JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 120, No. 1, pp. 73–95, January 2004 (©2004)

An Algorithm for Approximate Multiparametric Linear Programming[1]

C. Filippi[2]

Communicated by H. P. Benson

Abstract. Multiparametric programming considers optimization problems where the data are functions of a parameter vector and describes the optimal value and an optimizer as explicit functions of the parameters. In this paper, we consider a linear program where the right-hand side is an affine function of a parameter vector; we propose an algorithm for approximating its solution. Given a full-dimensional simplex in the parameter space and an optimizer for each simplex vertex, the algorithm formulates the linear interpolation of the given solutions as an explicit function of the parameters, giving a primal feasible approximation of an optimizer inside the simplex. If the resulting absolute error in the objective exceeds a prescribed tolerance, then the algorithm subdivides the simplex into smaller simplices where it applies recursively. We propose both a basic version and a refined version of the algorithm. The basic version is polynomial in the output size, provided a polynomial LP solver is used; the refined version may give a smaller output. A global error bound for the optimizer is derived and some computational tests are discussed.

Key Words. Multiparametric programming, linear programming, approximation methods, error bounds, complexity analysis, model predictive control.

1. Introduction

In parametric programming, an optimization problem is considered where the data are functions of one or more parameters. Parametric

[1]The author thanks the anonymous referees and Professor H. P. Benson, whose valuable comments greatly helped to improve the quality of the paper.

[2]Research Associate, Department of Pure and Applied Mathematics, University of Padova, Padova, Italy.

0022-3239/04/0100-0073/0 © 2004 Plenum Publishing Corporation

programming techniques systematically subdivide the parameter space into characteristic regions where the optimal value and an optimizer are given as explicit functions of the parameters, hence providing the decision maker with a complete map of possible outcomes.

A vast literature is concerned with parametric programming, especially for the linear case. However, most of the research was concerned with the monoparametric case, where the parameter is scalar, whereas the multiparametric case, where the parameters form a vector, was far less studied. The reason for this may be detected in at least two facts: (i) the number of characteristic regions may grow exponentially with both the dimension of the optimization problem and the number of parameters; thus, subdividing a multidimensional parameter space may require a huge amount of computing time and memory space; (ii) the visualization of multidimensional characteristic regions is impossible; hence, the vast information generated by a parametric programming technique is almost useless for a human decision maker.

In recent years, a new interest in multiparametric programming arose from the so-called model predictive control (MPC), a well-known technique in the system theory and optimal control community (Refs. 1–2). MPC is used widely in the process industry for the automatic regulation of plants under operating constraints. In MPC, the next command action is obtained by solving an optimization problem where the constraint right-hand sides depend on the current sensor measurement. In the classic setting, the optimization problem is solved online at each time step. However, the optimization effort may be moved offline by solving a multiparametric program where the variables correspond to command inputs and the parameters correspond to sensor measurements (Refs. 3–4). Since in the final application the characteristic regions and corresponding functions describing an optimizer are implemented on an electronic device, objection (ii) above is no more valid. Nevertheless, objection (i) still remains, and this fact motivated our present work.

All approaches to multiparametric linear programming that have appeared in the literature are exact and use bases to get a description of the characteristic regions. A vast bibliography on parametric linear programming, updated to the early nineties, can be found in Ref. 5. In particular, Gal and Nedoma (Ref. 6) were the first to propose a general and systematic procedure to solve a multiparametric right-hand side linear programming problem. They proposed to cover the parameter space by critical regions, where a critical region is the set of parameters such that a given basis is optimal for the parametric program. The proposed procedure is a dual pivoting method which searches a connected component of the graph whose vertices are the dual feasible bases of the problem and whose edges are the

dual pivots. More recently, Borrelli, Bemporad, and Morari (Ref. 7) proposed a new method to cover the parameter space by critical regions, which is not founded on pivoting. The method is based on a recursive subdivision of the parameter space around a critical region. Technical issues concerning the deletion of unnecessary subdivisions are addressed, and a preprocessing technique which possibly reduces a priori the number of parameters is also suggested. For a multiparametric transportation problem, Filippi and Romanin-Jacur (Ref. 8) proposed a lexicographic network pivoting method under which degeneracy is virtually eliminated. Unfortunately, the characteristics of the method are not extendable to a general multiparametric linear program.

As far as we know, no method has been proposed in the literature to get an approximate solution of a multiparametric linear programming problem or even a monoparametric linear program. Bemporad and Filippi (Ref. 9) suggested a method to get an approximate solution to a multiparametric strictly convex quadratic programming problem. They proposed to enlarge the exact characteristic region corresponding to a fixed active constraint set by relaxing the first-order optimality conditions, while preserving primal feasibility. Such an approach cannot be specialized satisfactorily to the linear case. In particular, the most interesting results in Ref. 9 are obtained when only the nonnegativity of the dual variables corresponding to the active constraints is relaxed: in the linear case, this relaxation would leave the characteristic region unchanged.

In the context of general convex parametric nonlinear programming, Fiacco (Ref. 10, Chapter 9) sketched a strategy for approximating the optimal value functions along a monodimensional cut of the parameter space. Essentially, he noted that the optimal primal solutions associated with two fixed parameter vectors may be used to compute an affine upper bound along the line segment joining the same parameter vectors; furthermore, the optimal dual solutions associated with the two parameter vectors may be used to compute a piecewise affine lower bound along the same line segment. Similar observations have inspired our work.

In this paper, we consider a multiparametric linear program where the right-hand side vector is an affine function of a parameter vector, and we propose an algorithm which solves approximately the program. The algorithm is based on a simple and direct recursive idea and may be outlined as follows. Consider a full-dimensional simplex in the parameter space such that all the linear programs corresponding to the vertices have a finite optimizer. The algorithm formulates the linear interpolation of such optima as an explicit function of the parameters, giving a primal feasible approximation of an optimizer inside the simplex. Then, the algorithm evaluates a lower bound and an upper bound on the absolute error in the optimal value which

may occur by using the approximation obtained inside the simplex. If the upper bound is less than a fixed positive tolerance, then the algorithm stops. If the lower bound is greater than the tolerance, then the simplex is subdivided into two or more full-dimensional nonoverlapping simplices; then, the algorithm is called recursively on each new simplex. If the tolerance is between the lower and the upper bound, then they are both refined, and the process may continue until they coincide. Each lower and upper bound evaluation corresponds to the solution of an auxiliary linear program.

We develop and analyze both a basic version and a refined version of the algorithm. Both versions are completely independent from the LP solver used for the auxiliary linear programs; in particular, both versions are completely independent from both the bases and optimal partitions of the variables. This is a clear distinction with respect to all the exact methods proposed in the literature and allows us to use the most convenient solver in each possible application. We show that the basic version of our algorithm has polynomial complexity in the output size, provided a polynomial LP solver is used; the refined version has a slightly higher complexity, but may give a smaller output. We derive also a global error bound on the optimizer; such a bound is computable practically and can be used to bound a priori the distance of the approximate optimizer obtained from a genuine one under any L_p norm. Finally, some computational tests on randomly generated instances are discussed.

2. Multiparametric Linear Programming

We consider the following multiparametric linear program:

$$(\mathrm{LP}_\theta) \quad \max_x \{c^T x : Ax \le b + F\theta\},$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $F \in \mathbb{R}^{m \times p}$ are data, $x \in \mathbb{R}^n$ is the decision variable vector, and $\theta \in \mathbb{R}^p$ is the parameter vector. Let $\bar{\Theta}$ denote the set of vector parameters θ such that (LP_θ) has a feasible solution, and let Θ be any polyhedral subset of $\bar{\Theta}$. The dual of problem (LP_θ) is

$$(\mathrm{LD}_\theta) \quad \min_\lambda \{(b + F\theta)^T \lambda : A^T \lambda = c, \lambda \ge 0\}.$$

Note that the feasible region of problem (LD_θ) does not depend on the parameter. Since problem (LP_θ) is feasible for all $\theta \in \Theta$, problem (LD_θ) cannot be unbounded for all $\theta \in \Theta$. If (LD_θ) is infeasible, then (LP_θ) is unbounded for all $\theta \in \Theta$. On the other hand, if (LD_θ) is feasible, then (LP_θ) has a finite optimum for all $\theta \in \Theta$. In order to avoid trivial cases, we assume that (LD_θ) is feasible. As a consequence, we may further assume that the columns

of A are linearly independent. Furthermore, we assume that $\Theta \subseteq \mathbb{R}^p$ is full-dimensional; i.e., the smallest affine subspace of $\mathbb{R}^p$ containing Θ is $\mathbb{R}^p$ itself. This condition can be achieved always by a suitable linear transformation, as shown in Ref. 7, and implies the linear independence of the columns of F.

The value function $\phi: \Theta \mapsto \mathbb{R}$ associates with every $\theta \in \Theta$ the correspond-responding optimal objective value of (LP_θ) and (LD_θ). Under the assumption that the columns of A are linearly independent, the feasible region of (LD_θ) has at least one vertex; let V denote its vertex set. For all $\theta \in \Theta$, we may write

$$\begin{aligned}\phi(\theta) &= \max_x \{c^T x: Ax \leq b + F\theta\} \\ &= \min_\lambda \{(b + F\theta)^T \lambda: A^T \lambda = c, \lambda \geq 0\},\end{aligned}$$

where the above assumptions imply that both problems have a finite optimum; then,

$$\phi(\theta) = \min_\lambda \{\lambda^T b + \lambda^T F\theta: \lambda \in V\}. \tag{1}$$

Since ϕ is the pointwise minimum of a finite set of affine functions of θ, ϕ is a continuous, concave, and piecewise affine function of θ. Multiparametric right-hand side linear programming amounts to determining the value function $\phi(\theta)$ and an optimizer $x^*(\theta)$ of (LP_θ) as explicit functions of θ, for all $\theta \in \Theta$.

Let $\bar{\theta} \in \Theta$ and let $\bar{x}$ and $\bar{\lambda}$ be optimizers for $(\mathrm{LP}_{\bar{\theta}})$ and $(\mathrm{LD}_{\bar{\theta}})$ respectively. Let $\bar{P}$ be the matrix formed by the rows of A corresponding to the constraints of $(\mathrm{LP}_{\bar{\theta}})$ satisfied as equalities at $\bar{x}$, and let $\bar{D}$ be the matrix formed by the rows of A corresponding to strictly positive entries of $\bar{\lambda}$. We say that primal degeneracy occurs if the rows of $\bar{P}$ are linearly dependent; dual degeneracy occurs if the columns of $\bar{D}$ are linearly dependent.

Degeneracy plays a central role in multiparametric linear programming: it is a frequent phenomenon and implies usually the existence of multiple optimizers, which may cause redundancies in the description of both ϕ and x^*. For this reason, giving exact form to ϕ and x^* is theoretically handy, but involves technical complications that seem unavoidable, also when interior-point methods are used. The main purpose of this paper is to design an approximate method that is conceptually simple and unaffected by degeneracy.

3. Algorithm

In this section and in the next one, our terminology is taken from Ref. 11. In particular, given a real matrix M, we denote by cone(M) the conical hull of the column vectors of M. In addition, we define a simplex as the convex hull of affinely independent vectors.

Let $\theta^0, \theta^1, \ldots, \theta^p \in \mathbb{R}^p$ be affinely independent points in $\bar{\Theta}$; define Θ as the following p-dimensional simplex:

$$\Theta = \left\{ \theta \in \mathbb{R}^p : \theta = \sum_{i=0}^{p} \mu_i \theta^i, \ \sum_{i=0}^{p} \mu_i = 1, \mu_i \geq 0, i = 0, \ldots, p \right\}.$$

Let x^i and λ^i be optimizers (LP_{θ^i}) and (LD_{θ^i}), respectively, for all $i = 0, 1, \ldots, p$. Define the matrices

$$M = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \theta^0 & \theta^1 & \ldots & \theta^p \end{bmatrix}, \qquad X = [x^0, x^1, \ldots, x^p], \tag{2}$$

and note that M is nonsingular by construction. Define the functions

$$\hat{x}(\theta) = XM^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix}, \qquad \hat{\phi}(\theta) = c^T \hat{x}(\theta). \tag{3}$$

Finally, define

$$f(\theta) = \min\{ b^T \lambda^i + \theta^T F^T \lambda^i : i = 0, 1, \ldots, p \}.$$

Proposition 3.1. The system of linear inequalities $M^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix} \geq 0$ is a minimal representation of Θ.

Proof. Consider the following equivalence relations:

$$\theta \in \Theta \Leftrightarrow \exists \mu \in \mathbb{R}^{p+1} : \mu \geq 0,$$

$$\begin{bmatrix} 1 \\ \theta \end{bmatrix} = M\mu \ \Leftrightarrow \ M^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix} \geq 0.$$

Minimality follows from the linear independence of the rows of M^{-1}. □

Proposition 3.2. If $\theta \in \Theta$, then $\hat{x}(\theta)$ is a feasible solution of (LP_θ) and $\phi(\theta) - \hat{\phi}(\theta) \geq 0$.

Proof. The vector

$$\mu = M^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix}$$

is the unique solution of

$$M\mu = \begin{bmatrix} 1 \\ \theta \end{bmatrix}.$$

If $\theta \in \Theta$, then

$$\mu \geq 0 \quad \text{and} \quad \sum_i \mu_i = 1.$$

Hence,

$$\begin{aligned} A\hat{x}(\theta) &= \sum_i \mu_i A x^i \\ &\leq \sum_i \mu_i (b + F\theta^i) \\ &= b + F\theta. \end{aligned}$$

Since $\hat{\phi}(\theta)$ is the objective value corresponding to $\hat{x}(\theta)$, we have

$$\hat{\phi}(\theta) \leq \phi(\theta). \qquad \square$$

Proposition 3.3. If $\theta \in \Theta$, then $f(\theta) \geq \phi(\theta)$.

Proof. This is an immediate consequence of the definition of f and equation (1). $\square$

Proposition 3.4. Let $a = c^T X M^{-1}$; let a_0 be the first entry of a; and let $\bar{a}$ be obtained from a by removing a_0. Let

$$\epsilon^* = \max_{t,\,\theta} \{t - \bar{a}^T\theta : M^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix} \geq 0,\ t - (\lambda^i)^T F\theta \leq (\lambda^i)^T b,\ i = 0, 1, \ldots, p\} - a_0, \quad (4)$$

where $t \in \mathbb{R}$ is an auxiliary variable. Then, $\phi(\theta) - \hat{\phi}(\theta) \leq \epsilon^*$ for all $\theta \in \Theta$.

Proof. Since

$$\phi(\theta) \leq f(\theta),$$

we have

$$\phi(\theta) - \hat{\phi}(\theta) \leq f(\theta) - \hat{\phi}(\theta), \qquad \text{for all } \theta \in \Theta.$$

Maximizing the right-hand side of the last inequality amounts to computing

$$\begin{aligned}
&\max_{\theta}\{f(\theta)-\hat{\phi}(\theta):\theta\in\Theta\}\\
&=\max_{\theta}\{f(\theta)-c^T\hat{x}(\theta):\theta\in\Theta\}\\
&=\max_{\theta}\left\{f(\theta)-c^TXM^{-1}\begin{bmatrix}1\\ \theta\end{bmatrix}:\theta\in\Theta\right\}\\
&=\max_{t,\theta}\left\{t-c^TXM^{-1}\begin{bmatrix}1\\ \theta\end{bmatrix}:\theta\in\Theta,\ t\le b^T\lambda^i+\theta^TF^T\lambda^i,\ i=0,1,\ldots,p\right\}\\
&=\max_{t,\theta}\{t-\bar{a}^T\theta:\theta\in\Theta,\ t\le b^T\lambda^i+\theta^TF^T\lambda^i,\ i=0,1,\ldots,p\}-a_0\\
&=\epsilon^*.
\end{aligned}$$

□

We are in a position to state a basic approximation algorithm for (LP_θ).

Algorithm 3.1

Input. A simplex Θ given by its vertices θ^i, $i=0,1,\ldots,p$, a solution x^i of (LP_{θ^i}), and a solution λ^i of (LD_{θ^i}), $i=0,1,\ldots,p$.

Output. Function $\hat{\phi}$ and corresponding $\hat{x}$ such that $\phi(\theta)-\hat{\phi}(\theta)\le\epsilon$ for all $\theta\in\Theta$.

Step 1. Build M and X as defined in (2).

Step 2. If M is nonsingular, then compute M^{-1} and

$$a=(a_0,\bar{a})=c^TXM^{-1};$$

solve

$$\max\left\{t-\bar{a}^T\theta:M^{-1}\begin{bmatrix}1\\ \theta\end{bmatrix}\ge 0,\ t-(\lambda^i)^TF\theta\le(\lambda^i)^Tb,\ i=0,1,\ldots,p\right\},$$

getting (t^*,θ^*); else, return.

Step 3. If $a_0-t^*+\bar{a}^T\theta^*\le\epsilon$, then save M^{-1} and X, and return; else, solve (LP_{θ^*}) and (LD_{θ^*}), getting x^* and λ^*.

Step 4. For all $i=0,1,\ldots,p$, replace the ith vertex of Θ by θ^*; call Algorithm 3.1 on the new simplex.

Note that, if M is singular, then the simplex Θ is not full-dimensional. In this case, Algorithm 3.1 simply returns control. Note further that saving the matrices M^{-1} and X is justified by Proposition 3.2 and definitions (3).

In general, let x^* and λ^* be the optimizers obtained in Step 3 of Algorithm 3.1. Then,

$$\begin{aligned}\epsilon^{**} &= \phi(\theta^*) - \hat{\phi}(\theta^*)\\ &= c^T x^* - a_0 - \bar{a}^T \theta^*\end{aligned}$$

is a lower bound on the absolute error obtained by using $\hat{x}$ instead of the exact optimizer inside the whole Θ. If $\epsilon^{**} < \epsilon$, then there is the possibility that the actual error is less than ϵ. Hence, we may solve the problem of Step 2 with the new constraint

$$t - (\lambda^*)^T F\theta \leq (\lambda^*)^T b$$

in order to get a new upper bound, tighter than the previously computed ϵ^*. This process can be repeated until either we get an upper bound less than ϵ or we get a lower bound greater than ϵ; the current simplex will be partitioned only if the latter case occurs. In general, this refinement may reduce the number of simplices, while maintaining the prescribed approximation. However, the algorithm may be further improved, as illustrated by the following example.

Example 3.1. Consider as (LP_θ) the following simple problem, with a scalar parameter:

$$\begin{aligned}\max\quad & 6x_1 + 10x_2,\\ \text{s.t.}\quad & x_1 + x_2 \leq -1.25 + \theta,\\ & x_2 \leq 0.75 - \theta,\\ & -x_1 \leq 0,\\ & 2x_1 + x_2 \leq 0.\end{aligned}$$

Let the interval $\Theta = [\theta^0, \theta^1] = [1.00, 1.20]$ be the current simplex, and let $\epsilon = 0.200$. The primal and dual solutions are reported below:

$$x^0 = (0.000, -0.250), \quad \lambda^0 = (10, 0, 4, 0), \quad (\lambda^0)^T F\theta + (\lambda^0)^T b = 10\theta - 12.50,$$
$$x^1 = (0.225, -0.450), \quad \lambda^1 = (0, 7, 0, 3), \quad (\lambda^1)^T F\theta + (\lambda^1)^T b = -7\theta + 5.25.$$

In Figure 1(a), the solid lines depict the upper approximation $f(\theta)$ of $\phi(\theta)$ inside Θ, obtained from λ^0 and λ^1; the lower approximation $\hat{\phi}(\theta)$ is depicted by the dashed line. Step 2 of Algorithm 3.1 generates $(t^*, \theta^*) = (-2.06, 1.044)$, and the upper bound on the error evaluated in Step 3 is

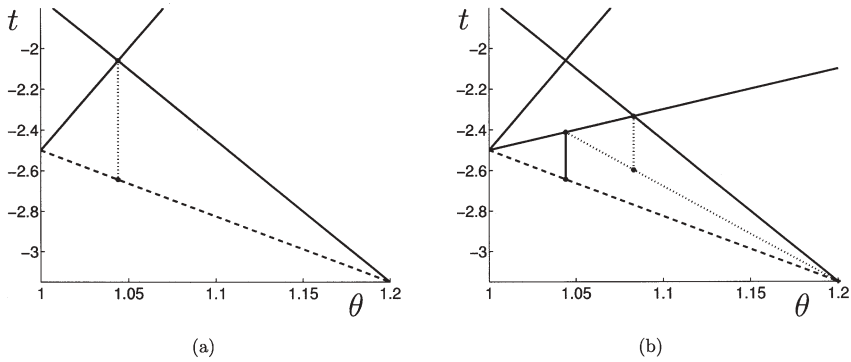

Fig. 1. A graphical representation of Example 3.1.

$\epsilon^* = 0.583$, which geometrically is the length of the dotted line in Figure 1(a). Since $\epsilon^* > \epsilon$, we solve (LP_{θ^*}) and (LD_{θ^*}), getting

$$x^* = (0.088, -0.294), \quad \lambda^* = (6, 4, 0, 0), \quad (\lambda^*)^T F\theta + (\lambda^*)^T b = 2\theta - 4.50.$$

The corresponding lower bound on the error is $\epsilon^{**} = \phi(\theta^*) - \hat{\phi}(\theta^*) = 0.352$. Since the lower bound [length of the solid vertical line in Figure 1(b)] is still greater than ϵ, we should split Θ in two parts: $\Theta_1 = [1.000, 1.044]$ and $\Theta_2 = [1.044, 1.200]$. In the next recursive call, Θ_1 has a zero upper bound on the error and then it is not divided. On the other hand, in Θ_2 the upper and lower bounds are both equal to $0.263 > \epsilon$. As a consequence, Θ_2 should be split into $\Theta_3 = [1.044, 1.083]$ and $\Theta_4 = [1.083, 1.200]$. For both these last simplices, an exact description is readily available. In conclusion, even using lower bound computation, the basic algorithm builds up an exact description of the value function using three simplices, while for an exact description just two simplices are sufficient.

Example 3.1 points out a problem due to the presence of multiple optimizers of the dual problem (LD_θ) for some given θ; this may happen because of primal degeneracy. More specifically, our basic algorithm requires three simplices because it takes $\lambda^0 = [10, 0, 4, 0]$ as optimizer of (LD_{θ^0}). However, we may note that $\lambda^* = [6, 4, 0, 0]$, the optimizer of (LD_{θ^*}), is an optimizer of (LD_{θ^0}) too. If we take $\lambda^0 = [6, 4, 0, 0]$, then the algorithm would return the exact description of ϕ in Θ. In principle, we may overcome completely the trouble pointed out by Example 3.1 enumerating all dual basic feasible solutions at each vertex of Θ. In practice, such enumeration may be very expensive in computational terms. Nevertheless, we can make the algorithm more robust at low expense by using the following argument.

Consider any vertex of Θ, say θ^0. Let $v \in \mathbb{R}^p$ be any direction pointing from θ^0 to the interior of Θ, so that $\theta^0 + \gamma v \in \Theta$ for any sufficiently small

$\gamma > 0$. Consider the following monoparametric linear program:

$$(\widetilde{\mathrm{LP}}_\gamma) \quad \max\{c^T x: Ax \le b^0 + \gamma v^0\},$$

and its dual

$$(\widetilde{\mathrm{LD}}_\gamma) \quad \max\{(b^0 + \gamma v^0)^T \lambda: A^T \lambda = c, \lambda \ge 0\},$$

where

$$b^0 = b + F\theta^0 \quad \text{and} \quad v^0 = Fv.$$

Let g denote the value function of $(\widetilde{LP}_\gamma)$. Since g is a 1-dimensional cut of ϕ, we have that g is concave, continuous, and piecewise affine on the real closed interval $\bar{\Gamma} = [\gamma_{\inf}, \gamma_{\sup}]$, where

$$\gamma_{\inf} = \inf\{\gamma: \theta^0 + \gamma v^0 \in \bar{\Theta}\},$$

$$\gamma_{\sup} = \sup\{\gamma: \theta^0 + \gamma v^0 \in \bar{\Theta}\}.$$

For proof of the following result, see e.g. Theorem 8.2 in Ref. 12.

Proposition 3.5. For any $\gamma \in [\gamma_{\inf}, \gamma_{\sup}]$, the left and right derivatives $g'_-(\gamma)$ and $g'_+(\gamma)$ of g at γ are respectively

$$g'_-(\gamma) = \max_\lambda\{(v^0)^T \lambda: \lambda \in \Lambda(\gamma)\},$$

$$g'_+(\gamma) = \min_\lambda\{(v^0)^T \lambda: \lambda \in \Lambda(\gamma)\},$$

where $\Lambda(\gamma)$ denotes the set of optimizers of $(\widetilde{\mathrm{LD}}_\gamma)$.

If x^0 is an optimizer of (LP_{θ^0}), then x^0 is also an optimizer of $(\widetilde{\mathrm{LP}}_0)$, and by complementary slackness we may write

$$\Lambda(0) = \{\lambda: A^T \lambda = c, (b^0 - Ax^0)^T \lambda = 0, \lambda \ge 0\}.$$

Let λ^0 be an optimizer of

$$\min_\lambda\{(v^0)^T \lambda: A^T \lambda = c, (b^0 - Ax^0)^T \lambda = 0, \lambda \ge 0\}. \tag{5}$$

Then, as a consequence of Proposition 3.5, $(v^0)^T \lambda^0$ is the right derivative of g in $\gamma = 0$, i.e., the exact slope of ϕ on θ^0 along direction v. For this reason, if we take λ^0 instead of an arbitrary point inside $\Lambda(0)$, we may in general obtain a better upper approximation f of ϕ in a neighbor of θ^0 contained in Θ. By introducing this device, we get our final algorithm, described as follows.

Algorithm 3.2.

Input. A simplex Θ given by its vertices θ^i, $i = 0, 1, \ldots, p$, a solution x^i of (LP_{θ^i}), and a solution of λ^i of (LD_{θ^i}), $i = 0, 1, \ldots, p$.

Output. Function $\hat{\phi}$ and corresponding $\hat{x}$ such that $\phi(\theta) - \hat{\phi}(\theta) \leq \epsilon$ for all $\theta \in \Theta$.

Step 1. Build M and X as defined in (2).

Step 2. If M is nonsingular, then compute M^{-1},

$$a = (a_0, \bar{a}) = c^T X M^{-1},$$

and set $q = p$; else, return.

Step 3. Solve

$$\max\left\{t - \bar{a}^T\theta : M^{-1}\begin{bmatrix}1\\ \theta\end{bmatrix} \geq 0,\ t - (\lambda^i)^T F\theta \leq (\lambda^i)^T b,\ i = 0, 1, \ldots, q\right\},$$

getting (t^*, θ^*). If

$$a_0 - t^* + \bar{a}^T\theta^* \leq \epsilon,$$

then save M^{-1} and X, and return.

Step 4. Solve (LP_{θ^*}) and (LD_{θ^*}), getting x^* and λ^*. If $a_0 - c^T x^* < \epsilon$, then set $\lambda^{q+1} = \lambda^*$ and $q = q + 1$, and go to Step 3.

Step 5. For all $i = 0, 1, \ldots, p$, let

$$v^i = \sum(\theta^j - \theta^i : j = 0, 1, \ldots, p;\ j \neq i), \quad w^i = Fv^i;$$

solve

$$\min\{(w^i)^T\lambda : A^T\lambda = c, (b + F\theta^* - Ax^*)^T\lambda = 0, \lambda \geq 0\},$$

getting $\hat{\lambda}$; set $\lambda^i = \hat{\lambda}$; replace the ith vertex of Θ by θ^*; call Algorithm 3.2 on the new simplex.

In Algorithm 3.2, the direction v is obtained by summing up the difference vectors $\theta^1 - \theta^0, \ldots, \theta^p - \theta^0$.

We remark that both Algorithm 3.1 and Algorithm 3.2 yield two piecewise affine functions $\hat{x}: \Theta \mapsto \mathbb{R}^n$ and $\hat{\phi}: \Theta \mapsto \mathbb{R}$ such that:

(i) $\hat{x}(\theta)$ is a primal feasible solution of (LP_θ), for all $\theta \in \Theta$;
(ii) $\hat{\phi}(\theta) = c^T\hat{x}(\theta)$, for all $\theta \in \Theta$;
(iii) $0 \leq \phi(\theta) - \hat{\phi}(\theta) \leq \epsilon$, for all $\theta \in \Theta$.

Note that $\hat{x}$ and $\hat{\phi}$ may be not continuous on the boundary of the returned simplices. As a consequence, the approximate optimizer and value function

may be defined more than once for some $\theta \in \Theta$, though this fact can happen only on a subset of Θ with null measure.

Example 3.2. We can visualize the output of Algorithm 3.2 by using the following two-parameter example, proposed in Ref. 7:

$$\min \quad x_1 + x_2 + x_3 + x_4, \tag{6a}$$

$$\text{s.t.} \quad -x_1 + x_5 \le 0, \quad -x_1 - x_5 \le 0, \quad -x_2 + x_6 \le 0, \tag{6b}$$

$$-x_2 - x_6 \le 0, \quad -1 \le x_5 \le 1, \quad -1 \le x_6 \le 1, \tag{6c}$$

$$-x_3 \le \theta_1 + \theta_2, \quad -x_3 - x_5 \le \theta_2, \quad -x_3 \le -\theta_1 - \theta_2, \tag{6d}$$

$$-x_3 + x_5 \le -\theta_2, \quad -x_4 - x_5 \le \theta_1 + 2\theta_2, \quad -x_4 - x_5 - x_6 \le \theta_2, \tag{6e}$$

$$-x_4 + x_5 \le -\theta_1 - 2\theta_2, \quad -x_4 + x_5 + x_6 \le -\theta_2. \tag{6f}$$

In Ref. 7, the problem is studied in the square

$$\{(\theta_1, \theta_2): -2.5 \le \theta_1 \le 2.5, \ -2.5 \le \theta_2 \le 2.5\}.$$

Since the solution is completely symmetric with respect to the line

$$\theta_1 + \theta_2 = 0,$$

we study the problem only inside

$$\Theta = \{(\theta_1, \theta_2): \theta_1 \ge -2.5, \theta_2 \ge -2.5, \theta_1 + \theta_2 \le 0\}.$$

The exact solution proposed in Ref. 7 is depicted in Figure 2(a), where the gray scale is related to the slope of the value function. We note that the value function could be described by four convex polygonal regions, corresponding to different gray scale. However, by dual degeneracy, it is not possible to give a unique expression for the optimizer inside each polygonal region. In fact, the exact solution proposed in Ref. 7 divides Θ into 7 characteristic regions. The triangulations of Θ obtained by Algorithm 3.2 for increasing tolerances are depicted in Figures 2(b)–2(d). When $\epsilon = 0.0$, we get an exact solution of the parametric problem, obtained by 15 simplices. When $\epsilon = 1.2$, we get 5 simplices; and when $\epsilon = 1.4$, only 3 triangles are needed. If we consider Θ as a unique region, we get an absolute error bound of 4.333.

4. Optimizer Error Bounding

The algorithms proposed in the previous section produce an approximate solution of problem (LP_θ) where the absolute error in the objective

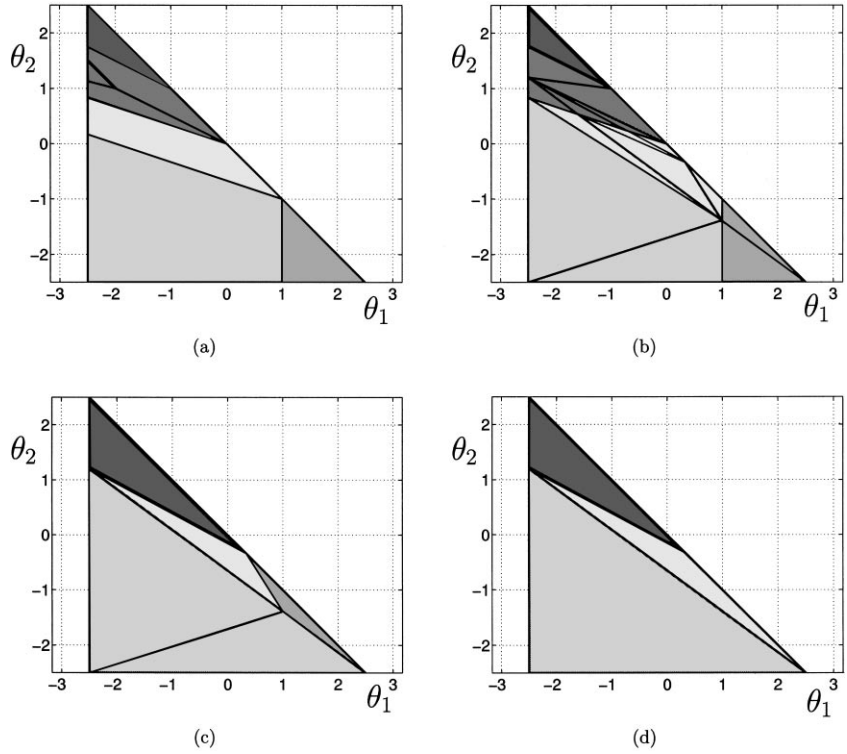

Fig. 2. Exact and approximate characteristic regions in problem (6): (a) exact solution proposed in Ref. 7; (b) $\epsilon = 0.0$, 15 regions; (c) $\epsilon = 1.2$, 5 regions; (d) $\epsilon = 1.4$, 3 regions.

value is at most ϵ. In this section, we discuss a global bound on the absolute error with respect to the optimizer. In the following, let $\|\cdot\|$ denote any L_p norm. Given a set $S \subseteq \mathbb{R}^n$ and a point $x \in \mathbb{R}^n$, we define the distance of x from S as

$$d(x, S) = \inf\{\|x - s\| : s \in S\}.$$

Let $\hat{x}: \Theta \mapsto \mathbb{R}^n$ and $\hat{\phi}: \Theta \mapsto \mathbb{R}$ denote the functions produced by Algorithm 3.1 or Algorithm 3.2, and let $X^*(\theta)$ denote the set of optimizers of (LP_θ), for all $\theta \in \Theta$. Although some Lipschitzian properties of convex polyhedra and linear programming can be found in the literature (see e.g. Refs. 13–14), explicit results relating the distance of a feasible solution from an optimal one and the gap on the objective value are not known to the author. For this reason, we state and prove the following theorem.

Theorem 4.1. There exists a positive scalar C such that $d(\hat{x}(\theta), X^*(\theta)) \leq C[\phi(\theta) - \hat{\phi}(\theta)]$ for all $\theta \in \Theta$.

Proof. Let $\theta \in \Theta$. If

$$\phi(\theta) - \hat{\phi}(\theta) = 0,$$

then $\hat{x}(\theta) \in X^*(\theta)$ and we are done. So, let

$$\phi(\theta) - \hat{\phi}(\theta) > 0.$$

Under the assumption that the matrix A has linearly independent columns, the polyhedron

$$P(\theta) = \{x: Ax \leq b + F\theta\}$$

has at least one vertex, for all $\theta \in \Theta$. Let the rows of A be indexed by $\{1, 2, \ldots, m\}$. For any $S \in \{1, 2, \ldots, m\}$, let A_B denote the submatrix of A consisting of rows indexed by S. Every one-dimensional face of $P(\theta)$ is identified by a subset B of $n-1$ indices in $\{1, 2, \ldots, m\}$ such that the rows of A_B are linearly independent (cf. Ref. 11, p. 105). Let $\mathscr{B}$ denote the collection of all such subsets, and note that every one-dimensional face of $P(\theta)$ not parallel to the hyperplane $\{x: c^T x = \phi(\theta)\}$ can be identified by a set $B \in \mathscr{B}$ such that system

$$A_B x = 0, \qquad c^T x = -1$$

has a unique solution. Let R be the finite real matrix whose columns are all the solutions of the previous systems for all $B \in \mathscr{B}$.

Let $L^*(\theta)$ be the set of all one-dimensional faces of $P(\theta)$ with exactly one point lying on $X^*(\theta)$. Each element of $L^*(\theta)$ may be written as $\{x^* + \alpha r: 0 \leq \alpha \leq \bar{\alpha}\}$, where:

(i) x^* is a vertex of $X^*(\theta)$;
(ii) r is a column of R;
(iii) $\bar{\alpha}$ is either a positive scalar or $+\infty$.

Define the polyhedron

$$P_\delta(\theta) = P(\theta) \cap \{x: c^T x = \phi(\theta) - \delta\}.$$

We prove the following claim:

For sufficiently small $\delta > 0$, if $\tilde{x} \in P_\delta(\theta)$, then

$$\tilde{x} - x^* \in \text{cone}(R) \qquad \text{for some } x^* \in X^*(\theta). \tag{7}$$

Note first that

$$P_0(\theta) = X^*(\theta).$$

Further note that, for sufficiently small $\delta > 0$, there is a one-to-one correspondence between the vertices of $P_\delta(\theta)$ and the elements of $L^*(\theta)$. In particular, if v^i is a vertex of $P_\delta(\theta)$, then there is a vertex x^i of $X^*(\theta)$ and a column r^i of R such that

$$v^i = x^i + \alpha_i r^i, \qquad \text{for some } \alpha_i > 0.$$

Moreover, for sufficiently small $\delta > 0$, $P_\delta(\theta)$ and $P_0(\theta)$ have the same characteristic cone. Let $v^1, v^2, \ldots, v^p$ and $u^1, u^2, \ldots, u^q$ be respectively the vertices and the external rays of $P_\delta(\theta)$. We have that $\tilde{x} \in P_\delta(\theta)$ if and only if there are nonnegative weights $\mu_1, \mu_2, \ldots, \mu_p$ and $\nu_1, \nu_2, \ldots, \nu_q$, with $\sum_{i=1}^p \mu_i = 1$, and such that

$$\tilde{x} = \sum_{i=1}^{p} \mu_i v^i + \sum_{k=1}^{q} \nu_k u^k.$$

But then, for sufficiently small $\delta > 0$, we have

$$\tilde{x} = \sum_{i=1}^{p} \mu_i x^i + \sum_{k=1}^{q} \nu_k u^k + \sum_{i=1}^{p} \mu_i \alpha_i r^i = x^* + r^*,$$

for some $x^* \in X^*(\theta)$ and $r^* \in \text{cone}(R)$. This proves claim (7).

The next step is to prove the following claim:

$$\text{There exists a point } x^* \in X^*(\theta) \text{ such that } \hat{x}(\theta) - x^* \in \text{cone}(R). \tag{8}$$

Suppose by contradiction that

$$\hat{x}(\theta) - x^* \notin \text{cone}(R), \qquad \text{for all } x^* \in X^*(\theta).$$

Then, for any positive scalar τ, we have

$$\tau(\hat{x}(\theta) - x^*) \notin \text{cone}(R), \qquad \text{for all } x^* \in X^*(\theta).$$

In particular, since δ in claim (7) can be taken arbitrarily small, we may assume that

$$0 < \delta < \phi(\theta) - \hat{\phi}(\theta)$$

and fix

$$\tau = \delta / [\phi(\theta) - \hat{\phi}(\theta)],$$

getting $0 < \tau < 1$. For any $x^* \in X^*(\theta)$, let

$$\tilde{x} = x^* + \tau(\hat{x}(\theta) - x^*).$$

Then,

$$\tilde{x} = \tau \hat{x}(\theta) + (1 - \tau) x^* \in P(\theta),$$

and since

$$c^T \tilde{x} = \phi(\theta) - \delta,$$

we have $\tilde{x} \in P_\delta(\theta)$. But

$$\tilde{x} - x^* = \tau(\hat{x}(\theta) - x^*) \notin \text{cone}(R).$$

Since this happens for all $x^* \in X^*(\theta)$, we contradict claim (7). This proves claim (8).

To conclude the proof, let $x^* \in X^*(\theta)$ be such that

$$\hat{x}(\theta) - x^* \in \text{cone}(R),$$

and let s be the number of columns of R. Then,

$$\hat{x}(\theta) - x^* = \sum_{j=1}^{s} \rho_j r^j$$

for nonnegative scalars $\rho_1, \rho_2, \ldots, \rho_s$. Since by construction

$$c^T r^j = -1, \qquad \text{for all } j,$$

we have

$$\begin{aligned} \sum_{j=1}^{s} \rho_j &= -\sum_{j=1}^{s} \rho_j c^T r^j \\ &= -c^T(\hat{x}(\theta) - x^*) \\ &= \phi(\theta) - \hat{\phi}(\theta). \end{aligned}$$

As a consequence,

$$\begin{aligned} \|\hat{x}(\theta) - x^*\| &= \left\| \sum_{j=1}^{s} \rho_j r^j \right\| \leq \sum_{j=1}^{s} \rho_j \left\| r^j \right\| \\ &\leq \max_{j=1,\ldots,s} \{\left\| r^j \right\|\} \sum_{j=1}^{s} \rho_j \\ &= \max_{j=1,\ldots,s} \{\left\| r^j \right\|\} (\phi(\theta) - \hat{\phi}(\theta)). \end{aligned}$$

Since

$$d(\hat{x}(\theta),\ X^*(\theta)) \leq \|\hat{x}(\theta) - x^*\|,$$

by letting

$$C = \max_{j=1,\ldots,s} \{||r^j||\},$$

we get the assertion. □

As

$$\phi(\theta) - \hat{\phi}(\theta) \leq \epsilon, \qquad \text{for all } \theta \in \Theta,$$

we get immediately the following global error bound.

Corollary 4.1. There exists a positive scalar C such that $d(\hat{x}(\theta), X^*(\theta)) \leq C\epsilon$ for all $\theta \in \Theta$.

Corollary 4.1 may be used to bound a priori the absolute error on the obtained suboptimizer with respect to any L_p norm. Note that we may obtain the constant C by computing all basic (possibly infeasible) solutions of the polyhedron

$$\{x \in \mathbb{R}^n : Ax \leq 0, c^T x = -1\}.$$

This can be done efficiently using the reverse search algorithm by Avis and Fukuda (Ref. 15). Hence, the constant C may be practically computed, though the required computational time may exponentially grow with the input size.

5. Complexity Analysis

The complexity of Algorithm 3.1 and Algorithm 3.2 is dominated by the solution of linear programs. Since the time complexity of linear programming depends on the algorithm implemented by the used solver, we consider an LP solver as an oracle, and we evaluate the complexity of Algorithm 3.1 and Algorithm 3.2 by the maximum number of linear programs that they have to solve. Accordingly, we denote by $\mathrm{lp}(m, n)$ a time complexity of solving $\max_y \{d^T y : Ey \leq w\}$, where $d \in \mathbb{R}^n$, $E \in \mathbb{R}^{m \times n}$, and $w \in \mathbb{R}^m$. For a deeper discussion of this notation, see Ref. 16.

We first focus on Algorithm 3.1. The complexity of each recursive call is dominated by the solution of two linear programs, namely, the problem solved in Step 2 (say problem LP′) and the problem solved in Step 3 (say problem LP″). Note that LP′ has $2(p+1)$ constraints and $p+1$ free variables, whereas LP″ has m constraints and n free variables. Then, the time complexity of each call of Algorithm 3.1 is

$$O(\mathrm{lp}(2p+2, p+1) + \mathrm{lp}(m, n)).$$

The total number of calls depends on the number of simplices to be generated in order to build an approximation of $\phi(\theta)$ satisfying the required tolerance. Such a number is a complex function of the input, though we may guess that it is exponential in m, n, p (cf. Ref. 17). On the other

hand, according to the literature on vertex enumeration of polyhedra (see e.g. Ref. 15), we can express the overall complexity of Algorithm 3.1 as a function of the output size, as stated in the following result.

Theorem 5.1. Algorithm 3.1 builds up an approximate description of ϕ using N_1 simplices in time $O((2N_1 - 1)\mathrm{lp}(2p + 2, p + 1) + (N_1 + p)\mathrm{lp}(m, n))$.

In order to prove Theorem 5.1, we need a few definitions and a technical lemma. Given a rooted tree T, a node u of T is a leaf if u has no child; otherwise, we say that u is internal. A rooted tree T is full if every node of T either is a leaf or has at least two children (cf. Ref. 18, Chapter 5). The smallest full tree is composed by three nodes: the root and its two children. Finally, let $t(N)$ denote the maximum number of nodes of a full tree with N leaves.

It is easy to prove by contradiction that, if T is a full tree with N leaves and $t(N)$ nodes, then T must be binary. Moreover, it is easy to prove by induction on N that, if T is a full binary tree with N leaves, then T has exactly $2N - 1$ nodes. We deduce the next result.

Lemma 5.1. The following identity holds: $t(N) = 2N - 1$.

Proof of Theorem 5.1. Each time Algorithm 3.1 is called, a simplex in the parameter space is explored. If the upper bound on the error inside the simplex is less than the prescribed tolerance, the current simplex is returned; otherwise, the simplex is subdivided in two or more smaller simplices and a recursive call is performed for each of them. Thus, the exploration of the parameter space performed by Algorithm 3.1 may be visualized by a search tree, where the nodes correspond to the explored simplices and the arcs correspond to the recursive calls. The assumed N_1 simplices returned by the algorithm are the leaves of the search tree. Let I denote the number of internal nodes. When Algorithm 3.1 stops, a problem LP′ has been solved for every node of the search tree and, in addition, a problem LP″ has been solved for each internal node. Furthermore, $p + 1$ problems LP″ have been solved to generate the input for the first call. Thus, in order to generate N_1 simplices, Algorithm 3.1 has solved $I + N_1$ problems LP′ and $I + p + 1$ problems LP″. If $N_1 = 1$, then $I = 0$ and the stated time complexity holds trivially. If $N_1 > 1$, we have to derive an upper bound for I. With this respect, note that each internal node of the search tree has a number of children ranging from 2 to $p + 1$. Thus, the search tree is full, and we may write

$$I + N_1 \leq t(N_1).$$

As a consequence of Lemma 5.1, the last inequality implies

$$I \leq N_1 - 1,$$

from which the stated time complexity follows. □

Corollary 5.1. Algorithm 3.1 has a time complexity polynomial in the output size, provided a polynomial LP solver is used.

We now turn to Algorithm 3.2. Here, at each recursive call, a problem LP′ could be solved several times, each time with a new constraint added. Let K be the maximum number of times a problem LP′ may be solved during a recursive call. Then, the time complexity of each call of Algorithm 3.2 is $O(K\text{lp}(2(p+K), p+1) + \text{lp}(m, n))$. Though in general we cannot exclude that K grows exponentially with m and n, our preliminary tests indicate that in practice K is generally very low, as will be shown in Section 6. Similarly to Theorem 5.1, we can prove the following theorem.

Theorem 5.2. Algorithm 3.2 builds up an approximate description of ϕ using N_2 simplices in time $O((2N_2 - 1)(1 + K)\text{lp}(2p + 2K, p + 1) + (N_2 + p)\text{lp}(m, n))$.

6. Computational Tests

In order to test the practical behavior of Algorithm 3.1 and Algorithm 3.2, we implemented them in Matlab 6 using the standard Optimization Toolbox. The program were run on a Pentitum II PC. No attempts were made to optimize the code; in particular, each linear program is solved from scratch. We tested Algorithm 3.1 and Algorithm 3.2 on randomly generated instances, having the form

$$\max\left\{\sum_{j=1}^{n} x_j : Ax \leq b + F\theta, 0 \leq x_j \leq 10, j = 1, \ldots, n\right\},$$

where A is a dense $m \times n$ matrix, b is an m vector, and F is a dense $m \times p$ matrix. All instances are considered in the parameter set

$$\Theta = \left\{\theta \in \mathbb{R}^p : \theta \geq 0, \sum_{k=1}^{p} \theta_k \leq 1\right\}.$$

The entries of both A and F are obtained by rounding numbers with normal standard distribution. The vector b is obtained from F in such a way that $x = 0$ is always feasible in the vertices of Θ.

In Table 1, we compare the performance of Algorithm 3.1 and Algorithm 3.2 for different number of parameters and different tolerances. The

Table 1. Comparison of the performance of Algorithm 3.1 and Algorithm 3.2 on randomly generated instances.

		A1	A2		
$m \times n \times p$	ϵ	N_1	N_2	K_{MAX}	K_{MEAN}
$20 \times 5 \times 2$	0.50	3.2	2.0	1	0.2857
	0.20	5.2	5.0	0	0.0000
	0.10	8.4	8.2	0	0.0000
	0.01	16.0	15.2	0	0.0000
$20 \times 5 \times 3$	0.50	5.0	4.4	1	0.2857
	0.20	10.6	10.4	1	0.0215
	0.10	23.2	21.6	1	0.0408
	0.01	115.4	79.2	2	0.0335
$20 \times 5 \times 4$	0.50	13.8	12.2	1	0.1165
	0.20	67.4	52.8	2	0.0450
	0.10	110.4	106.2	2	0.0087
	0.01	314.4	265.8	1	0.0199
$20 \times 5 \times 5$	0.50	63.2	55.2	2	0.1043
	0.20	481.4	400.4	2	0.0437
	0.10	1437.6	1175.2	3	0.0712
	0.01	6564.4	5264.4	6	0.0259

values reported are averages among five generated instances. We use the following notation: N_i denotes the number of simplices returned by Algorithm 3.1i, with $i = 1, 2$; K_{MAX} is the maximum number of times a problem LP′ is solved during a recursive call as a consequence of the lower bound computation, evaluated over all five instances; K_{MEAN} is the mean number of times of problem LP′ is solved during a recursive call, averaged over all recursive calls in the five instances.

By comparing the values of N_1 and N_2, we see that the size of the output of Algorithm 3.2 is on the average 19% less than the size of the output of Algorithm 3.1. This fact evidences the effectiveness of the refinements introduced. Focusing on Algorithm 3.2, we see that the values of K_{MEAN} and K_{MAX} are very low. In particular, the overall average number of times a problem LP′ is solved in a recursive call of Algorithm 3.2 turns out to be 0.0329. This fact suggests that, in practice, the constant K introduced in Theorem 5.2 takes very low values.

For what concerns the CPU times, on the tested instances Algorithm 3.1 required times ranging from less than 1 second for the simplest instances with 2 parameters to 4 hours and 52 minutes for the hardest instance with 5 parameters; on the average, Algorithm 3.1 required about 68 seconds. For Algorithm 3.2, the CPU times were roughly doubled. Anyway, we remark that an optimized code should reduce dramatically the computational effort, especially for what concerns Algorithm 3.2.

7. Conclusions

In this paper, we propose a recursive algorithm which generates approximate explicit descriptions of the value function and of an optimizer of a multiparametric right-hand side liner program.

For any given instance, the final outcome of the algorithm is a lookup table of the form

$$\hat{x}(\theta) = X^k (M^k)^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix}, \quad \text{if } (M^k)^{-1} \begin{bmatrix} 1 \\ \theta \end{bmatrix} \geq 0, \qquad \text{for all } k = 1, \ldots, K(\epsilon),$$

where the matrices X^k and M^k are defined according to (2) and $K(\epsilon)$ is the number of returned simplices, that is an unknown nonincreasing function of the prescribed tolerance ϵ. We note that $K(0)$ is at least equal to the number of simplices required by a triangulation of the value function; thus, $K(0)$ may be greater than the number of characteristic regions K^* returned by any one the exact methods proposed in the literature. We note also that, under our working assumptions, there exists a finite ϵ^{MAX} such that $K(\epsilon) = 1$ for all $\epsilon \geq \epsilon^{\mathrm{MAX}}$. As a consequence, there must exist a threshold value $\epsilon^* \geq 0$ such that

$$K(\epsilon) \leq K^*, \qquad \text{for all } \epsilon \geq \epsilon^*;$$

after such a threshold, our method becomes convenient with respect to an exact one in terms of output size. We may expect that the value of ϵ^* heavily relies on the specific instance at hand; evaluating ϵ^* on a significant set of instances coming from real applications may be the topic of future research. Another issue for future research is the data structure used to record all the generated information (Ref. 19). We believe that the recursion tree generated implicitly by our algorithm (and defined in the proof of Theorem 5.1) may be exploited to speedup the on-line use of the lookup table.

Finally, a relevant issue is the extension of our results to more general multi-parametric programming problems, following the lines suggested by Fiacco (Ref. 10, Chapter 9).

References

1. Morari, M., and Lee, J., *Model Predictive Control*: *Past*, *Present*, *and Future*, Computers and Chemical Engineering, Vol. 23, pp. 667–682, 1999.
2. Mayne, D., Rawlings, J., Rao, C., and Scokaert, P., *Constrained Model Predictive Control*: *Stability and Optimality*, Automatica, Vol. 36, pp. 789–814, 2000.

3. Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N., *The Explicit Linear-Quadratic Regulator for Constrained Systems*, Automatica, Vol. 38, pp. 3–20, 2002.
4. Tøndel, P., Johansen, T. A., and Bemporad, A., *An Algorithm for Multi-parametric Quadratic Programming and Explicit MPC Solutions*, Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida, pp. 1199–1204, 2001.
5. Gal, T., *Postoptimal Analysis*, *Parametric Programming*, *and Related Topics*, 2nd Edition, de Gruyter, Berlin, Germany, 1995.
6. Gal, T., and Nedoma, J., *Multiparametric Linear Programming*, Management Science, Vol. 18, pp. 406–422, 1972.
7. Bemporad, A., Borrelli, F., and Morari, M., *A Geometric Algorithm for Multi-parametric Linear Programming*, Journal of Optimization Theory and Applications, Vol. 118, pp. 515–540, 2003.
8. Filippi, C., and Romanin-Jacur, G., *Multiparametric Demand Linear Transportation Problem*, European Journal of Operational Research, Vol. 139, pp. 206–219, 2002.
9. Bemporad, A., and Filippi, C., *Suboptimal Explicit Receding–Horizon Control via Approximate Multiparametric Quadratic Programming*, Journal of Optimization Theory and Applications, Vol. 117, pp. 9–38, 2003.
10. Fiacco, A. V., *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, Academic Press, London, UK, 1983.
11. Schrijver, A., *Theory of Linear and Integer Programming*, Wiley-Interscience, Chichester, UK, 1986.
12. Murty, K. G., *Linear Programming*, Wiley-Interscience, New York, NY, 1983.
13. Walkup, D. W., and Wets, R. J. B., *A Lipschitzian Characterization of Convex Polyhedra*, Proceedings of the American Mathematical Society, Vol. 23, pp. 167–173, 1969.
14. Robinson, S. M., *A Characterization of Stability in Linear Programming*, Operations Research, Vol. 25, pp. 435–447, 1977.
15. Avis, D., and Fukuda, K., *A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra*, Discrete Computational Geometry, Vol. 8, pp. 295–313, 1992.
16. Bemporad, A., Fukuda, K., and Torrisi, F. D., *Convexity Recognition of the Union of Polyhedra*, Computational Geometry, Vol. 18, pp. 141–154, 2001.
17. Murty, K. G., *Computational Complexity of Parametric Linear Programming*, Mathematical Programming, Vol. 19, pp. 213–219, 1980.
18. Cormen, T. H., Leiserson, C. E., and Rivest, R. L., *Introduction to Algorithms*, McGraw-Hill, New York, NY, 1990.
19. Borrelli, F., Baotic, M., Bemporad, A., and Morari, M., *Efficient Online Computation of Constrained Optimal Control Laws*, Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida, pp. 1187–1192, 2001.