2013 European Control Conference (ECC)
July 17-19, 2013, Zürich, Switzerland.

Evaluation of piecewise affine control law via graph traversal

Martin Herceg, Sébastien Mariéthoz, and Manfred Morari

***Abstract*— This paper presents a constructive algorithm for the effective evaluation of the piecewise affine (PWA) control laws generated in explicit model predictive control (MPC). One of the main limitations of explicit MPC is the construction of the search trees that are used to efficiently compute the PWA control law in real-time. The key idea developed in the paper consists in replacing these search trees by graphs that are directly generated while solving the multiparametric problems. It is shown that if the parameter space is explored in the breadth-first search (BFS) fashion, the bounds for the worst case runtime implementation can be obtained after the construction of an explicit solution. The proposed method traverses the graph in linear time that corresponds to a breadth of the graph on the average. Effectiveness of the approach is demonstrated numerically on a power electronics benchmark.**

I. Introduction

The application of explicit MPC consists of two phases. In the first *offline* synthesis phase, an optimal control problem is solved parametrically for the whole set of operating conditions. The solution of the first phase is a PWA control law that is referred to as the explicit solution. In the second *online* phase, the explicit solution is executed on a real-time platform, where the PWA control law is repetitively evaluated based on incoming measurements. As the online evaluation part is cheap to implement, this favors explicit MPC to run in very high sampling rates. Applications of this type are very common in power electronics, where the optimal control law is executed in orders of microseconds [4], [8].

The algorithm that is executed in the online part solves a *point location problem* and evaluates the associated affine function. In the explicit MPC concept there are additional limitations to be considered in the solution of the point location problem. Firstly, the implemented algorithm together with the explicit solution must fit the memory constraints that were allocated by the real-time platform. Secondly, the runtime of the algorithm must be strictly less than the sampling time and this must be ensured for all cases.

With respect to these requirements several algorithms have been proposed in the literature, which offer detailed complexity analysis with respect to memory requirements and execution time. The simplest way of solving the point location problem is referred to as *sequential search*. The principle is to consecutively test each region for set membership until the region where the point lies is found. The worst case runtime of the sequential algorithm is linear in the number of regions.

An efficient solution to the point location problem is a transformation of the explicit solution to a *binary search tree* and evaluation using the algorithm proposed in [12]. Once transformed, the running time of the algorithm is logarithmic in the number of regions and it is one of the fastest evaluation methods of explicit MPC. Recently, [10] suggested to transform the binary search tree to a multiway tree that can be evaluated effectively using parallel architectures. The disadvantage of these approaches, however, lies in the transformation step, which is only tractable for small MPC problems where the explicit solution consist at most a few thousand regions.

Other approaches solve the point location problem by exploiting the properties of the value function that arises from the MPC problem. The approach of [1] utilizes the sequential search to find the correct region by comparing values for the PWA descriptor function. It has been shown in [6] that using the *approximate nearest neighbor search* the point location problem can be solved in logarithmic time. The postprocessing step in this approach is of negligible complexity, but the method is only applicable for problems with linear cost functions. Postprocessing is also involved in the method of [3], which requires the computation of bounding boxes and the construction of an interval search tree.

In [13] the point location problem is solved by exploiting the adjacency list that is inherently associated with the polytopic partition of the PWA function. It traverses the associated graph in real-time along a line. The approach does not need any postprocessing step for implementation as this graph is constructed while solving the multiparametric problem. The authors provide an algorithm for computing the worst case runtime bounds via generation of a cross-tree.

Similarly to the approach introduced in [13], the present paper also exploits the graph defined by the adjacency list. The paper contribution is an alternative *graph traversal* approach that converges in less iterations with less number of arithmetic operations. The worst case bounds on algorithm runtime is obtained by a simple enumeration technique without generation of a cross-tree. The effectiveness of the graph traversal approach is demonstrated on a power electronics benchmark.

The authors are with the Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH Zürich), Physikstrasse 3, CH - 8092, Zürich, Switzerland. E-mail: {herceg,mariethoz,morari}@control.ee.ethz.ch

Notations and Definitions

Definition 1.1 (Polyhedron): A *polyhedron* is the intersection of a finite number of halfspaces: $\mathcal{P} := \{x \in \mathbb{R}^n \mid H_{\mathcal{P}} x \leq l_{\mathcal{P}}\}$. A *polytope* is a bounded polyhedron.

Definition 1.2 (Face): A subset of a polyhedron is called a *face* $\mathcal{F}$ of $\mathcal{P}$ if it can be represented as $\mathcal{F} = \mathcal{P} \cap \{x \in \mathbb{R}^n \mid h^T x = l\}$ for some valid inequality $h^T x \leq l$. The faces of polyhedron $\mathcal{P}$ of dimension 0, 1, $(n-2)$, and $(n-1)$ are called vertices, edges, ridges and facets, respectively.

Definition 1.3 (Partition): The set of polytopes $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_\mathcal{P}}\}$ is a *partition* if the polytopes are non-intersecting, $\mathcal{P}_i \cap \mathcal{P}_j, i \neq j$ is not full dimensional and $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$.

Definition 1.4 (Adjacency): Two polytopes $\mathcal{P}_i$ and $\mathcal{P}_j$, $i \neq j$ are adjacent if they share a common facet.

II. Problem Definition

A. MPC Formulation and Explicit Solution

Consider the class of linear time invariant (LTI) systems described in discrete-time setting

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

which dynamics is determined by the real matrices $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$. The manipulated inputs are denoted as u and the system state x is assumed to be known either by direct measurement or estimation. It is assumed that the pair (A, B) is controllable and observable. The variable $k = 0, 1, \ldots$ denotes the discrete time instant. The LTI system (1) can operate under constraints which act either on state variables $x \in \mathcal{X}$ or input variables $u \in \mathcal{U}$. The constraints $\mathcal{X}, \mathcal{U}$ can be expressed as linear inequality constraints

$$\mathcal{X} := \{x \in \mathbb{R}^n \mid H_\mathcal{X} x \leq l_\mathcal{X}\}, \tag{2a}$$

$$\mathcal{U} := \{u \in \mathbb{R}^m \mid H_\mathcal{U} u \leq l_\mathcal{U}\}. \tag{2b}$$

In the MPC approach, an optimization problem is formulated that involves the open-loop predictions of LTI model (1), operating constraints (2) and performance criterion $J(x, u)$ as follows:

$$\min_{u_k, \ldots, u_{N-1}} \quad J(x, u) \tag{3a}$$

$$\text{s.t.:} \quad x_{k+1} = Ax_k + Bu_k, \tag{3b}$$

$$x_N \in \Omega, x_k \in \mathcal{X}, u_k \in \mathcal{U}, k \in 0, 1, \ldots, N. \tag{3c}$$

In the receding horizon implementation, the predicted control actions $u_{k+1}, \ldots, u_N$ are discarded and only the first computed input u_0 is applied back to the plant. Due to closed loop stability reasons [9], the formulation of the optimization problem (3) contains a terminal set constraint Ω that is given as a polyhedral set of the form as (2). The performance criterion is usually expressed as

$$J(x, u) = \|Px_N\|_p + \sum_{k=0}^{N-1} (\|Qx_k\|_p + \|Ru_k\|_p) \tag{4}$$

with symmetric weighting factors $P \succ 0$, $Q \succeq 0$, and $R \succ 0$ that accounts the predictions from the actual measurement x_0 up to the finite prediction horizon N. If the norm $\|\cdot\|_p$ in (4) belongs to $p = \{1, \infty\}$ then the optimization problem (3) can be reformulated as a linear program and for $p = 2$ the problem boils down to a quadratic program [2]. In both cases the optimization problem (3) can be solved explicitly where the vector of initial conditions x_0 is treated as a parameter.

The explicit solution is characterized by the following theorem which applies to both linear and quadratic case.

Theorem 2.1 (Explicit solution): [2] Consider the multiparametric problem (3) with a linear (quadratic) objective function (4). Then, the set of feasible parameters $\mathcal{P}_\mathrm{f}$ is convex, the optimizer $u^*(x_0) : \mathcal{P}_\mathrm{f} \mapsto \mathbb{R}^n$ is continuous and piecewise affine over a polyhedral partition of $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$, and the optimal solution $J^*(x_0) : \mathcal{P}_\mathrm{f} \mapsto \mathbb{R}$ is continuous, convex and piecewise affine (quadratic).

An important consequence of Theorem 2.1 is that the resulting optimal control law is given as a PWA function

$$u^*(x_0) = F_i x_0 + g_i \text{ if } x_0 \in \mathcal{P}_i, i = 1, \ldots, n_\mathcal{P} \tag{5}$$

that is defined over a partition $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$ comprising of $n_\mathcal{P}$ polytopes. The PWA controller (5) is actually a solution of the offline phase in the explicit MPC approach. In the remaining online phase, this controller is evaluated for a particular value of the state x_0 that is described in the following section.

B. Point Location Problem

Definition 2.1 (Point location problem): [6] Given a vector x_0 and a set of non-intersecting polytopes $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$, determine any integer $i(x_0)$ $i \in \{1, \ldots, n_\mathcal{P}\}$ such that the polytope $\mathcal{P}_{i(x_0)}$ contains x_0. If $x_0 \notin \mathcal{P}_\mathrm{f}$, then $i(x) = 0$.

Once the integer i has been found, the remaining part is the evaluation of the affine control law, which is of negligible effort. For the high-speed evaluation of the PWA control law (5), it is required that the point location problem is solved within restricted sampling time. The key factor that influences the selection of the appropriate algorithm is driven by the number of regions $n_\mathcal{P}$ which determines the worst case bound for the online runtime. If the generated partition contains more than a few thousands regions, then most of the existing methods will not be applicable, because of the prohibitive postprocessing step. In the next section, the core of the algorithm will be presented that uses a search graph to solve the point location problem.

III. Graph Traversal Method

The domain of the PWA function forms a connected undirected graph where each polytope $\mathcal{P}_i$ in the partition represents a graph *vertex* and facet between two adjacent polytopes forms a graph *edge*. The graph is created by the multiparametric solver in the exploration phase and is

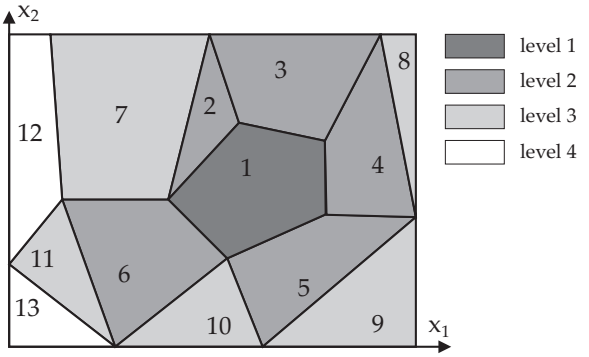

Fig. 1. Exploration of the parameter space using breadth-first search.

defined by an adjacency list. The main idea of the graph traversal method is to exploit the graph that is built in the exploration phase of the multiparametric solver such that the resulting adjacency list can be efficiently traversed without any postprocessing steps. The first step of the method requires a graph construction, which is actually implemented in the multiparametric solver. The multiparametric solver allows region discovery either in depth-first search (DFS) or breadth-first search (BFS) fashion. For the proposed graph traversal method the BFS is preferred in order to exploit the associated structure.

A. Graph Construction

It is important that the parameter space in multiparametric solver is explored in BFS fashion in order to ensure spatial ordering of regions in the graph. To understand the principle of BFS in parametric optimization, consider the example shown in Fig. 1. In the initialization phase, the optimization problem is solved for a particular value of the parameter x_c that is usually the center of the set given by constraints (2). Based on the optimizer and the optimal active set, the first region $\mathcal{P}_1$ is generated which is shown in the darkest gray shade on Fig. 1. The region $\mathcal{P}_1$ will be referred to as first level. In the exploration phase, all adjacent regions are found by crossing each facet of the region $\mathcal{P}_1$. The corresponding polytopes are denoted with numbers 2, 3, 4, 5, and 6 on Fig. 1 and this set will be referred to as second level. The exploration proceeds by finding new adjacent regions by consecutively crossing each facets for regions that were discovered in the second level. The third level comprises of regions 7, 8, 9, 10, and 11. By crossing each facets of polytopes contained in the third level, the fourth level is found that is built by regions 12 and 13. If facets of all discovered regions have been traversed, the BFS terminates. From Fig. 1 it is obvious that BFS evolves outwards in levels from the initial region 1 and the forming graph expands in breadth. The leveled organization of regions is shown in Fig. 1. The number of levels denotes the breadth of the graph b and for the example shown in Fig. 1, $b = 4$.

For the proposed graph search method it is important how the regions are stored in the adjacency list. In particular, from the adjacency list must be evident which regions are adjacent across given facets. One possibility is to introduce the following format of the adjacency list:

```
region) facet - adjacent regions
```

Using this format, the adjacency list for the example shown in Fig. 1 is given as:

```
1) 1-2, 2-3, 3-4, 4-5, 5-6
2) 1-7, 2-3, 3-1
3) 1-2, 2-{}, 3-4, 4-1
4) 1-1, 2-3, 3-8, 4-5
5) 1-1, 2-4, 3-9, 4-10
6) 1-7, 2-1, 3-10, 4-11
7) 1-12, 2-{}, 3-2, 4-6
8) 1-4, 2-{}, 3-{}
9) 1-5, 2-{}, 3-{}
10) 1-6, 2-5, 3-{}
11) 1-12, 2-6, 3-13
12) 1-{}, 2-{}, 3-7, 4-11
13) 1-{}, 2-11, 3-{}
```

Basically, the adjacency list defines a function $r_{\text{adj}} = G(r, f)$ where r is the region index, f is the facet index, and r_{adj} is the index set of a region that is adjacent to region r across the f-th facet. In the next part it will be shown how this information is crucial in determining the direction for graph traversal shown in the subsequent section.

B. Graph Traversal

The main idea of the algorithm is to exploit the spatial ordering of regions from BFS and to determine the particular polytope by traversing the levels outwards from the initial region. Consider again the example shown in Fig. 2. Assume that the given point is located in region 12. The algorithm starts in region $\mathcal{P}_1 := \{x \in \mathbb{R}^2 \mid H_{\mathcal{P}_1} x \leq l_{\mathcal{P}_1}\}$ by evaluating the inequality description for given value of the vector x_0. The result of the comparison determines the facets of polytope $\mathcal{P}_1$ that need to be crossed in order to move to the second level, i.e. the indices of rows $\{H_{\mathcal{P}_1} x_0 > l_{\mathcal{P}_1}\}$ that are true. The comparison operation can be viewed as determining the direction in which the algorithm should proceed. In Fig. 2 the comparison gives facets 1 and 5. From the adjacency list it follows that the neighboring region across facet 1 is region 2 and across facet 5 is region 6. Here it does not matter which facet to choose because both lead to the second level. Choosing the facet 1, the next region to explore is $\mathcal{P}_2 := \{x \in \mathbb{R}^2 \mid H_{\mathcal{P}_2} x \leq l_{\mathcal{P}_2}\}$. In region $\mathcal{P}_2$, the evaluation and comparison operation are performed again and it is found that across the first facet the next region in the third level to be explored is $\mathcal{P}_7$. The last comparison gives the result that the vector is located in region 12.

The graph traversal algorithm is given in Alg. 1. It comprises of a recursive formula that traverses always from the first level (or first region $\mathcal{P}_1$) to the outermost

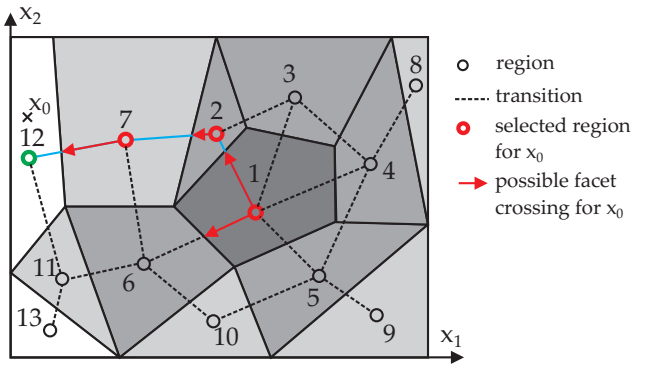

Fig. 2. Traversing the graph is based on finding a set of possible facets to cross via (6) which determines the direction.

Algorithm 1 Graph traversal

Input: Initial condition $x_0 \in \mathbb{R}^n$
Input: Partition $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$
Input: Adjacency list $r_\mathrm{adj} = G(r, f)$

Procedure i =GraphSearch(x, $\mathcal{P}_\mathrm{f}$, $G(r,f)$, i)
if $i = \{\}$ **then**
$i = 1$
end if
Find halfplanes that do not contain the point x_0

$$F = \{j \in \{1, \ldots, n_v\} \mid v = H_{\mathcal{P}_i} x_0,\ v_j > l_{\mathcal{P}_i}\} \quad (6)$$

if $F = \{\}$, the region is found. **then**
return i.
else
Pick one facet $f \in F$ and get the region index from the next level

$$r_\mathrm{adj} = G(i, f) \quad (7)$$

if $r_\mathrm{adj} = \{\}$, $x_0 \notin \mathcal{P}_\mathrm{f}$. **then**
return $\{\}$.
else
Pick one index r from the set r_adj and call

$$i = \mathrm{GraphSearch}(x, \mathcal{P}_\mathrm{f}, G(r, f), r) \quad (8)$$

end if
end if

level. For proper implementation of the algorithm, following assumptions are needed.

Assumption 3.1: All polytopes $\mathcal{P}_i$ in the partition $\mathcal{P}_\mathrm{f} = \cup_{i=1}^{n_\mathcal{P}} \mathcal{P}_i$ are in minimal hyperplane representation, i.e. each hyperplane defines a facet.

Assumption 3.2: The partition $\mathcal{P}_\mathrm{f}$ forms a convex union.

Theorem 3.1: [13] Alg. 1 returns the index i of the region $\mathcal{P}_i$ containing the point x_0 after at most n_r iterations where n_r is the number of regions visited along the path and $n_\mathrm{r} \leq n_\mathcal{P}$.

Proof See [13].

The Alg. 1 is an extended version of algorithm proposed in [13] because it considers all possible paths due (6). More importantly, the selected path does not affect the convergence because crossing any of the facets causes that the algorithm moves to a region closer to the point x_0. At each stage of the algorithm thus a subproblem (8) of a smaller size of the original point location problem is solved which is a principle of dynamic programming.

The choice of facets f from the set F depends on the particular implementation of the algorithm. Here are several possible selections available:

a) Generate a point inside the region $\mathcal{P}_i$ and determine the facet which is closest to x_0 by computing intersections with a ray that defines the direction. This approach is implemented in [13].
b) The simplest implementation is by picking always the first facet in the set F, or the last facet which does not require any additional computations.
c) Choose the facet based on the distance d between the point x_0 and the hyperplanes in the set F. The distance is given as $d_j = (h_j^T x_0 - l_j)/\|h_j\|, j \in F$ and the value with the maximum distance determines the facet that is the closest to the point x_0. Otherwise, the value with the minimum distance corresponds to a facet that is the furthest from x_0. Comparing with option a), here no point needs to be generated inside the actual region. Furthermore, since multiparametric solvers return polytopes in normalized form (i.e. $\|h_i\| = 1$), additional computational savings are possible.

Applying the selection of the facet that is always the closest to the point x_0 in the dynamic programming fashion reminds a behavior that solves the shortest path problem in a graph. In this case the solution is only approximation of the shortest path because the optimal solution is given by minimal number of facets to be crosses which cannot be guaranteed by Alg. 1. In the best case, the maximum number of regions to be traversed along the path is given by the maximum number of levels which corresponds to a breadth of the graph, i.e. $n_\mathrm{r} = b$. However, for partitions it cannot be ruled out that the Alg. 1 always moves towards ascending level so in general $n_\mathrm{r} \geq b$.

Remark 3.1: Degenerate cases in explicit solutions are characteristic that there are multiple regions adjacent to one facet. This case is ordinary handled by Alg. 1 because for the convergence it does not matter which path the algorithm takes and it will always proceed through regions towards x_0.

Remark 3.2: In principle, for the graph traversal Alg. 1 it does not matter if the graph is constructed using BFS or DFS because the graph lookup (7) will always return index of region that is closest to the initial point x_0. The approach is implemented such that BFS is used in the offline phase for the construction of the leveled structure which is then traversed online in DFS fashion to solve the point location problem as shown in Fig. 2.

Remark 3.3: If the convexity Assumption 3.2 was violated, the Alg. 1 could end up in an internal cycle and the graph traversal approach is not applicable. This danger comes into consideration when the explicit solution is not complete and there are missing regions in the partition caused by numerical problems in the multiparametric solver. Practically, the explicit solution is always checked for convexity and incomplete solution is discarded.

IV. Complexity Analysis

A. Runtime

The two main parts that build the algorithm are (6) the evaluation with comparison operation and (7) the graph lookup. At every step of the algorithm the matrix-vector product is computed in (6) which requires $n_v n$ multiplications and $n_v(n-1)$ summations. In addition, it is needed n_v comparisons to find the halfplanes. Moreover, if the selection of the facet is based on computing the maximum distance, $n_f - 1$ comparisons are needed where n_f is the number of facets in the set F. The graph lookup (7) does not require any arithmetic operation because the value is directly accessed from the memory. The number of hyperplanes $n_{v,i}$ defining the polytope i can vary as well as number of possible facets $n_{f,i}$, so for maximum n_r iterations it gives $\sum_{i=1}^{n_\mathrm{r}}(n_{v,i}n + n_{v,i}(n-1) + n_{v,i} + n_{f,i} - 1) = \sum_{i=1}^{n_\mathrm{r}} n_{v,i}(2n + n_{f,i} - 1)$ arithmetic operations. Define the number $n_{v,\max}$ as the maximum over the set of $n_{v,i}$. The number of possible facets $n_f \leq n_v - 1$ because if $n_f = n_v$ the region has been found. The conservative estimate of maximum number of arithmetic operations is given as $\sum_{i=1}^{n_\mathrm{r}} n_{v,i}(2n + n_{f,i} - 1) \leq \sum_{i=1}^{n_\mathrm{r}} n_{v,i}(2n + n_{v,i} - 2) \leq n_{v,\max}(2n + n_{v,\max} - 2)n_\mathrm{r}$. The runtime computational complexity is thus linear in the number of regions crossed along the path $O(n_\mathrm{r})$.

In the literature, it is of interest to express the worst case runtime complexity in the number of regions $n_\mathcal{P}$. Theoretically, the worst possible bound is linear in the number of regions $O(n_\mathcal{P})$ but this number occurs very rarely in practice. To have more realistic estimate, assume that the partition is given as hyperball or hypercube[1] and the graph is always initialized from the center. The volume of both hypecube or hyperball can be represented as $V_n = \mathrm{const} \cdot R^n$ where const is a constant, R is the radius of the ball. The volume can be rewritten as $V = n_\mathcal{P}\tilde{v}$ where $\tilde{v}$ is an average volume per one region and the radius as $R = b\tilde{d}$ where $\tilde{d}$ is an average diameter per one region. Putting these expressions together, the relation between the number of regions and the graph breadth can be given as $n_\mathcal{P} = \mathrm{const} \cdot b^n$. The relation leads to the result $b = \mathrm{const} \cdot \sqrt[n]{n_\mathcal{P}}$ and since $b \approx n_\mathrm{r}$, it reveals that the average runtime complexity can be estimated in number of regions as $O(\sqrt[n]{n_\mathcal{P}})$.

[1] The volume of a regular n-dimensional simplex with edge length s is given as $V_n = \frac{\sqrt{n+1}}{n!\sqrt{2^n}} s^n$ which can be generalized as $V_n = \mathrm{const} \cdot s^n$ and related to any convex polytope with average edge length s.

B. Computation of the Worst Case Bound

An algorithm that determines the exact worst case bound for graph traversal has been proposed in [13]. It is an enumerative approach that constructs a cross-tree from the underlying graph where the worst case is given as the depth of the resulting tree. The algorithm could be still applied in this case but it can easily become prohibitive for partitions with large number of polytopes. Therefore, we propose an alternative way of computing the worst case bound that avoids computation of a cross-tree. A reasoning behind is to exploit the leveled structure of the solution that allows only search paths that start from the first level and go to the last level. Hence, to get the worst bound, it suffices to evaluate the Alg. 1 for multiple initial conditions x_0 contained in regions only from the last level, count the iterations and find the maximum over that set. Since the number of regions in the last level is a fraction of the total number of regions, this test can be run very efficiently. Although the analysis gives an approximation of the worst case bound, the proposed approach is tractable and applicable to partitions with large number of polytopes.

C. Storage

The memory requirements for the Alg. 1 consist of allocated memory to store the partition $\mathcal{P}_\mathrm{f}$ in the irredundant hyperplane representation and the allocated memory to store the adjacency list $G(r, f)$. Assuming that the matrices in the hyperplane form of $\mathcal{P}_\mathrm{f}$ are stored as doubles and indices in $G(r, f)$ as integers, the total amount of allocated memory is given as $\sum_{i=1}^{n_\mathcal{P}}(n_{v,i}(n+1)\mathrm{sizeof(double)} + (n_{v,i} + n_{\mathrm{adj},i})\mathrm{sizeof(int)})$ where $n_{\mathrm{adj},i}$ is the total number of adjacent regions for the region i. The memory complexity thus depends linearly on the number of regions forming the partition $O(n_\mathcal{P})$. For actual implementation of the algorithm in the MPC concept, one has to store in addition the PWA control law (5) where the memory complexity also depends linearly on the number of regions.

V. Benchmark Example

The graph construction using BFS/DFS has been implemented in the parametric linear-complementarity problem (PLCP) solver [7] together with the graph search algorithm that is a part of MPT 3.0 [5]. The algorithm has been successfully applied on multiple test cases and a practically relevant example is presented next.

A. Benchmark Problem

To demonstrate the effectiveness of the graph search method, consider the model predictive control of the currents of a three-phase voltage source inverter. This problem is too complex to be implemented using the conventional binary search tree algorithm, therefore it was dealt with using online Fast Gradient MPC in [11]. The objective of the control strategy is to perform reference

TABLE I

Complexity assessment

	$N=4$, $n_{\mathcal{P}}=2197$ $b=9, n_{\mathcal{H}}=8283, n_l=96$		$N=5, b=11, n_{\mathcal{P}}=23129$ $n_{\mathcal{H}}=112795, n_l=488$	
	best-worst cases			
approach	# iter.	# oper.	# iter.	# oper.
a)	9-17	11.4k-30.6k	11-23	16.8k-44.0k
b)	9-21	5.0k-12.6k	11-33	6.3k-19.2k
c)	9-13	5.1k-8.0k	11-15	5.6k-9.6k

tracking while fulfilling polygonal input constraints. The associated MPC problem is formulated as

$$\begin{aligned} \min_{u_k,\ldots,u_{N-1}} \quad & \|Q(x_N - z_N)\|_2 + \ldots \\ & + \textstyle\sum_{k=0}^{N-1}(\|Q(x_k - z_k)\|_2 + \|Ru_k\|_2) \\ \text{s.t.:} \quad & x_{k+1} = Ax_k + Bu_k + B_w w_k, \\ & x_0 \in \mathcal{X}, u_k \in \mathcal{U}, w_0 \in 0.05\mathcal{U}, r_0 \in \mathcal{U} \end{aligned}$$

where the time-varying reference trajectory z is computed at each sampling period based on the power reference r_k and estimated grid voltage w_k. The state and input constraints are given as $\mathcal{X} = \{x \in \mathbb{R}^6 \mid \|x\|_\infty \leq 10\}$, $\mathcal{U} = \{u \in \mathbb{R}^2 \mid H_{\mathcal{U}} u \leq l_{\mathcal{U}}\}$, and the prediction horizon $N = \{4, 5\}$. The optimization problem features 10 parameters and is solved explicitly as a function of the vector of parameters $(x_0^T, w_0^T, r_0^T)^T \in \mathbb{R}^{10}$.

B. Comparison of the Three Graph Traversal Approaches

The results returned from the PLCP solver are displayed in Tab. I where n_{l} is the number of regions in the last level, and $n_{\mathcal{H}}$ corresponds to unique hyperplanes defining the partition. The results shown in Tab. I have been computed by running the graph traversal Alg. 1 for 6 randomly generated points [2] contained in each region from the last level and taking the best and the worst case. Tab. I displays the minimum and the maximum number of iterations needed for the graph traversal Alg. 1 to converge, and arithmetic operations to be performed for three possible implementations reviewed in Sec. III-B. The results indicate that in the best case the number of iterations corresponds to the breadth of the graph for all approaches. In the worst case, the number of iterations is different and the approach c) achieved the smallest number. Differences are also visible in the number of operations that favor the approach c) that chooses the facet based on the maximum distance from the point x_0. This supports the belief that the approach c) solves the shortest path problem in a graph.

C. Comparison to the Other Methods

In comparison, the generation of a binary search tree for this example requires solving of $2n_{\mathcal{H}}n_{\mathcal{P}}$ linear programs (LPs) that gives the number 36 395 002 for $N = 4$ and 5 217 671 110 LPs for $N = 5$, which makes the method of [12] prohibitive in this case. Similarly, the approach of [3] requires solving of $2nn_{\mathcal{P}}$ LPs. This gives 43 940 for $N = 4$ and 462 580 LPs for $N = 5$ which is tractable, but still time consuming. Furthermore, the approach of [1] requires $2n_{\mathcal{H}}n_{\mathcal{P}}$ operations that gives 50 026 for $N = 4$ and 154 538 for $N = 5$.

[2] One point is the Chebyshev centre of a polytope and five other points have been generated in random direction from the center.

VI. Conclusions

The paper has proposed an efficient graph traversal method for the evaluation of PWA control laws arising from linear MPC. The proposed method exploits the properties of the adjacently list that is inherently associated with the PWA control law and solves a shortest path problem in graph using approximated dynamic programming. The graph is built while constructing the PWA control law using multiparametric programming, which eliminates the postprocessing bottleneck. The proposed method is applicable for cases with large number of regions where other methods are prohibitive. The associated algorithm has a runtime complexity that is on average $O(\sqrt[n]{n_{\mathcal{P}}})$ and $O(n_{\mathcal{P}})$ in memory. The effectiveness of the approach has been demonstrated on a power electronics benchmark.

References

[1] M. Baotic, F. Borrelli, A. Bemporad, and M. Morari. Efficient On-Line Computation of Constrained Optimal Control. *SIAM Journal on Control and Optimization*, 47(5):2470–2489, September 2008.

[2] F. Borrelli. Constrained Optimal Control of Linear and Hybrid Systems. In *Lecture Notes in Control and Information Sciences*, volume 290. Springer, 2003.

[3] F.J. Christophersen, M. Kvasnica, C.N. Jones, and M. Morari. Efficient Evaluation of Piecewise Control Laws defined over a Large Number of Polyhedra. In *Proc. of the European Control Conf.*, Kos, Greece, July 2007.

[4] S. Mariéthoz et al. Comparison of Hybrid Control Techniques for Buck and Boost DC-DC Converters. *IEEE Trans. on Control Systems Technology*, 18(5):1126–1145, September 2010.

[5] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In *Proc. of the European Control Conf.*, Zurich, Switzerland, July 2013.

[6] C.N. Jones, P. Grieder, and S. Rakovic. A Logarithmic-Time Solution to the Point Location Problem for Parametric Linear Programming. *Automatica*, 42(12):2215–2218, December 2006.

[7] C.N. Jones and M. Morari. Multiparametric Linear Complementarity Problems. In *IEEE Conf. on Decision and Control*, December 2006.

[8] S. Mariéthoz, A. Domahidi, and M. Morari. High-Bandwidth Explicit Model Predictive Control of Electrical Drives. *IEEE Trans. on Industry Appl.*, 48(6):1980 – 1992, December 2012.

[9] D.Q. Mayne, J.B. Rawlings C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.

[10] M. Monnigmann and M. Kastsian. Fast explicit model predictive control with multiway trees. In S. Bittanti, A. Cenedese, and S. Zampieri, editors, *Preprints of the 18th IFAC World Congress*, volume 18, pages 1356–1361, Milano, Italy, 2011.

[11] S. Richter, S. Mariéthoz, and M. Morari. High-Speed Online MPC Based on a Fast Gradient Method Applied to Power Converter Control. In *American Control Conf.*, pages 4737 – 4743, Baltimore, MD, USA, June 2010.

[12] P. Tøndel, T.A. Johansen, and A. Bemporad. Evaluation of Piecewise Affine Control via Binary Search Tree. *Automatica*, 39(5):945–950, May 2003.

[13] Y. Wang, C.N. Jones, and J.M. Maciejowski. Efficient point location via subdivision walking with application to explicit MPC. In *European Control Conf.*, Kos, Greece, July 2007.