A reduction type penalty algorithm for nonlinear semi-infinite programming ⋆

Alzira Mota * **A. Ismael F. Vaz** **

* *Institute Polytechnic of Porto, Mathematic Department, Engineering Institute, Porto, Portugal (e-mail:*atm@isep.ipp.pt*).*
** *University of Minho, Engineering School, Production and Systems Department, Portugal (e-mail:*aivaz@dps.uminho.pt*).*

Abstract: Semi-infinite programming (SIP) problems arise in several engineering areas such as, for example, robotic trajectory planning, production planning, digital filter design and air pollution control. In spite of being an active research area with many seminal works it lacks available software that could be used by the research community. The only exceptions are the `fseminf` MATLAB function, available in the Optimization Toolbox, and the NSIPS solver, but neither of them provide an implementation of a method belonging to the well known reduction type class.
This paper proposes an implementation of a reduction type algorithm base on a penalty technique and provides a compare between several well known penalty functions.
The provided numerical results with a significant number of SIP test problems are reported as performance profiles.

Keywords: Semi-infinite programming, Reduction type algorithm, Particle swarm optimization, Penalty functions

1. INTRODUCTION

Semi-infinite programming (SIP) problems arise in many engineering areas such as robotic trajectory planning, production planning, Chebyshev approximation theory and air pollution control (see, for example, Goberna and López (2001); Hettich and Kortanek (1993); Reemtsen and Rückmann (1998) and the references therein).

While algorithms for solving SIP problems are reported in the SIP literature during the last decades (e.g. Goberna and López, 2001; Hettich and Kortanek, 1993; Reemtsen and Rückmann, 1998; Polak, 1997; Fiacco and Kortanek, 1983; Hettich, 1979) there is not much software available. The MATLAB (MATLAB, 2004) `fseminf` function is available in the Optimization Toolbox. The NSIPS (Vaz et al., 2002) solver is a public domain software dedicated to SIP. The FSQP [1] (Lawrence and Tits, 1998) is also reported to be useful for SIP.

We introduce now some notation and definitions used throughout the paper. The SIP problem can be described in the following form:

$$\begin{aligned} &\min_{x \in R^n} \ f(x) \\ &s.t. \ \ g_i(x,t) \leq 0, \ i = 1, \ldots, m \\ &\qquad \forall t \in T \subset R^p, \end{aligned} \tag{1}$$

where $f(x)$ is the objective function, $g_i(x,t)$, $i = 1, \ldots, m$, are the (infinite) constraint functions and T is an infinite set. f and g_i are assumed to be at least twice continuously differentiable in all arguments. The SIP formulation could be more general, by including equality and inequality constraints depending only on the x variables. The set T could also depend on the x variables resulting in a generalized SIP problem, but the given definition just suits our purpose.

A natural way to solve the SIP problem (1) is to replace the infinite set T by a finite one. There are several ways to do this. Discretization, exchange and reduction type methods (see Hettich and Kortanek (1993), for a more detailed explanation) are the major classes.

Reduction type methods are characterized by the need to compute the global solutions of the auxiliary problems

$$\max_{t \in T} \ g_i(\bar{x}, t) \tag{2}$$

for each $i = 1, \ldots, m$ and a given $\bar{x}$.

The extra burden to compute all the global solution to problem (2) is compensated by the good theoretical properties enjoyed by the reduction type methods.

Discretization methods are implemented in the MATLAB toolbox and in the NSIPS solvers.

In the next section we introduce the reader to penalty techniques in the context of reduction type methods for SIP. In Section 3 we describe the implemented penalty technique algorithm. Numerical results are presented in Section 4 and we conclude the paper in Section 5.

2. PENALTY APPROACH

In a penalty technique a constrained optimization problem is transformed into a sequence of unconstrained optimiza-

⋆ Support was provided by FCT under grant POCI/MAT/58957/2004, and by the Algoritmi research center
[1] www.aemdesign.com

tion subproblems. This transformation is done by defining an auxiliary function (penalty function ϕ) depending on the objective or Lagrangian function and on the problem constraints. The unconstrained optimization subproblems are controlled by one or more parameters (penalty parameters) that determine the relative importance of the constraints in the unconstrained subproblems.

Note 1. A penalty function $\phi(x,\mu)$ is called exact when there exists a finite μ^*, such that a local minimizer of $\phi(x,\mu)$, $x^*(\mu)$, is a solution of (1), x^*, for $\mu > \mu^*$.

Penalty functions for semi-infinite programming have been proposed during the last decades (see, for example, Conn and Gould (1987); Price (1992); Price and Coope (1990, 1996); Watson (1981)).

In this paper we are reporting results for the L_1, L_2, L_∞ and L_P penalty functions. The first three penalty functions considered are based on the 1, 2 and ∞ norms of the constraints, respectively. The L_P penalty function was proposed in Price and Coope (1996) and combines two terms based on the ∞ norm of the constraints.

The L_1 penalty function considered for problem (1) is formulated as

$$\phi_1(x,\mu) = f(x) + \mu\theta_1(x) \tag{3}$$

where

$$\theta_1(x) = \sum_{i=1}^{m} \left(\max_{t\in T} [g_i(x,t)]_+ \right),$$

μ is a positive penalty parameter and for $z \in R$, $[z]_+ = \max\{0, z\}$. The major drawback with this penalty function is its non-differentiability due to the use of the $[]_+$ function.

The L_2 penalty function is defined as

$$\phi_2(x,\mu) = f(x) + \mu\theta_2(x) \tag{4}$$

where

$$\theta_2(x) = \sum_{i=1}^{m} \left(\max_{t\in T} [g_i(x,t)]_+^2 \right).$$

This penalty function is known not to be exact (and therefore a solution to problem (1) is only obtained when $\mu \to \infty$, resulting in ill conditioned subproblems to be solved). In spite of this drawback the ϕ_2 penalty function has the advantage of being once continuously differentiable (recall that f and g_i, $i = 1,\ldots,m$, are assumed twice continuous differentiable).

The L_∞ penalty function uses the ℓ_∞ norm of the constraints and is described as

$$\phi_\infty(x,\mu) = f(x) + \mu\theta_\infty(x) \tag{5}$$

where

$$\theta_\infty(x) = \max_{i\in\{1,\ldots,m\}} \left(\max_{t\in T} [g_i(x,t)]_+ \right).$$

The penalty function proposed by Price and Coope (1996); Price (1992) uses an augmented L_∞ penalty function. The augmented penalty function uses two penalty parameters to control feasibility and is defined as follows:

$$\phi_P(x,\mu,\nu) = f(x) + \mu\theta_\infty(x) + \frac{1}{2}\nu\theta_\infty^2(x), \tag{6}$$

where $\mu > 0$ and $\nu \geq 0$ are the penalty parameters. The second term was added in order to prevent μ to be set extremely large. Under mild assumptions it can be shown that an optimal to problem (1) is a critical point of $\phi_P(x,\mu,\nu)$ and conversely (see Price (1992) for details).

The penalty function L_∞ is exact (in a similar way as defined in Note 1).

When a penalty function is known to be exact μ^* (and ν^*) is (are) not known in advance so a unique minimization of the penalty function is not possible. Often μ^* (and/or ν^*) is (are) related with the Lagrange multipliers of the active constraints at the solution of the constrained (original) problem.

For a practical purpose the μ (and ν) parameter(s) is (are) updated along the iterations as new estimations for the Lagrange multipliers becomes available, or simply by increasing the penalty parameters if the current iterate shows to be unfeasible for the constrained problem (1).

Updating schemes for the penalty parameters and the penalty framework are described in the next section.

3. THE PENALTY FRAMEWORK

In this section we describe the used penalty framework.

The penalty framework comprises two main phases (or iterations type). The first phase, called external iterations, is described in subsection 3.1 and is related with solving a sequence of penalty functions parameterized by μ (for the penalty functions L_1, L_2 and L_∞) or parameterized by μ and ν (for the penalty function L_P). In this phase the computation of all the global (and as many as possible local) solutions to problems (2) is also addressed. The second phase, called internal iterations, is described in subsection 3.2 and is related with the minimization of the penalty function when μ (and ν) are fixed.

3.1 External iterations

This phase computes a sequence $\{x_k\}$, where k is the external iteration counter, by solving a sequence of subproblems

$$x_{k+1} \quad = \quad arg \min_{x\in R^n} \phi(x)$$

parameterized by the penalty parameters (μ when (3), (4), or (5) are considered and μ and ν when (6) is considered).

If the optimality stopping criteria is met the algorithm stops with an approximate solution to (1), otherwise the penalty parameters are updated and another external iteration begins.

In order to motivate the penalty update equation for penalty L_P we introduce the Lagrangian function, $L(x,\lambda)$, considering only one infinite constraint to simplify the notation (*i.e.*, considering $m = 1$).

Let

$$L(x,\lambda) = f(x) + \sum_{i=1}^{s} \lambda_i g(x,t_i) \tag{7}$$

where λ_i are the Lagrange multiplier associated with the points t_i that make the $g(x,t) \leq 0$ constraint active. Note that the t_i, $i = 1,\ldots,s$, are the global optima for problem (2).

Consider also the penalty L_P defined for $m = 1$,