

Reverse Search for Parametric Linear Programming

Colin N. Jones

Automatic Control Laboratory
Swiss Federal Institute of Technology
Physikstrasse 3, CH-8092 Zurich
Switzerland
colin.jones@cantab.net

Jan M. Maciejowski

Control Group, Department of Engineering
University of Cambridge
Trumpington St, Cambridge, UK
jmm@eng.cam.ac.uk

Abstract—This paper introduces a new enumeration technique for (multi)parametric linear programs (pLPs) based on the reverse-search paradigm. We prove that the proposed algorithm has a computational complexity that is linear in the size of the output (number of so-called critical regions) and a constant space complexity. This is an improvement over the quadratic and linear computational and space complexities of current approaches.

Current implementations of the proposed approach become faster than existing methods for large problems. Extensions of this method are proposed that make the computational requirements lower than those of existing approaches in all cases, while allowing for efficient parallelisation and bounded memory usage.

I. INTRODUCTION

It is standard practice to implement a model predictive controller (MPC) by solving an optimisation problem on-line. For example, when the system is linear, the constraints are polyhedral and the cost is linear (e.g. 1- or ∞ -norm), this amounts to computing a single linear program at each sampling instant. In recent years, it has become well-known that for this class of systems the optimal input is a piecewise affine function (PWA) defined over a polyhedral partition of the feasible states. By pre-computing this PWA function off-line, the on-line calculation of the control input then becomes one of evaluating the PWA function at the current measured state, which allows for significant improvements in sampling speed [1].

The computation of the optimal PWA function, mapping the measured state to the control input, can be posed as the following (multi)parametric linear program (pLP) [1]:

$$\min_u \{c^T u \mid (x, u) \in P\}, \quad (1)$$

where $x \in \mathbb{R}^d$ is the parameter, or state, $u \in \mathbb{R}^m$ is the optimiser, or control input and slack variables and P is a polyhedron¹, which incorporates the system constraints and is assumed bounded.

Several methods of computing the solution to pLP (1) can be found in the literature (e.g., [1]–[4]). All of these approaches enumerate the affine regions of the optimal PWA function one at a time. As each new region is discovered it must be compared with the list of regions seen so far in order to determine if it is new. This requires that the

set of discovered regions be stored in main memory for comparison, and makes the complexity of discovering a new region a function of the number of regions discovered so far. The number of regions in the optimal PWA function is known to be worst-case exponential in the size of the data [1], and to grow very quickly with the dimension of the parameter, and therefore the requirement to store and search against the entire set of regions is the factor that limits the size of problems that can be tackled by currently available algorithms.

In this paper we present a new method of computing the optimiser of pLP (1) based on a *reverse-search* approach [5]. By defining a tree over the set of regions in the solution, reverse-search is a depth-first enumeration method in which all decisions can be made based on *local* information and therefore the set of regions seen so far does not need to be stored in main memory, or consulted when a new region is discovered. There are three benefits of the proposed method over existing approaches: First, the online memory requirements are fixed, second the computational complexity is not a function of the number of regions in the solution and finally, the approach is readily parallelisable. These properties make the reverse-search approach very efficient for large pLP problems.

The remainder of this paper is organised as follows. Section II provides required background on parametric linear programming. Section III introduces a polytope such that there is a one-to-one mapping from its vertices to the affine pieces of the solution. The reverse-search approach described in [5] is then applied to this polytope, and hence to the associated pLP in Section IV. The complexity of the method is analysed and compared with existing methods in Section V and two important extensions are given in Section VI.

NOTATION

If $A \in \mathbb{R}^{m \times n}$ and $I \subseteq \{1, \dots, n\}$, then $A_{*,I} \in \mathbb{R}^{m \times |I|}$ is the matrix formed by the columns of A indexed by I . If $c \in \mathbb{R}^n$ is a vector then c_I is the vector formed by the elements of c in I . If $R \subseteq \{1, \dots, m\}$ then we will use the notation $A_{R,*} \in \mathbb{R}^{|R| \times n}$ to denote the matrix formed by the rows of A indexed by R . Wherever the ordering of the index set is important, we shall assume that the set is ordered from the smallest index to the largest.

¹A polyhedron is the intersection of a finite number of halfspaces.

A *polyhedron* is the intersection of a finite number of halfspaces and a *polytope* is a bounded polyhedron. If $P = \{x \mid Ax \leq b\}$ is a polyhedron and $H = \{x \mid a^T x \leq d\}$ is a halfspace such that $P \subseteq H$, then $P \cap \{x \mid a^T x = d\}$ is a *face* of P . One- and zero-dimensional faces are called *edges* and *vertices* respectively.

A vector $r \in \mathbb{R}^d$ defines a *ray* as $R = \{r\alpha \mid \alpha \geq 0\}$. A set C is called a cone if for every $x \in C$ and scalar $\alpha \geq 0$, we have $\alpha x \in C$. The columns of a matrix $F \in \mathbb{R}^{m \times n}$ are called the *generators* of the cone $C = \text{cone}(F) \triangleq \{F\alpha \mid \alpha \geq 0\}$. The generator $F_{*,i}$ is called *redundant* if $F_{*,i} \in \text{cone}(F_{*,\{1,\dots,n\} \setminus \{i\}})$ and *irredundant*, or *extreme* otherwise.

The *Minkowski sum* of two sets, denoted $A \oplus B$ is defined as $A \oplus B \triangleq \{x + y \mid x \in A, y \in B\}$.

II. PRELIMINARIES

A. Linear Programming

Consider the following linear program:

$$\min_{\lambda} \{c^T \lambda \mid \lambda \in D\}, \quad (2)$$

where $\lambda \in \mathbb{R}^n$ is the optimiser, $c \in \mathbb{R}^n$ is a vector and the constraint polytope D is defined by the matrix $A \in \mathbb{R}^{m \times n}$, $m \leq n$, $\text{rank } A = m$ and the vector $b \in \mathbb{R}^m$ as

$$D \triangleq \{\lambda \mid A\lambda = b, \lambda \geq 0\}. \quad (3)$$

Any set $B \subset \{1, \dots, n\}$ such that $|B| = m$ and $\text{rank } A_{*,B} = m$ is called a *basis* and we write $N = \{1, \dots, n\} \setminus B$ for its complement and call λ_B and λ_N the basic and non-basic variables respectively. Every basis B defines a *basic solution*² λ^B to the linear equations in (3), which is given by restricting the non-basic constraints to zero

$$\lambda_B^B = A_{*,B}^{-1} b, \quad \lambda_N^B = 0. \quad (4)$$

A basis is called *primal feasible* if the resulting solution also satisfies the inequality constraints in (3): $A_{*,B}^{-1} b \geq 0$. Note that all solutions represented by bases occur at a *vertex* of the constraint polyhedron D .

B. Optimality Conditions

Definition 1 (Tangent cone [6]): Let λ be an element of the polyhedron $D \subseteq \mathbb{R}^n$. A vector $\gamma \in \mathbb{R}^n$ is said to be a *feasible direction* at λ if there exists a strictly positive scalar α for which $\lambda + \alpha\gamma \in D$. The set of all feasible directions at λ is called the *tangent cone* and is written $\mathcal{T}_D(\lambda)$.

Given a basis B , the extreme feasible directions at the solution λ^B are given by increasing each non-basic variable in a feasible (positive) direction:

$$\lambda_B = \lambda_B^B - A_{*,B}^{-1} A_{*,i} \lambda_i, \quad \lambda_i \geq 0, \quad \lambda_{N \setminus \{i\}} = 0 \quad \forall i \in N$$

The set of all convex combinations of the extreme rays give the tangent cone:

$$\mathcal{T}_D(\lambda^B) = \text{cone}(F), \quad (5)$$

²Where clear from the context, we will refer to the basic solution as simply the solution.

where $F_{B,*} \triangleq -A_{*,B}^{-1} A_{*,N}$, $F_{N,*} \triangleq I$.

Theorem 1 (Optimality Condition): Let λ be an element of the polyhedron D . A necessary and sufficient condition for λ to be a global minimum of the linear program (2) is $c^T \gamma \geq 0$ for all feasible directions γ at λ .

Definition 2: The normal cone to D at λ is the orthogonal complement of the tangent cone:

$$\mathcal{N}_D(\lambda) \triangleq \{v \mid v^T \gamma \leq 0, \forall \gamma \in \mathcal{T}_D(\lambda)\}$$

From the above definition and (5), the normal cone to the basic feasible solution λ^B is:

$$\mathcal{N}_D(\lambda^B) = \{v \mid F^T v \leq 0\} \quad (6)$$

A direct result of Theorem 1 and (6) is that the basic solution λ^B , and hence the basis B , is optimal if and only if³

$$-c \in \mathcal{N}_D(\lambda^B). \quad (7)$$

C. Parametric Linear Programming

The problem we will consider in this paper is the following parametric linear program:

$$\min_{\lambda} \{(Ex)^T \lambda \mid \lambda \in D\}, \quad x \in \mathcal{X} \quad (8)$$

where $x \in \mathcal{X}$ is the parameter, $\mathcal{X} \subseteq \mathbb{R}^d$ is a polyhedron and $E \in \mathbb{R}^{n \times d}$ is a matrix of rank d , $d < n$. It is assumed throughout this paper that the set of feasible parameters \mathcal{X} is full-dimensional, which is common to most pLP algorithms [2], [7]. This assumption can be easily guaranteed through a pre-processing operation [8]. The standard assumption is also made that pLP (8) has an optimal bounded solution for every $x \in \mathcal{X}$.⁴

Definition 3 (Critical Region [4]): If B is a basis of pLP (8), then the *critical region* \mathcal{R}_B is defined as the set of all parameters $x_0 \in \mathcal{X}$ such that B is optimal for $x = x_0$.

It can be seen from (7) and (8) that the critical region \mathcal{R}_B is the polyhedral set

$$\mathcal{R}_B = \{x \mid F^T E x \leq 0\} \cap \mathcal{X}$$

Our goal is to enumerate all full-dimensional critical regions.

The approach adopted to handle potential degeneracy is *lexicographic perturbation* of the problem, where an arbitrarily small *symbolic* vector is added to the right-hand side of the constraints.

Theorem 2: [3] There exists a positive number $\delta > 0$, such that whenever $0 < \epsilon_0 < \delta$, the following perturbed problem is primal non-degenerate for every value of the parameter $x \in \mathcal{X}$:

$$\min_{\lambda} \{(Ex)^T \lambda \mid A\lambda = b + \epsilon, \lambda \geq 0\} \quad (9)$$

where $\epsilon^T \triangleq [\epsilon_0 \quad \epsilon_0^2 \quad \dots \quad \epsilon_0^m]$. Furthermore, each critical region is uniquely defined by a single basis, the relative interiors of critical regions do not overlap and the union

³The vector $-F^T c$ is often referred to as the reduced cost \bar{c} and condition (7) then becomes $\bar{c} \geq 0$.

⁴Note that this also implies that the dual solution is feasible and bounded.

of all full-dimensional critical regions is the set of feasible parameters \mathcal{X} .

In the remainder of this paper we will assume that the problem has been lexicographically perturbed and will therefore not discuss possible degeneracy of the solution.⁵ The reader is referred to [3] for details on the computational mechanics of a lexicographically perturbed problem.

D. Affine Offset

The standard parametric linear program resulting from model predictive control problems takes the form [1]:

$$u^*(x) = \arg \min_u \{ -b^T u \mid A^T u \leq Ex + c \}, \quad (10)$$

where $x \in \mathcal{X}$ is the measured state vector, and the decision variable u is a sequence of inputs that will be applied to the system and some slack variables. The goal is to compute an optimal input $u^*(x)$ as a function of the state x . The set of feasible states \mathcal{X} is assumed to be a polytope and therefore bounded, and the pLP (10) is assumed to have a bounded optimiser for every feasible value of the parameter $x \in \mathcal{X}$.

In this section we will demonstrate how to transform pLP (10) into an equivalent pLP of the form used in this paper (8). We begin by posing the dual of pLP (10):

$$\min_{\lambda} \{ (Ex + c)^T \lambda \mid A\lambda = b, \lambda \geq 0 \} \quad (11)$$

The pLP (10) is assumed to have a bounded optimal solution for every feasible value of the parameter. Therefore, a basis is optimal for the dual problem (11) if and only if it is optimal for the primal (10) [9]. Given an optimal basis B of pLP (11), the primal optimiser of (10) is given by:

$$u^*(x) = A_{*,B}^{-T} (E_{B,*} x + c_B), \quad x \in \mathcal{R}_B$$

The parametric linear program (11) differs from that considered in this paper by the affine offset c . The following theorem demonstrates that any problem of the form (11) can be *homogenized* into an equivalent problem of the form used in this paper.

Theorem 3: B is an optimal basis of pLP (11) for the parameter x if and only if it is an optimal basis of

$$\min_{\hat{\lambda}} \left\{ \left(\begin{bmatrix} E & c \end{bmatrix} \begin{pmatrix} \hat{x} \\ t \end{pmatrix} \right)^T \hat{\lambda} \mid A\hat{\lambda} = b, \hat{\lambda} \geq 0 \right\}, \quad (12)$$

for parameter $\hat{x} = x/t$ and $t > 0$.

Proof: Let B be a feasible basis of pLP (11). Clearly B is also a feasible basis of (12) and the normal cone $\mathcal{N}(\lambda^B)$ is the same in both cases for the basic solution λ^B . Let F be a generator matrix for the normal cone at λ^B , $\text{cone}(F) = \mathcal{N}(\lambda^B)$. Then the optimality condition (7) is satisfied if and only if $F^T(Ex + c) \geq 0$ and $F^T \begin{bmatrix} E & c \end{bmatrix} \begin{pmatrix} \hat{x} \\ t \end{pmatrix} \geq 0$ for pLP (11) and pLP (12) respectively. Since t is strictly

positive, one can see that the basis is optimal for pLP (11) for the value of the parameter x if and only if it is also optimal for pLP (12) for $\hat{x} = x/t$. ■

Remark 1: Note that in order to satisfy the assumption made in Section II-C, we must here assume that $\begin{bmatrix} E & c \end{bmatrix}$ is full rank.

III. PLP AS VERTEX ENUMERATION

The following theorem demonstrates that the goal of enumerating all bases that define full-dimensional critical regions can be re-posed as a vertex enumeration problem of a linear transform of the constraint polytope D .

Theorem 4: If B is a feasible basis of pLP (8) and λ^B is the basic solution, then B defines a full-dimensional critical region if and only if $E^T \lambda^B$ is a vertex of the polytope $E^T D \triangleq \{ E^T \lambda \mid A\lambda = b, \lambda \geq 0 \}$.

Proof: pLP (8) can be re-written in the following form through the change of variable $z \triangleq E^T \lambda$:

$$\min_z \{ x^T z \mid z \in E^T D \}, \quad (13)$$

It follows from (7) and Definition 3 that x is in the critical region \mathcal{R}_B if and only if $-x$ is in the normal cone $\mathcal{N}_{E^T D}(E^T \lambda^B)$. Finally, the normal cone of a point $E^T \lambda^B$ in a polytope $E^T D$ is full-dimensional if and only if $E^T \lambda^B$ is a vertex of $E^T D$ [10, §13.2.2]. ■

IV. REVERSE SEARCH

In this section an algorithm based on the reverse search approach of Avis and Fukuda [11] is introduced. The goal of reverse search is to remove the need to store or operate on the critical regions that have been discovered so far, resulting in constant space complexity and a time complexity that is linear in the number of regions in the solution.

Reverse search achieves its aim by converting the *skeleton* of a polytope, or graph formed from its vertices and edges, into a tree. The root of the tree is taken to be any critical-region defining basis; we shall call this root R . If V is the set of vertices of the polytope, a single-valued mapping $\mathcal{G} : V \setminus \{R\} \rightarrow V$ is defined over the remaining vertices. This function is defined in such a way that if $\mathcal{G}(\cdot)$ is applied recursively to any vertex we will eventually arrive at the root R . In other words, $\mathcal{G}(\cdot)$ defines a unique directed path from every vertex to the root. Note that the definition implies that there are no circuits in the graph and that for each vertex there is exactly one edge pointing towards the root and possibly several pointing away. As a result, the mapping $\mathcal{G}(\cdot)$ defines a tree over the polytope skeleton. The reverse search algorithm proceeds to enumerate all vertices by *reversing* the paths taken to the root and enumerating this tree in a depth-first fashion.

The algorithm begins at the root, chooses the first incoming path and follows it backwards one step to a new vertex. A recursive call is then made on this new vertex, which also defines a tree. Once the first path has been enumerated, the algorithm moves onto the second, and so on until all branches have been enumerated.

⁵Note that our definition of a critical region differs from that generally found in the literature. However, under the assumption that the problem is non-degenerate, or equivalently, lexicographically perturbed, the two definitions are equivalent.

The key to the efficiency of this approach is that all decisions can be made based on *local* information. When a new vertex is discovered it does not need to be checked against previously discovered vertices because if it is further down the tree, we can be certain that it has not been seen before.

Each vertex $v \in E^T D$ is joined along the edges that intersect at v to a set of adjacent vertices, which are the neighbours of v . The algorithm must determine from these edges which ones define paths moving down the tree, which one up and which neither. There are a maximum of $n - m$ edges leaving each vertex, which is the dimension of the polytope D^6 . We assume a function $\text{neighbour} : V \times \mathbb{N} \rightarrow V \cup \{\emptyset\}$ that maps a vertex $v \in E^T D$ and an integer $i \in \{1, \dots, n - m\}$ to the set of vertices that neighbour v , or to the empty set if integer i does not define a neighbour. This function is described in detail in Section IV-A below.

The proposed reverse-search method is shown as Algorithm 1, which is a standard implementation and is not much different from the generic algorithm of [5]. The differences arise in the definitions of the functions $\mathcal{G}(\cdot)$ and $\text{neighbour}(\cdot, \cdot)$, which allow the reverse-search algorithm to be applied to the parametric linear programming problem.

The process is a simple depth-first search, where the cost function $\mathcal{G}(\cdot)$ is used to avoid the requirement of a stack and hence achieve fixed memory requirements. The inner ‘while’ loop (Steps 5–11) first moves the algorithm down to a leaf of the tree. Step 13 then backtracks up the tree one level and Steps 14–18 determine which branch the algorithm followed to get to the leaf. The neighbour counter cnt is then incremented by one (Step 6) and the next branch is followed back down to a leaf. By continuing this until there are no more branches to follow, the entire tree will be visited in a depth-first manner.

An example reverse-search tree is shown in Figure 1. The figure on the left shows the path that the function $\mathcal{G}(\cdot)$ would take from each vertex of the cube, and the figure on the right is the resulting reverse search tree.

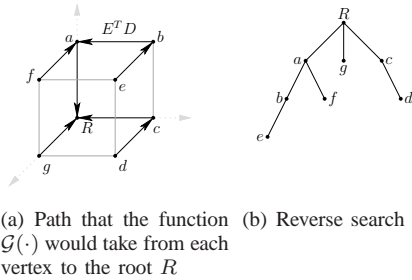


Fig. 1. Reverse Search Illustration (Figure taken from [11])

A. Neighbour Function

This section defines the function $\hat{B} = \text{neighbour}(B, i)$, which takes an integer i and a basis B of the polytope D

⁶Recall that the polytope D is assumed to be in general position due to the lexicographic perturbation 2

Algorithm 1 Multiparametric Linear Programming: Reverse Search for Enumeration

Input: Basis R of pLP (8) such that $E^T \lambda$ is a vertex of $E^T D$
 Functions $\mathcal{G}(\cdot)$ and $\text{neighbour}(\cdot, \cdot)$
 The dimension maxcnt of the polytope D

Output: All bases that define critical regions

```

1:  $B \leftarrow R$                                      Begin at the root
2:  $\text{cnt} \leftarrow 0$                                    Neighbour counter
3: Report  $R$ 
4: loop
5:   while  $\text{cnt} < \text{maxcnt}$  do
6:     Increment  $\text{cnt} \leftarrow \text{cnt} + 1$ 
7:     Compute neighbour  $\text{next} \leftarrow \text{neighbour}(B, \text{cnt})$ 
8:     if  $\text{next} \neq \emptyset$  and  $\mathcal{G}(\text{next}) = B$  then
9:        $B \leftarrow \text{next}$ ,  $\text{cnt} \leftarrow 0$ , Report  $B$ 
10:    end if
11:  end while

Downward traverse
12:  if  $B \neq R$  then
13:     $B' \leftarrow B$ ,  $B \leftarrow \mathcal{G}(B)$ ,  $\text{cnt} \leftarrow 0$       Move up the tree
14:    repeat                                           Restore cnt
15:       $\text{cnt} \leftarrow \text{cnt} + 1$ 
16:    until  $\text{Neighbour}(B, \text{cnt}) = B'$ 
17:  end if
18: end loop

```

that defines a vertex $E^T \lambda^B$ of $E^T D$. The function returns a basis \hat{B} of D that defines the vertex $E^T \lambda^{\hat{B}}$, which is the i^{th} neighbour of $E^T \lambda^B$. The notion of ordering over neighbours and an efficient computational method to find them is given in this section.

Two vertices of a polytope are called neighbours, or adjacent, if they are contained in the same edge, or one-dimensional face of the polytope. The edges of a polytope P that intersect at a vertex $v \in P$ are given by the intersection of P with the extreme rays of the tangent cone $\mathcal{T}_P(v)$ [9]. The following lemma describes the tangent cone of a basic solution of $E^T D$.

Lemma 1: If λ is an element in the polytope D , then

$$\mathcal{T}_{E^T D}(E^T \lambda) = E^T \mathcal{T}_D(\lambda)$$

Proof: From Definition 1, $\gamma \in \mathbb{R}^d$ is in $E^T \mathcal{T}_D(\lambda)$ if and only if there exists a scalar $\alpha > 0$ and a vector g such that

$$\gamma = E^T g, \quad \lambda + \alpha g \in D. \quad (14)$$

By assumption, E is rank d and therefore such a g always exists. Under the mapping E^T , (14) becomes

$$E^T \lambda + \alpha \gamma \in E^T D,$$

which is true if and only if $\gamma \in \mathcal{T}_{E^T D}(E^T \lambda)$. \blacksquare

Lemma 1 and (5) give a description of the tangent cone to a vertex $E^T \lambda^B$ of $E^T D$ defined by the basis B :

$$\mathcal{T}_{E^T D}(E^T \lambda^B) = \text{cone}(E^T F)$$

where $F_{B,*} \triangleq -A_{*,B}^{-1} A_{*,N}$, $F_{N,*} \triangleq I$. Note however, that not every column $E^T F_{*,i}$ defines an extreme ray of $\mathcal{T}_{E^T D}(E^T \lambda^B)$, as some of them may well be redundant.

Remark 2: Testing if a ray is extreme or redundant requires a single linear program of dimension d . This is a standard redundancy elimination operation, which is in fact equivalent to the redundancy elimination operations in other pLP approaches [1]–[4]. The reader is referred to [12] for computational details.

We can now define the function $\hat{B} = \text{neighbour}(B, i)$. If the ray $r^i \triangleq \{E^T F_{*,i} \alpha \mid \alpha \geq 0\}$ is an extreme ray of the tangent cone $\mathcal{T}_{E^T D}(E^T \lambda^B)$, then the neighbour function returns the basis \hat{B} such that $E^T \lambda^{\hat{B}}$ is the vertex of $(r^i \oplus \{E^T \lambda^B\}) \cap E^T D$, which is different from $E^T \lambda^B$. Note that there are exactly two vertices on each edge. If another index j defines the same ray, i.e. $E^T F_{*,i} = E^T F_{*,j}$, then the neighbour function returns \hat{B} for the smallest index, and the empty set for all others. Similarly, if an index defines a redundant ray, then the empty set is returned.

1) *Efficient Computation of Neighbour:* The following theorem provides an efficient method of computing the basis that represents the adjacent vertex given an irredundant ray of the tangent cone.

Theorem 5: If B is a feasible basis of D , $E^T \lambda^B$ is a vertex of $E^T D$ and $r^i = \{E^T F_{*,i} \alpha \mid \alpha \geq 0\}$ is an irredundant ray of the tangent cone $\mathcal{T}_{E^T D}(E^T \lambda^B)$, then the adjacent vertex of $E^T \lambda^B$ in the direction r^i is the optimal basis of the LP:

$$\min_{\lambda} \{(E^T F_{*,i})^T E^T \lambda \mid \lambda \in D, \lambda_j = 0, \forall j \notin T\}, \quad (15)$$

where $T \triangleq \{j \mid \exists \epsilon > 0, E^T F_{*,i} = E^T F_{*,j} \epsilon\}$ and $F_{B,*} \triangleq -A_{*,B}^{-1} A_{*,N}, F_{N,*} \triangleq I$.

Proof: The adjacent vertex is reached by moving along edge $(r^i \oplus \{E^T \lambda^B\}) \cap E^T D$ away from $E^T \lambda^B$. Every point $\lambda \in D$ can be written as $\lambda = \lambda^B + F\gamma$, for some $\gamma \geq 0$, where F is as defined in the statement of the theorem. Consider the column $F_{*,j}$ and the resulting points $\lambda = \lambda^B + F_{*,j} \gamma_j$, $\gamma_j \geq 0$. Clearly, $E^T \lambda \in (r^i \oplus \{E^T \lambda^B\})$ if and only if there exists an $\alpha \geq 0$ such that $E^T F_{*,j} = E^T F_{*,i} \alpha$. Therefore, the face Q of D such that $E^T Q = (r^i \oplus \{E^T \lambda^B\}) \cap E^T D$ is given by $Q = \{\lambda \mid \lambda_i = 0, \forall i \notin T\} \cap D$.

The LP given in the statement of the theorem then maximises in the direction of the ray r^i , while restricting λ to be in Q . ■

2) *neighbour(\cdot, \cdot) Complexity:* Computing the i^{th} neighbour of the basis B requires two linear programs. The first tests the redundancy of the ray r^i in the tangent cone $\mathcal{T}_{E^T D}(E^T \lambda^B)$, and the second, LP (15), computes the optimal basis of the neighbour. If $LP(v, w)$ is the complexity of a single linear program of dimension v , with w constraints⁷, then in the worst case, the complexity of the neighbour function is

$$\mathcal{O}_{\text{neigh}} = \mathcal{O}(LP(d, e) + LP(m, n)), \quad (16)$$

where the dimension of the parameter x is d , the constraint matrix A is in $\mathbb{R}^{m \times n}$ and we define the variable $e \triangleq n - m$.

⁷If the LP is written in the form (2)(3), then the dimension is m and the number of constraints is n , for $A \in \mathbb{R}^{n \times m}$.

B. Function $\mathcal{G}(\cdot)$

We follow a similar approach to [11] and define the function $\mathcal{G}(\cdot)$ via the simplex algorithm. Consider the following linear program:

$$\min_z \{\rho^T z \mid z \in E^T D\}, \quad (17)$$

where the cost ρ is chosen such that it is not parallel to any of the edges of $E^T D$ ⁸. At each vertex $v \in E^T D$, there are a number of edges intersecting at v that can be followed to an adjacent vertex (the extreme rays of the tangent cone). The function $\mathcal{G}(\cdot)$ chooses the edge along which the cost ρ decreases most rapidly. Note that this edge is unique via the above assumption. If the cost decreases most rapidly along the ray r^i of the tangent cone, then the function $\mathcal{G}(\cdot)$ returns the neighbour in the direction r^i by computing LP (15).

C. $\mathcal{G}(\cdot)$ Complexity

Evaluating the function $\mathcal{G}(\cdot)$ requires the determination of the irredundant extreme ray of the tangent cone along which the cost decreases most rapidly. Finding the set of all irredundant rays requires, in the worst case, $n - m$ d -dimensional redundancy elimination LPs. Computing the neighbour requires one LP (15) of dimension m .

$$\mathcal{O}_{\mathcal{G}} = \mathcal{O}(eLP(d, e) + LP(m, n)), \quad (18)$$

where $e \triangleq n - m$.

V. COMPLEXITY

A. Time Complexity

The time complexity of the reverse search enumeration method is clearly a function of Steps 7, 8, 13 and 16 in Algorithm 1.

For each critical region in the solution, the reverse search algorithm traverses once down the tree past the region (Steps 7–10) and once up away from the region (Steps 13–16). In the worst case, each region can have $e \triangleq n - m$ neighbours (the dimension of the polytope D) and therefore the computational complexity of the reverse-search algorithm is:

$$\mathcal{O}_{rs} = \mathcal{O}(N_r(e(\underbrace{\mathcal{O}_{\text{neigh}}}_{\text{Steps 7–10}} + \underbrace{\mathcal{O}_{\mathcal{G}(\cdot)}}_{\text{Steps 13–16}})) + e\mathcal{O}_{\text{neigh}} + \mathcal{O}_{\mathcal{G}(\cdot)}), \quad (19)$$

where N_r is the number of critical regions in the solution. From Equations (16), (18) and (19), the complexity can be re-written as:

$$\mathcal{O}_{rs} = \mathcal{O}(N_r(e^2 LP(d, e) + eLP(m, n))). \quad (20)$$

Equation (20) demonstrates that the complexity of the reverse-search algorithm is a *linear* function of the size of the output (number of critical regions in the solution).

⁸This ensures that there will be no dual-degeneracy in LP (17), which is a condition that can be made almost certain via a random selection of ρ . Certainty can be achieved through a primal-dual lexicographic perturbation as outlined in [3].

B. Space Complexity

One of the most important features of reverse-search is the small memory requirement. Since the algorithm does not need to compare a newly discovered region with those seen before, the only data that must be stored in main memory is the data E, A, b of the pLP, the current basis B and the neighbour count cnt . The space complexity of the algorithm is therefore constant $\mathcal{O}(1)$.

C. Comparison to Existing Approaches

All current approaches for computing solutions to (multi)parametric linear programs require that as each new region is discovered, an operation must be carried out that is a function of all regions discovered so far. It follows that both the space and time complexities of these algorithms grows in a super-linear fashion with the size of the output.

It is important to state that while the complexity of the proposed reverse search approach is superior to current algorithms, there is a large complexity coefficient hidden in the big- \mathcal{O} notation. As a result, the reverse-search method as given as Algorithm 1 is slower for small problems than the current most efficient implementations available in the literature and only comes into its own on larger examples. However, the following section introduces extensions to the reverse-search approach that allow the parallelisation of the method and a significant computational efficiency improvement at the cost of an increase in memory usage.

VI. EXTENSIONS

One of the main strengths of the reverse-search method is that it can be parallelised very efficiently, as each branch of the tree can be assigned to a different processor and no data will need to pass between the machines, since all decisions can be made locally using the function $\mathcal{G}(\cdot)$. If the tree is well-balanced this will result in a speed improvement equal to the number of processors used, although there is no *a priori* guarantee that a problem will be well-balanced.

Second, the computation speed can be significantly improved through the use of memory. Storing the path taken down the tree in a stack removes the requirement to do any computations while traversing up the tree (Steps 13–18). Recording adjacent critical regions that are discovered during the downward traverse (Steps 5–11), but are not children of the current critical region removes any duplication of work. Note that because none of the stored data is strictly required for the implementation of the algorithm, it is possible to set the amount of memory used to a fixed value. This use of memory makes the number of linear programs required exactly equal to that of the current fastest approaches, while still maintaining the complexity as a linear function of the number of regions, the ability to parallelise the algorithm, and arbitrarily bound memory usage. Future implementations will take advantage of this ability.

VII. EXAMPLE

Consider the following problem:

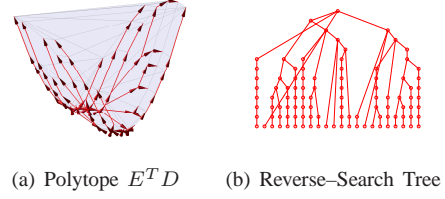


Fig. 2. Reverse Search Tree for Example VII

$$J^*(x_0) = \min \sum_{k=0}^{N-1} (\|Qx_k\|_\infty + \|Ru_k\|_\infty) + \|Q_f x_N\|_\infty$$

subject to

$$x_{k+1} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} x_k + \begin{pmatrix} 1 \\ 0.5 \end{pmatrix} u_k$$

$$\|x_k\|_\infty \leq 5, \quad \|u_{k-1}\|_\infty \leq 1, \quad \forall k \in \{1, \dots, N\},$$

where the objective weight matrices are set to $Q = Q_f = I$ and $R = I$. The task is to regulate the system to the origin while fulfilling the input and state constraints. Conversion of the example to the form of pLP (1) is discussed in [2]. The solution consists of 84 critical regions and can be seen as the polytope $E^T D$ and the resulting reverse-search tree, which are shown in Figure 2.

Space limitations prevent the inclusion of detailed examples in this paper. However, code and examples will be available as part of the Multiparametric Toolbox MPT [13].

ACKNOWLEDGEMENTS

The authors would like to thank Eric Kerrigan and Komei Fukuda for their valuable discussions on this paper.

REFERENCES

- [1] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming - the explicit solution," *IEEE Transactions on Automatic Control*, vol. 47, no. 12, pp. 1974–1985, 2002.
- [2] F. Borrelli, A. Bemporad, and M. Morari, "A Geometric Algorithm for Multi-Parametric Linear Programming," *Journal of Optimization Theory and Applications*, vol. 118, no. 3, pp. 515–540, 2003.
- [3] C. Jones, E. Kerrigan, and J. Maciejowski, "Lexicographic perturbation for multiparametric linear programming with applications to control," *Automatica*, 2005, submitted.
- [4] T. Gal, *Postoptimal Analyses, Parametric Programming and Related Topics*, 2nd ed. Walter de Gruyter, 1995.
- [5] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Math*, vol. 65, pp. 21–46, 1996.
- [6] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*. Athena Scientific, 1997.
- [7] J. Spjøtvold, P. Tøndel, and T. Johansen, "A method for obtaining continuous solutions to multiparametric linear programs," in *Proceedings of the 16th IFAC World Congress*, Prague, 2005.
- [8] F. Borrelli, *Constrained Optimal Control Of Linear And Hybrid Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2003, vol. 290.
- [9] K. Murty, *Linear Programming*. John Wiley & Sons, 1983.
- [10] J. E. Goodman and J. O'Rourke, Eds., *Handbook of Discrete and Computational Geometry*. CRC Press, 1997.
- [11] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete and Computational Geometry*, vol. 8, pp. 295–313, 1992.
- [12] K. Fukuda, "Frequently asked questions in polyhedral computation," <http://www.ifor.math.ethz.ch/fukuda/polyfaq/polyfaq.html>.
- [13] M. Kvasnica, P. Grieder, and M. Baotić, "Multi-Parametric Toolbox (MPT)," 2004, <http://control.ee.ethz.ch/ mpt/>.