Computational Optimization and Applications manuscript No.
(will be inserted by the editor)

Comparative study of RPSALG algorithm for convex semi-infinite programming

**A. Auslender · A. Ferrer · M.A. Goberna ·
M.A. López**

the date of receipt and acceptance should be inserted later

Abstract The Remez penalty and smoothing algorithm (RPSALG) is a unified framework for penalty and smoothing methods for solving min-max convex semi-infinite programing problems, whose convergence was analyzed in a previous paper of three of the authors. In this paper we consider a partial implementation of RPSALG for solving ordinary convex semi-infinite programming problems. Each iteration of RPSALG involves two types of auxiliary optimization problems: the first one consists of obtaining an approximate solution of some discretized convex problem, while the second one requires to solve a non-convex optimization problem involving the parametric constraints as objective function with the parameter as variable. In this paper we tackle the latter problem with a variant of the cutting angle method called ECAM, a global optimization procedure for solving Lipschitz programming problems. We implement different variants of RPSALG which are compared with the unique publicly available SIP solver, NSIPS, on a battery of test problems.

Keywords convex semi-infinite programming - Remez-type methods - penalty methods - smoothing methods - cutting angle method

This research was partially supported by MINECO of Spain, Grants MTM2011-29064-C03-01/02.

Institut Camille Jordan, Université de Lyon, CNRS, and Department of Economics, Ecole Polytechnique, France, email: auslender.alfred@gmail.com

Departament de Matemàtica Aplicada I, Universitat Politècnica de Catalunya, Spain, email: alberto.ferrer@upc.ed

Department of Statistics and Operations Research, University of Alicante, Spain, email: mgoberna@ua.es

Department of Statistics and Operations Research, University of Alicante, Spain, email: marco.antonio@ua.es

1 Introduction

Ordinary convex SIP problems arise in a natural way in a variety of fields, such as finance [31], controller design problems [23], sensor selection [22], system identification [24], Chebyshev systems [16], convex geometry [19] or probability distributions [15], among others.

A Remez penalty and smoothing algorithm (RPSALG in short) was proposed in [1] to solve min-max convex semi-infinite programming (SIP) problems of the form

$$(P_0)\ F_* := \inf\{F(x) :\ x \in C\}, \tag{1}$$

where the objective function is $F(x) := \sup\{f_t(x) : t \in T_1\}$, the feasible set is $C := Q \cap D$, with Q being a fixed closed convex subset of $\mathbb{R}^n$ and $D := \{x : G(x) \leq 0\}$, $G(x) := \sup\{g_t(x) : t \in T_2\}$, T_1 and T_2 are compact metric spaces, and $f : T_1 \times \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ and $g : T_2 \times \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ are finite and continuous functions on $T_1 \times Q$ and $T_2 \times Q$, respectively, and such that for each t the functions $f_t(\cdot) := f(t, \cdot)$ and $g_t(\cdot) := g(t, \cdot)$ are lower semicontinuous and convex on $\mathbb{R}^n$, and at least $\mathcal{C}^1$ on Q. In this general version of RPSALG, the objective function $F(x)$ is smoothed and the constraint function is replaced with a penalty function involving finitely many constraints g_t. In the article we confine ourselves to consider ordinary convex SIP problems in which $Q = \mathbb{R}^n$ and T_1 is a singleton set. Then, the convex semi-infinite programming problem considered here can be described in the form:

$$(P)\ f_* = \inf_{x \in C}\{f(x) :\ g(t,x) \leq 0,\ t \in T\}, \tag{2}$$

where T is a compact metric space, $f : \mathbb{R}^n \to \mathbb{R}$ is convex on $\mathbb{R}^n$ and level bounded on the feasible set $C := \{x \in \mathbb{R}^n : G(x) \leq 0\}$, with $G(x) := \max\{g_t(x) : t \in T\}$, $g : T \times \mathbb{R}^n \to \mathbb{R}$ is continuous, and the constraint functions g_t are convex on $\mathbb{R}^n$ for all $t \in T$. Moreover, the involved functions, f and g_t, $t \in T$, are assumed to be $\mathcal{C}^1$. We also consider problems with constraints in blocks (also called parametric constraints), i.e. convex SIP problems where the feasible set is the intersection of finitely many sets of the form $\{x \in \mathbb{R}^n : g(t,x) \leq 0,\ t \in T\}$, with T and g as above. We say that the convex SIP problem (P) satisfies the Slater condition whenever there exists $\widehat{x} \in \mathbb{R}^n$ such that $g(t, \widehat{x}) < 0$ for all $t \in T$. This is a stability condition for (P) in the sense that sufficiently small perturbations of the constraints preserve the feasibility of the problem [13, Theorem 5.1].

Our version of RPSALG is a particular case of the unified framework described in [1], inspired by the first algorithm of Remez [29], which was proposed for approximating functions in the framework of linear SIP. The basic Remez's algorithm for solving (2) is described in Table 1, in which the step $S1$ consists of computing a minimizer x^{k+1} of the ordinary convex program (P_k) obtained by replacing in (2) the index set T by a grid T^k, while step $S2$ provides the index t^{k+1} of a most violated constraint at x^{k+1}. In practice, x^{k+1} is an approximate solution of (P_k) while t^{k+1} is the index of some constraint sufficiently violated by x^{k+1}. The approximate optimal solution $t^{k+1} \in T$ obtained in step $S2$ is aggregated to T^k for the next iteration.

Table 1 REMEZ general framework

```
Procedure: REMEZ
Initialization: determine $T^0$ and $x^0$; $k := 0$; non_stop:=true (binary);
begin
  while (non_stop) do
    $S1$ : Solve $(P_k)$ $f(x^{k+1}) = \min\{f(x) : g(t,x) \leq 0, t \in T^k\}$;
    $S2$ : Solve $g(t^{k+1}, x^{k+1}) = \max\{g(t, x^{k+1}) : t \in T\}$;
    $S3$ : $T^{k+1} := T^k \cup \{t^{k+1}\}$;
    $S4$ : $k := k+1$;
    $S5$ : If the stopping condition is satisfied then, non_stop:=false;
  endwhile
  return bestSolution $x^k$;
end
```

Accordingly, RPSALG is structured as the basic Remez's algorithm, but replacing the constrained convex program (P_k) in step $S1$ by the minimization without constraints of the regularized convex program

$$\min\{H_k(x) + \varphi_k(x) : x \in \mathbb{R}^n\}, \tag{3}$$

where φ_k is a suitable regularizing convex function guaranteeing the strong convexity of the objective function of (3), and H_k is the corresponding merit function,

$$H_k(x) := f(x) + G_k(x), \tag{4}$$

with $G_k(x)$ defined as

$$G_k(x) := \frac{\gamma_k}{|T^k|} \sum_{t \in T^k} \frac{\theta(g(t,x)\delta_k)}{\delta_k}, \tag{5}$$

where θ belongs to some family of penalty functions, and $\{\gamma_k\}$ and $\{\delta_k\}$ are appropriated sequences of positive scalars; $G_k(x)$ is a penalty function for approaching of the feasible set of the discrete subproblem (P_k), i.e. $\{x \in \mathbb{R}^n : g(t,x) \leq 0, t \in T^k\}$. RPSALG algorithm for solving (2) is described in Table 2, where $\{\epsilon_k\}$ and $\{\mu_k\}$ denote two sequences of positive tolerances such that $\epsilon_k \downarrow 0$ and $\mu_k \downarrow 0$.

Table 2 RPSALG general framework

```
Procedure: RPSALG
Initialization: determine $T^0$ and $x^0$; $k := 0$; non_stop:=true (binary);
begin
  while (non_stop) do
    $S1$ : Solve $H_k(x^{k+1}) + \varphi_k(x^{k+1}) \leq \min\{H_k(x) + \varphi_k(x)\} + \epsilon_k$;
    $S2$ : Solve $g(t^{k+1}, x^{k+1}) \geq \max\{g(t, x^{k+1}) : t \in T\} - \mu_k$,
    $S3$ : $T^{k+1} := T^k \cup \{t^{k+1}\}$;
    $S4$ : $k := k+1$;
    $S5$ : If the stopping condition is satisfied then, non_stop:=false;
  endwhile
  return bestSolution $x^k$;
end
```

Unfortunately there is not much software for SIP. In this paper we compare RPSALG with NSIPS, the unique solver publicly available so far for solving SIP problems. NSIPS is a set of solvers for semi-infinite programming problems designed without assumptions of convexity. NSIPS uses the SIPAMPL software package, which allows the codification of semi-infinite programming problems in AMPL and includes a database with a large battery of coded SIP problems (see [32] and the SIPAMPL manual[1] for additional information). NSIPS is publicly available on the NEOS server platform[2]. NSIPS includes four solvers: a discretization solver, a penalty technique solver, a sequential quadratic programming solver (SQP), and an infeasible quasi-Newton interior point solver. Some of them need to use commercial software NPSOL [18].

Of all the solvers included in NSIPS only the penalty technique solvers are considered in the article. Penalty methods include two versions based on a quasi-Newton method applied to penalty functions. The first method solves the unconstrained problem, and it is based on penalty functions (several penalty functions can be selected), and no reference to Lagrange multipliers is made. The second one solves the unconstrained problem using two possible options, namely an Augmented Lagrangian penalty function or a multiplier penalty function.

The paper is organized as follows. Section 2 describes different versions of RPSALG for problems with a unique block of constraints. Section 3 analyzes, from a computational efficiency point of view, implementations of RPSALG based on optimal gradient algorithms and variable metric schemes. Section 4 proposes stopping rules for both auxiliary optimization subproblems. Section 5 adapts RPSALG to problems with constraints in blocks. Section 6 compares the numerical results obtained for two particular implementations of RPSALG, and for two particular NSIPS solvers on a large collection of test problems. Finally, Section 7 provides some conclusions.

2 Versions of RPSALG

The implementation of RPSALG for the problem (P) formulated in (2) depends on the optimization algorithms used in steps $S1$ and $S2$ (see Table 2), and also on the regularizing convex function φ, the penalty function θ, and the couple of sequences of positive scalars $\{\gamma_k\}$ and $\{\delta_k\}$. Notice that for each choice of these parameters we have a different instance of RPSALG.

Thus, once a standard algorithm has been chosen for solving $S1$ (e.g., a Gradient-type, a Newton-type, or a Quasi-Newton-type algorithm), the following question arises: how to solve efficiently the non-convex program in step $S2$ when either the dimension of T is greater than one or the constraint functions are non-standard? A sensible answer to this question consists of using the so-called Extended Cutting Angle Method (ECAM), a global optimization procedure for Lipschitz programs that allows us to solve the subproblems $S2$ regardless of the dimension of T. To the authors' knowledge, the use in this article of global optimization software, such as ECAM, to solve the non-convex program at the step $S2$ in algorithms based on Remez's approximation is an innovation in the field.

[1] http://plato.la.asu.edu/ftp/sipampl.pdf

[2] http://www.neos-server.org/neos/

2.1 The choice of the regularizing convex function φ

The most relevant choice concerns the regularizing convex function φ, as it determines the convergence behavior of the corresponding variants of our method. In this paper we consider two versions of RPSALG, named RPSALG1 and RPSALG2, that use different regularizing convex functions, φ^1 and φ^2, guaranteeing the strong convexity of the objective function in the unconstrained convex problem (3). Consider the regularizing functions $\varphi^1, \varphi^2 : \mathbb{R}^n \to \mathbb{R}$ defined by

$$\varphi^1(x) = \epsilon \|x\|^2, \text{ for } \epsilon > 0, \text{ and } \varphi^2(x) = \frac{1}{2}\|x - \overline{x}\|^2, \text{ for } \overline{x} \in \mathbb{R}^n,$$

i.e. the Tihonov and the Moreau-Yosida regularizing functions, respectively.

In RPSALG1 we associate with a given positive sequence $\{\epsilon_k\}$ such that $\epsilon_k \searrow 0$ the sequence of regularized subproblems

$$(P_k^1) \quad \min\{H_k(x) + \varphi_k^1(x) \ : \ x \in \mathbb{R}^n\}, \quad k = 1, 2, ...,$$

where $\varphi_k^1(x) := \epsilon_k \|x\|^2$, $k = 1, 2, ...$

In RPSALG2 we consider the sequence of regularized subproblems

$$(P_k^2) \min\{H_k(x) + \varphi_k^2(x) : \ x \in \mathbb{R}^n\}, \quad k = 1, 2, ..., \tag{6}$$

where $\varphi_k^2(x) := \frac{1}{2}\left\|x - x^{k-1}\right\|^2$, $k = 1, 2, ...$, with x^{k-1} denoting an approximate solution of (P_{k-1}^2).

2.2 The choice of the penalty function θ

In order to guarantee the convergence of RPSALG, the penalty function $\theta : \mathbb{R} \to \mathbb{R}_+$ in (5) is required to be $\mathcal{C}^1$, convex, non-decreasing, non-constant and with $\lim_{u\to-\infty} \theta(u) = 0$. These conditions are satisfied by well-known penalty functions as the following:

$$\theta_1(u) = \log(1 + \exp(u)),$$

$$\theta_2(u) = 2^{-1}(u + \sqrt{u^2 + 4}),$$

$$\theta_3(u) = \begin{cases} 0, & u \leq -1, \\ \frac{1}{4}(u+1)^2, & -1 < u < 1, \\ u, & u \geq 1, \end{cases}$$

$$\theta_4(u) = \frac{1}{2}(u^+)^2,$$

where $u^+ := \max\{u, 0\}$,

$$\theta_5(u) = (u^+)^3,$$

and

$$\theta_6(u) = \exp(u).$$

The assumptions on θ entail $\theta_\infty(-1) = 0$ and $\theta_\infty(1) > 0$, where θ_∞ denotes the asymptotic function of θ, i.e. $\text{epi}(\theta_\infty) = (\text{epi}\,\theta)_\infty$ (the so-called recession cone of the epigraph of θ).

2.3 The choice of the positive sequences $\{\gamma_k\}$ and $\{\delta_k\}$

Once the regularizing function has been fixed, each triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ in the expression

$$G_k(x) = \frac{\gamma_k}{|T^k|} \sum_{t \in T^k} \frac{\theta(g(t,x)\delta_k)}{\delta_k}$$

determines a different instance of RPSALG. To ensure convergence, we consider the following conditions involving a sequence of integer numbers $\{m_k\}$ such that $m_k \geq |T^k|,\ k = 1, 2, \dots$:

(a) $\theta_\infty(1) < +\infty$, $\lim_{k\to\infty} \gamma_k/\delta_k = 0$, and $\lim_{k\to\infty} \gamma_k/m_k = +\infty$.
(b) $\theta_\infty(1) = +\infty$, $\lim_{k\to\infty} \gamma_k/\delta_k = 0$, and $\gamma_k/m_k > \varepsilon\ \forall k$ and a certain $\varepsilon > 0$.
(c) $\theta_\infty(1) = +\infty$, $\lim_{k\to\infty} \delta_k = +\infty$, $\gamma_k/m_k > \varepsilon\ \forall k$ and a certain $\varepsilon > 0$, $\{\gamma_k/\delta_k\}$ is bounded, and $\theta(0) = 0$ or the Slater condition holds.

Observe that the three conditions (a), (b) and (c) imply

$$\lim_{k\to\infty} \gamma_k = \lim_{k\to\infty} \delta_k = +\infty.$$

The convergence of RPSALG1 derives from the following result (for RPSALG2 no counterpart is still available):

Theorem 1 *[1, Theorem 3.1] If the triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ satisfies at least one of the conditions (a), (b), (c), then the sequence $\left\{x^k\right\}$ built by RPSALG1 is bounded and each limit point of this sequence is an optimal solution of (P).*

With respect to the choice of the sequences $\{\gamma_k\}$ and $\{\delta_k\}$ three cases are considered in our implementation. If we take $m_k := |T^0| + k$, we can verify the following statements:

i) $\gamma_k := (m_k)^{1.5}$ and $\delta_k := (m_k)^{2.5}$ satisfy (a), (b) and (c). Then, any triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ with $\theta \in \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ can be used.
ii) $\gamma_k := m_k$ and $\delta_k := (m_k)^{1.5}$ satisfy (b) and (c). Then, any triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ with $\theta \in \{\theta_4, \theta_5, \theta_6\}$ can be used.
iii) $\gamma_k = \delta_k = m_k$ satisfy (c). Then, any triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ with $\theta \in \{\theta_4, \theta_5\}$, or $\theta = \theta_6$ together the Slater condition can be used. Nevertheless, Slater condition will not be taken into account in our analysis because it is difficult to be checked.

3 Implementing RPSALG

As we have seen, each triplet $(\theta, \{\gamma_k\}, \{\delta_k\})$ determines a different instance of RPSALG. In this section, we compare the implementations corresponding to cases i), ii) and iii), in which RPSALG1 converges. Nevertheless, some considerations must be taken into account. Indeed, the standardization of floating point arithmetics follows the IEEE 754 standard. This standard has some major shortcomings. One of them is that it does not specify the behavior of standard transcendental functions so as the exponential function. As J.M. Muller states in [26], some transcendental functions are even very badly implemented in common run-time libraries that produce wrong results on some arguments. Thus, the defective results generated could seriously affect the numerical stability of the implementations. In addition, the lack of an exact definition of the results to be returned by standard libraries prohibits portability over different platforms. These drawbacks suggest not to use the penalization functions θ_1 and θ_6 in our implementation of RPSALG. However, in order to emphasize the previous comments on numerical instability we have included the results of the function θ_1 in Table 3, where we can see that it has the worst computational behavior from the point of view of successfully completing the program run. On the other hand, penalty functions θ_4 and θ_5 have a similar performance. For this reason, since our purpose is to show the best penalty function, in Table 3 we only compare θ_4 with θ_i, $i = 1, 2, 3$. Tests with θ_5 and θ_i, $i = 1, 2, 3$, have shown similar results on RPSALG.

3.1 Choosing an efficient version of RPSALG

The benchmark results are generated by running RPSALG1 and RPSALG2 on the set of test problems described in [21]. Then, we report information of interest for each instance from its performance profile (see Appendix for a summary or [14] for complete description) with respect the number of function evaluations and the CPU-time. From Table 3 we can compare the instances with the probability of win over the rest, *Best*, and the probability of success, *Success*, in solving the test problems with respect the number of function evaluations, *nfeval*, and the CPU-time, *time*.

Table 3 Performance profile results for instance evaluations

	RPSALG1				RPSALG2			
	nfeval		*time*		*nfeval*		*time*	
Instance	*Best*	*Success*	*Best*	*Success*	*Best*	*Success*	*Best*	*Success*
i) with θ_1	7.1%	93.1%	7.1%	93.1%	7.1%	71.5%	7.1%	71.5%
i) with θ_2	7.1%	100%	7.1%	100%	7.1%	86%	7.1%	86%
i) with θ_3	7.1%	100%	0%	100%	7.1%	78.6%	0%	78.6%
i) with θ_4	93.1%	100%	100%	100%	71.5%	78.6%	78.6%	78.6%
ii) with θ_4	7.1%	100%	7.1%	100%	7.1%	86%	0%	86%
iii) with θ_4	7.1%	100%	0%	100%	7.1%	86%	0%	86%

As we can see, the case i) with function θ_4 is the best for all options. It requires less functions evaluations in the 93.1% of the cases with RPSALG1 (with an 100% of success), and in the 71.5% of the cases with RPSALG2 (78.5% of success). Moreover, it is the best option for CPU-time since it spends less time in the 100% of the cases with RPSALG1 (100% of success), and in the 78.6% of the cases with RPSALG2 (78.6% of success). So, we shall use the case i) with function θ_4 in all RPSALG implementations.

3.2 Building a starting grid T^0

As explained in Section 1, we confine ourselves to consider ordinary convex SIP problems of the form (2) such that T is a compact metric space, the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is convex on $\mathbb{R}^n$ and level bounded on $C = \{x \in \mathbb{R}^n : G(x) \leq 0\}$, $g : T \times \mathbb{R}^n \to \mathbb{R}$ is continuous, and the constraint functions g_t are convex on $\mathbb{R}^n$ for all $t \in T$; we also assume that the involved functions, f and g_t, $t \in T$, are $\mathcal{C}^1$. These assumptions guarantee that the optimal set of (2) is nonempty and compact (by the same argument as [1, Prop. 2.1]). Moreover, by [1, Lemma 3.1], there exists a finite nonempty subset $T^0 \subset T$ such that f is level bounded on $C^0 := \{x \in \mathbb{R}^n : G^0(x) \leq 0\}$, with $G^0(x) := \max\{g_t(x) : t \in T^0\}$. There are some particular cases in which the set T^0 is easily obtainable. For instance, when $T = \operatorname{cl}\operatorname{int} T \subset \mathbb{R}^m$ and $\beta_r \searrow 0$, since $\operatorname{dist}(T \cap \beta_r \mathbb{Z}^m, T) \to 0$, it is possible to take the regular grid $T^0 = T \cap \beta_r \mathbb{Z}^m$ for sufficiently large r (see [1, Remark 3.1]).

When T is either a full dimensional closed convex sets or the finite union of pairwise disjoint sets of this class (typically a box or the union of finitely many disjoint boxes, as it happens in almost all test problems), then $T = \operatorname{cl}\operatorname{int} T$ by the accessibility lemma.

When T has a finite number of isolated elements (indices), then they must be included in T^0. So, it is easy to get a starting grid T^0 whenever T is the union of a finite set with finitely many pairwise disjoint boxes.

3.3 Solving the programs at step $S1$

The aim of this subsection is to discuss the optimization methods allowing to solve efficiently the subproblems

$$(P_k^1) \quad \inf\{H_k(x) + \epsilon_k \|x\|^2 \ : \ x \in \mathbb{R}^n\}, \quad k = 1, 2, ...,$$

and

$$(P_k^2) \quad \inf\{H_k(x) + \frac{1}{2} \left\| x - x^{k-1} \right\|^2 \ : \ x \in \mathbb{R}^n\}, \quad k = 1, 2, ...,$$

where $H_k(x) = f(x) + G_k(x)$. These problems have the common form

$$\inf\{f(x) : \ x \in Q\},$$

where f is a strongly convex objective function $\mathcal{C}^1$ on $Q = \mathbb{R}^n$. When the number n of variables is too large we cannot use Newton type methods but only gradient based methods. We summarize now, very shortly, the accelerating gradient methods based on Nesterov's ideas. Let Q be a closed convex set in $\mathbb{R}^n$ and let $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ be a proper lower semicontinuous convex function, $\mathcal{C}^1$ on Q. We suppose the existence of a global minimizer x^* of f on Q and that ∇f is globally Lipschitz on Q with Lipschitz constant L. This constant must be known since it is used in the construction of Nesterov-type method. More precisely, if $\{x^k\}$ is a sequence given by such an algorithm that we shall denote OGA (optimal gradient algorithm), then there exists a constant $D(x^*, x^0)$, depending on x^* and the starting point x^0 such that

$$f(x^k) - f(x^*) \leq LD(x^*, x^0) = \mathcal{O}(1/k^2). \tag{7}$$

Furthermore, Nesterov [27] has shown that this estimate is "optimal" for the class of convex $\mathcal{C}^1$ functions for which the gradient is globally Lipschitz (this last assumption is essential). It is worthwhile to note that Q must be "simple" in the following sense: all the formulas in OGA are given by analytic formulas, without any subroutine for solving a minimization subproblem, so that (7) is really a complexity estimation. As examples of "simple" sets Q we have Euclidean balls, affine sets, half-spaces, box constrained sets, simplex sets, etc. This kind of "optimal methods" have been extended with different versions to constrained optimization independently by Nesterov [27] and by Auslender and Teboulle [2] for "simple" feasible sets.

Since the comparative study tackled in this paper requires to solve (P_k^1) and (P_k^2) for the convex SIP problems collected at the SIPAMPL database, where $Q = \mathbb{R}^n$, n is small, and the objective functions are very general (so that it is not possible to give analytic formulas), OGA methods are not so advantageous in this framework. We illustrate this sentence analyzing the particular case of linearly constrained convex SIP problems, i.e. problems as (P) in (2) with $g(t, x) = \langle a_t, x\rangle - b_t$, with $a_t \in \mathbb{R}^n$ and $b_t \in \mathbb{R}$, for all $t \in T$. The continuity of g on $T \times \mathbb{R}^n$ entails that $t \mapsto a_t$ is continuous on the compact set T, so that

$$\mu := \sup_{t\in T} \|a_t\|$$

is attained.

Proposition 1 *Let $\theta : \mathbb{R} \to \mathbb{R}_+$ be a $\mathcal{C}^1$, convex, non-decreasing, non-constant function such that $\lim_{u\to-\infty} \theta(u) = 0$ and θ' is globally Lipschitz on $\mathbb{R}$ with constant α. Assume that L_0 is a Lipschitz constant for ∇f. Then $\nabla\left(H_k + \varphi_k^1\right)$ and $\nabla\left(H_k + \varphi_k^2\right)$, are globally Lipschitz with constants L_1 and L_2 given by*

$$L_1^k = L_0 + 2\epsilon_k + \alpha\gamma_k\delta_k\mu^2 \text{ and } L_2^k = L_0 + 1 + \alpha\gamma_k\delta_k\mu^2,$$

respectively.

Proof: Let θ be as above, with Lipschitz constant α. Given an affine function $h(x) = \delta(\langle a, x\rangle - b)$, with $\delta \geq 0$, $a \in \mathbb{R}^n$, and $b \in \mathbb{R}$, we have $\nabla\theta(h(x)) = \delta\theta'(h(x))a$. Thus, for any two points $x, y \in \mathbb{R}^n$, one has

$$\begin{aligned}\|\nabla\theta(h(x)) - \nabla\theta(h(y))\| &= \delta\|a\|\left|\theta'(h(x)) - \theta'(h(y))\right| \\ &\leq \alpha\delta\|a\|\|h(x) - h(y)\| \\ &\leq \alpha\delta^2\|a\|^2\|x - y\|,\end{aligned}$$

so that $\nabla(\theta \circ h)$ is globally Lipschitz on $\mathbb{R}^n$ with constant $\alpha\delta^2\|a\|^2$. Hence,

$$\nabla G_k(x) = \frac{\gamma_k}{|T_k|}\frac{\sum_{t\in T_k}\nabla\theta(\delta_k g(t, x))}{\delta_k}$$

is globally Lipschitz too, with Lipschitz constant $\frac{\alpha\gamma_k\delta_k}{|T_k|}\sum_{t\in T_k}\|a_t\|^2 \leq \alpha\gamma_k\delta_k\mu^2$. Thus, $\nabla H_k(x) = \nabla f(x) + \nabla G_k(x)$ is globally Lipschitz with Lipschitz constant $L_0 + \alpha\gamma_k\delta_k\mu^2$. $\square$

The penalty functions θ_1 and θ_2 satisfy the assumptions of Proposition 1 because they are $\mathcal{C}^2$ with $0 \leq \theta_1''(u) \leq \frac{1}{4}$ and $0 \leq \theta_2''(u) \leq \frac{1}{8}$ for all $u \in \mathbb{R}$. Also the non-$\mathcal{C}^2$ functions θ_3 and θ_4 satisfy the assumptions as their derivatives have Lipschitz constants equal to 2 and 1, respectively. Observe that the derivatives of θ_5 and θ_6 are not globally Lipschitz. Concerning the objective function f, ∇f is Lipschitz with Lipschitz constant $L_0 = 0$ whenever f is linear (i.e. in linear SIP).

Nevertheless, in practice, OGA is inconvenient for solving the problems considered because the product $\gamma_k \delta_k$ tends to infinity as the number of iterations increases so that $L_i^k \to +\infty$ as $k \to +\infty$, $i = 1, 2$. Indeed, in each iteration new points must be calculated through steps whose length depends on $1/L_i^k$, which tends to zero as $k \to +\infty$, this makes OGA increasingly slow and inefficient. This phenomenon is illustrated in the Tables 4 and 5, corresponding to Example 1 below. For this reason, we propose to use the Limited-memory Broyden-Fletcher-Goldfarb-Shanno method [28] to solve the subproblems in Step 1 of Table 2. This is a quasi-Newton method (denoted by QN in the sequel) for unconstrained optimization that iteratively finds a minimizer by approximating the inverse Hessian matrix using information from last iterations, which drastically saves the memory storage and computational time for large-scaled problems.[3]

Example 1 Consider the well-known test problem from linear semi-infinite literature consisting on computing the polynomial of degree less than n that best approximates the tangent curve over $[0, 1]$ in the L^1 norm whose exact solution is known (see [21]).

$$\begin{aligned} &\text{minimize} \quad f(x) = \textstyle\sum_{i=1}^{n} \frac{x_i}{i} \\ &\text{subject to: } -\textstyle\sum_{i=1}^{n} x_i t^{i-1} \leq -\tan t, \ \ t \in [0, \ 1]. \end{aligned}$$

We solve it for $n = 3$, taking the penalization function θ_2 and the positive sequences $\gamma_k := (|T^0| + k)^{1.5}$ and $\delta_k = (|T^0| + k)^{2.5}$.

We can observe the results in Table 4 and Table 5. The column *Method* in Table 4 indicates the use of QN or OGA methods, φ indicates the regularized function which has been used, *Iter* indicates the number of major iterations required, f_* represents the optimal value obtained by RPSALG, *f-eval* is the number of objective functions evaluations, *g-eval* is the number of gradient evaluations, (P^i)*-mean* is the average of the iterations performed at each major iteration in the problems (P_k^i), $i = 1, \ 2$ and *Time* is the CPU time in seconds. On the other hand, Table 5 represents the evaluation of the numbers γ_k, δ_k and $\gamma_k/|T|$ for the problem number 4 in Table 5, and *EPS* indicates the stopping criterion evaluation.

Table 4 Results and CPU time for the tangent sample with precision $\epsilon = 0.001$

Num	*Method*	φ	*Iter*	f^*	*f-eval*	*g-eval*	(P^i)*-mean*	*Time*
1	QN	φ^1	47	6.49994e-001	586	331	6	2.603
2	QN	φ^2	45	6.49923e-001	479	227	4	1.612
3	OGA	φ^1	47	6.50003e-001	0	83590	890	457.447
4	OGA	φ^2	46	6.50037e-001	0	76510	833	390.191

In Table 5 we can see the inconvenience of using OGA algorithm since the product $\gamma_k \delta_k$ tends to infinity as the number of iterations increases. In Table

[3] For additional information: http://www.chokkan.org/software/liblbfgs/

4 we can compare the results of the different versions of the RPSALG and the advantages of using a Quasi-Newton method for solving subproblems in Step 1 of Table 2. Also, we can see that the use of φ_2 as the regularizing function allows one to reduce the CPU time in comparison with the use of φ_1.

Table 5 $\{\gamma_k\}$ and $\{\delta_k\}$ for the problem 4 in Table 4

Iter	γ_k	δ_k	$\gamma_k/\lvert T\rvert$	*EPS*
0	22.63	181.02	3.23	7.92611
10	76.37	1374.62	4.49	0.01230
20	148.16	4148.54	5.49	0.00395
30	234.25	8901.41	6.33	0.00201
40	332.55	15962.58	7.08	0.00126
46	396.82	21428.14	7.49	0.00097

3.4 Solving the programs at step $S2$

The subproblems to be solved in Step 2 consist of finding the optimal set of $g(\cdot,x):T\to\mathbb{R}$ for some $x\in\mathbb{R}^n$. Assume that $g(\cdot,x)$ is $\mathcal{C}^1$ on T, for all $x\in\mathbb{R}^n$, and denote $\mu:=\max_{t\in T}\|\nabla_t g(t,x)\|\in\mathbb{R}$. Then, given $t_1,t_2\in T$, there exists $\lambda\in\,]0,1[$ such that

$$|g(t_1,x)-g(t_2,x)|=|\langle\nabla_t g((1-\lambda)\,t_1+\lambda t_2,x),(t_1-t_2)\rangle|\le\mu\,\|t_1-t_2\|\,,$$

so that $g(\cdot,x)$ is Lipschitz continuous on T with Lipschitz constant μ.

Assume further that T is a convex polyhedron (typically, it is a box). Then we have to find the solution set of a problem of the form

$$\inf\{f(x):x\in S\}\,, \tag{8}$$

where f is Lipschitz continuous and S is a convex polyhedron.

In all the implementations of RPSALG considered in this paper we use an extension of the Cutting Angle method of Bagirov and Rubinov [3], due to Beliakov ([6],[7],[8],[9],[10]) and called Extended Cutting Angle Method (ECAM in short), in order to solve this very hard optimization problem. In ECAM the objective function is assumed to be Lipschitz continuous and it is optimized by building a sequence of piecewise linear underestimates. ECAM is inspired in the classical Cutting Plane method by Kelley [25] and Cheney and Golstein [12] to solve linearly constrained convex programs of the form (8), where S is the solution set of a given linear system and $f:\mathbb{R}^n\to\mathbb{R}$ is convex. Since f is lower semicontinuous, it is the upper envelope of the set of all its affine minorants, i.e.

$$f=\sup\,\{h\ :\ h\text{ affine function, }h\le f\}. \tag{9}$$

Indeed, it is enough to consider in (9) the affine functions of the form $h(x)=f(z)+\langle u,x-z\rangle$, where $u\in\partial f\,(z)$ (the subdifferential of f at $z\in\mathbb{R}^n$), the graph of h being a hyperplane which supports the epigraph of f at $(z,f(z))$. Let $x^1,...,x^k\in S$

be given and consider the affine functions $h^j(x) = f(x^j) + \langle u^j, x - x^j \rangle$, for some $u^j \in \partial f\left(x^j\right)$, $j = 1, ..., k$. The function

$$f_k := \max_{j=1,\ldots,k} h^j \tag{10}$$

is a convex piecewise affine underestimate of the objective function f, in other words, a polyhedral convex minorant of f. The k-th iteration of the Cutting Plane method consists of computing an optimal solution x^{k+1} of the approximating problem $\inf\{f_k(x) : x \in S\}$ which results of replacing f with f_k in (8) or, equivalently, solving the linear programming problem in $\mathbb{R}^{n+1}$

$$\inf\left\{x_{n+1} : x \in S, x_{n+1} \geq h^j(x), j = 1, ..., k\right\}, \tag{11}$$

where $x = (x_1, ..., x_n)$. Then the next underestimate of f, $f_{k+1} := \max\left\{f_k, h^{k+1}\right\}$, is a more accurate approximation to f, and the method iterates.

The Generalized Cutting Plane method for (8), where $f : \mathbb{R}^n \to \mathbb{R}$ is now a nonconvex function while $S = \left\{x \in \mathbb{R}^n_+ : \sum_{i=1}^n x_i = 1\right\}$ is the unit simplex, follows the same script, except that the underestimate f_k is built using the so-called H-subgradients (see [30]) instead of ordinary subgradients, so that minimizing f_k on S is no longer a convex problem. The Cutting Angle method ([3],[4]), of which ECAM is a variant, is an efficient numerical method for minimizing the underestimates when f belongs to certain class of abstract convex functions. Assume that f is Lipschitz continuous with Lipschitz constant $M > 0$ and take a scalar $\gamma \geq M$. Let $x^1, ..., x^k \in S$ be given. For $j = 1, ..., k$, we define the *support vector* $l^j \in \mathbb{R}^n$ by

$$l_i^j := \frac{f(x^j)}{\gamma} - x_i^j,\ i = 1, \ldots, n, \tag{12}$$

and the *support function* h^j by

$$h^j(x) := \min_{i=1,\ldots,n} (f(x^j) - \gamma(x_i^j - x_i)) = \min_{i=1,\ldots,n} \gamma(l_i^j + x_i). \tag{13}$$

Since the functions h^j are concave piecewise affine underestimates of f (i.e. polyhedral concave minorants of f), the underestimate f_k defined in (10) is now a saw-tooth underestimate of f and its minimization becomes a hard problem as (11) is no longer a linear program. ECAM locates the set V^k of all local minima of the function f_k which, after sorting, yields the set of global minima of f_k (see [8] and [9] for additional information). A global minimum x^{k+1} of f_k is aggregated to the set $\left\{x^1, ..., x^k\right\}$ and the method iterates with $f_{k+1} := \max\left\{f_k, h^{k+1}\right\}$.

Remark 1 Notice that the transformation of variables

1) $\bar{x}_i = x_i - a_i$, $i = 1, \ldots, n$, $d = \sum_{i=1}^n (b_i - a_i)$ with $\bar{x}_i \geq 0$ and $\sum_{i=1}^n \bar{x}_i \leq d$
2) $z_i = \frac{\bar{x}_i}{d}$, $i = 1, \ldots, n$, $z_{n+1} = \sum_{i=1}^n z_i$,

allows us to substitute the program

$$\min\{f(x) : x \in [a, b]\}$$

by the following one:

$$\min\{g(z_1, \ldots, z_{n+1}) : (z_1, \ldots, z_{n+1}) \in S\},$$

where S denotes the unit simplex in $\mathbb{R}^{n+1}$.

4 Stopping rules

4.1 Stopping rule for programs at the step $S1$

Since H_k is $\mathcal{C}^1$, any usual convergent gradient method will provide the iterate x^k in a finite number of steps if suitable stopping rules are adopted.

The regularized objective functions

$$H_k^{reg_i}(x) := H_k(x) + \varphi_k^i(x),\ \ i = 1, 2;\ \ k = 1, 2, ...,$$

are strongly convex and so, they have a unique global minimizer y_i^k. According to $S1$ in Table 2, for each k, (P_k^i), $i = 1, 2$, has to be solved within the error ϵ_k, with $\epsilon_k \searrow 0$, i.e. x^k must satisfy

$$H_k^{reg_i}(x^k) \leq H_k^{reg_i}(x) + \epsilon_k, \quad \forall x \in \mathbb{R}^n,\ \ i = 1, 2;\ \ k = 1, 2, ... \tag{14}$$

Stopping rule for (P_k^1) in RPSALG1: According to [1, Remark 3.2], (14) will be satisfied, i.e.

$$H_k(x^k) + \epsilon_k \left\|x^k\right\|^2 \leq H_k(x) + \epsilon_k \left\|x\right\|^2 + \epsilon_k \quad \forall x \in \mathbb{R}^n. \tag{15}$$

provided that we use the stopping rule

$$\left\|\nabla H_k(x^k) + 2\epsilon_k x^k\right\| \leq \sqrt{2}\epsilon_k, \tag{16}$$

where

$$\nabla H_k(x^k) = \nabla f(x^k) + \frac{\gamma_k}{|T_k|} \sum_{t \in T_k} \theta'(\delta_k g_t(x^k))\nabla g_t(x^k)$$

is obtained by the chain rule.

Stopping rule for (P_k^2) in RPSALG2: Now (14) will be satisfied, i.e.

$$H_k(x^k) + \frac{1}{2}\left\|x^k - x^{k-1}\right\|^2 \leq H_k(x) + \frac{1}{2}\left\|x - x^{k-1}\right\|^2 + \epsilon_k \quad \forall x \in \mathbb{R}^n. \tag{17}$$

provided that we use the stopping rule

$$\left\|\nabla H_k(x^k) + x^k - x^{k-1}\right\| \leq \sqrt{\epsilon_k}. \tag{18}$$

In fact, the function $H_k^{reg_2}$ is strongly convex with modulus 1 [20, IV, Theorem 4.3.1], and applying (18) and [20, IV, Theorem 4.1.4] to the couple of points x^k and y_2^k where $\{y_2^k\} = \operatorname{argmin}_{\mathbb{R}^n} H_k^{reg_2}$ (and so, $\nabla H_k^{reg_2}(y_2^k) = 0_n$):

$$\begin{aligned}
\left\|x^k - y_2^k\right\|^2 &\leq \left\langle \nabla H_k^{reg_2}(x^k) - \nabla H_k^{reg_2}(y_2^k), x^k - y_2^k \right\rangle \\
&\leq \left\|\nabla H_k^{reg_2}(x^k) - \nabla H_k^{reg_2}(y_2^k)\right\| \left\|x^k - y_2^k\right\| \\
&= \left\|\nabla H_k^{reg_2}(x^k)\right\| \left\|x^k - y_2^k\right\| \\
&\leq \sqrt{\epsilon_k} \left\|x^k - y_2^k\right\|,
\end{aligned}$$

entailing $\left\|x^k - y_2^k\right\| \leq \sqrt{\epsilon_k}$. Moreover, by convexity,

$$\begin{aligned} H_k^{reg_2}(x^k) &\leq H_k^{reg_2}(y_2^k) - \left\langle \nabla H_k^{reg_2}(x^k), x^k - y_2^k \right\rangle \\ &\leq H_k^{reg_2}(y_2^k) + \left\|\nabla H_k^{reg_2}(x^k)\right\| \left\|x^k - y_2^k\right\| \\ &\leq H_k^{reg_2}(y_2^k) + \sqrt{\epsilon_k}\sqrt{\epsilon_k}, \end{aligned}$$

and we get (17).

Proposition 2 *Assume that the triple $(\theta, \{\gamma_k\}, \{\delta_k\})$ satisfies at least one of the conditions (a), (b), (c) in Theorem 1. Let ξ and η be given positive numbers (tolerances) and let $\left\{x^k\right\}$ be the sequence generated by RPSALG 1. Then*

$$\left\|\nabla H_k(x^k)\right\| \leq \xi \text{ and } G(x^k) \leq \eta \tag{19}$$

holds for some $k \in \mathbb{N}$.

Proof: We can assume without loss of generality (w.l.o.g., in short) that $\left\{x^k\right\}$ is convergent. It is sufficient to show that $\lim_{k\to\infty} \left\|\nabla H_k(x^k)\right\| = 0$ and $\lim_{k\to\infty} G(x^k) \leq 0$.

Let $x^* = \lim_{k\to\infty} x^k$. By Theorem 1 x^* is an optimal solution of (P). On the one hand, taking limits in (16) as $k \to \infty$, we get $\lim_{k\to\infty} \left\|\nabla H_k(x^k)\right\| = 0$. On the other hand, $G(x) = \max\{g_t(x) : t \in T\}$ is a convex finite-valued function, so that it is continuous and, so, $\lim_{k\to\infty} G(x^k) = G(x^*) \leq 0$. □

Remark 2 The proof of Theorem 1 does not make use of the differentiability of the functions g_t, $t \in T$ (see [1] for the details) Nevertheless, if g_t is not differentiable, the same may happen with G_k and H_k, so that the new iterate x^k could be non-approachable by gradient methods and the stopping rules (16) and (18) may not apply. So, convergence of RPSALG1 is conditioned to the fact that all the constraint functions are continuously differentiable at the elements of the sequence $\left\{x^k\right\}$ built by RPSALG1, a condition which cannot be checked a priori.

4.2 Global stopping rule for RPSALG1

Theorem 1 established the existence of a subsequence of iterates $\{x^k\}_{k\in K}$ such that $\lim_{k\in K,\ k\to\infty} x^k = x^*$, where x^* is optimal for problem (P) in (2). In [1], the Lagrangian dual of (P) is studied by considering the dual pair formed by:

a) $\mathcal{C}(T)$: the Banach space of real-valued continuous functions on T, equipped with the maximum norm

$$\|h\| = \max\{|h(t)| : t \in T\}.$$

b) $\mathcal{M}(T)$: the topological dual of $\mathcal{C}(T)$, i.e. the space of all the finite signed Borel measures on T, embedded with the total variation norm.

c) The pairing

$$\langle \sigma, h \rangle = \int_T h(t)\sigma(dt) \text{ with } \sigma \in \mathcal{M}(T) \text{ and } h \in \mathcal{C}(T).$$

In [1, Section 4], a sequence of discrete measures $\{\sigma^k\}_{k\in K}$ associated with $\{x^k\}_{k\in K}$ is introduced by means of the expression

$$\sigma^k := \frac{\gamma_k}{|T^k|} \sum_{t\in T^k} \theta'(g(t,x^k)\delta_k)\alpha_t, \tag{20}$$

where α_t is the Dirac measure concentrated at t, i.e. for any continuous function $h \in \mathcal{C}(T)$

$$\langle \alpha_t, h \rangle = h(t).$$

Assuming that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is convex on $\mathbb{R}^n$ and level bounded on the feasible set $C := \{x \in \mathbb{R}^n : G(x) \leq 0\}$, that the Slater constraint qualification holds, and that $\nabla g(.,.)$ exists and is continuous on $T \times \mathbb{R}^n$, Theorem 4.2 in [1] establishes the existence of a subsequence $\{\sigma^k\}_{k\in K'}$, $K' \subset K$, which is weak*-convergent to a measure σ^*. The measure σ^* is an optimal solution for the Lagrangian dual problem (D) given in [1, (47)] and satisfies

$$\left\langle \sigma^*, g(\cdot, x^*) \right\rangle = 0.$$

Then, applying for instance [11, Proposition 2.24(iii)], we have

$$\lim_{k\in K',\ k\to\infty} \langle \sigma^k, g(\cdot, x^k) \rangle = \left\langle \sigma^*, g(\cdot, x^*) \right\rangle = 0.$$

Inspired by this fact we can use the following global stopping rule:

$$\left| \langle \sigma^k, g(\cdot, x^k) \rangle \right| = \frac{\gamma_k}{|T^k|} \sum_{t\in T^k} \theta'(g(t,x^k)\delta_k) \left| g_t(x^k) \right| \leq \eta, \tag{21}$$

where $\eta > 0$ is a tolerance parameter.

In the particular case of linearly constrained convex SIP, i.e. $g(t,x) = \langle a(t), x \rangle - b(t)$, if we use $\theta(u) = \theta_4(u) = \frac{1}{2}(u^+)^2$, the stopping rule (21) becomes

$$\frac{\gamma_k \delta_k}{|T^k|} \sum_{t\in T^k} (\left[\left\langle a(t), x^k \right\rangle - b(t) \right]^+)^2 \leq \eta. \tag{22}$$

5 Adapting RPSALG to constraints in blocks

Some SIP problems arising in practice can be formulated as

$$(P_0)\ f_* = \inf\{f(x) :\ g_i(t,x) \leq 0,\ t \in T_i,\ i = 1, ..., m\}, \tag{23}$$

where T_i is a (possibly degenerate) compact interval in $\mathbb{R}^{d_i}$, $i = 1, ..., m$, $f : \mathbb{R}^n \to \mathbb{R}$ is convex on $\mathbb{R}^n$ and level bounded on the feasible set $C := \{x \in \mathbb{R}^n : G_i(x) \leq 0, i = 1, ..., m\}$, with $G_i(x) := \max\{g_i(t,x) : t \in T_i\}$. Assume that for each $i = 1, ..., m$, $g_i : T_i \times \mathbb{R}^n \to \mathbb{R}$ is continuous, and the functions $g_i(t, \cdot)$ are convex on $\mathbb{R}^n$ for all $t \in T_i$. Assume also that the involved functions, f and $g_i(t, \cdot)$, $t \in T$, are $\mathcal{C}^1$.

We shall now describe a procedure to reformulate (P_0), when T_i is a (possibly degenerate) interval for all $1, ..., m$, with a unique index set, in three steps which preserve the objective function f and the feasible set C.

Step 1: Embedding all index sets in the same space.

Let $d := \max\{d_i : i = 1, ..., m\}$ and $T_i = \prod_{j=1,...,d_i} \left[\alpha_j^i, \beta_j^i\right],\ \alpha_j^i \leq \beta_j^i,\ j = 1, ..., d_i$. If $d_i < d$, we define $\left[\alpha_j^i, \beta_j^i\right] := [0,1],\ j = d_i + 1, ..., d,\ \widetilde{T_i} = T_i \times [0,1]^{d-d_i}$ and $\widetilde{g}_i : \widetilde{T_i} \to \overline{\mathbb{R}}$ such that $\widetilde{g}_i(t,s,x) = g_i(t,x)$ for all $(t,s,x) \in T_i \times [0,1]^{d-d_i} \times \mathbb{R}^n$. Then, we can replace $g_i(t,x)$ in (P) by $\widetilde{g}_i(t,s,x)$, where $\widetilde{g}_i$ enjoys the same properties as g_i. At the end of Step 1, we have an optimization problem of the form

$$(P_1)\ f_* = \inf\{f(x) :\ g_i(t,x) \leq 0, t \in T_i, i = 1, ..., m\},$$

where all the index sets T_i have the same dimension.

Step 2: Unifying the index sets.

Assume that $T_i = \prod_{j=1,...,d} \left[\alpha_j^i, \beta_j^i\right],\ \alpha_j^i \leq \beta_j^i,\ i = 1, ..., m$. Given $i \in \{1, ..., m\}$ and $j \in \{1, ..., d\}$, define $h_j^i : \mathbb{R} \to \mathbb{R}$ such that $h_j^i(\lambda) = (1-\lambda)\alpha_j^i + \lambda\beta_j^i$. For $i \in \{1, ..., m\}$, let $h^i : \mathbb{R}^d \to \mathbb{R}^d$ be the affine mapping such that $h^i(\lambda_1, ..., \lambda_d) = \left(h_1^i(\lambda_1), ..., h_d^i(\lambda_d)\right)$. Since $T_i = h^i\left([0,1]^d\right)$, defining $\widetilde{g}_i(s,x) := g_i(h^i(s), x)$, we can replace the constraint system in (P) with $\{\ \widetilde{g}_i(s,x) \leq 0, s \in T, i = 1, ..., m\}$, where $T = [0,1]^d$ is the common index set of all blocks of constraints. So, Step 2 provides a reformulation of (P_1) of the form

$$(P_2)\ f_* = \inf\{f(x) :\ g_i(t,x) \leq 0,\ i = 1, ..., m,\ t \in T\},$$

whose constraint functions satisfy the same properties as those in the initial model (P).

For most SIP problems arising in functional approximation Steps 1-2 are usually unnecessary as the index sets T_i coincide.

Example 2 Consider the problem consisting of computing a best uniform approximation from above to a given function $h : [\alpha, \beta] \to \mathbb{R}$, $\alpha < \beta$, by means of polynomials of degree less than $n-1$, with $n > 2$, under the condition that they are non-decreasing and convex on $[\alpha, \beta]$. Since the unknown polynomial $\sum_{i=1}^{n-1} t^{i-1}x_i$ can be represented by its vector of coefficients $(x_1, ..., x_{n-1})$, denoting by x_n the uniform error bound, the problem to be solved is

$$(P_2)\ f_* = \inf\{f(x) :\ g_i(t,x) \leq 0,\ i = 1, ..., 5,\ t \in [\alpha, \beta]\},$$

where $x = (x_1, ..., x_n)$, $f(x) = x_n$, and the constraints are:

- ∘ Approximation from above: $g_1(t,x) = h(t) - \sum_{i=1}^{n-1} t^{i-1}x_i \leq 0$.
- ∘ Monotonicity: $g_2(t,x) = -\sum_{i=1}^{n-1} (i-1)\, t^{i-2}x_i \leq 0$.
- ∘ Convexity: $g_3(t,x) = \sum_{i=1}^{n-1} (i-1)(i-2)\, t^{i-3}x_i \leq 0$.
- ∘ x_n is a lower uniform error bound: $g_4(t,x) = -\sum_{i=1}^{n-1} t^{i-1}x_i - x_n + h(t) \leq 0$.
- ∘ x_n is an upper uniform error bound: $g_5(t,x) = \sum_{i=1}^{n-1} t^{i-1}x_i - x_n - h(t) \leq 0$.

Step 3: Reduction to a unique block.

Defining $g(t,\cdot) := \max\{g_i(t,\cdot), i = 1,...,m\}$, we get the following reformulation of (P_2) :

$$(P_3)\ f_* = \inf\{f(x) : g(t,x) \leq 0,\ t \in T\}.$$

Since $g_i : T \times \mathbb{R}^n \to \mathbb{R}$ is continuous and the functions $g_i(t,\cdot)$ are $\mathcal{C}^1$ and convex on $\mathbb{R}^n$ for all $t \in T$ and $i = 1,..,m$, (P_3) satisfies the same assumptions required to the problem (P) in (2), except the possible lack of differentiability of g_t at those points $x \in \mathbb{R}^n$ whose corresponding *set of active indices*, defined as

$$I_t(x) = \left\{ i \in 1,...,m : g_i(t,x) = \max_{j=1,...,m} g_j(t,x) \right\},$$

is not a singleton (i.e. such that $\max\{g_i(t,x), i = 1,...,m\}$ is attained at more than one index). Thus, the set

$$\bigcup_{1\leq i<j\leq m} \left\{ x \in \mathbb{R}^n : g_i(t,x) = g_j(t,x) \right\}. \tag{24}$$

contains all points where g_t fails to be $\mathcal{C}^1$. When solving (P_3) with RPSALG, the failure of the $\mathcal{C}^1$ property of g_t at x^k for some $t \in T^k$ entails the non-smoothness of the auxiliary problem (P_k^1), which should be solved with some subgradient method (instead of a gradient one). Fortunately, the subdifferential of H_k at x^k, $\partial H_k(x^k)$, can be easily computed through the closed formula (25), where conv X stands for the convex hull of X.

Proposition 3 *Let (P_0) be as in (23), the triple $(\theta, \{\gamma_k\}, \{\delta_k\})$ be as in Theorem 1, and H_k be defined as in (4) and (5). Then, given $x^k \in \mathbb{R}^n$,*

$$\partial H_k(x^k) = \nabla f(x^k) + \frac{\gamma_k}{|T^k|} \sum_{t\in T^k} \theta'\left(g_t\left(x^k\right)\right) \operatorname{conv}\left\{\nabla g_i(t,x^k) : i \in I_t\left(x^k\right)\right\}. \tag{25}$$

Proof. Let $t \in T$ be given. By [20, Theorem 4.3.1 and Corollary 4.3.2] one has

$$\begin{aligned} \partial(\theta \circ g_t)(x) &= \theta'(g_t(x))\,\partial g_t(x) \\ &= \theta'(g_t(x)) \operatorname{conv}\left(\bigcup_{i\in I_t(x)} \nabla g_i(t,x) \right). \end{aligned} \tag{26}$$

Observing that all functions in the right hand-side of the equation

$$H_k(x) = f(x) + \frac{\gamma_k}{|T^k|} \sum_{t\in T^k} \frac{\theta(g_t(x)\,\delta_k)}{\delta_k}$$

are finite-valued and convex, we can combine [20, Theorem 4.1.1] and (26) to obtain

$$\begin{aligned} \partial H_k(x^k) &= \nabla f(x^k) + \tfrac{\gamma_k}{|T^k|\delta_k} \textstyle\sum_{t\in T^k} \partial\theta(g(t,x^k)\delta_k) \\ &= \nabla f(x^k) + \tfrac{\gamma_k}{|T^k|} \textstyle\sum_{t\in T^k} \theta'\left(g_t\left(x^k\right)\right) \operatorname{conv}\left\{\nabla g_i(t,x^k) : i \in I_t\left(x^k\right)\right\}. \end{aligned}$$

□

The unique possible drawback of RPSALG applied to (P_3) is related with the stopping rule $\partial H_k(x^k) \cap \xi\mathbb{B} \neq \emptyset$ (where $\mathbb{B}$ denotes the closed unit ball in $\mathbb{R}^n$), the natural extension of $\left\|\nabla H_k(x^k)\right\| \leq \xi$, which does not guarantee finite termination.

If the constraints in (P_0) are linear and non-repeated (as in Example 2), the sets $\left\{x \in \mathbb{R}^n : g_i(t,x) = g_j(t,x)\right\}$, $i \neq j$, are hyperplanes and so the set in (24) is null. Actually, even though we may have $g_i(t,\cdot) = g_j(t,\cdot)$ on some set of positive measure in artificial examples, in most convex SIP applications the constraints of (P_3) are $\mathcal{C}^1$ almost everywhere for all $t \in T$. In that case, since the functions G_k and H_k are $\mathcal{C}^1$ except on some subset of

$$\bigcup_{t \in T^k} \bigcup_{1 \leq i < j \leq m} \left\{x \in \mathbb{R}^n : g_i(t,x) = g_j(t,x)\right\}$$

(union of $\left|T^k\right|$ null sets), these functions are $\mathcal{C}^1$ almost everywhere for all $t \in T$. So, one can expect the convergence of RPSALG applied to (P_3) provided at least one of the conditions (a), (b), (c) in Theorem 1 holds.

The convex SIP problem (P_0) with blocks formed by a unique constraint ($g_i(t,x) \leq 0$, such that $|T_i| = 1$) can be reformulated as (P_3). When T^1 is infinite while $|T_i| = 1$, $i = 2, .., m$, it can be convenient to replace (P_0) by a suitable approximating problem with $\mathcal{C}^1$ constraints. Indeed, let $\theta : \mathbb{R} \to \mathbb{R}$ be a non-decreasing $\mathcal{C}^1$ function such $\theta(u) = 0$ for all $u \leq 0$ and $\theta(u) > 0$ for all $u > 0$ (conditions satisfied by the penalty functions θ_4 and θ_5); choose "big" positive numbers $M_1, ..., M_m$ and consider the convex SIP approximating problem

$$(P_a) \; f_* = \inf\{f(x) + \sum_{i=2}^{m} M_i \theta(g_i(x)) : \; g_1(t,x) \leq 0, t \in T^1\},$$

where $g_i(x)$ stands for $g_i(t,x)$ as the latter function does not depend on t for $i = 2, ..., m$. Obviously, (P_a) has the same feasible set as (P_0) and satisfies all assumptions required to the problem (P) in (2).

Example 3 Consider the convex SIP problem (P_0) in $\mathbb{R}^2$ [32, page 49] with objective function $f(x) = \|x\|^2$ and constraints $g_1(t,x) = tx_1 + t^2 x_2 \leq 0, t \in [0,1]$, $g_2(t,x) = x_1 + x_2 - 10 \leq 0$ $(t = 2)$, and $g_3(t,x) = -x_1 - x_2 - 10 \leq 0$ $(t = 3)$. Despite the fact that $T_1 = [0,1]$, $T_2 = \{2\}$ and $T_3 = \{3\}$ have different dimensions, we can replace T_2 and T_3 by T_1, obtaining the following reformulation of (P_0) :

$$(P_3) \qquad f_* = \inf\{\|x\|^2 : g_t(x) \leq 0, t \in [0,1]\},$$

whose constraint function

$$g_t(x) = \max\left\{tx_1 + t^2 x_2, x_1 + x_2 - 10, -x_1 - x_2 - 10\right\}$$

is $\mathcal{C}^1$ except on the union of the straight lines $\left\{x \in \mathbb{R}^2 : (t+1)x_1 + \left(t^2+1\right)x_2 = -10\right\}$ and $\left\{x \in \mathbb{R}^2 : (t-1)x_1 + \left(t^2-1\right)x_2 = -10\right\}$ (a null set).
The alternative approach consists of taking two big numbers $M_1 > 0$ and $M_2 > 0$ and a function $\theta \in \{\theta_4, \theta_5\}$, and replacing (P_0) with

$$\inf\{\|x\|^2 + M_1\theta(x_1 + x_2 - 10) + M_2\theta(-x_1 - x_2 - 10) : tx_1 + t^2 x_2 \leq 0, t \in [0,1]\}.$$

Observe that the linear constraints could be replaced by the non-smooth convex constraint $|x_1 + x_2| \leq 10$, getting the simpler approximating problem

$$\inf\{\|x\|^2 + M\theta\left(|x_1 + x_2| - 10\right) : tx_1 + t^2x_2 \leq 0, t \in [0,1]\},$$

with $M > 0$. The disadvantage of the latter problem is the failure of the $\mathcal{C}^1$ property of the objective function on the parallel straight lines $x_1 + x_2 = \pm 10$.

6 Numerical results

In this section we present the results of numerical experiments to compare the two versions of RPSALG with the penalty solvers included in NSIPS.

6.1 Options to the Solvers

As we know RPSALG can solve convex semi-infinite programs of class $\mathcal{C}^1$, without any limitation on the number of parametric and non-parametric constraints, the initial guess, and the dimension of T. The public version of NSIPS without using the NPSOL commercial software [18] cannot start an instance unless an initial guess has been defined. Moreover, the index set of any parametric constraint is a one-dimensional interval. On the other hand, penalty methods in NSIPS include two versions based on a quasi-Newton method applied to penalty functions. The first method solves the unconstrained problem, based on penalty functions (that can be chosen on three options), where no reference to Lagrange multipliers is made. The second one, in which an estimation of the Lagrange multipliers is made, solves the unconstrained problem using two possible options: an Augmented Lagrangian penalty function or a multiplier penalty function. For making a more streamlined presentation of the results we previously have tested all the option solvers and finally the two most efficient of them (for each penalty method) have been selected to be compared against the two versions of RPSALG by using the best option described in Table 3, i.e., case i) with function θ_4.

As penalty functions we have selected the following AMPL command options (indeed they are the penalty functions with integral representations in [32, (5.11) and (5.14)]): option nsips_options 'method=penalty pf_type=p1' and option nsips_options 'method=penalty_m pf_type=p1'. In this article we refer to this couple of selected methods as Penalty1 and Penalty2, respectively.

6.2 Test problems

A total of 71 semi-infinite test problems have been selected satisfying the convexity and differentiability hypothesis of RPSALG with an initial guess. The test problems have been selected as follows: 37 of them have been obtained from the SIAMPL database[4] (numbers from 1 to 29 in Table 6 and from 50 to 57 in Tables 9, 10) and the remaining 34 problems have been generated by using the procedure described in [17]. In this latter case we can generate test problems with known solution and without limitations on n and $\dim T$.

[4] http://plato.la.asu.edu/ftp/sipampl.pdf

6.3 Computational results

The numerical experiments were carried out on a PC with Processor Intel(R) Core(TM)2 Duo CPU E8500, 3.16 GHz and 3.49 GB of RAM (MS Windows XP, professional). For comparing the different solvers on the same computer we have used the AMPL Student Version 20111121 (MS VC++ 6.0) to run NSIPS. The AMPL student version can be downloaded for free but it is limited to solve problems with 300 variables and a total of 300 objectives and constraints.

The numerical experiments are summarized in four tables. In Tables 6 and 7, *Num* denotes the number assigned to the instance selected of the chosen solver, *Name* indicates the name of the instance in SIAMPL database, n is the number of variables, *Itr1* is the number of iterations required for RPSALG1, *Tfc1* indicates the total number of functions and constraints evaluations for RPSALG1, *Itr2* is the number of iterations required for RPSALG2, *Tfc2* indicates the total number of functions and constraints evaluations for RPSALG2, *ItrP1* is the number of iterations required for Penalty1, *TfcP1* indicates the total number of functions and constraints evaluations for Penalty1, *ItrP2* is the number of iterations required for Penalty2, *TfcP2* indicates the total number of functions and constraints evaluations for Penalty2. In Tables 9 and 10 $\dim T$ represents the dimension of the space of parameters, f^* the optimal value of the objective function, *Tfce* the total number of functions and constraints evaluations, and *Time* the CPU time.

The results shown in the Tables 6 and 7 are compared in Table 8. The precise meanining of the entries in the latter table, $\rho_s(1)$ (probability of success in solving a problem) and ρ_s^* (probability of win over the rest) is explained in the Appendix.

The maximum number of iterations was limited to 400 and the precision is $\epsilon = 0.001$ for all instances. If a solver needs more than 400 iterations to obtain a solution and/or the accuracy of the obtained solution is greater than the chosen value, then we will consider that the solver has failed in solving the problem. The failure of a solver is indicated with a star $(*)$, in the column that indicates the number of iterations of the corresponding solver.

Due to the mentioned limitations of the NSIPS, the computational results presented in the article have been divided in two cases:

i) **Instances that satisfy the NSIPS limitations.** We compare the total number of functions an constraints evaluations for the solvers RPSALG1, RPSALG2, Penalty1 and Penalty2 as described in Tables 6 and 7. For the sake of brevity and clarity, we have included the numerical results as performance profiles in Table 8. Figure 1 plots the performance profile of the results. From Table 8 we can compare the four solvers with respect to the probability of win over the rest, $\rho_s(1)$, and the probability of success in solving a problem, ρ_s^*.

ii) **Instances that do not satisfy the NSIPS limitations.** We compare the solvers RPSALG1, RPSALG2 for problems from 50 to 71 described in Tables 9 and 10, that do not satisfy the NSIPS limitations so we can only solve them by using the RPSALG versions. In Tables 9 and 10 the specific numerical results for the solvers RPSALG1, RPSALG2 with $\dim T > 1$ are described.

Table 6 Results for RPSALG1, RPSALG2, Penalty1 and Penalty2, dim T = 1, $\epsilon = 0.001$, $(*)$ indicates the failure of corresponding solver

Num	*Name*	*n*	*Itr1*	*Tfc1*	*Itr2*	*Tfc2*	*ItrP1*	*TfcP1*	*ItrP2*	*TfcP2*
1	coopeL	2	0*	0000	8	1120	5	19530	9	26720
2	coopeM	2	9	6480	7	4046	5	17175	10	28294
3	hettich4	2	9	3392	11	2184	5	33635	12	65103
4	leon12	2	8	4072	9	2540	5	28556	10	34600
5	leon13	2	8	3118	401*	12848	4	361854	9	42670
6	leon14	2	12	5751	12	3095	5	655248	9	55853
7	liu1	2	401*	24120	401*	20530	5	11791	11	18056
8	liu2	2	10	3312	10	2461	5	22677	9	28030
9	watson1	2	10	3312	401*	25440	6	22504	11*	40865
10	hettich2	3	8	5534	9	6898	6	85232	11	125432
11	watson4a	3	8	6689	8	3372	5	28974	9	76059
12	watson5	3	12	7624	12	5204	5*	27036	11	44009
13	leon1	4	7	7388	401*	106582	5	72977	11	89331
14	hettich3	5	13	38532	9	22342	6	40318	12	53189
15	leon6	5	6	17621	10	15031	4	57090	8	70905
16	leon7	5	7	16908	8	7311	5	528993	9	427003
17	leon2	6	6	23042	8	20799	4	61317	7	78341
18	leon3	6	7	28942	8	18971	4	294449	9	337566
19	watson4b	6	7	18023	9	17205	5	31368	9	141536
20	leon4	7	7	29793	401*	135756	5	91313	9	122092
21	leon8	7	8	60767	7	22607	9	298892	9	217280
22	leon9	7	8	458573	8	171424	5*	250714	9*	217387
23	ferris1	7	7	33655	7	15069	7	208777	9	265833
24	ferris2	7	8	36763	9	16900	5	59729	7	178273
25	leon5	8	8	27727	8	27727	9	304946	9	269204
26	watson4c	8	7	55682	9	29687	5	54011	9	205438
27	fang1	50	15	900654	10	231185	4*	42913	6*	167138
28	fang2	50	8	582136	11	437440	4*	44996	6	170520
29	fang3	50	20	1443832	11	332947	5*	44972	9	137194

7 Conclusions

This paper reports on the implementation of a penalty and smoothing method for solving convex semi-infinite programing problems inspired by the first algorithm of Remez, the so-called RPSALG. It is known that one of the main computational difficulties to solve semi-infinite programs comes from the non-convex optimization problem associated with the constraints, $S2$, which must be solved efficiently at each iteration. As an innovation of this paper, we tackle this problem with the so-called Cutting Angle Method, a global optimization procedure for solving Lipschitz programming problems. As far as we know, a global optimization software for solving $S2$ has not been used before.

Two versions of RPSALG are proposed (RPSALG1 and RPSALG2), implemented in C++ and run on Visual C++ 6.0. We compare them with the best options of the two penalty methods solvers included in NSIPS, called Penalty1 (based on penalty functions where no reference to Lagrange multipliers is made) and Penalty2 (using an Augmented Lagrangian penalty function), and run on the student version of AMPL (with a maximum of 300 variables and a total of 300 objectives and constraints, and $\dim T = 1$). We verify its performance by conducting some numerical results with a set of test problems. All the results have been

Table 7 Results for RPSALG1, RPSALG2, Penalty1 and Penalty2, dim T =1 $\epsilon = 0.001$, (∗) indicates the failure of corresponding solver (Continued)

Num	*Name*	n	*Itr1*	*Tfc1*	*Itr2*	*Tfc2*	*ItrP1*	*TfcP1*	*ItrP2*	*TfcP2*
30	ftpeallT1	5	8	23717	10	10486	3	68219	9	107806
31	ftpeallT1	10	8	38294	10	21980	4	104427	7	122150
32	ftpeallT1	15	9	64560	10	32020	4	128913	9	125002
33	ftpeallT1	20	9	74922	11	49486	4	149475	7	150263
34	ftpeallT1	25	9	113051	11	77726	3	134452	7	169104
35	ftpeallT1	50	8	204996	11	205008	4	152510	9	143445
36	ftpeallT1	100	9	370370	11	535044	3	217982	7	106212
37	ftpeallT1	150	9	631567	10	723818	4	72148	7	68182
38	ftpeallT1	200	9	888875	9	591287	3	59251	7	60756
39	ftpeallT1	250	9	1147834	8	564614	4	57262	7	62128
40	ftpeaqlT1	5	12	14823	13	14320	4	36017	9	53840
41	ftpeaqlT1	10	12	29001	13	24923	4	40330	10	62838
42	ftpeaqlT1	15	12	44718	12	32900	4	39372	10	66483
43	ftpeaqlT1	20	12	64793	13	49820	5	42169	9	67780
44	ftpeaqlT1	25	12	79442	13	63712	5	45696	8	50821
45	ftpeaqlT1	50	12	149173	12	109142	5	47843	9	67782
46	ftpeaqlT1	100	12	243641	12	222307	4	42026	10	60512
47	ftpeaqlT1	150	12	420680	12	325963	4	38261	9	68638
48	ftpeaqlT1	200	12	583250	12	444920	4	35394	7	47597
49	ftpeaqlT1	250	12	672318	12	573394	4	35427	7	48350

Table 8 Solvers evaluation

Solver	$\rho_s(1)$	ρ_s^*
RPSALG1	12.25%	96.43%
RPSALG2	59.18%	90.38%
Penalty1	18.37%	90.38%
Penalty2	6.70%	94.67%

obtained on the same computer. Notice that, the termination criterion (21) used in RPSALG is computationally expensive due to the number of function evaluations performed at each iteration. NSIPS just requires the evaluation of a relatively small variable change at each iteration (with such termination criterion the number of function evaluations performed for RPSALG would have been much lower). Despite of this drawback there are several reasons to use it: a) the theoretical coherence of the article, b) it provides accurate approximations of the optimal value of the objective function together with good estimations of the optimal solution, and c) it minimizes the risk of a false statement of convergence.

The preliminary numerical considerations are as follows. From the summary results of the solvers evaluation of Table 8 (and Figure 1), we can conclude that RPSALG2 is much faster than the other solvers while RPSALG1 is slightly more stable than the others (it solves more than 96% of the instances in contrast with percentages in the interval 90%-95% for the three other solvers). On the other hand, RPSALG can solve convex semi-infinite programs of class C^1, without any limitation on the number of parametric constraints, the number of general finite constraints, and $\dim T$ as described on the Tables 9 and 10.

Table 9 Results for RPSALG1, dim T > 1, $\epsilon = 0.001$, (∗) indicates failure of the solver

Num	*Name*	*n*	dim T	*Iter*	f^*	*Tfce*	*Time*
50	andreson1	3	2	10	-0.3340	6409	0.189
51	hettich5	3	2	7	0.5368	6007	0.484
52	lin1	6	2	7	-1.5070	37750	0.531
53	reemtsen3	10	2	10	-0.8015	81066	3.062
54	reemtsen4	37	2	401*	-1.4433	2275119	260.375
55	potchinkov2	65	3	1	-0.0005	89150	27.500
56	potchinkov3	66	2	7	0.9996	236747	59.609
57	potchinkovPL	122	2	6	-0.0008	1351250	338.865
58	fpeallT2	5	2	11	-0.6408	46679	1.421
59	fpeallT2	10	2	11	-0.6407	81360	2.959
60	fpeallT2	50	2	12	-0.6409	609637	49.985
61	fpeallT2	100	2	13	-0.6409	1613578	218.964
62	fpeallT2	250	2	12	-0.6408	2758994	799.040
63	fpeallT2	500	2	11	-0.6409	4402209	2396.774
64	fpeallT2	1000	2	10	-0.6406	2000102	2065.064
65	fpeaqlT2	5	2	12	0.9992	16235	0.703
66	fpeaqlT2	10	2	12	0.9992	32569	1.515
67	fpeaqlT2	50	2	13	0.9992	163137	14.562
68	fpeaqlT2	100	2	12	0.9990	310612	44.173
69	fpeaqlT2	250	2	12	0.9990	727781	214.893
70	fpeaqlT2	500	2	12	0.9991	1695534	919.218
71	fpeaqlT2	1000	2	12	0.9991	3428577	3525.101

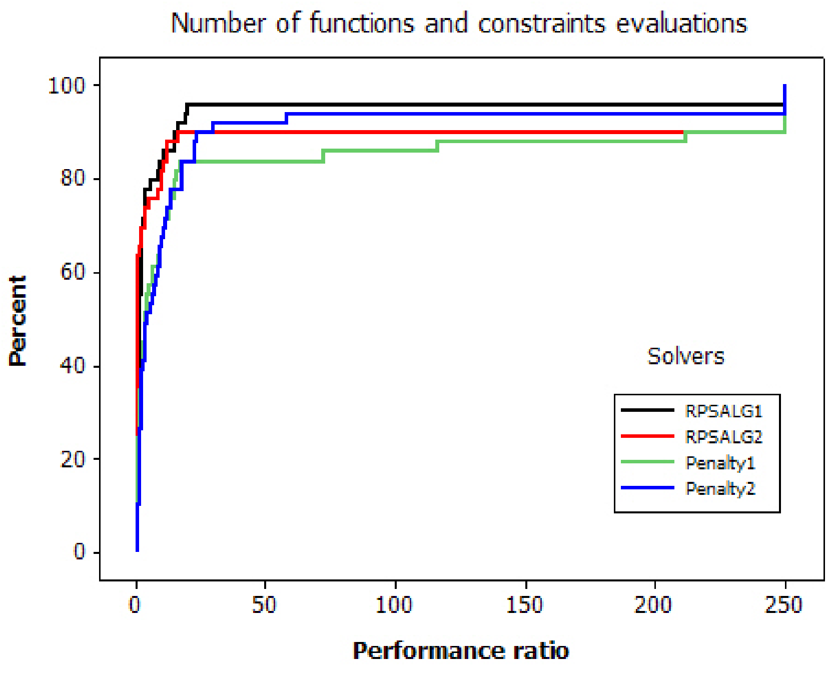

Fig. 1 Performance profiles of the number of function and constraints evaluations

Table 10 Results for RPSALG2, dim T > 1, $\epsilon = 0.001$, $(*)$ indicates failure of the solver

Num	*Name*	*n*	dim T	*Iter*	f^*	*Tfce*	*Time*
50	andreson1	3	2	401*	-0.3333	117798	7.505
51	hettich5	3	2	7	0.5392	2293	0.359
52	lin1	6	2	8	-1.6265	41903	0.562
53	reemtsen3	10	2	11	-0.8013	69738	2.671
54	reemtsen4	37	2	401*	-1.4433	2275119	260.985
55	potchinkov2	65	3	1	-0.0005	10698	3.828
56	potchinkov3	66	2	6	0.9995	164254	41.656
57	potchinkovPL	122	2	6	-0.0008	1143037	287.322
58	fpeallT2	5	2	9	-0.6409	8481	0.453
59	fpeallT2	10	2	9	-0.6406	15038	0.860
60	fpeallT2	50	2	10	-0.6407	92860	8.406
61	fpeallT2	100	2	10	-0.6405	180172	25.625
62	fpeallT2	250	2	10	-0.6407	481768	140.951
63	fpeallT2	500	2	10	-0.6407	983064	534.158
64	fpeallT2	1000	2	10	-0.6406	2000102	2065.064
65	fpeaqlT2	5	2	12	0.9990	9970	0.593
66	fpeaqlT2	10	2	13	0.9992	20290	1.156
67	fpeaqlT2	50	2	13	0.9991	121333	10.969
68	fpeaqlT2	100	2	13	0.9992	225129	31.909
69	fpeaqlT2	250	2	13	0.9991	593726	173.499
70	fpeaqlT2	500	2	13	0.9993	1137878	613.258
71	fpeaqlT2	1000	2	13	0.9992	2467579	2558.739

The results obtained are promising enough to suggest that RPSALG could be a competitive solver for CSIP problems.

Appendix: performance profiles

The benchmark results are generated by running the three solvers to be compared on the collection of problems gathered in the SIPAMPL database and recording the information of interest, in this case the number of function evaluations (as it is independent of the available hardware). In this paper we use the notion of performance profile due to Dolan and Moré [14]) as a tool for comparing the performance of a set of solvers $\mathcal{S}$ on a test set $\mathcal{P}$. For each couple $(p, s) \in \mathcal{P} \times \mathcal{S}$ we define

$$f_{p,s} := \text{number of function evaluations required to solve problem } p \text{ by solver } s.$$

Let $p \in \mathcal{P}$ be a problem solvable by solver $s \in \mathcal{S}$. We compare the performance on problem p of solver s with the best performance of any solver on the same problem by means of the *performance ratio*

$$r_{p,s} := \frac{f_{p,s}}{\min\{f_{p,s} : \ s \in \mathcal{S}\}} \geq 1,$$

with $r_{p,s} = 1$ if and only if s is a winner for p (i.e. it is at least as good, for solving p, as any other solver of $\mathcal{S}$). We also define $r_{p,s} = r_M$ when solver s does not solve problem p, where r_M is some scalar greater than the maximum of the performance ratios $r_{p,s}$ of all couples $(p, s) \in \mathcal{P} \times \mathcal{S}$ such that p is solved by solver s. The choice of r_M does not affect the performance evaluation.

The performance of solver s on any given problem may be of interest, but we would like to obtain an overall assessment of the performance of the solver. To this aim, we associate with each $s \in \mathcal{S}$ a function $\rho_s : \mathbb{R}_+ \to [0,\ 1]$, called *performance profile* of s, defined as the ratio

$$\rho_s(t) = \frac{\text{size}\{p \in \mathcal{P} : \ r_{p,s} \leq t\}}{\text{size}\mathcal{P}}, \ t \geq 0.$$

Obviously, ρ_s is a stepwise non-decreasing function such that $\rho_s(t) = 0$ for all $t \in [0, 1[$ and $\rho_s(1)$ is the relative frequency of wins of solver s over the rest of the solvers. If p is taken at random from $\mathcal{P}$, then $r_{p,s}$ can be interpreted as a random variable and $\rho_s(1)$ as the probability of solver s to win over the rest of the solvers while, for $t > 1$, $\rho_s(t)$ represents the probability for solver $s \in \mathcal{S}$ that a performance ratio $r_{p,s}$ is within a factor $t \in \mathbb{R}$ of the best possible ratio. So, in probabilist terms, ρ_s can be seen as a distribution function.

The definition of the performance profile for large values requires some care. We assume that $r_{p,s} \in [1, r_M]$ and that $r_{p,s} = r_M$ only when problem p is not solved by solver s. As a result of this convention, $\rho_s(r_M) = 1$, and the number

$$\rho_s^* := \lim_{t \searrow r_M} \rho_s(t)$$

is the probability that the solver $s \in \mathcal{S}$ solves problems of $\mathcal{P}$.

Choosing a best solver for $\mathcal{P}$ is a bicriteria decision problem, the objectives being the probability of winning and the probability of solving a problem, i.e.

$$\text{“}\min\text{”} \left\{ (\rho_s(1), \rho_s^*) : s \in \mathcal{S} \right\}.$$

Performance profiles are relatively insensitive to changes in results on a small number of problems. Additionally, they are also largely unaffected by small changes in results over many problems.

Acknowledgments

The authors wish to thank A. Vaz for his valuable help concerning the implementation of SIPAMPL.

The research of the second author has been partially supported by MINECO of Spain, Grant MTM2011-29064-C03-01. The research of the third and fourth authors has been supported by MINECO of Spain, Grant MTM2011-29064-C03-02 and Generalitat Valenciana, Grant ACOMP/2013/062.

References

1. Auslender, A., Goberna, M.A., López, M.A.: Penalty and smoothing methods for convex semi-infinite programming. Math. Oper. Res. 34, 303-319 (2009)
2. Auslender, A., Teboulle, M.: Interior gradient and proximal methods for convex and conic optimization. SIAM J. Optim. 16, 697-725 (2006)
3. Bagirov A.M., Rubinov A.M.: Global minimization of increasing positively homogeneous functions over the unit simplex. Ann. Oper. Res. 98, 171-187 (2000)
4. Bagirov, A.M., Rubinov, A.M.: Modified versions of the cutting angle method. In: Hadjisavvas, N., Pardalos, P.M. (eds.) Advances in Convex Analysis and Global Optimization, pp. 245-268. Kluwer, Netherlands (2001)
5. Batten, L.M., Beliakov, G.: Fast algorithm for the cutting angle method of global optimization. J. Global Optim. 24, 149-161 (2002)
6. Beliakov, G.: Geometry and combinatorics of the cutting angle method. Optimization 52, 379-394 (2003)
7. Beliakov, G.: Cutting angle method. A tool for constrained global optimization. Optim. Method. Softw. 19, 137-151 (2004)
8. Beliakov, G.: A review of applications of the cutting angle method. In: Rubinov, A.M., Jeyakumar, V. (eds.) Continuous Optimization, pp. 209-248. Springer, New York (2005)
9. Beliakov, G.: Extended cutting angle method of global optimization. Pacific J. Optim. 4, 153-176 (2008)
10. Beliakov, G., Ferrer, A.: Bounded lower subdifferentiability optimization techniques: applications. J. Global Optim. 47 211-231 (2010)
11. Bonnans, J.F., Shapiro, A.: Perturbation Analysis of Optimization Problems. Springer, New York (2000)

12. Cheney, E.W., Goldstein, A.A.: Newton method for convex programming and Tchebycheff approximation. Numer. Math. 1, 253-268 (1959)
13. Dinh, N., Goberna, M.A., López, M.A.: On the stability of the feasible set in optimization problems. SIAM J. Optim. 20, 2254-2280 (2010)
14. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Programming, 91 201-213 (2002)
15. Fackrell, M.: A semi-infinite programming approach to identifying matrix-exponential distributions. Internat. J. Systems Sci. 43, 1623-1631 (2012)
16. Faybusovich, L., Mouktonglang, T., Tsuchiya, T.: Numerical experiments with universal barrier functions for cones of Chebyshev systems. Comput. Optim. Appl. 41, 205-223 (2008)
17. Ferrer, A., Miranda, E.: Random test examples with known minimum for convex semi-infinite programming problems. E-prints UPC, 2013 (http://hdl.handle.net/2117/19118)
18. Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H.: User's guide for NPSOL: A Fortran Package for Nonliner Programing. Stanford University, Stanford, CA (1986)
19. Gürtuna, F.: Duality of ellipsoidal approximations via semi-infinite programming. SIAM J. Optim. 20, 1421-1438 (2009)
20. Hiriart-Urruty, J.-B., Lemaréchal, C.: Convex Analysis and Minimization Algorithms. I. Fundamentals. Springer-Verlag, Berlin (1993)
21. Ito, S., Liu, Y., Teo, K.L.: A dual parametrization method for convex semi-infinite programming. Ann. Oper. Res. 98, 189-213 (2000)
22. Joshi, S., Boyd, S.: Sensor selection via convex optimization. IEEE Trans. Signal Processing 57, 451-462 (2009)
23. Karimi, A., Galdos, G.: Fixed-order H_∞ controller design for nonparametric models by convex optimization. Automatica 46, 1388-1394 (2010)
24. Katselis, D., Rojas, C., Welsh, J., Hjalmarsson, H.: Robust experiment design for system identification via semi-infinite programming techniques. In: Kinnaert, M. (ed.), Preprints of the 16th IFAC Symposium on System Identification, pp. 680-685. Brussels, Belgium (2012)
25. Kelley, J.E., Jr.: The cutting-plane method for solving convex programs. J. Soc. Indust. Appl. Math. 8, 703-712 (1960)
26. Muller, J.M.: Elementary Functions: Algorithms and Implementation (2nd Ed.). Birkhäuser, Boston (2006)
27. Nesterov, Y.: A method for solving the convex programming problem with convergence rate $O(\frac{1}{k^2})$. Soviet Math. Dokl. 27, 372-376 (1983)
28. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, New York (1999)
29. Remez, E.: Sur la détermination des polynômes d'approximation de degré donné. Commun. Soc. Math. Kharkoff et Inst. Sci. Math. et Mecan. 10, 41-63 (1934)
30. Rubinov, A.M.: Abstract Convexity and Global Optimization. Kluwer, Dordrecht/Boston (2000)
31. Tichatschke, R., Kaplan, A., Voetmann, T., Böhm, M.: Numerical treatment of an asset price model with non-stochastic uncertainty, TOP 10, 1-50 (2002)
32. Vaz, A., Fernandes, E., Gomes, M.: SIPAMPL: Semi-infinite programming with AMPL. ACM Trans. Math. Software 30, 47-61 (2004)