



PERGAMON

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Automatica 39 (2003) 945–950

automatica

www.elsevier.com/locate/automatica

Technical Communique

Evaluation of piecewise affine control via binary search tree[☆]

P. Tøndel^{a,*}, T.A. Johansen^a, A. Bemporad^b

^aDepartment of Engineering Cybernetics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway

^bDipartimento di Ingegneria dell'Informazione, University of Siena, 53100 Siena, Italy

Received 3 April 2002; accepted 26 November 2002

Abstract

We present an algorithm for generating a binary search tree that allows efficient computation of piecewise affine (PWA) functions defined on a polyhedral partition. This is useful for PWA control approaches, such as explicit model predictive control, as it allows the controller to be implemented online with small computational effort. The computation time is *logarithmic* in the number of regions in the PWA partition.

© 2003 Elsevier Science Ltd. All rights reserved.

Keywords: Piecewise linear controllers; Hybrid systems

1. Introduction

Piecewise affine (PWA) controllers arise naturally in various applications, e.g. in the presence of constraints or as approximations of nonlinear maps. In this paper we address the problem of evaluating a PWA function. At first sight, this may seem a trivial task, but when the function is complex, a straightforward evaluation is computationally expensive. The main motivation behind this work is the recent development of explicit solutions to model predictive control (MPC) problems, in which the solutions are complex PWA state feedback laws. In [Bemporad, Morari, Dua, and Pistikopoulos \(2002b\)](#) it was recognized that the linear MPC problem can be formulated as a multi-parametric quadratic program (mp-QP) and solved explicitly, with a PWA solution. An algorithm to solve the mp-QP is also provided, however, a more efficient algorithm is developed by [Tøndel, Johansen, and Bemporad \(2003\)](#). An alternative solution strategy is given by [Johansen, Petersen, and Slupphaug \(2002\)](#), where pre-determination of a small set of sampling

instants where the active set is allowed to change gives a suboptimal solution, and in [Seron, DeDoná, and Goodwin \(2000\)](#) based on a geometric interpretation of the QP problem. Suboptimality of mp-QP is also introduced in [Bemporad and Filippi \(2003\)](#) by relaxing the optimality conditions, and in [Johansen and Grancharova \(2002\)](#), by imposing an orthogonal structure to the state space partitioning. In [Bemporad, Borrelli, and Morari \(2002\)](#) MPC problems with $1/\infty$ -norms are formulated as multi-parametric linear programs (mp-LP) and solved explicitly, while extensions to hybrid systems using multi-parametric mixed-integer LP (mp-MILP), can be found in [Bemporad, Borrelli, and Morari \(2000b\)](#), and explicit robust MPC is treated in [Bemporad, Borrelli, and Morari \(2003\)](#). All of these approaches lead to PWA state feedback laws. Evaluation of PWA functions is also of interest with other PWA control structures than explicit MPC control (e.g. [Garcia & Tarbouriech, 1999](#); [Wredenhagen & Belanger, 1994](#); [Rantzer & Johansson, 2000](#); [Hassibi & Boyd, 1998](#); [Slupphaug & Foss, 1999](#)).

The most immediate way of evaluating a PWA function is to do a sequential search through the regions representing the PWA function (see [Algorithm 1](#) below). The use of neighboring relations between the regions do not necessarily reduce the worst case computational complexity in a practical system, since there may be large changes in the state between any consecutive samples. Reasons for this may include sudden setpoint changes, mode switches, integrator resetting, disturbances and slow sampling.

[☆] This work was in part financed by the Norwegian Research Council. This paper was not presented at any IFAC meeting. This paper was recommended for publication in revised form by Associate Editors Jay H. Lee under the direction of Editor Paul van den Hof.

* Corresponding author.

E-mail addresses: petter.tondel@itk.ntnu.no (P. Tøndel), tor.arne.johansen@itk.ntnu.no (T.A. Johansen), bemporad@diu.unisi.it (A. Bemporad).

Nevertheless, this is similar to “warm start” in numerical optimization and will usually give some reduction in average computation effort. For the case of exact solutions to the mp-QP and mp-LP problems, [Borrelli, Baotic, Bemporad, and Morari \(2001b\)](#) propose a more efficient method regarding both search time and storage by exploiting properties of the value function. This method is however not feasible to more general PWA function evaluation, and is still fairly time consuming since it requires a sequential search. The evaluation of a PWA function is similar to the point location problem ([Snoeyink, 1997, Chap. 30](#); [Goodrich & Ramaiyer, 1999](#)) which has been subject to some research in the computational geometry field. However, this research has been mainly focused on planar problems, and also a few treatments of problems in three dimensions. These solutions are not suitable for the problems faced when evaluating the PWA solutions to control problems, which may have higher dimensions. The off-line mp-QP algorithm of ([Bemporad et al., 2002b](#)) has the property that a binary tree structure could be generated while the mp-QP problem is solved, but it is not obvious how to modify the algorithm such that the search tree will be balanced.

In this paper, we present an efficient data structure for the representation of PWA functions, in an effort to minimize the time needed to evaluate the function. We also seek to minimize the storage required by this data structure, although this is considered of secondary importance. The proposed method is general, in the sense that it does not have special requirements on the PWA function. Overlapping regions and holes in the partition are handled by the method. The proposed method gives evaluation times which are *logarithmic* in the number of regions in the PWA function, while the storage required by the data structure is polynomial in the number of regions. It can also be used for evaluating piecewise quadratic as well as piecewise nonlinear functions, as long as the functions are defined on a polyhedral partition.

2. Explicit constrained linear MPC

Below we give a short summary of linear MPC problems and their explicit solutions. Consider the linear system

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t, \\ y_t &= Cx_t, \end{aligned} \quad (1)$$

where $x_t \in \mathbb{R}^n$ is the state variable, $u_t \in \mathbb{R}^m$ is the input variable, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{\dim(y) \times n}$ and (A, B) is a controllable pair. The output and the control input are subject to the bounds $y_{\min} \leq y_t \leq y_{\max}$ and $u_{\min} \leq u_t \leq u_{\max}$, where $y_{\min} < y_{\max}$ and $u_{\min} < u_{\max}$. For the current x_t , MPC solves the optimization problem

$$\begin{aligned} J^*(x_t) &= \min_{u_t, \dots, u_{t+M-1}} \|x_{t+N}\|_p^p \\ &+ \sum_{k=0}^{N-1} (\|x_{t+k+1}\|_p^q + \|u_{t+k}\|_p^r) \end{aligned} \quad (2)$$

subject to $x_{t|t} = x_t$ and

$$y_{\min} \leq y_{t+k|t} \leq y_{\max}, \quad k = 1, \dots, N, \quad (3)$$

$$u_{\min} \leq u_{t+k} \leq u_{\max}, \quad k = 0, \dots, N-1, \quad (4)$$

$$x_{t+k+1|t} = Ax_{t+k|t} + Bu_{t+k}, \quad k \geq 0, \quad (5)$$

$$y_{t+k|t} = Cx_{t+k|t}, \quad k \geq 0, \quad (6)$$

$$u_{t+M+k} = 0 \quad \forall k \geq 0. \quad (7)$$

Additionally, we may require the terminal constraint

$$Lx_{t+N} \leq l \quad (8)$$

to be satisfied. For $p = 2$, $\|x\|_p^E = x^T E x$,¹ $Q = Q^T \geq 0$, $R = R^T > 0$ and $P \geq 0$. For $p = 1$ and ∞ , $\|x\|_p^E = \|E x\|_p$. For ease of notation, we may in the sequel skip the index t , and use u for u_t and x for x_t . These problems can be reformulated as the following multi-parametric programs:

(1) mp-QP ($p = 2$):

$$\min_U U^T H U + x^T F U \quad (9)$$

$$\text{s.t. } GU \leq W + Sx, \quad (10)$$

where $U = [u_t^T, \dots, u_{t+M-1}^T]^T$, see [Bemporad et al. \(2002b\)](#) for details.

(2) mp-LP ($p = 1$ or ∞):

$$\min_U h^T U \quad (11)$$

$$\text{s.t. } GU \leq W + Sx, \quad (12)$$

where $U = [u_t^T, \dots, u_{t+M-1}^T, \varepsilon^T]^T$ and ε is a vector of slack variables, see [Bemporad, Borrelli, and Morari \(2000a\)](#) for details.

(3) Robust MPC ($p = \infty$). In [Bemporad, Borrelli and Morari \(2003\)](#) the authors show that when introducing uncertainty to the linear model (1), that is

$$x_{t+1} = A(w(t))x_t + B(w(t))u_t + Ev(t), \quad (13)$$

where $v(t)$ and $w(t)$ are unknown but bounded input disturbances and parametric uncertainties, respectively, a min–max optimization problem analogous to (2)–(7) can be solved by N mp-LPs.

(4) mp-MILP ($p = 1$ or ∞): Here the linear model (1) is replaced by the piecewise affine model.

$$x_{t+1} = A_i x_t + B_i u_t + f_i, \quad \text{if } \begin{bmatrix} x_t \\ u_t \end{bmatrix} \in \mathcal{X}_i, \quad (14)$$

where $A \in \mathbb{R}^{n \times n}$, $B_i \in \mathbb{R}^{n \times m}$ and $f_i \in \mathbb{R}^n$ are constant vectors and $\{\mathcal{X}_i\}$ is a polyhedral partition of the state+input space. The problem can be reformulated as a mp-MILP, see [Bemporad, Borrelli and Morari \(2000b\)](#).

¹ Although this does not strictly define a norm, we choose this description for ease of notation.

- (5) mp-MIQP ($p=2$): The linear model is here as in (14), but the cost function is quadratic, see Bemporad et al., (2002a). The explicit solution to this problem can in some cases be PWA with a polyhedral partition, but in general the partition is not polyhedral, as it can be defined by quadratic functions. This kind of functions is not treated in this paper, but our methods can be extended to cover this.
- (6) mp-NLP: This class of parametric programs is more general than form (1)–(7). Both the cost function and constraints may be non-linear functions. Exact solutions to these problems have in general not been established, but an approximation based on mp-QP giving a PWA solution, is developed by Johansen (2002).

Definition 1. A function $u: X \rightarrow \mathbb{R}^s$, is piecewise affine (PWA) if $X = \bigcup_{i=1}^{n_r} X_i \subseteq \mathbb{R}^n$, where X_i are convex polyhedral regions with mutually disjoint interiors and $u(x) = H_i x + k_i$, $\forall x \in X_i$.

In case of discontinuities over overlapping boundaries, namely for some i, j $H_i x + k_i \neq H_j x + k_j$ for $x \in X_i \cap X_j \neq \emptyset$, we assume that $u(x)$ is defined as one of the possible values.

The solutions to the problems above together with the other problems mentioned in Section 1 are PWA functions, which gives the control input (the first m elements of the optimal U) as an explicit function of x . We will in the next section present an efficient data structure which allows efficient evaluation of PWA functions.

3. On-line search tree

When a PWA controller is executed, the problem is to decide which polyhedral region X_i the current state x_i belongs to, and then compute the control input using the corresponding affine control law. The most direct way of doing this is by the following sequential search through the polyhedral regions of the partition.

Algorithm 1 (Sequential search).

- 1 $i \leftarrow 1$
- 2 while $x \notin X_i$ and $i \leq n_r$
- 3 $i \leftarrow i + 1$
- 4 end (while)
- 5 if $i = n_r + 1$, then $x \notin X$, (problem infeasible), terminate.
- 6 evaluate the control input, $u(x) = H_i x + k_i$

In the worst case Algorithm 1 checks every region (and every hyperplane) in the partition. We want a method to find the region to which a given x belongs by evaluating as few hyperplanes as possible.

An efficient way to exploit the convexity of polyhedral sets is to build off-line a binary search tree (for on-line use)

where at each level one linear inequality is evaluated. Consider the set of polyhedra $\{X_1, X_2, \dots, X_{n_r}\}$, and the corresponding set of affine functions $\{F_1, F_2, \dots, F_K\}$ defining an affine control law. Note that $K \leq n_r$ since several regions can have the same control law. Let all unique hyperplanes defining the polyhedra in the partition be denoted by $a_j^\top x = b_j$ for $j = 1, 2, \dots, L$, and define $d_j(x) = a_j^\top x - b_j$. Let the index representation \mathcal{J} of a polyhedron denote a combination of indexes combined with the sign of d_j , e.g. $\mathcal{J} = \{1^+, 4^+, 6^-\}$ would mean that $d_1(x) \geq 0$, $d_4(x) \geq 0$ and $d_6(x) \leq 0$. Such an index representation obviously defines a polyhedron in the state space, $\mathcal{P}(\mathcal{J})$. We can further define the set of polyhedral regions corresponding to \mathcal{J} as the index set $\mathcal{I}(\mathcal{J}) = \{i | X_i \cap \mathcal{P}(\mathcal{J}) \text{ is full-dimensional}\}$. For a set \mathcal{I} of polyhedra, we can also define an index set of corresponding affine functions $\mathcal{F}(\mathcal{I}) = \{k | F_k \text{ corresponds to } X_i, i \in \mathcal{I}\}$.

The idea is to construct a binary search tree such that for a given $x \in X$, at each node we will evaluate one affine function $d_j(x)$ and test its sign. Based on the sign we select the left or the right sub-tree. Traversing the tree from the root to a leaf node, one will end up with a leaf node giving a unique affine control law F_k . The main challenge is to design a tree of minimum depth such that we minimize the number of hyperplanes to be evaluated to determine the solution. Of secondary priority, is the desire to keep the total number of nodes in the tree at a minimum, as this would decrease the on-line memory requirements.

Each node of the tree will be denoted by N_k , and we will use a list \mathcal{U} to keep the indices of the nodes which are currently unexplored. An unexplored non-leaf node N_k will consist of $(\mathcal{I}_k, \mathcal{J}_k)$, where \mathcal{J}_k is the index set (with signs) of hyperplanes obtained by traversing the tree from the root node to N_k and $\mathcal{I}_k = \mathcal{I}(\mathcal{J}_k)$. An explored non-leaf node will contain an index j_k to a hyperplane, while a leaf node will contain an affine control law, F_k . See Fig. 1 for an example of a simple search tree. We will use the notation ‘ \pm ’ for statements which should be repeated for both ‘+’ and ‘-’. Let $|\cdot|$ denote the number of elements in a set.

Note that $\mathcal{I}(\mathcal{J} \cup j^\pm) \subseteq (\mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^\pm))$, and that the difference between these two sets can be characterized by the following lemma:

Lemma 1. If $i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+)$ but $i \notin \mathcal{I}(\mathcal{J} \cup j^+)$, then X_i is split into two full-dimensional polyhedra by the hyperplane j , i.e. $i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. The same result holds when j^+ and j^- are interchanged.

Proof. Since $i \notin \mathcal{I}(\mathcal{J} \cup j^+)$ then $\mathcal{P}(\mathcal{J} \cup j^+) \cap X_i$ is not full-dimensional. But since $i \in \mathcal{I}(\mathcal{J})$ and $\mathcal{P}(\mathcal{J}) = \mathcal{P}(\mathcal{J} \cup j^-) \cup \mathcal{P}(\mathcal{J} \cup j^+)$ we have that $\mathcal{P}(\mathcal{J} \cup j^-) \cap X_i$ is full-dimensional, and so is $\mathcal{P}(j^-) \cap X_i$, which implies $i \in \mathcal{I}(j^-)$ and completes the proof. \square

When exploring a node of the tree, the main goal is to reduce the number of remaining control laws as much as possible from the current to the next level of the tree. More

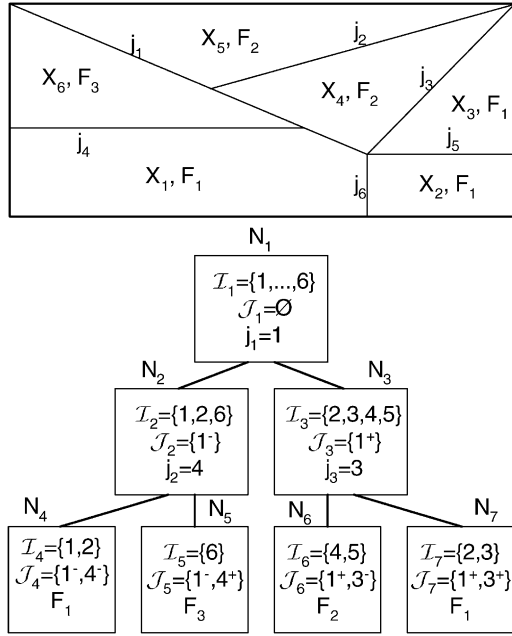


Fig. 1. Search tree generated from partition with $n_r = 6$ and $K = 3$.

precisely, for a node $N_k = (\mathcal{I}_k, \mathcal{J}_k)$, we want to select the hyperplane j_k as $\arg \min_j \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|)$, where $\mathcal{I}_k^\pm = \mathcal{I}(\mathcal{J}_k \cup j^\pm)$. This does, however, require the computation of \mathcal{I}_k^\pm for every j . Lemma 1 provides a computationally efficient approximation of \mathcal{I}_k^\pm as $\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^\pm)$. One can further get the exact \mathcal{I}_k^\pm by for each $i \in \mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$ solving the two LPs

$$\min_{x \in X_i} \pm d_j(x). \quad (15)$$

For each hyperplane j , one can from the solution of these LPs say on which side of the hyperplane the region X_i is, or that X_i is split by the hyperplane. As the approximation can be used to select a few candidate hyperplanes, there is only a small number of LPs which have to be solved. We can now present an algorithm to build a binary search tree:

Algorithm 2 (Build search tree).

- 1 Compute the index sets $\mathcal{I}(j^+)$ and $\mathcal{I}(j^-)$ for every $j \in \{1, \dots, L\}$.
- 2 The root node of the tree is initialized as $N_1 \leftarrow (\{1, \dots, n_r\}, \emptyset)$.
- 3 The set of unexplored nodes is initialized as $\mathcal{U} \leftarrow \{N_1\}$.
- 4 Select any unexplored node $N_k \in \mathcal{U}$ and let $\mathcal{U} \leftarrow \mathcal{U} \setminus N_k$.
- 5 Compute the approximations $\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^\pm)$ for all j , and sort the hyperplanes by the quantity $\max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^+))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^-))|)$.
- 6 Compute (the exact) $\mathcal{I}_k^\pm = \mathcal{I}(\mathcal{J}_k \cup j^\pm)$ for each of the first n_j elements of the sorted list of step 5 (see below for how n_j is selected). This is done by solving the LPs (15) for each $i \in \mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. Select j_k among these as $j_k = \arg \min_j \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|)$.

- 7 Complete the node as $N_k \leftarrow j_k$, and create two child nodes, $N^\pm \leftarrow (\mathcal{I}_k^\pm, \mathcal{J}_k \cup j^\pm)$.
- 8 If $|\mathcal{F}(\mathcal{I}^\pm)| > 1$, add N^\pm to \mathcal{U} . Else N^\pm is a leaf node, let $N^\pm \leftarrow \mathcal{F}(\mathcal{I}^\pm)$.
- 9 If $\mathcal{U} \neq \emptyset$, go to step 4, else terminate.

The computationally most expensive steps of this algorithm are steps 1 and 6. In step 1, one has to determine for each hyperplane, which side every region X_i lies on. This can be implemented by solving $2Ln_r$ LPs (15), which is computationally expensive for large problems. If the vertices of every X_i are available, these LPs can be replaced by simple arithmetic operations, giving considerably faster computation. If computation of the vertices is considered to expensive, one can for each X_i compute a set of points V_i , such that $X_i \subseteq \text{Conv}(V_i)$ (*Conv* denotes the convex hull). Such vertices can e.g. be found by using outer parallelotopic approximations as in Vicino and Zappa (1996). Each of the $2Ln_r$ cases can now be determined by simple arithmetic operations, except when *Conv*(V_i) is split by a hyperplane, when LPs still has to be solved.

In step 6, one also has to solve LPs to find the exact \mathcal{I}_k^\pm . The number n_j of hyperplanes which are checked in step 6 can be varied to trade-off between the off-line time required to generate the search tree and the complexity of the tree. In the examples of Section 5, n_j has been chosen to be $\lfloor j_{\text{approx}} \rfloor$, where $j_{\text{approx}} = \lfloor j \mid \max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j))|) = \min_{j_i} \max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j_i))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j_i))|) \rfloor$, which means that only hyperplanes which minimize the criterion in step 5 are considered in step 6. To further decrease off-line computation time, one can in step 5 consider only hyperplanes corresponding to remaining polyhedral regions \mathcal{I}_k (e.g. j_4 and j_6 for node N_2 in Fig. 1). Moreover, hyperplanes defining the boundary only between regions with the same control law (as j_2 , j_5 and j_6 in Fig. 1) can also be disregarded in step 5, as they are not needed to complete the search tree.

Often the best hyperplane j_k from step 6 is not unique. Among the set of hyperplanes which are best from the criterion in step 6, one can further refine the selection. Consider

1. $\min_j (\max(|\mathcal{I}_k^-|, |\mathcal{I}_k^+|))$,
2. $\min (|\mathcal{I}_k^-|, |\mathcal{I}_k^+|)$ and $\min (|\mathcal{F}(\mathcal{I}_k^-)|, |\mathcal{F}(\mathcal{I}_k^+)|)$.

By considering the first of these additional criteria, one tries not only to reduce the number of possible control laws from one level of the tree to the next, but also the number of polyhedral regions in which the state x_i may be. Reducing the complexity between tree levels in this way, has in examples shown beneficial results. The second criterion considers the least complex of the two child nodes. By reducing the complexity of this node, one can reduce the total number of nodes in the tree. This will however not contribute to reducing the depth of the tree. The next algorithm is used on-line to traverse the binary tree (see e.g. Aho, Hopcroft, & Ullman, 1983).

Algorithm 3 (Traverse search tree).

- 1 Let the current node N_k be the root node of the tree.
- 2 while N_k is not a leaf node
- 3 Evaluate the hyperplane $d(x) = a^T x - b$ corresponding to N_k .
- 4 Let N_k be the child node according to the sign of $d(x)$.
- 5 end (while)
- 6 Evaluate the control input $u(x)$ corresponding to N_k .

In general, the worst-case number of arithmetic operations required to search the tree and evaluate the PWA function is $(2n + 1)D + 2nm$, where D is the depth of the tree, m is the number of inputs and n is the number of states. At each node there are n multiplications, n additions and 1 comparison. Moreover, $2nm$ operations are required to evaluate the affine state feedback of the leaf node. Regarding memory requirements for the data structure, the most efficient is to store each of these solutions in a table, and give a pointer to an element in this table for each leaf node in the tree. Similarly, there is only a small subset of all the hyperplanes representing the regions X_i which is used in the search tree. Moreover, each of these hyperplanes are usually used in several nodes of the tree. So the hyperplanes should also be stored in a table, while using pointers to this table in the non-leaf tree nodes. This would require each leaf node in the tree to contain one pointer to a table of control laws, while each non-leaf node would contain one pointer to a table of hyperplanes, and two additional pointers to its child nodes.

4. Estimated complexity of the tree

This section will give an estimate of the depth and number of nodes in a tree for a given problem size. Such an estimate has to be based on how good the hyperplanes selected in step 6 of Algorithm 2 are. This estimate is given for the case when we want to find the exact region a state x belongs to without considering that several regions can have the same affine control law. In the best case we will in each node of the tree be able to select a hyperplane which has half of the remaining regions on each side. This will obviously give a tree where the depth would be $D = \lceil \log_2(n_r) \rceil$, and each hyperplane would be stored once in the tree. Obviously this best case estimate would not be possible for anything else than problems with a very special partition.

We can however give a more realistic estimate. Assume that the hyperplane selected in a node N_k has the property $\max(|\mathcal{S}_k^-|, |\mathcal{S}_k^+|) / |\mathcal{S}_k| \leq \alpha$, $\alpha \in [0.5 \ 1)$, where $\alpha = 0.5$ corresponds to the best case. Since $|\mathcal{S}_k| = n_r$ for the root node, the depth of the tree would then be given by

$$n_r \alpha^D = 1, \tag{16}$$

or equivalently,

$$D = \left\lceil \frac{\ln 1/n_r}{\ln \alpha} \right\rceil = \left\lceil -\frac{\ln n_r}{\ln \alpha} \right\rceil. \tag{17}$$

If the tree is ‘full’, that is the depth is the same for all leaf nodes, the approximate number of nodes in the tree is

$$2^D = 2^{\left\lceil -\frac{\ln n_r}{\ln \alpha} \right\rceil} \approx n_r^{-\ln 2 / \ln \alpha}. \tag{18}$$

In our experience, an α of $\frac{2}{3}$ is a conservative estimate when using Algorithm 2. This would give $D = \lceil 1.7 \log_2 n_r \rceil$ and the number of nodes would be $n_r^{1.7}$. However, regardless of the size of α , the depth of the tree would be a logarithmic function of n_r , while the number of nodes would be polynomial in n_r . Note that the complexity of the tree would be considerably reduced in the case of explicit MPC solutions, where we can stop dividing the tree when we know the affine control law which is optimal, without knowing the exact polyhedral region in which the state is. Moreover, the tree is usually far from ‘full’, so the estimate of number of nodes is conservative. The examples in the next section therefore show a considerably lower complexity than the given estimate.

5. Examples

In the examples of this section, Algorithm 1 is implemented by storing each region in the partition, represented by its hyperplanes, and the corresponding affine function parameters. Obviously this algorithm could be improved both in terms of computational complexity and storage, e.g. by computing unions of polyhedra where the affine control law is the same as in Bemporad, Fukuda, and Torrisi (2001).

Example 1. Consider the linear system (Borrelli et al., 2001b)

$$y(t) = \frac{1}{s^4} u(t) \tag{19}$$

which is discretized with sampling time $T_s = 1$. The system is subject to input constraints, $-1 \leq u(t) \leq 1$ and output constraints, $-10 \leq y(t) \leq 10$. For the quadratic case (mp-QP), an MPC controller is designed with $N = 6$, $Q = I_{4 \times 4}$, $R = 0.01$ and $P = 0$. The explicit solution consists of 213 regions. Table 1 reports the comparison between Algorithm 1, the algorithm from (Borrelli et al., 2001b) and Algorithm 3 in terms of required storage and arithmetic operations to compute the control input. The generated search tree has a depth of 12, containing 1473 nodes. Six hundred and seventy four unique hyperplanes occur in the tree, and there are 59 different affine control laws representing the PWA function.

Table 1
Performance of search tree for mp-QP solution

	Alg. 1	Alg. from ^a	Alg. 3
Storage (real numbers)	9740	1065	1615
Storage (pointers)	—	—	2945
Arith. ops. (worst case)	20668	3489	116

^aBorrelli et al. (2001a).

Table 2
Performance of search tree for mp-MILP solution

	Alg. 1	Alg. 3
Storage (real numbers)	34776	1350
Storage (pointers)	—	2277
Arithmetic ops. (worst case)	68834	156

The off-line computations to generate the tree was done in less than 1 min, using Matlab 6.0 on a 1 GHz Pentium III.

Example 2. In Borrelli, Bemporad, Fodor, and Hrovat (2001a) the authors gave a solution to a constrained optimal control problem, solving a traction control problem using a hybrid model. This was formulated as an mp-MILP, and solved explicitly. The resulting controller was a PWA function consisting of 508 polyhedral regions, giving a single control input as a function of 5 states. The performance of using a search tree to represent this PWA function compared to a sequential search is shown in Table 2. The search tree has a depth of 12, and consists of 1139 nodes.

6. Conclusions

We have presented a binary tree structure designed to give very efficient evaluation of PWA functions. Our method gives a PWA function evaluation time which is *logarithmic* in the number regions representing the PWA function. This allows considerably faster PWA function evaluation than existing methods. As the explicit solutions to MPC problems are (often complex) PWA functions, the method is expected to widely increase the sampling rates by which MPC can be applied.

References

- Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data structures and algorithms*. Reading, MA: Addison-Wesley.
- Bemporad, A., Borrelli, F., & Morari, M. (2000a). Explicit solution of LP-based model predictive control. In *Proceedings of the 39th IEEE conference on decision and control* (pp. 632–637). Sydney, Australia.
- Bemporad, A., Borrelli, F., & Morari, M. (2000b). Piecewise linear optimal controllers for hybrid systems. In *Proceedings of the American control conference* (pp. 1190–1194). Chicago, IL.
- Bemporad, A., Borrelli, F., & Morari, M. (2002a). On the optimal control law for linear discrete time hybrid systems. In M. Greenstreet & C. Tomlin (Eds.), *Hybrid systems: computation and control* (pp. 105–119). In *Lecture notes in computer science*, Vol. 2289. Berlin: Springer.
- Bemporad, A., Borrelli, F., & Morari, M. (2003). Min–max control of constrained uncertain discrete-time linear systems. *IEEE Transactions on Automatic Control*, submitted for publication.
- Bemporad, A., Borrelli, F., & Morari, M. (2002). Model predictive control based on linear programming—the explicit solution. *IEEE Transactions on Automatic Control*, 47(12), 1974–1985.
- Bemporad, A., & Filippi, C. (2003). Suboptimal explicit RHC via approximate multiparametric quadratic programming. *Journal of Optimization Theory and Applications*, 117, 1 (to appear).
- Bemporad, A., Fukuda, K., & Torrisi, F.D. (2001). Convexity recognition of the union of polyhedra. *Computational Geometry*, 18, 141–154.
- Bemporad, A., Morari, M., Dua, V., & Pistikopoulos, E. N. (2002b). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3–20.
- Borrelli, F., Baotic, M., Bemporad, A., & Morari, M. (2001a). Efficient on-line computation of constrained optimal control laws. In *Proceedings of the 40th IEEE conference on decision and control* (pp. 1187–1192). Orlando, FL.
- Borrelli, F., Bemporad, A., Fodor, M., & Hrovat, D. (2001b). A hybrid approach to traction control. In: A. Sangiovanni-Vincantelli & M.D. Di Benedetto (Eds.), *Hybrid Systems: Computation and Control* (pp. 162–174). Springer-Verlag.
- Garcia, G., & Tarbouriech, S. (1999). Piecewise-linear robust control of systems with input constraints. *European Journal of Control*, 5(1), 157–166.
- Goodrich, M. T., & Ramaiyer, K. (1999). Point location. In J.-R. Sack & J. Urrutia (Eds.), *Handbook of computational geometry* (pp. 121–153). Amsterdam. Elsevier Science Publishers B.V. North-Holland.
- Hassibi, A., & Boyd, S. (1998). Quadratic stabilization and control of piecewise linear systems. In *Proceedings of the American control conference*, Philadelphia, PA, USA (pp. 3659–3664).
- Johansen, T. A. (2002). On multi-parametric nonlinear programming and explicit nonlinear model predictive control. In *Proceedings of the 41st IEEE conference on decision and control*. Las Vegas.
- Johansen, T. A., & Grancharova, A. (2002). Approximate explicit model predictive control implemented via orthogonal search tree partitioning. In *Preprints XV IFAC world congress*. Barcelona.
- Johansen, T. A., Petersen, I., & Slupphaug, O. (2002). Explicit suboptimal linear quadratic regulation with input and state constraints. *Automatica*, 38(7), 1099–1111.
- Rantzer, A., & Johansson, M. (2000). Piecewise linear quadratic optimal control. *IEEE Transactions on Automatic Control*, 45, 629–637.
- Seron, M. M., DeDoná, J. A., & Goodwin, G. C. (2000). Global analytical model predictive control with input constraints. In *Proceedings of the 39th IEEE conference on decision and control* (pp. 154–159). Sydney.
- Slupphaug, O., & Foss, B. A. (1999). Quadratic stabilization of discrete-time uncertain nonlinear multi-model systems using piecewise affine state feedback. *International Journal of Control*, 72, 686–701.
- Snoeyink, J. (1997). Point location. In J. E. Goodman & J. O'Rourke (Eds.), *Handbook of discrete and computational geometry* (pp. 559–574). Boca Raton, FL: CRC Press.
- Tøndel, P., Johansen, T. A., & Bemporad, A. (2003). An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3), 489–497.
- Vicino, A., & Zappa, G. (1996). Sequential approximation of feasible parameter sets for identification with set membership uncertainty. *IEEE Transactions on Automatic Control*, 41, 774–785.
- Wredenhagen, G. F., & Belanger, P. R. (1994). Piecewise-linear LQ control for systems with input constraints. *Automatica*, 30(3), 403–416.