Research report manuscript No.
(will be inserted by the editor)

Regularized optimization methods for convex MINLP problems

Welington de Oliveira

October 24, 2014

Abstract We propose regularized cutting-plane methods for solving mixed-integer nonlinear programming problems with nonsmooth convex objective and constraint functions. In order to avoid tailing-off effect that makes calculations unstable as the iteration process progresses, the given algorithms require solving a regularized mixed-integer linear programming subproblem at each iteration. The goal is to perform as few function evaluations as possible in the quest of solving convex mixed-integer nonlinear optimization problems. This is an important feature for some industrial applications resulting in hard-to-evaluate objective and/or constraint functions. Numerical experiments comparing the proposed algorithms with classical methods in this area show the effectiveness of our approach.

Keywords Mixed-integer programming · Cutting-plane method · Nonsmooth optimization

1 Introduction

Many real-life optimization problems are modeled in a mixed-integer setting, involving discrete and continuous decision variables. Optimization algorithms for solving *mixed-integer nonlinear programming* – MINLP – problems have become an important focus of research over the last years, [2,4,15,16]. As discussed in [11], some of these algorithms require differentiability of the involved functions. Following the lead of [29,11], in this work we do not assume differentiability to deal with problems of the form:

$$f_{\min} := \min_{x \in X} f(x) \;\; \text{s.t.} \;\; c(x) \leq 0\,, \tag{1}$$

where f, $c : \mathbb{R}^n \to \mathbb{R}$ are convex functions and $X \subset \mathbb{R}^n$ is a mixed-integer set, which is assumed to be nonempty and bounded. Since c can be a nonsmooth function, there is no loss of generality in assuming that $c : \mathbb{R}^n \to \mathbb{R}$. Indeed, function c can be the maximum of finitely many constraints $f_j(x) \leq 0 \in \mathbb{R}$, i.e., $c(x) = \max_j f_j(x)$.

Many methods have been proposed for solving convex MINLP problems, [15,4]. The *outer-approximation* method presented in [10,15] and [12] require differentiability of both functions f and c in order to efficiently solve the problem; see [11, § 4.1]. A technique that is closely related to the outer-approximation method is the *generalized Benders decomposition* proposed in [14]. Publications [26] and [20] propose *branch-and-bound* algorithms for solving problem (1), and [13, 7] explore valid inequalities in these techniques. In [22] a branch-price-and-cut strategy combined with decomposition techniques is proposed to provide valid inequalities and strong bounds to

Welington de Oliveira
IMPA - Instituto Nacional de Matemática Pura e Aplicada. Rua Dona Castorina 110, 22460-320, Rio de Janeiro, Brazil
BCAM - Basque Center for Applied Mathematics. Alameda de Mazarredo, 14, 48009 Bilbao, Basque Country - Spain
E-mail: wlo@impa.br

guide the search in a branch-and-bound tree. Other alternatives for solving (1) are based in Lagrangian relaxation; see for instance [24,18,3] for column generation approaches and [25,21] for other techniques applied to stochastic optimization. Interested readers are referred to [2,16] for more information on the aforementioned techniques for MINLP problems.

Another important method for solving nonsmooth convex MINLP problems is the *extended cutting-plane method* – ECPM – proposed in [29] and further studied in [30,11,28]. The ECPM is based on the Kelly's classical cutting-plane algorithm given in [17] and thus, convexity of the involved functions is exploited in order to approximate f and c by supporting hyperplanes. More precisely, the method generates a sequence of points $\{x^k\} \in X$ by solving a sequence of MILP subproblems, approximating the considered MINLP. It is worth mentioning that intermediate subproblems need not be solved exactly. This is an interesting feature for dealing with MINLP problems whose MILP subproblems is too time consuming and functions f and c in (1) can be easily evaluated by a *black-box* or *oracle*.

In general, ECPM algorithms require more iterations to solve problem (1) than outer-approximation based algorithms. However, the cost-per-iteration of the former method is lower because no additional nonlinear subproblem is required to be solved, in contrast to outer-approximation methods. One additional difficulty arises when in problem (1) at least one of the functions f or c is difficult to evaluate, a common feature in industrial optimization problems; see [24,27] for some examples. Under this assumption, ECPM seems to be so far the method of choice: outer-approximation techniques are not suitable, since solving the nonlinear optimization subproblems coming from relaxations of (1) can be too time consuming. Moreover, branch-and-bound approaches might not be suitable either: these techniques usually require too many function evaluations to solve the problem. Therefore, it is of interest obtaining algorithms for solving MINLP problems that require fewer function evaluations than ECPM but with a comparable cost-per-iteration. In order to seek for such algorithms, we then propose in this work to regularize the extended cutting-plane method of [29]. The proposed approaches are natural extensions of the level bundle methods given in [19,27] for continuous convex minimization problems. For this reason, we name our proposals *extended level bundle methods* – ELBM. We mention that the ELBM approaches are for the level bundle methods in [19] what the ECPM of [29] is for the Kelley's cutting-plane method [17]. In this manner, this article combines the two different areas: nonlinear mixed-integer optimization and nonsmooth optimization. As mentioned in [11], combination of these two optimization areas is rare, although, both bundle methods for nonsmooth optimization and cutting-plane methods for MINLP have their origin in the same classical cutting-plane method of Kelley [17].

This work is organized as follows: in Section 2 we give an algorithm pattern that will be employed for all presented ELBM variants. Its convergence analysis is also presented in Section 2, without specifying precisely the level bundle type. Some variants for the method are proposed in Section 3, aiming at solving the nonsmooth optimization problem (1) by requiring fewer function evaluations than an ECPM based algorithm. In Section 4 we provide numerical experiments comparing some of the proposed level bundle variants with an implementation of the ECPM and solver BONMIN [5]. Finally, Section 5 contains some concluding comments and remarks.

2 General considerations and algorithm

Given a point $x^k \in X$, we assume the existence of an oracle that provides us with:

$$\begin{bmatrix} f\text{-oracle information: } f(x^k) \text{ and } g_{\mathrm{f}}^k \in \partial f(x^k) \\ c\text{-oracle information: } c(x^k) \text{ and } g_{\mathrm{c}}^k \in \partial c(x^k)\,. \end{bmatrix} \tag{2}$$

It is well known that if both f and c are convex and finite-valued functions on an open set containing X, then their subdifferentials are nonempty for all $x \in X$. We emphasize that the requirement in (2) is a much milder assumption than demanding knowledge of the whole subdifferentials $\partial f(x^k)$ and $\partial c(x^k)$, as required by the outer-approximation algorithm of [11].

With the oracle information in (2) it is useful to define, at iteration k, two polyhedral *cutting-plane* models:

$$\check{f}_k(x) := \max_{j\in\mathcal{B}_k}\{f(x^j) + \langle g_{\mathrm{f}}^j, x - x^j\rangle\} \;\text{ and }\; \check{c}_k(x) := \max_{j\in\mathcal{B}_k}\{c(x^j) + \langle g_{\mathrm{c}}^j, x - x^j\rangle\}\,. \tag{3}$$

The index set $\mathcal{B}_k \subseteq \{1,\ldots,k\}$ corresponds to the method's memory: $\{x^j,\, (f(x^j), g_{\mathrm{f}}^j),\, (c(x^j), g_{\mathrm{c}}^j)\}_{j\in\mathcal{B}_k}$. Throughout this work we will refer to $\mathcal{B}_k$ as *information bundle*. Convexity of f and c ensures that the above models are under approximations of the respective functions, i.e.,

$$\check{f}_k(x) \leqslant f(x) \;\text{ and }\; \check{c}_k(x) \leqslant c(x) \;\text{ for all } k \text{ and all } x \in \mathbb{R}^n\,.$$

The above inequalities are crucial for cutting-plane based methods. For instance, the extended cutting-plane method of [29] deals with problem (1) by solving at each iteration k a MILP problem of the form

$$x^{k+1} \in \arg\min_{x\in X}\; \check{f}_k(x) \;\text{ s.t. }\; \check{c}_k(x) \leq 0\,, \tag{4}$$

which provides $\check{f}_k(x^{k+1})$ as a lower bound to the optimal value $f_{\min}$ of (1). Therefore, the measure $\max\{f(x^{k+1}) - \check{f}_k(x^{k+1}),\, c(x^{k+1})\}$ gives a certificate of optimality for the ECPM algorithm, which can be stopped with a δ_{Tol}-solution x^{k+1} if $\max\{f(x^{k+1}) - \check{f}_k(x^{k+1}),\, c(x^{k+1})\} \leqslant \delta_{\mathrm{Tol}}$, for some $\delta_{\mathrm{Tol}} > 0$.

In this work we are interested in regularized variants of (4) aiming at reducing the number of oracle calls required to solve (1). When regularization terms come into play, the value $\check{f}_k(x^{k+1})$ may no longer be a valid lower bound for $f_{\min}$; see (6) and (7) below. Therefore, other alternatives for the above certificate of optimality must be employed instead: given a lower bound f_{low}^k for $f_{\min}$ at iteration k, we proceed as [27] to define the following certificate of optimality

$$\mathtt{O}_k := \min_{j\leq k}\left[\max\{f(x^j) - f_{\mathrm{low}}^k,\, c(x^j)\}\right], \;\text{ with } x_{\mathrm{best}}^k \text{ yielding the minimum } \mathtt{O}_k. \tag{5}$$

Remark 1 Sequence $\{\mathtt{O}_k\}$ is non-increasing and non-negative by construction. It is easy to see that if $\mathtt{O}_k = 0$ for an arbitrary k, then x_{best}^k is a solution to problem (1). We refer the reader to [27, Lemma 1] for a formal proof of this claim. □

Given a parameter $\gamma \in (0,1)$, the certificate of optimality $\mathtt{O}_k$ is also useful to define the so called level parameter and level set, respectively given by

$$f_{\mathrm{lev}}^k := f_{\mathrm{low}}^k + \gamma\mathtt{O}_k \;\text{ and }\; \mathbb{X}^k := \{x \in X :\, \check{f}_k(x) \leq f_{\mathrm{lev}}^k,\, \check{c}_k(x) \leq 0\}\,. \tag{6}$$

Provided that $\mathbb{X}^k$ is nonempty, a new iterate x^{k+1} for a certain extended level bundle method is a properly chosen point in $\mathbb{X}^k$. A general way to choose the next iterate x^{k+1} in $\mathbb{X}^k$ is by solving the following optimization problem:

$$x^{k+1} = \arg\min_{x\in\mathbb{X}^k} \varphi(x;\hat{x}^k)\,, \tag{7}$$

where $\hat{x}^k \in X$ is a given stability center and $\varphi : \mathbb{R}^n \to \mathbb{R}_+$ a given stability function. Different choices for the function φ and stability center $\hat{x}^k$ define different level bundle methods, as discussed in Section 3 below. For instance, in the convex setting, [19] takes $\hat{x}^k = x^k$ and $\varphi(x;\hat{x}^k)$ as the Euclidean distance $\|x - \hat{x}^k\|_2$. In this article we are interested in a mixed-integer setting; thus, other choices for the stability center $\hat{x}^k$ and stability function φ in (7) might be preferable (for instance, setting φ as the ℓ_1-norm).

Remark 2 As discussed in [27], if the set $\mathbb{X}^k$ is empty, then $f_{\mathrm{lev}}^k < f(x)$ for all $x \in \{\tilde{x} \in X :\, \check{c}_k(\tilde{x}) \leqslant 0\}$. Since $\check{c}_k(\cdot) \leqslant c(\cdot)$, it follows that f_{lev}^k is a lower bound for the optimal value $f_{\min}$ whenever $\mathbb{X}^k$ is an empty set. We can thus update the lower bound f_{low}^k by the simple rule presented in [27]:

$$\text{set } f_{\mathrm{low}}^k \leftarrow f_{\mathrm{lev}}^k \text{ whenever } \mathbb{X}^k = \emptyset\text{; then use (6) to increase the level parameter.} \quad \square$$

We now give our algorithm, which is an extension of [27, Algorithm 1] to the mixed-integer setting, leaving unspecified for the moment a rule to define the next iterate x^{k+1}. We will come back to this subject in Section 3 below.

Algorithm 1 (General framework for extended level bundle methods)

Step 0. Select $\gamma \in (0,1)$ and a stopping tolerance $\delta_{\text{Tol}} > 0$. Choose $x^0 \in X$, $f^0_{\text{low}} \leq f_{\min}$, a stability function φ and a rule to define the stability center $\hat{x}^k$ in (7). Call the oracle to obtain $(f(x^0), g^0_{\text{f}})$ and $(c(x^0), g^0_{\text{c}})$. Set $\mathcal{B}_0 \leftarrow \{0\}$, $\ell \leftarrow 0$, $k(\ell) \leftarrow 0$, and $k \leftarrow 0$.

Step 1. Find x^k_{best} and $\mathtt{O}_k$ as in (5). If $\mathtt{O}_k \leq \delta_{\text{Tol}}$, stop: x^k_{best} is a δ_{Tol}-solution.

Step 2. If $\mathtt{O}_k \leq (1-\gamma)\mathtt{O}_{k(\ell)}$, set $k(\ell+1) \leftarrow k$ and $\ell \leftarrow \ell+1$. Choose $\mathcal{B}_k \supset \{k\}$.

Step 3. Define f^k_{lev} and $\mathbb{X}^k$ as in (6). Choose a stability center $\hat{x}^k$ according to the given rule.

Step 4. If $\mathbb{X}^k = \emptyset$, set $f^k_{\text{low}} \leftarrow f^k_{\text{lev}}$, $k(\ell+1) \leftarrow k$, $\ell \leftarrow \ell+1$, and go back to Step 1.
Otherwise, find a point $x^{k+1} \in \mathbb{X}^k$ by solving (7).

Step 5. Call the oracle to compute $(f(x^{k+1}), g^{k+1}_{\text{f}})$ and $(c(x^{k+1}), g^{k+1}_{\text{c}})$.

Step 6. Set $\mathcal{B}_{k+1} = \mathcal{B}_k \cup \{k+1\}$, $f^{k+1}_{\text{low}} \leftarrow f^k_{\text{low}}$, $k \leftarrow k+1$ and go back to Step 1.

Suppose that the initial lower bound f^0_{low} is not available, then f^0_{low} can be obtained by solving the MILP problem $\min_{x\in X} f(x^0) + \langle g^0_{\text{f}}, x - x^0\rangle$, or its linear programming relaxation.

From the practical viewpoint, usually optimality and feasibility tolerances are different, reflecting the fact that one may be satisfied with an approximated solution, but needs the constraints to be strictly satisfied. Moreover, tolerances are typically given as relative to avoid problems with scaling. For these reasons, an alternative stopping test could be of the form

$$\frac{f(x^k_{\text{best}}) - f^k_{\text{low}}}{(1+|f^k_{\text{low}}|)} \leq \delta^f_{\text{Tol}} \;\text{ and }\; c(x^k_{\text{best}}) \leq \delta^c_{\text{Tol}}\,,$$

for given tolerances $\delta^c_{\text{Tol}}, \delta^f_{\text{Tol}} > 0$, with δ^c_{Tol} usually less than δ^f_{Tol}. In order to ease notation, we consider in the convergence analysis of Algorithm 1 the simpler stopping test $\mathtt{O}_k \leq \delta_{\text{Tol}}$.

Remark 3 A natural procedure in Step 5 corresponds to try to find x^{k+1} directly, instead of verifying whether $\mathbb{X}^k$ is an empty set. If an optimization solver is employed to solve subproblem (7) to define x^{k+1}, we may expect an exit flag providing information on feasibility of $\mathbb{X}^k$. □

Since a regularized subproblem as (7) is solved to define the next iterate x^{k+1}, the index set $\mathcal{B}_k$ can be reduced in Step 2 whenever the algorithm provides a sufficient decrease of the improvement function. This is akin to a complete restart of the algorithm, but with some form of memory, yielded by $\mathtt{O}_k$, $\hat{x}^k$ and φ, that ensures overall convergence.

In what follows, consider *cycles* denoted by

$$K^\ell := \{k(\ell), \ldots, k(\ell+1)-1\}, \quad \text{for } \ell \geq 0\,. \tag{8}$$

In Step 6, Algorithm 1 adds one more index into $\mathcal{B}_{k+1}$. Therefore, two more constraints $f(x^{k+1}) + \langle g^{k+1}_{\text{f}}, x - x^{k+1}\rangle \leq 0$ and $c(x^{k+1}) + \langle g^{k+1}_{\text{c}}, x - x^{k+1}\rangle \leq 0$ are added into the level set given in (6). We thus conclude that

$$\mathbb{X}^{j+1} \subset \mathbb{X}^j \subset \mathbb{X}^{k(\ell)} \;\text{ for all }\; j, j+1 \in K^\ell\,, \tag{9}$$

because from (8), it follows by construction that $f^{k(\ell)}_{\text{low}} = f^j_{\text{low}}$ and $f^{k(\ell)}_{\text{lev}} \geq f^j_{\text{lev}}$ for all $j \in K^\ell$.

2.1 Convergence analysis

We first recall that f and c are convex functions and X is a bounded set. Thus, there exist two constants L_f, $L_c > 0$ (possibly unknown) such that

$$|f(x) - f(\tilde{x})| \leq L_f\|x - \tilde{x}\| \text{ and } |c(x) - c(\tilde{x})| \leq L_c\|x - \tilde{x}\|\,, \tag{10}$$

for all $x, \tilde{x} \in X$. We start our analysis with the following two lemmas that hold regardless of how $x^{k+1} \in \mathbb{X}^k$ is chosen (for example, x^{k+1} can be any feasible point for (7)).

Lemma 1 *If $\delta_{\text{Tol}} = 0$ and $\ell \to \infty$, then $\mathtt{O}_k \to 0$.*

Proof Let $\mathcal{A}$ be an index set gathering indices ℓ updated only in Step 4 of Algorithm 1. Suppose that $\mathcal{A}$ has infinitely many indices. Whenever the level set is found to be empty in Step 4, the lower bound f_{low}^k is increased by an amount of $\gamma \mathtt{O}_k > 0$ and the cycle-counter ℓ is incremented by one. Moreover, Remark 2 ensures that $f_{\text{low}}^k \leqslant f_{\min}$ for all $k \geqslant 0$. We conclude that $f_{\text{low}}^{k(\ell+1)} > f_{\text{low}}^{k(\ell)}$ because the stopping test in Step 2 fails. Since $f_{\min} < \infty$ is an upper bound on this sequence, we must therefore have that $\mathtt{O}_k \to 0$. (Hence, if $\delta_{\text{Tol}} > 0$ there cannot be an infinite loop between Steps 4 and 1.)

Suppose now that $\mathcal{A}$ is a finite set. Then, there exists an index $\bar{\ell} \geq 0$ such that $\mathbb{X}^k \neq \emptyset$ for all $k \geq k(\bar{\ell})$. It follows from Step 2 that $(1-\gamma)^{\ell-\bar{\ell}+1}\mathtt{O}_{k(\bar{\ell})} \geq \mathtt{O}_{k(\ell+1)}$ for $\ell \geq \bar{\ell}$. The result is now proven because $\gamma \in (0,1)$, $\{\mathtt{O}_k\}$ is non-increasing and $\ell \to \infty$. □

If the tolerance δ_{Tol} is strictly positive, Lemma 1 ensures that the stopping test of Algorithm 1 will be triggered as long as ℓ increases sufficiently. Accordingly, our goal is to show that each cycle represented by K^ℓ in (8) has only finitely many indices. For this we need the following result, which is a particular case of Lemma 3 in [27].

Lemma 2 *Let $\ell \geq 0$ be arbitrary, $\gamma \in (0,1)$ be given, and $\Lambda = \max\{L_f, L_c\}$, where L_f and L_c are the Lipschitz constants for f and c given in* (10)*. At iteration $k \in K^\ell$ of Algorithm 1, the following estimates hold:*

$$\|x^{k+1} - x^j\| \geq \frac{(1-\gamma)}{\Lambda}\mathtt{O}_k \quad \textit{for} \quad j = k(\ell)+1, \ldots, k\,.$$

Proof We recall that $K^\ell \subset \mathcal{B}_k$. Therefore, since x^{k+1} is chosen in $\mathbb{X}^k$ given in (6), we get

$$f(x^j) + \langle g_{\text{f}}^j, x^{k+1} - x^j\rangle \leqslant \check{f}_k(x^{k+1}) \leqslant f_{\text{lev}}^k \quad \text{and} \quad c(x^j) + \langle g_{\text{c}}^j, x^{k+1} - x^j\rangle \leqslant \check{c}_k(x^{k+1}) \leqslant 0$$

for all $j \in K^\ell$ such that $j > k(\ell)$. By applying the Cauchy-Schwarz inequality we thus derive

$$f(x^j) - f_{\text{lev}}^k \leqslant \|g_{\text{f}}^j\|\|x^{k+1} - x^j\| \leqslant \Lambda\|x^{k+1} - x^j\| \quad \text{and} \quad c(x^j) \leqslant \|g_{\text{c}}^j\|\|x^{k+1} - x^j\| \leqslant \Lambda\|x^{k+1} - x^j\|\,.$$

Since $f_{\text{lev}}^k = f_{\text{low}}^k + \gamma\mathtt{O}_k$ and $\mathtt{O}_k \geqslant 0$ by Remark 1, we conclude that

$$\begin{aligned}\Lambda\|x^{k+1} - x^j\| &\geqslant \max\{f(x^j) - (f_{\text{low}}^k + \gamma\mathtt{O}_k),\, c(x^j)\} \;\geqslant \max\{f(x^j) - f_{\text{low}}^k - \gamma\mathtt{O}_k,\, c(x^j) - \gamma\mathtt{O}_k\}\\ &= -\gamma\mathtt{O}_k + \max\{f(x^j) - f_{\text{low}}^k,\, c(x^j)\} \geqslant (1-\gamma)\mathtt{O}_k\,,\end{aligned}$$

where the last inequality is due to (5). Hence, the result follows. □

As long as the iterate x^{k+1} is a point belonging to the level set $\mathbb{X}^k$, Algorithm 1 ensures $\mathtt{O}_k \to 0$. This statement is formalized in the following result.

Theorem 1 *Suppose the mixed-integer feasible set $X \neq \emptyset$ in* (1) *is bounded, and that f_{low}^0 is chosen to satisfy $f_{\text{low}}^0 \leq f_{\min}$. Then, if $\delta_{\text{Tol}} = 0$ Algorithm 1 ensures that $\mathtt{O}_k \to 0$ regardless the employed rule to define $x^{k+1} \in \mathbb{X}^k$. If $\delta_{\text{Tol}} > 0$, Algorithm 1 finds a δ_{Tol}-solution to problem* (1) *in finitely many iterations.*

Proof By Remark 1 and Lemma 1 we just need to show that each cycle K^ℓ given in (8) with $\ell \geq 0$ has finitely many indices. In order to do so, let us suppose by contradiction that there is an index $\ell \geq 0$ such K^ℓ has infinitely many indices k. Thus, it holds that $\mathbb{X}^k \neq \emptyset$ and $\mathtt{O}_k > (1-\gamma)\mathtt{O}_{k(\ell)} \geq \epsilon > 0$ for all $k \in K^\ell$ (if such $\epsilon > 0$ does not exist, then $\mathtt{O}_{k(\ell)} = 0$ and the algorithm would have terminated). It follows from (9) that the (infinite) sequence $\{x^k\}_{k\in K^\ell}$ is contained in the bounded level set $\mathbb{X}^{k(\ell)}$. Lemma 2 gives

$$\|x^{k+1} - x^j\| \geq \frac{(1-\gamma)\mathtt{O}_k}{\Lambda} \geq \frac{(1-\gamma)\epsilon}{\Lambda} > 0 \;\text{ for all }\; j,k \in K^\ell \;\text{ such that }\; j \leqslant k\,.$$

The above relation shows that the bounded (and infinite) sequence $\{x^k\}_{k\in K^\ell} \subset X$ has no accumulation point in the closed and bounded set X, which is impossible in a finite-dimensional space. Hence, each cycle K^ℓ must have only finitely many indices and the result follows. □

Theorem 1 shows convergence of Algorithm 1 by assuming that x^{k+1} is arbitrarily chosen in the set $\mathbb{X}^k$ given in (6) (for example by solving (4), the ECPM subproblem, or by using a single tree strategy, [2, § 3.3]). Notwithstanding, the number of oracle calls to be performed by Algorithm 1 depends strongly on the rule to determine the iterates in Step 4. In the next section we give some alternatives to determine iterates by solving a subproblem as (7).

3 Obtaining new iterates: extended level bundle variants

Notice that when $\mathbb{X}^k$ is nonempty, the point x^{k+1} determined by solving (4) belongs to $\mathbb{X}^k$. In this case, Algorithm 1 is nothing but the extended cutting-plane algorithm proposed in [29].

It is worth mentioning that cutting-plane algorithms have three known drawbacks: (a) no matter how good is the initial point x^0 given to the algorithm, the iteration process will not exploit quality of x^0 for defining new iterates (initial iterates x^k can be far away from the "good candidate" x^0); (b) tailing-off effect that makes calculations unstable as the iteration process progresses; (c) optimization subproblem (4) determining new iterates has more and more constraints and, hence, gets more and more ill-conditioned and difficult to solve.

In the convex setting, bundle methods [19,8] are able to overcome all the three drawbacks mentioned above. A special class of bundle methods, known as level bundle methods [8], employs subproblem (7) for defining new iterates. Depending on the choice of the stability function φ and stability center $\hat{x}^k$ in (7), the resulting algorithm provides:
(i) an upper bound for the maximum number of iterations performed in each cycle K^ℓ; and
(ii) a strategy to keep the information bundle $\mathcal{B}_k$ bounded, i.e., restricted memory.
Characteristics (i) and (ii) above are well-known in the level bundle literature, [8]. However, the mixed-integer setting is rather intricate. We thus rely on the essence of level bundle methods to provide techniques that overcome items (a) and (b) above, and try to remedy, in the practical viewpoint, item (c). No theoretical guarantees for (i) and (ii) are given.

3.1 Choosing stability centers

There is considerable freedom to choose the stability centers $\hat{x}^k$ in subproblem (7), solved at each iteration by Algorithm 1. We now enumerate some possibilities:

– *Fixed stability center.* A simple rule to update $\hat{x}^k$ along the iterative process is

$$\hat{x}^{k+1} = \hat{x} \ \text{ for all iteration } k\,,$$

where $\hat{x} \in \mathbb{R}^n$ is an arbitrary but fixed point; for example $\hat{x} = x^0$, the initial point.

– *Current iterate.* Proposed in [19], this rule consists in taking

$$\hat{x}^{k+1} = x^{k+1} \ \text{ for all iteration } k\,.$$

– *Incumbent iterate.* As in [27], the rule consists in updating the stability center only after obtaining enough decrease of the certificate of optimality, that is,

$$\text{if } \mathtt{O}_k \le (1-\gamma)\mathtt{O}_{k(\ell)}, \text{ then set } \hat{x}^{k+1} \leftarrow x^k_{\text{best}}. \text{ Otherwise, set } \hat{x}^{k+1} \leftarrow \hat{x}^k.$$

– *Serious iterate.* If $f(x^{k+1}) \le f(\hat{x}^k) - (1-\gamma)\mathtt{O}_k$ and $c(x^{k+1}) \le 0$, or[1] $c^+(x^{k+1}) \le c^+(\hat{x}^k) - (1-\gamma)\mathtt{O}_k$, then set $\hat{x}^{k+1} \leftarrow x^{k+1}$. Otherwise, set $\hat{x}^{k+1} \leftarrow \hat{x}^k$.

A good choice for the stability-center rule will depend on the problem's assumptions. For example, if a good estimate $\hat{x}$ for the optimal solution is known, one may employ the *fixed stability center* rule. On the other hand, if nothing about the optimal solution is known, one may prefer the *current iterate* rule, aiming to escape from bad quality initial iterates x^k. Overall, a satisfactory rule is the *incumbent iterate* one, which is a hybrid of the two previous rules. The *serious iterate* rule is designed to overcome the cutting-plane method's drawback mentioned in item (a) above.

We now deal with another important ingredient for Algorithm 1: the stability function φ.

[1] The notation c^+ stands for $c^+(x) = \max\{c(x), 0\}$.

3.2 Choosing stability functions

As for the stability center, there is also a fair amount of freedom to choose the stability function φ in (7). The main importance of φ is to keep the next iterate near to the given center $\hat{x}^k$. Below we list some alternatives for φ.

– ℓ_2-norm. Most level bundle methods found in literature employ the Euclidean norm as a stabilization function; see [8]. Accordingly, subproblem (7) results in a *mixed-integer quadratic programming* problem - MIQP, that is

$$x^{k+1} \in \arg\min_{x\in\mathbb{X}^k} \|x-\hat{x}^k\|_2 \quad \equiv \quad \min_{x\in\mathbb{X}^k} \langle x,x\rangle - 2\langle x,\hat{x}^k\rangle\,.$$

(If both x and $\hat{x}$ belong to $\{0,1\}^n$, then the objective function above can be written as $\langle \frac{\mathbb{1}}{2}-\hat{x}^k, x\rangle$, where $\mathbb{1}\in\mathbb{R}^n$ is the vector of ones. In this case, subproblem above is a binary linear programming problem.)

– ℓ_1-norm. Differently from the Euclidean norm, the ℓ_1-norm choice for the stability function φ leads to a MILP problem:

$$x^{k+1} \in \arg\min_{x\in\mathbb{X}^k} \|x-\hat{x}^k\|_1 \quad \equiv \quad \min_{x\in\mathbb{X}^k, s\in\mathbb{R}^n} \sum_{i=1}^{n} s_i \;\text{ s.t. }\; x_i-\hat{x}_i^k \le s_i,\; \hat{x}_i^k - x_i \le s_i \;\forall i=1,\ldots,n\,.$$

(For binary variables, $\|x-\hat{x}^k\|_1$ can be written as $\sum_{j\in\{i:\hat{x}_i^k=0\}} x_j + \sum_{j\in\{i:\hat{x}_i^k=1\}}(1-x_j)$, which is a linear function.)

– ℓ_∞-norm. Taking ℓ_∞ as the stability function also leads to a MILP problem:

$$x^{k+1} \in \arg\min_{x\in\mathbb{X}^k} \|x-\hat{x}^k\|_\infty \quad \equiv \quad \min_{x\in\mathbb{X}^k, s\in\mathbb{R}} s \;\text{ s.t. }\; x_i-\hat{x}_i^k \le s,\; \hat{x}_i^k - x_i \le s \;\forall i=1,\ldots,n\,.$$

– Doubly stabilization. Given a norm $\|\cdot\|$ and a prox-step $t_k>0$, the next iterate can be obtained by solving the following doubly stabilized subproblem:

$$x^{k+1} \in \arg\min_{x\in\mathbb{X}^k} \check{f}_k(x) + \frac{1}{2t_k}\|x-\hat{x}^k\|\,.$$

With this choice for subproblem (7), Algorithm 1 becomes a mixed-integer variant of the method proposed [9].

In general, MIQP problems are more difficult to solve than MILP. Therefore, the ℓ_1 and ℓ_∞ norms might be preferable to the ℓ_2-norm as stabilization functions. A possible advantage of the ℓ_∞-norm over the ℓ_1 is that when dealing with the above formulation of problem (7), less extra variables are required to put the problem in a canonical form.

We have presented four alternatives for defining the stability function, and four rules for choosing the stability centers. In a total, sixteen variants of Algorithm 1 are provided. Naturally, more variants can be proposed by simply changing φ or the stability-center rule.

4 Numerical assessment

In this section we report some numerical experiments comparing four level bundle variants with an extended cutting-plane algorithm and software BONMIN [5]. The goal is to verify empirically that one can reduce the number of oracle calls required to solve a convex MINLP problem by regularizing the MILP subproblem as in (7). As mentioned in the Introduction, performing fewer oracle calls is of interest in some industrial applications, where the involved functions are costly.

Solvers. In order to solve the corresponding subproblems (7) at each iteration we employed the Gurobi toolbox for MATLAB; see `http://www.gurobi.com`. In a total, seven solvers were considered. Four of them employ Algorithm 1 (coded in MATLAB), with two different stability functions, and two rules to define stability centers in (7). The solvers are named:

– ℓ_1 – *Current iterate* rule to define $\hat{x}^k$, and ℓ_1-norm for the stability function;
– ℓ_∞ – *Current iterate* rule to define $\hat{x}^k$, and ℓ_∞-norm for the stability function;
– $\hat{\ell}_1$ – *Incumbent iterate* rule to define $\hat{x}^k$, and ℓ_1-norm for the stability function;
– $\hat{\ell}_\infty$ – *Incumbent iterate* rule to define $\hat{x}^k$, and ℓ_∞-norm for the stability function;
– `CP` – Algorithm 1 solving subproblem (4) to define x^{k+1}.
– `B-BB` – A NLP-based branch-and-bound algorithm implemented in BONMIN [5];
– `B-QG` – An implementation of the branch-and-cut algorithm proposed in [23] and available in BONMIN.

Notice that solver `CP` is nothing but a simple variant of the extended cutting-plane method proposed in [29]. Solvers `B-BB` and `B-QG` are available in the software BONMIN (`https://projects.coin-or.org/Bonmin`), which were employed through the OPTI toolbox for Matlab [6].

Algorithm's parameters. All the solvers were initialized with x^0, a point that solves the following MILP problem:

$$\min_{x\in X} \langle g_{\mathrm{f}}, \tilde{x} - x\rangle \;\text{ s.t. }\; c(\tilde{x}) + \langle g_{\mathrm{c}}, \tilde{x} - x\rangle \leq 0\,,$$

where $\tilde{x} \in \mathbb{R}^n$ is (in most of test-problems) the vector of ones, and $g_{\mathrm{c}} \in \partial f(\tilde{x})$ (respectively, $g_{\mathrm{c}} \in \partial c(\tilde{x})$). A lower bound f^0_{low} was set as the optimal value of the above MILP problem. Moreover, we set $\gamma = 0.2$ in Algorithm 1 and use the relative stopping test $\mathtt{O}_k/(1+|f^k_{\mathrm{low}}|) \leq 10^{-4}$. Since the five first solvers use the same stopping test, obtained solutions have the same accuracy. Solvers `B-BB` and `B-QG` were set with the same relative tolerance. If the stopping test is not satisfied, the five first algorithms stop by the maximum number of iterations, which was set as 3000. All solvers were set with a CPU time limit of one hour. In order to illustrate the solvers' performance we have considered several test problems described below.

Problems from `MINLP Library`. We consider a set of 25 test-problems from the *MINLP Library* available at `http://www.gamsworld.org/minlp`. These problems have the form

$$\min_{x\in X} f(x) \;\text{ s.t. }\; f_i(x) \leq 0, \;\text{ for }\; i = 1,\ldots,m,$$

with $f, f_i : \mathbb{R}^n \to \mathbb{R}$ convex and continuously differentiable functions. Solvers `B-BB` and `B-QG` consider the original problem structure, while solvers `CP`, ℓ_1, ℓ_∞, $\hat{\ell}_1$ and $\hat{\ell}_\infty$ consider the nonsmooth reformulation (1), by taking $c(x) = \max_{j=1,\ldots,m} f_j(x)$.

Table 4 presents the problem's names, dimensions (n, m) (which are the number of variables and nonlinear constraints, respectively) and optimal values. CPU time (in seconds) and number of oracle calls required to solve these 25 problems are also presented in the Table 4 for five solvers: `CP`, ℓ_1, ℓ_∞, `B-BB` and `B-QG`. Total CPU time and oracle calls are presented in the last line of the table. As reported, solvers `CP`, ℓ_1, ℓ_∞ and `B-QG` could solve all these problems within the CPU time limit of one hour (in contrast to `B-BB`), being `B-QG` much faster than the other considered solvers. However, solvers `B-QG` and `B-BB` required significantly more oracle calls than the cutting-plane based algorithms `CP`, ℓ_1 and ℓ_∞. This shows that the latter class of methods is preferable when function evaluation is expensive. Concerning number of oracles calls, solver ℓ_∞ was the most effective one for this set of problems: ℓ_∞ performed 23.6% (respectively 99.97%) fewer oracles calls than `CP` (respectively `B-QG`).

Results for solvers $\hat{\ell}_\infty$ and $\hat{\ell}_1$ on these problems are summarized in Table 4, where total CPU time and number of oracle calls are reported. Table 4 also presents results for the considered cutting-plane based methods applied to the test-problems by adopting formulation (1), but with $c : \mathbb{R}^n \to \mathbb{R}^{\tilde{m}}$ defined as follows: by partitioning the set $\{1,\ldots,m\}$ into $\tilde{m} \leq m$ subsets $I_1,\ldots,I_{\tilde{m}}$ such that $I_i \cap I_j = \emptyset$ if $i \neq j$, we reformulate each test-problem as (1) with $c_j(x) = \max_{i\in I_j} f_j(x)$,

Problem	Data (n,m)	f^*	CPU time (s) CP	ℓ_1	ℓ_∞	B-BB	B-QG	# Oracle calls CP	ℓ_1	ℓ_∞	B-BB	B-QG
st_e14	(11,13)	4.58	0.2	0.3	0.2	0.9	0.2	21	15	13	36	11
synthes2	(11,13)	73.04	0.1	0.2	0.2	0.3	0.2	11	11	11	94	39
synthes3	(17,23)	68.01	0.2	0.4	0.4	0.6	0.5	18	15	15	145	129
ex1223	(11,13)	4.58	0.2	0.3	0.2	0.2	0.1	21	15	13	36	11
ex1223b	(7,9)	4.58	0.2	0.3	0.2	0.3	0.1	18	16	14	21	11
flay02h	(46,51)	37.95	0.8	2.4	2.4	0.3	0.2	53	52	59	13	43
flay02m	(14,11)	37.95	0.5	1.2	1.1	0.2	0.1	38	41	38	6	15
flay03h	(122,144)	48.99	7.7	21.4	18.6	2.8	1.7	172	165	157	1519	1884
flay03m	(26,24)	48.99	1.6	5.2	4.2	1.6	0.2	73	88	76	1067	73
flay04m	(42,42)	54.41	12.7	19.5	18.8	43.9	2.8	184	119	129	41352	12015
m3	(26,43)	37.80	0.5	1.0	1.1	1.2	0.1	36	35	36	827	272
m6	(86,157)	82.26	11.3	15.5	10.7	255.7	7.7	153	129	103	212356	124191
m7	(114,211)	106.76	21.1	47.0	44.6	2359.1	37.9	204	185	140	1886624	605724
m7_ar2_1	(112,269)	190.23	41.8	67.5	56.7	3600.1	42.4	234	213	164	2798895	1131309
m7_ar3_1	(112,269)	143.59	114.6	168.3	89.3	3600.3	50.0	227	204	161	2642214	926234
m7_ar4_1	(112,269)	106.76	38.9	54.3	28.4	3600.1	20.2	211	194	145	2553283	296164
m7_ar5_1	(112,269)	106.46	37.4	58.9	63.1	3600.1	141.3	222	195	153	2613613	4820486
m7_ar25_1	(112,269)	143.59	25.9	41.6	27.3	426.8	10.0	217	200	151	326416	308859
gear	(4,0)	0.00	0.0	0.1	0.1	0.4	0.0	6	11	10	136	14
tls2	(37,24)	5.30	0.2	0.5	0.5	13.6	1.1	15	16	16	11891	4912
tls4	(105,64)	8.30	1279.4	2063.2	1640.4	2568.2	71.4	276	271	242	2345105	981449
batch	(46,69)	285506.51	2.3	3.9	5.0	1.4	0.3	78	62	77	89	224
batchdes	(19,19)	167427.66	0.2	0.4	0.4	0.3	0.1	23	18	20	10	10
batch0812	(100,217)	2687026.78	4.0	6.9	7.3	5.1	1.6	90	66	79	634	1711
batchs101006m	(278,1019)	769440.42	317.6	384.6	533.6	35.6	23.6	1000	572	727	7998	102142
Sum	-	-	1919.4	2965.0	2554.8	20119.1	414.1	3601	2908	2749	15444380	9317932

Table 1 Comparison of five solvers on 25 test-problems from *MINLP library*.

$\tilde{m}$	CPU time (s) CP	ℓ_1	ℓ_∞	$\hat{\ell}_1$	$\hat{\ell}_\infty$	# Oracle calls CP	ℓ_1	ℓ_∞	$\hat{\ell}_1$	$\hat{\ell}_\infty$
1	1919.4	2965.0	2554.8	5094.2	3695.9	3601	2908	2749	3426	3088
50	727.9	678.6	789.9	710.2	1218.7	544	526	522	536	514
m	666.6	865.9	821.9	1042.7	918.2	355	341	342	362	338

Table 2 Analysis of total CPU time and number of oracle calls to solve 25 MINLP problems.

for $j = 1, \ldots, \tilde{m}$. We consider in Table 4 $\tilde{m} \in \{1, 50, m\}$. For the case $\tilde{m} = 1$, all the level bundle variants require fewer oracle calls than the extended cutting-plane solver CP. When all the problem's constraints are considered individually, i.e. $\tilde{m} = m$, all the five solvers have a comparable performance in terms of oracle calls, being CP slightly faster than the other variants.

In what follows we will be dealing only with nonsmooth MINLP problems, and thus solvers B-BB and B-QG will not be employed. It is already clear from Table 4 that such solvers require in general many oracle calls, an attribute that we aim to avoid.

MaxQuad *problems.* In this subsection we consider randomly generated MINLP problems of the form (1) with objective functions given by

$$f_1(x) = \max_{i=1,\ldots,10} \big\{ \langle Q_i x, x\rangle + \langle q_i, x\rangle \big\} + \alpha |x|_1 \quad \text{and} \quad f_\infty(x) = \max_{i=1,\ldots,10} \big\{ \langle Q_i x, x\rangle + \langle q_i, x\rangle \big\} + \alpha |x|_\infty \,,$$

where $Q_i \in \mathbb{R}^{n\times n}$ and $q_i \in \mathbb{R}^n$ are randomly generated matrices and vectors for all $i = 1, \ldots, 10$ (Q_i are symmetric and positive semidefinite matrices). Parameter α runs through the values $\alpha \in \{0.5, 1\}$. The problem's dimension n varies according to $n \in \{10, 20, 30, 40\}$, and feasible set is

$$X = \left\{ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : \frac{1}{n}\sum_{i=1}^{n} x_i \le -1 \ \text{ and } \ \begin{bmatrix} x_1 \in \{-3,-2,-1,0\} & \\ x_i \in \{-1,0\} & if\ i = 2,\ldots,p \\ x_i \in [-20,20] & if\ i = p+1\ldots,n \end{bmatrix} \right\}.$$

In the above set, the integer number p is defined as $p = \min\{n/2, 10\}$. The nonlinear and nonsmooth constraint function is given by $c(x) = \max\{e^{-\frac{1}{n}\sum_{i=1}^n x_i},\, e^{-x_n}\} - e$. With this setting, we have considered four different types of problems, named

- $\bullet\, 1 :\ \min f_1(x)$ s.t. $x \in X$ and $c(x) \le 0$,
- $\bullet\, 2 :\ \min f_\infty(x)$ s.t. $x \in X$ and $c(x) \le 0$,
- $\bullet\, 3 :\ \min f_1(x)$ s.t. $x \in X$,
- $\bullet\, 4 :\ \min f_\infty(x)$ s.t. $x \in X$.

For each problem and considered dimension n, ten instances were generated and solved by employing the five cutting-plane based solvers.

Table 4 provides the average (over ten instances) of CPU time, number of oracle calls, and number of empty level sets. If any solver reached the maximum number of iterations, which was fixed as 3000, we consider the problem not solved. If at least one problem belonging to a given set of ten instances was not solved, we provide the character "-" in Table 4. We mention that the number of iterations of solver `CP` coincides with the number of oracle calls. For the other solvers, the (average) number of iterations can be obtained by summing the (average) number of oracle calls and (average) number of empty level sets.

n	Prob	CPU time (s)					# Oracle calls					# Empty level set			
		`CP`	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	`CP`	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
10	1	1.3	2.9	2.6	3.2	2.8	38.7	32.8	31.3	34.1	33.1	18.4	17.4	18.8	18.0
10	2	1.3	3.0	3.1	3.1	3.1	38.6	37.0	38.5	39.1	38.2	18.6	18.0	18.1	17.1
10	3	1.5	3.4	3.4	3.5	3.4	52.5	40.8	42.5	43.4	44.1	22.2	21.6	23.2	21.8
10	4	1.6	3.3	3.1	3.3	3.3	53.0	41.1	39.7	42.1	42.6	23.0	21.9	23.0	20.8
20	1	12.0	15.4	19.0	18.0	19.2	157.1	119.3	131.1	134.3	136.5	32.7	30.1	32.2	30.5
20	2	5.8	8.4	9.9	8.9	10.1	153.5	121.1	130.6	129.0	135.9	32.9	31.0	32.1	30.0
20	3	7.5	11.1	12.9	11.9	13.7	198.7	144.5	160.2	156.1	169.8	35.5	33.1	34.8	34.0
20	4	6.7	10.0	12.0	11.0	11.7	190.5	138.6	155.5	152.0	155.5	35.0	33.7	34.6	33.3
30	1	108.5	67.2	123.7	74.0	95.5	656.8	381.6	515.8	415.5	469.6	41.6	40.0	41.2	39.7
30	2	116.8	66.1	119.3	72.5	94.0	656.4	378.1	503.5	408.5	466.0	42.8	40.0	41.9	40.1
30	3	200.4	83.5	160.7	86.4	113.5	920.0	446.8	619.6	477.2	555.3	43.0	40.4	43.0	40.5
30	4	135.3	74.2	146.1	82.4	104.7	797.2	421.2	589.7	464.2	527.6	43.0	40.2	42.8	40.1
40	1	-	299.9	789.8	292.3	423.9	-	740.4	1192.2	768.4	928.5	44.3	40.9	44.1	41.8
40	2	-	268.3	702.1	287.8	388.7	-	703.4	1131.5	759.6	890.2	45.5	42.1	44.4	42.5
40	3	-	347.2	999.7	337.5	492.8	-	810.6	1330.1	869.5	1044.1	46.0	43.4	45.6	43.2
40	4	-	327.7	1014.2	331.1	462.4	-	789.2	1333.6	846.8	1011.1	45.9	43.6	45.5	43.3

Table 3 Average over ten instances. Total of 160 different problems.

Table 4 shows that solver `CP` was faster than the extended level bundle solvers ($\hat{\ell}_1$, $\hat{\ell}_\infty$, ℓ_1, and ℓ_∞) for the small instances, corresponding to $n = 10$ and $n = 20$. However, for instances with dimension $n \geq 30$, all the level bundle solvers but $\hat{\ell}_\infty$ were faster than `CP`, which failed to solve 10 of the largest instances within the maximum number of iterations. Notice also that the level bundle solvers solved all the 160 problems without reaching the maximum number of iterations, and required significantly fewer oracle calls than `CP`.

Table 4 gives the total CPU time and number of oracle calls to solve all 150 successful instances of the considered problems. As shown, all the level bundle solvers performed better (in terms of CPU time and number of oracle calls) than `CP`. The table also presents the percentage of CPU time and oracle call reduction with respect to `CP`.

Total	`CP`	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
Oracle calls	89955	45775	67273	49245	56622
Reduction % (w.r.t `CP`)	-	49.1	25.2	45.3	37.1
CPU Time (min)	615.1	207.4	542.5	213.6	291.3
Reduction % (w.r.t `CP`)	-	66.3	11.8	65.3	52.6

Table 4 Number of oracle calls and CPU time over 150 successful instances.

As one can notice, regularizing subproblem (4) pays off. Solver $\hat{\ell}_1$ reduced the number of oracle calls in 49%, and CPU time in 66%. Its counterpart ℓ_1, with current iterate rule, also provided good results. Reduction of oracle calls provided by the level solvers avoids the MILP subproblems to become large and too difficult to be solved. This explains why reducing oracle calls in 45% (solver ℓ_1) provided a CPU time reduction of 65%.

`QR` problems. In this subsection we consider a class of problems denoted by QR in [1, § 4.2]. These test problems have objective functions given by $f(x) = \max_{j=1,\ldots,10}\{b_j\|x-y_j\|_2^2+a_j\}$, where each

a_j and every component of each fixed center y_j is a random number uniformly drawn in [-5,5], and each b_j is a random number uniformly drawn in [0,5]. Feasible set is, for $p = n/2$, given by

$$X = \left\{ x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} : \begin{bmatrix} x_i \in \{0,1\} \;\; if \; i = 1, \ldots, p \\ x_i \in [-2,2] \; if \; i = p+1 \ldots, n \end{bmatrix} \right\}.$$

For each problem's dimension $n \in \{10, 20, 30, 40\}$, ten instances were generated and solved by employing the five cutting-plane based solvers. Table 4 presents the average (over ten instances) of CPU time, number of oracle calls, and number of empty level sets.

n	CPU time (s)					# Oracle calls					# Empty level set			
	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
10	0.5	0.9	0.9	0.9	0.9	40.5	31.5	32.2	33.8	34.4	13.5	12.9	13.4	13.0
20	9.4	7.1	14.5	8.6	12.2	205.8	114.8	168.1	132.0	159.5	19.8	19.8	19.7	19.9
30	46.2	39.2	109.5	43.5	74.0	480.9	263.7	443.8	293.7	378.1	22.3	21.4	22.6	21.2
40	148.2	82.3	368.5	90.7	185.7	928.2	371.8	747.3	403.3	552.2	23.3	21.5	22.7	21.8

Table 5 Average over ten instances. Total of 40 different problems.

Table 6 gives the total CPU time and number of oracle calls required to solve all 40 instances. Although level solvers performed fewer oracle calls, variants with ℓ_∞ norm stabilization required

Total	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
Oracle calls	16554	7818	13914	8628	11242
Reduction % (w.r.t CP)	-	52.8	15.9	47.9	32.1
CPU Time (min)	34.1	21.6	82.2	23.9	45.5
Reduction % (w.r.t CP)	-	36.6	-141.5	29.7	-33.5

Table 6 Number of oracle calls and CPU time over 40 instances.

more CPU time than the CP solver. This shows that the resulting MILP subproblem (7) with ℓ_∞ norm is (for these class of problems) more difficult to solve than (4).

Stochastic programming problems. In this subsection we consider a linearly constrained stochastic programming problem written as (1), with $f(x) = q^\top x + \frac{1}{N}\sum_{i=1}^N \mathcal{Q}_i(x)$ and feasible set $X = \{(z,y) \in \mathbb{Z}^{10} \times \mathbb{R}^{50}_+ : Ay + Bz = b, 0 \le z \le 10\}$, where $q \in \mathbb{R}^{60}$, $b \in \mathbb{R}^{30}$, and $\mathcal{Q}_i(x) := \min_{r \in \mathbb{R}^{90}_+} d^\top r$ s.t. $Wr = h_i - Tx$ is the recourse function (which is convex) and depends on the i-th scenario $h_i \in \mathbb{R}^{60}$. Matrices W and T have appropriate dimensions, and no nonlinear constraint c is considered. Note that f is an expensive function if the number N of scenarios is large.

We have considered five configurations of the problem, obtained by varying N according to {50, 100, 200, 300, 500}. For each number of scenarios N, 10 different problems were randomly generated by changing the sample $\{h_1, \ldots, h_N\}$.

N	CPU time (s)					# Oracle calls					# Empty level set			
	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
50	49.1	45.8	50.8	49.4	51.0	168.3	113.6	131.6	125.7	135.3	23.9	23.0	22.3	22.7
100	86.7	70.3	80.6	76.0	84.3	179.1	118.1	139.5	130.0	148.2	23.3	23.4	22.8	23.1
200	148.4	113.1	131.1	125.2	137.3	174.0	117.6	138.0	131.5	146.0	23.7	23.2	22.2	22.9
300	203.4	165.6	192.0	181.3	195.7	165.8	123.5	144.1	135.9	148.1	24.0	23.8	23.0	22.9
500	363.0	274.9	304.0	302.7	321.3	181.7	126.9	144.2	140.7	153.0	24.1	23.2	23.3	23.3

Table 7 Average over ten instances. Total of 50 different problems.

Table 8 gives the total CPU time and number of oracle calls to solve all 50 instances of the considered problem. As shown, all the level bundle solvers performed better (in terms of CPU time and number of oracle calls) than CP. The table also presents the percentage of CPU time and oracle call reduction with respect to the latter algorithm. Solver $\hat{\ell}_1$ reduced the number of oracle

Total	CP	$\hat{\ell}_1$	$\hat{\ell}_\infty$	ℓ_1	ℓ_∞
Oracle calls	8689	5997	6974	6638	7306
Reduction % (w.r.t CP)	-	31.0	19.7	23.6	15.9
CPU Time (min)	141.8	111.6	126.4	122.4	131.6
Reduction % (w.r.t CP)	-	17.3	6.3	10.5	2.7

Table 8 Number of oracle calls and CPU time over 50 instances

calls in 31%, and CPU time in 17%, showing the effectiveness of the regularized cutting-plane variant.

5 Concluding remarks

In this work we have proposed a class of regularized cutting-plane methods for convex and non-smooth mixed-integer nonlinear programming problems. Each iterate of the proposed variants is obtained by solving either a MIQP or a MILP subproblem. Although the given algorithm does not require solving a subproblem exactly at each iteration, solving it exactly for a given stability center and function may effectively reduce the number of oracle calls to solve the problem. As shown in Section 3, there is considerable freedom in choosing the subproblem structure. Moreover, the number of constraints (cutting-plane linearizations) can be reduced at some certain iterations, called critical iterations.

We have compared numerically some of the proposed variants with a non-regularized extended cutting-plane algorithm (CP) and the BONMIN solver on several academic MINLP problems. In some variants with fixed stability center, the number of oracle calls was reduced in around 50% with respect to CP, keeping approximately the same computational effort per iteration. Reduction of oracle calls was even more effective when compared to BONMIN. This is an important matter when dealing with MINLP problems whose involved functions are costly.

Acknowledgements. The author gratefully acknowledges financial support provided by Severo Ochoa Program SEV-2013-0323 and Basque Government BERC Program 2014-2017.

References

1. A. Astorino, A. Frangioni, M. Gaudioso, and E. Gorgone, *Piecewise-quadratic approximations in convex numerical optimization*, SIAM Journal on Optimization, 21 (2011), pp. 1418–1438.
2. P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, *Mixed-integer nonlinear optimization*, Acta Numerica, 22 (2013), pp. 1–131.
3. H. Ben Amor, J. Desrosiers, and A. Frangioni, *On the Choice of Explicit Stabilizing Terms in Column Generation*, Discrete Applied Mathematics, 157 (2009), pp. 1167–1184.
4. P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. WäChter, *An algorithmic framework for convex mixed integer nonlinear programs*, Discret. Optim., 5 (2008), pp. 186–204.
5. P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter, *An algorithmic framework for convex mixed integer nonlinear programs*, Discrete Optimization, 5 (2008), pp. 186 – 204. In Memory of George B. Dantzig.
6. J. Currie and D. I. Wilson, *OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User*, in Foundations of Computer-Aided Process Operations, N. Sahinidis and J. Pinto, eds., Savannah, Georgia, USA, 8–11 January 2012.
7. C. DAmbrosio, A. Frangioni, L. Liberti, and A. Lodi, *On interval-subgradient and no-good cuts*, Operations Research Letters, 38 (2010), pp. 341 – 345.
8. W. de Oliveira and C. Sagastizábal, *Bundle methods in the* XXIst *century: A birds'-eye view*, Pesquisa Operacional, 34 (2014), pp. 647–670.
9. W. de Oliveira and M. Solodov, *A doubly stabilized bundle method for nonsmooth convex optimization*, tech. report, 2013. Available at url = http://www.optimization-online.org/DB-HTML/2013/04/3828.html.
10. M. Duran and I. E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, 36 (1986), pp. 307–339.
11. V.-P. Eronen, M. M. Makela, and T. Westerlund, *On the generalization of* ECP *and* OA *methods to nonsmooth convex* MINLP *problems*, Optimization, 63 (2014), pp. 1057–1073.

12. R. Fletcher and S. Leyffer, *Solving mixed integer nonlinear programs by outer approximation*, Mathematical Programming, 66 (1994), pp. 327–349.
13. A. Frangioni and C. Gentile, *Perspective cuts for a class of convex 0-1 mixed integer programs*, Mathematical Programming, 106 (2006), pp. 225–236.
14. A. Geoffrion, *Generalized benders decomposition*, Journal of Optimization Theory and Applications, 10 (1972), pp. 237–260.
15. I. E. Grossmann, *Review of nonlinear mixed-integer and disjunctive programming techniques*, Optimization and Engineering, 3 (2002), pp. 227–252.
16. R. Hemmecke, M. Kppe, J. Lee, and R. Weismantel, *Nonlinear integer programming*, in 50 Years of Integer Programming 1958-2008, M. Jnger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, eds., Springer Berlin Heidelberg, 2010, pp. 561–618.
17. J. Kelley, Jr., *The cutting-plane method for solving convex programs*, Journal of the Society for Industrial and Applied Mathematics, 8 (1960), pp. 703–712.
18. K. Kiwiel and C. Lemaréchal, *An inexact bundle variant suited to column generation*, Math. Program., 118 (2009), pp. 177–206.
19. C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, *New variants of bundle methods*, Math. Program., 69 (1995), pp. 111–147.
20. S. Leyffer, *Integrating* sqp *and branch-and-bound for mixed integer nonlinear programming*, Computational Optimization and Applications, 18 (1998), pp. 295–309.
21. M. Lubin, K. Martin, C. G. Petra, and B. Sandk, *On parallelizing dual decomposition in stochastic integer programming*, Operations Research Letters, 41 (2013), pp. 252 – 258.
22. P. Munari and J. Gondzio, *Using the primal-dual interior point algorithm within the branch-price-and-cut method*, Computers & Operations Research, 40 (2013), pp. 2026 – 2036.
23. I. Quesada and I. Grossmann, *An* lp/nlp *based branch and bound algorithm for convex* minlp *optimization problems*, Computers & Chemical Engineering, 16 (1992), pp. 937 – 947.
24. C. Sagastizábal, *Divide to conquer: decomposition methods for energy optimization*, Mathematical Programming, 134 (2012), pp. 187–222.
25. P. Schütz, A. Tomasgard, and S. Ahmed, *Supply chain design under uncertainty using sample average approximation and dual decomposition*, European Journal of Operational Research, 199 (2009), pp. 409 – 419.
26. R. A. Stubbs and S. Mehrotra, *A branch-and-cut method for 0-1 mixed convex programming*, Mathematical Programming, 86 (1999), pp. 515–532.
27. W. van Ackooij and W. de Oliveira, *Level bundle methods for constrained convex optimization with various oracles*, Computational Optimization and Applications, 57 (2014), pp. 555–597.
28. T. Westerlund and K. Lundqvist, *Alpha-*ecp*, version 5.101: An interactive* minlp*-solver based on the extended cutting plane method*, Tech. Report 01-178-A, Process Design Laboratory at Abo Akademi University, 2005. Updated version of 2005-10-21. Available at `http://www.abo.fi/~twesterl/A-ECPManual.pdf`.
29. T. Westerlund and F. Pettersson, *An extended cutting plane method for solving convex* minlp *problems*, Computers & Chemical Engineering, 19, Supplement 1 (1995), pp. 131 – 136. European Symposium on Computer Aided Process Engineering.
30. T. Westerlund and R. Pörn, *Solving pseudo-convex mixed integer optimization problems by cutting plane techniques*, Optimization and Engineering, 3 (2002), pp. 253–280.