# Computation and Approximation of Piecewise Affine Control Laws via Binary Search Trees

P. Tøndel[1], T.A. Johansen[1] and A. Bemporad[2]

**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 (MPC), as it allows the controller to be implemented on-line with small computational effort. The computation time is *logarithmic* in the number of regions in the PWA partition. A method for generating an *approximate* PWA function based on a binary search tree is also presented, giving further simplification of PWA control.

## 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 [1] 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 in [2]. An alternative solution strategy is given in [3], where pre-determination of a small set of sampling instants where the active set is allowed to change gives a suboptimal solution, and in [4] based on a geometric interpretation of the QP problem. Suboptimality of mp-QP is also introduced in [5] by relaxing the optimality conditions, and in [6], by imposing an orthogonal structure to the state space partitioning. In [7] 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 [8], and explicit robust MPC is treated in [9]. 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. [10, 11, 12, 13, 14]).

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). For the case of exact solutions to the mp-QP and mp-LP problems, the authors of [15] propose a more efficient method regarding both search time and storage by exploiting properties of the value function. This method is however not feasible for 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 [16, 17] 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 MPC problems, which may have higher dimensions. The off-line mp-QP algorithm of [1] 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 so that the resulting 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. Some preliminary results were presented in [18]. We also present an algorithm for suboptimal evaluation of explicit MPC solutions that allows to trade-off between the complexity of the search tree and the optimality of the solution. Similar to the method of [6] the algorithm uses an orthogonal partition to get highly efficient on-line evaluation.

## 2 Explicit Constrained Linear MPC

Below we give a short description of linear MPC problems and their explicit solutions. For more details, see [1, 7]. For treatment of hybrid systems and the explicit solution of mp-MILP, see [8]. Consider the linear system

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ y_t &= Cx_t \end{aligned} \tag{1}$$

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

$$J^*(x_t) = \min_{u_t,\dots,u_{t+M-1}} \|x_{t+N}\|_p^P + \sum_{k=0}^{N-1} (\|x_{t+k+1}\|_p^Q + \|u_{t+k}\|_p^R) \tag{2}$$

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

$$\begin{aligned} y_{\min} \le y_{t+k|t} \le y_{\max}, k = 1, ..., N \\ u_{\min} \le u_{t+k} \le u_{\max}, k = 0, ..., N-1 \end{aligned}$$

[1]Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway, Petter.Tondel@itk.ntnu.no, Tor.Arne.Johansen@itk.ntnu.no.

[2]Dipartimento di Ingegneria dell'Informazione, University of Siena, 53100 Siena, Italy, bemporad@dii.unisi.it.

$$x_{t+k+1|t} = Ax_{t+k|t} + Bu_{t+k},\ k \geq 0 \quad (3)$$
$$u_{t+M+k} = 0,\ k \geq 0$$
$$y_{t+k|t} = Cx_{t+k|t},\ k \geq 0$$

For $p = 2$, $\|x\|_p^E = x^T Ex$, $Q = Q^T \geq 0$, $R = R^T > 0$ and $P \geq 0$. For $p = 1$ and $p = \infty$, $\|x\|_p^E = \|Ex\|_p$. For ease of notation, we will 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. $p = 2$: (mp-QP)

$$\min_{U=[u_t^T\ \dots\ u_{t+M-1}^T]^T} U^T HU + x^T FU \quad (4)$$
$$s.t.\ GU \leq W + Sx. \quad (5)$$

2. $p = 1$ or $p = \infty$: (mp-LP)

$$\min_{U=[u_t^T\ \dots\ u_{t+M-1}^T\ \epsilon^T]^T} h^T U \quad (6)$$
$$s.t.\ GU \leq W + Sx, \quad (7)$$

where $\epsilon$ is a vector of slack variables (see [7]).

**Definition 1** *A function $u : X \to \mathbb{R}^s$, is piecewise affine (PWA) if $X = \cup_{i=1}^{n_r} X_i \subseteq \mathbb{R}^n$, 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 mp-QP/mp-LP problems above are continuous PWA functions, which gives the control input as an explicit function of the state, [1, 7]. We will in the next section present an efficient data structure which allows very fast 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 $x_t$ 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* $\quad i \leftarrow i + 1$
*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

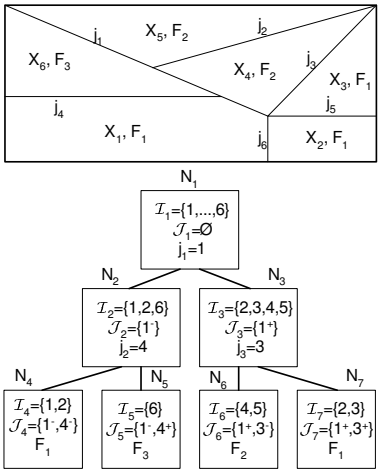


**Figure 1:** Partition and corresponding search tree

at each level one linear inequality is evaluated. Consider the set of polyhedra $\{X_1, X_2, ..., X_{n_r}\}$, and the corresponding set of affine functions $\{F_1, F_2, ..., F_K\}$ 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 $a_j^T x = b_j$ for $j = 1, 2, ..., L$, and define $d_j(x) = a_j^T x - b_j$. 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 a set 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 subtree. 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 Figure 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. □

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 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

$$\min_{x \in X_i} \pm d_j(x), \tag{8}$$

which decide on which side of hyperplane $j$ does polyhedron $i$ lie. 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 $j \in \{1, \ldots, L\}$.*
2 *The root node of the tree is initialized as $N_1 \leftarrow (\{1, \ldots, n_r\}, \emptyset)$.*
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 (8) 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, and 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 (8), 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 $X_i \subseteq Conv(V_i)$ ($Conv$ denotes the convex hull). Such vertices can e.g. be found by using outer parallellotopic approximations as in [19]. 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 $|j_{approx}|$, where $j_{approx} = \{j | \max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j))|) = \min_{j_l} \max(|\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j_l))|, |\mathcal{F}(\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j_l))|)\}$, 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 figure 1). Moreover, hyperplanes defining the boundary only between regions with the same control law (as $j_2$, $j_5$ and $j_6$ in Figure 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 $x_t$ 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. [20]).

**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 $\frac{\max(|\mathcal{I}_k^-|,|\mathcal{I}_k^+|)}{|\mathcal{I}_k|} \leq \alpha, \alpha \in [0.5 \;\; 1)$, where $\alpha = 0.5$ corresponds to the best case. Since $|\mathcal{I}_k| = n_r$ for the root node, the depth of the tree would then be given by

$$n_r \alpha^D = 1, \tag{9}$$

or equivalently,

$$D = \left\lceil \frac{\ln \frac{1}{n_r}}{\ln \alpha} \right\rceil = \left\lceil -\frac{\ln n_r}{\ln \alpha} \right\rceil. \tag{10}$$

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

$$2^D = 2^{\left\lceil -\frac{\ln n_r}{\ln \alpha} \right\rceil} \approx n_r^{-\frac{ln2}{\ln \alpha}}. \tag{11}$$

In our experience, an $\alpha$ 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 $\alpha$, 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 [21]).

**Example 1** *We have repeated the mp-QP example from [15] and generated a search tree for comparison. Consider the linear system*

$$y(t) = \frac{1}{s^4} u(t) \tag{12}$$

*which is discretized with sampling time $T_s = 1$. The system is subject to input constraints, $-1 \leq u(t+k) \leq 1$ and output constraints, $-10 \leq y(t+k) \leq 10$. For the quadratic case (mp-QP), an MPC controller is designed with $N = 6$, $Q = I_{4\times4}$, $R = 0.01$ and $P = 0$. The explicit PWA state feedback consists of 213 regions. Table 1 reports the comparison between Algorithm 1, the algorithm from [15] 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. 674 unique hyperplanes occur in the tree, and there are 59 different affine control laws representing the PWA function. The off-line computations to generate the tree was done in less than 1 minute, using Matlab 6.0 on a 1GHz Pentium III.*

**Table 1:** Performance of search tree for mp-QP solution

| | Alg. 1 | Alg. from [15] | Alg. 3 |
|---|---|---|---|
| Storage (real numbers) | 9740 | 1065 | 1615 |
| Storage (pointers) | - | - | 2945 |
| Arith. ops, worst case | 20668 | 3489 | 116 |

□

**Example 2** *In [22] 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.*

**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 |

□

## 6 Approximate Search Tree

As the complexity of the explicit MPC solution increases, it would be desirable to make a trade-off between complexity of the search tree and optimality of the solution. An approximate solution should be designed to take maximal advantage of the properties of the search tree presented in Section 3. The method proposed below does this by only allowing orthogonal hyperplanes in the tree nodes. This has two main advantages for the tree complexity: 1) The storage required by each hyperplane is only two numbers, an orthogonal direction and a position. 2) The on-line evaluation of each node is one comparison only. The complexity of this tree would be as in the best case analysis of section 4. For the algorithm to terminate, we have to require the underlying PWA function to be continuous. The proposed method will give a solution which is primary feasible, while the error compared to the exact solution is bounded. The following lemma will be used in the sequel to enforce primal feasibility on the solution, by considering only the first $m$ components of the solution $U(x)$.

**Definition 2** *The orthogonal projection of the polyhedron $A_1 x + A_2 y \leq b$ onto the $x$-space consists of every $x$ such that there exists $y$ with $A_1 x + A_2 y \leq b$.*

**Lemma 2** *Let $\overline{G}u \leq \overline{W} + \overline{S}x$ be the orthogonal projection of $GU \leq W + Sx$ onto the $(u,x)$-space, where $u \in \mathbb{R}^m$ are the first $m$ elements of $U \in \mathbb{R}^{Nm}$. Consider a polyhedral region $X_i$, and let $V = \{V_1, ..., V_{n_V}\}, V_j \in \mathcal{R}^n$ be a set of points such that $X_i \subseteq Conv(V)$. Then $\overline{G}u(V_i) \leq \overline{W} + \overline{S}V_i \;\; \forall \; V_i \in V$ implies that there exists $U(x)$ with $u(x)$ as the first $m$ components such that $GU(x) \leq W + Sx \;\; \forall \;\; x \in X_i$.*

*Proof:* Since $\overline{G}u(V_i) \leq \overline{W} + \overline{S}V_i \ \ \forall \ V_i \in V$ we have that $\overline{G}u(x) \leq \overline{W} + \overline{S}x \ \ \forall \ x \in X_i \subseteq Conv(V)$ due to convexity. It now follows from the definition of the orthogonal projection that for every $x \in X_i$ there exist $U(x)$ with $u(x)$ as the first $m$ components such that $GU(x) \leq W + Sx$. □

We will use

$$e_{abs} = \max_{x \in X_i} \|H_i x + k_i - u^*(x)\|_\infty \tag{13}$$

as a measure to how good an approximation $\hat{u}_i(x) = H_i x + k_i$ is in the region $X_i$. The algorithm can be summarized as follows: Given a hypercube $HC$, split $HC$ into two hypercubes $HC^-$ and $HC^+$ by an orthogonal hyperplane. Find affine control laws $H^- x + k^-$ and $H^+ x + k^+$ such the corresponding errors $e^+_{abs}$ and $e^-_{abs}$ are minimized in each hypercube. Iterate on the position of the hyperplanes until $e^+_{abs} \approx e^-_{abs}$. The reason we want $e^+_{abs} \approx e^-_{abs}$, is that one of these errors is an increasing function of the position of the hyperplane, while the other is decreasing. So $\max(e^+_{abs}, e^-_{abs})$ is minimized when $e^+_{abs} = e^-_{abs}$. If $e^\pm_{abs}$ is below an a priori provided bound, the hypercubes are kept as part of the approximate solution. Else, keep dividing $HC^-$ and $HC^+$ as was done with $HC$. Algorithm 4 shows how a hypercube $HC$ will be split into two parts by a hyperplane $h$.

**Algorithm 4** ***(Split hypercube)***

1. *Select a hyperplane $h$ which splits the hypercube $HC$ into two equal sized hypercubes, $HC^\pm$.*
2. *Let $V^\pm$ be the set of vertices (which are easily computed) of $HC^\pm$, and let $X_{HC^\pm}$ be a set of points. Initialize $X_{HC^\pm} \leftarrow V^\pm$.*
3. *Find a primary feasible affine function $u_{HC^\pm} = H^\pm x + k^\pm$ for $HC^\pm$, which minimizes $\|u_{HC^\pm}(x_i) - u^*(x_i)\|_\infty$ for all $x$ in $X_{HC^\pm}$, by solving*

$$\min_{H^\pm, k^\pm, \alpha^\pm} \ \alpha^\pm \tag{14}$$

$$s.t \quad \|H^\pm x_i + k^\pm - u^*(x_i)\|_\infty \leq \alpha^\pm \ \ \forall \ \ x_i \in X_{HC^\pm} \tag{15}$$

$$\overline{G}u(v_j) \leq \overline{W} + \overline{S}v_j \quad \forall \quad v_j \in V^\pm, \tag{16}$$

   *which can be formulated as an LP.*
4. *Find a set of points $X_{HC^\pm,new}$ in $HC^\pm$ such that $\|u_{HC^\pm}(x) - u^*(x)\|_\infty > \alpha^\pm_{max}, \ \ \forall \ \ x \in X_{HC^\pm,new}$, where $\alpha^\pm_{max}$ is the minimizing $\alpha^\pm$ from (14)–(16).*
5. *If $X_{HC^-,new} = \emptyset$ and $X_{HC^+,new} = \emptyset$ go to step 6, else let $X_{HC^\pm} \leftarrow X_{HC^\pm} \cup X_{HC^\pm,new}$, go to step 3.*
6. *Let $e^\pm_{abs} \leftarrow \alpha^\pm$. If $|e^-_{abs} - e^+_{abs}| \leq \epsilon$, where $\epsilon$ is some small tolerance, the best hyperplane has been found for this axis-orthogonal direction. Otherwise move the hyperplane $h$ accordingly to $e^-_{abs}$ and $e^+_{abs}$, form $HC^\pm$ and go to step 2.*

□

The computationally most expensive part in Algorithm 4 is step 4, which is equivalent to minimizing a PWA function. This can be done by Mixed Integer Linear Programming (MILP) and parametric-MILP as in [5], or by simply solving two LPs for each polyhedral region in the exact solution, which intersects the hypercube. The LP (14)–(16) may not have a feasible solution. We then split the hypercube into two equal sized parts.

We present an algorithm to create an approximate search tree below. An unexplored non-leaf node $N_k$ of the tree will contain a representation of the hypercube corresponding to the node, while an explored non-leaf node will contain an axis-orthogonal hyperplane.

**Algorithm 5** ***(Approximate search tree)***

1. *The root node of the tree is initialized as $N_1 \leftarrow (HC_1)$, where $HC_1$ is the hybercube in which the solution is to be approximated.*
2. *The set of unexplored nodes is initialized as $\mathcal{U} \leftarrow \{N_1\}$.*
3. *Select any unexplored node $N_k \in \mathcal{U}$. Let $\mathcal{U} \leftarrow \mathcal{U} \setminus N_k$.*
4. *For each axis-orthogonal direction $o$, find $e_{abs}(o) = \max(e^-_{abs}(o), e^+_{abs}(o))$ by applying Algorithm 4 and select the direction $o_{opt} = \arg\min_o e_{abs}(o)$. Let $h_k$ be the corresponding hyperplane, and $HC^\pm_k$ be the two hypercubes generated by the split.*
5. *Complete the node as $N_k \leftarrow (h_k)$ and create two child nodes $N^\pm \leftarrow (HC^\pm_k)$, where $V^\pm_k$ is the vertices of the corresponding hypercube.*
6. *Let $\epsilon_{max}$ be the maximum allowed error. If $e^\pm_{abs}(o_{opt}) > \epsilon_{max}$, add $N^\pm$ to $\mathcal{U}$.*
7. *If $\mathcal{U} \neq \emptyset$, go to step 3, else terminate.*

□

**Example 3** *We have tested Algorithm 5 on an example from [5], where the system $y(t) = 2(s-1)/(s^2+2s+5)u(t)$ is sampled with $T = 0.1s$, and an explicit MPC controller is computed for a time horizon $N = 6$ with input constraints, $-1 \leq u(t) \leq 1$. Because the hyperplanes generated by this method are forced to be in axis-orthogonal directions, the resulting controller will be complicated in parts of the state space where the shape of the exact solution differs from the axis-orthogonal directions. Often a linear transformation on the state would be beneficial before applying Algorithm 5, and in this case a rotation of the state space has been applied ($x_1$ and $x_2$ in the figures refers to the rotated coordinates). The inverse transformation must of course be applied to the state before traversing the tree on-line. The number of arithmetic operations to traverse this tree is $2nm + D$, as $1$ comparison is needed to evaluate each hyperplane and $2n$ operations are required to evaluate each element in the affine control law. Moreover, $n(n-1)$ additional operations are required to perform the inverse linear transformation. The exact solution to the mp-QP is shown in figure 2, while an approximate solution allowing $e_{abs} \leq 0.4$ is shown in figure 3. Table 3 shows the complexity of some approximate search trees compared to a search tree generated by applying Algorithm 2 to the exact solution. The approximate tree shows good performance in terms of arithmetic operations/storage when the chosen maximum allowed error is not too low.*

**Table 3:** Approximate search trees

| $e_{abs}$ | Depth | Arithmetic ops. | Storage (numbers) |
|---|---|---|---|
| 0.1 | 14 | 24 | 640 |
| 0.2 | 7 | 17 | 220 |
| 0.3 | 5 | 15 | 136 |
| 0.4 | 4 | 14 | 80 |
| Exact tree | 8 | 44 | 603 |

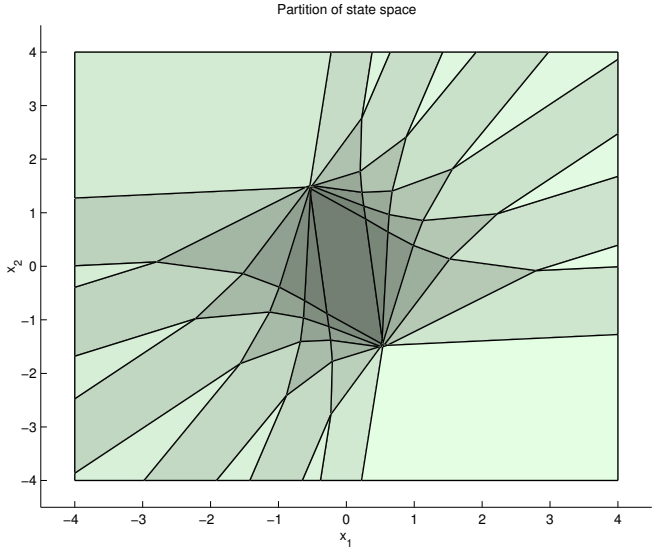


**Figure 2:** MPC controller, exact solution with 73 regions

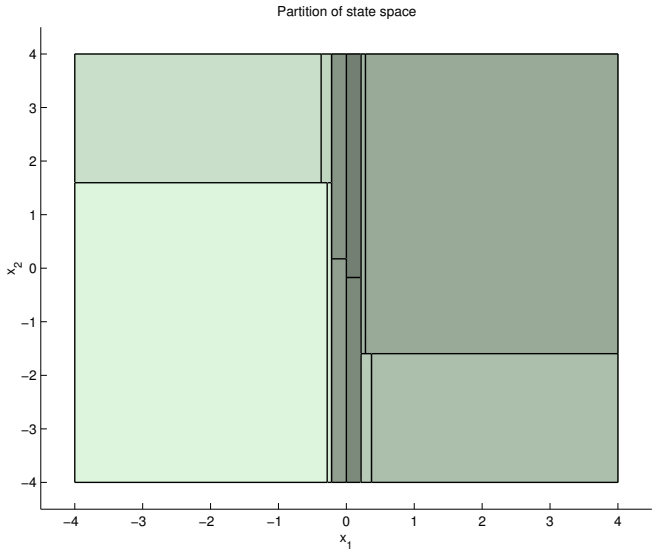


**Figure 3:** MPC controller, approximate solution with $e_{abs} = 0.4$

## 7 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 of 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. A method for generating an approximate solution to explicit MPC giving further improved on-line performance in terms of evaluation time and required storage is also presented.

### References

[1] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[2] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," in *Proc. 40th IEEE Conf. on Decision and Control*, (Orlando, Fl), 2001.

[3] T. A. Johansen, I. Petersen, and O. Slupphaug, "Explicit suboptimal linear quadratic regulation with input and state constraints," *Automatica*, vol. 38, no. 7, pp. 1099–1111, 2002.

[4] M. M. Seron, J. D. Doná, and G. Goodwin, "Global analytical model predictive control with input constraints," in *Proc. 39th IEEE Conf. on Decision and Control*, (Sydney), 2000.

[5] A. Bemporad and C. Filippi, "Suboptimal explicit RHC via approximate multiparametric quadratic programming," *Journal of Optimization Theory and Applications*, In press.

[6] T. A. Johansen and A. Grancharova, "Approximate explicit model predictive control implemented via orthogonal search tree partitioning," in *Preprints XV IFAC World Congress*, (Barcelona), 2002.

[7] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming - the explicit solution," *IEEE Trans. Automatic Control*, In press.

[8] A. Bemporad, F. Borrelli, and M. Morari, "Optimal controllers for hybrid systems: Stability and piecewise linear explicit form," in *Proc. 39th IEEE Conf. on Decision and Control*, 2000.

[9] A. Bemporad, F. Borrelli, and M. Morari, "Piecewise linear robust model predictive control," in *Proc. European Control Conference*, (Porto, Portugal), Oct. 2001.

[10] G. Garcia and S. Tarbouriech, "Piecewise-linear robust control of systems with input constraints," *European Journal of Control*, vol. 5, no. 1, pp. 157–166, 1999.

[11] G. Wredenhagen and P. Belanger, "Piecewise-linear LQ control for systems with input constraints," *Automatica*, vol. 30, no. 3, pp. 403–416, 1994.

[12] A. Rantzer and M. Johansson, "Piecewise linear quadratic optimal control," *IEEE Trans. Automatic Control*, vol. 45, pp. 629–637, 2000.

[13] A. Hassibi and S. Boyd, "Quadratic stabilization and control of piecewise linear systems," in *Proceedings American Control Conference*, 1998.

[14] O. Slupphaug and B. A. Foss, "Quadratic stabilization of discrete-time uncertain nonlinear multi-model systems using piecewise affine state feedback," *Int. J. Control*, vol. 72, pp. 686–701, 1999.

[15] F. Borrelli, M. Baotic, A. Bemporad, and M. Morari, "Efficient on-line computation of explicit model predictive control," in *Proc. 40th IEEE Conf. on Decision and Control*, (Orlando, Florida), 2001.

[16] J. Snoeyink, "Point location," in *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, eds.), ch. 30, pp. 559–574, CRC Press LLC, 1997.

[17] M. T. Goodrich and K. Ramaiyer, "Point location," in *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, eds.), pp. 121 – 153, Amsterdam: Elsevier Science Publishers B.V. North-Holland, 1999.

[18] P. Tøndel and T. A. Johansen, "Complexity reduction in explicit linear model predictive control," in *Preprints XV IFAC World Congress*, (Barcelona), 2002.

[19] A. Vicino and G. Zappa, "Sequential approximation of feasible parameter sets for identification with set membership uncertainty," *IEEE Trans. Automatic Control*, vol. 41, pp. 774–785, 1996.

[20] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*. Reading, MA.: Addison-Wesley, 1983.

[21] A. Bemporad, K. Fukuda, and F. D. Torrisi, "Convexity recognition of the union of polyhedra," *Computational Geometry*, no. 18, pp. 141–154, 2001.

[22] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "A hybrid approach to traction control," in *Proc. 4th International Workshop on Hybrid Systems: Comp. and Control*, (Rome), 2001.