.38	0.6	200

Table 2: Mid and High Dimension Test Results for Maximum of Quadratics

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
5, 3, 10	CPROX	0.302	0.616	0.936	85.2
	CPRFR	0.302	0.616	0.936	92.9
	N1CV2	0	0.217	2.33	55.9
	N2FC1	0	0.236	2.32	65.4
	RGS	0.0159	0.296	0.45	65.6
10, 5, 10	CPROX	0.368	0.755	1.68	95.8
	CPRFR	0.368	0.755	1.68	100
	N1CV2	0	0	0	82.5
	N2FC1	0	0	0	48.3
	RGS	0.108	0.484	1.15	89.8
20, 9, 20	CPROX	39.8	59.5	83.7	100
	CPRFR	39.8	59.5	83.7	100
	N1CV2	0	0	0	90.9
	N2FC1	0	0	0	92.1
	RGS	31.9	50.2	79.5	124

Table 3: Low Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
50, 5, 10	CPRox	2.7	4.03	5.14	95.9
	CPrFR	2.7	4.03	5.14	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	45
	RGS	0.184	1.49	2.69	153
50, 5, 50	CPRox	1.01	1.49	1.87	26.9
	CPrFR	0.882	1.36	1.87	100
	N1CV2	0	0	0	91.9
	N2FC1	0	0	0	86.6
	RGS	0.428	0.913	1.25	156
50, 5, 250	CPRox	0.243	0.357	0.445	67.1
	CPrFR	0.243	0.357	0.445	100
	N1CV2	0	0	0	92.5
	N2FC1	0	0	0	58.8
	RGS	0.069	0.189	0.272	158

Table 4: Medium Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
100, 3, 10	CPRox	10.3	11.8	13.6	41.2
	CPrFR	6.95	11.2	13.6	96.4
	N1CV2	0	0.197	8.23	100
	N2FC1	0	0.197	8.23	43.4
	RGS	7.67	8.75	9.94	103
100, 5, 10	CPRox	6.64	8.48	10.5	83.3
	CPrFR	6.64	8.48	10.5	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	44
	RGS	4.45	6.25	8.04	103
100, 7, 10	CPRox	44.9	66.9	80	100
	CPrFR	44.9	66.9	80	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	42.1
	RGS	30.8	39.4	43.4	103

Table 5: High Dimension Test Results for Piecewise Polynomials: $x_i^0 = 1$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
5, 3, 10	CProx	0.0578	0.14	0.288	90.1
	CPrFR	0.0578	0.142	0.288	100
	N1CV2	0	0.0043	0.0462	97.5
	N2FC1	0	0.0071	0.0442	63.4
	RGS	0.0192	0.0756	0.175	67.4
10, 5, 10	CProx	0.133	0.278	0.426	65.4
	CPrFR	0.15	0.289	0.495	100
	N1CV2	0	0.0040	0.0322	94.1
	N2FC1	0	0.0039	0.0204	55.6
	RGS	0.0513	0.159	0.234	76.0
20, 9, 20	CProx	0.156	0.257	0.359	100
	CPrFR	0.156	0.257	0.359	100
	N1CV2	0	0.0003	0.0015	92.5
	N2FC1	0	0.0005	0.0058	40.0
	RGS	$8.71\ 10^{-5}$	0.0866	0.154	99.7

Table 6: Low Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
50, 5, 10	CProx	1.07	1.41	1.79	44.8
	CPrFR	0.759	1.41	1.79	100
	N1CV2	0	0.0005	0.0032	100.0
	N2FC1	0	0.0004	0.0021	82.9
	RGS	0.096	0.843	1.08	155
50, 5, 50	CProx	0.205	0.249	0.313	100
	CPrFR	0.205	0.249	0.313	100
	N1CV2	0	4.3610^{-5}	0.0007	100.0
	N2FC1	0	7.9510^{-5}	0.0012	18.8
	RGS	0.073	0.119	0.18	157
50, 5, 250	CProx	0.0399	0.0487	0.0612	100
	CPrFR	0.0399	0.0487	0.0612	100
	N1CV2	0	9.1610^{-6}	0.0001	98.2
	N2FC1	0	0	0	41.9
	RGS	$6.46\ 10^{-5}$	0.0134	0.0278	159

Table 7: Mid Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

N, Deg, R	Alg.	Worst (T.D.)	Mean (T.D.)	Best (T.D.)	Mean Oracle Calls
100, 3, 10	CProx	2.34	2.79	3.22	19.9
	CPrFR	1.57	2.54	3.22	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	45
	RGS	0.63	1.11	1.55	103
100, 5, 10	CProx	2.54	2.91	3.25	20.6
	CPrFR	1.47	2.73	3.25	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	45
	RGS	0.936	1.29	1.63	103
100, 7, 10	CProx	2.58	2.91	3.27	19.2
	CPrFR	1.47	2.56	3.27	100
	N1CV2	0	0	0	100
	N2FC1	0	0	0	45
	RGS	0.97	1.33	1.64	103

Table 8: High Dimension Test Results for Piecewise Polynomials: $x_i^0 = 0$

R, N	Alg.	(R.A.)	Oracle Calls	R, N	Alg.	(R.A.)	Oracle Calls
0.1, 1	CProx	0.176	2	1, 1	CProx	0.301	1
	CPrFR	-0.106	100		CPrFR	0.15	100
	N1CV2	0	2		N1CV2	0.176	10
	N2FC1	0	2		N2FC1	1.08	5
	RGS	∞	5		RGS	∞	5
0.1, 10	RGS	0.154	107	1, 10	RGS	0.154	106
0.1, 100	RGS	0.154	103	1, 100	RGS	0.154	103
0.25, 1	CProx	15.477	2	2, 1	CProx	1.31	1
	CPrFR	15.477	3		CPrFR	9.47	100
	N1CV2	0.0669	7		N1CV2	0.993	10
	N2FC1	1.27	12		N2FC1	0.186	100
	RGS	∞	5		RGS	0.696	57
0.25, 10	RGS	0.154	109	2, 10	RGS	0.696	75
0.25, 100	RGS	0.154	103	2, 100	RGS	0.696	104

Table 9: Test Results for the Spike

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 1	CProx	$1.35\ 10^{-6}$	0.333	0.555	4
	CPrFR	$2.36\ 10^{-6}$	0.582	0.971	100
	N1CV2	0	$5.3\ 10^{-4}$	0.0019	1
	N2FC1	$3.69\ 10^{-5}$	0.617	1.48	50.5
	RGS	0.95	0.95	0.951	63.3
0.5, 1	CProx	$6.16\ 10^{-8}$	0.433	0.876	2.5
	CPrFR	$1.04\ 10^{-7}$	0.397	0.876	95.3
	N1CV2	0	$5.3\ 10^{-4}$	0.0019	1
	N2FC1	$3.69\ 10^{-5}$	0.503	1.4	37.4
	RGS	0.75	0.75	0.751	64.8
2.5, 1	CProx	$4.02\ 10^{-9}$	0.082	0.141	1.6
	CPrFR	$6.00\ 10^{-9}$	0.0231	0.0882	100
	N1CV2	0	$5.3\ 10^{-4}$	0.0019	1
	N2FC1	0.129	0.394	0.978	63.5
	RGS	0	0.16	0.188	60.5

Table 10: One Dimensional Test Results for the Waterfalls

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 10	CProx	3.41	8.17	9.74	21.7
	CPrFR	0	6.43	9.32	100
	N1CV2	$7.56\ 10^{-4}$	0.00234	0.00414	1
	N2FC1	0.183	1.91	5.79	83.2
	RGS	6.61	9.1	9.43	112
0.5, 10	CProx	1.14	3.91	6.6	4.9
	CPrFR	0	3.4	8.51	100
	N1CV2	$7.56\ 10^{-4}$	0.00234	0.00414	1
	N2FC1	0.113	1.58	6.66	90.1
	RGS	1.83	4.24	6.51	119
2.5, 10	CProx	0.273	0.665	1.11	2
	CPrFR	0.0411	0.167	0.781	100
	N1CV2	$7.56\ 10^{-4}$	0.00234	0.00414	1
	N2FC1	0.0897	1.01	4.98	87.7
	RGS	0.0934	0.489	0.703	109

Table 11: Dimension Ten Test Results for the Waterfalls

R, dim	Alg.	Minimal (T.D.)	Mean (T.D.)	Maximal (T.D.)	Mean Oracle Calls
0.1, 100	CProx	74.2	95.4	97.2	95.9
	CPrFR	95.3	96.4	97.2	100
	N1CV2	0.0194	0.025	0.0325	1
	N2FC1	0.119	5.43	15.1	92.6
	RGS	1.97	1.97	1.97	103
0.5, 100	CProx	25.3	50.4	66	7.25
	CPrFR	0	59.7	87.1	100
	N1CV2	0.0194	0.025	0.0325	1
	N2FC1	0.166	3.43	11.3	89.6
	RGS	1.77	1.77	1.78	103
2.5, 100	CProx	5.82	6.95	8.16	2
	CPrFR	0.592	0.71	0.84	100
	N1CV2	0.0194	0.025	0.0325	1
	N2FC1	0.129	0.56	1.16	81.6
	RGS	0.766	0.769	0.775	103

Table 12: Dimension One Hundred Test Results for the Waterfalls