Managing time-storage complexity in point location problem: Application to Explicit Model Predictive Control

Farhad Bayat and Tor Arne Johansen and Ali Akbar Jalali

Abstract— **The online computational burden of linear model predictive control (MPC) can be moved offline by using multi-parametric programming, so called explicit MPC. The explicit MPC is a piecewise affine (PWA) function defined over a polyhedral subdivision of the set of feasible states. The online evaluation of such a control law needs to determine the polyhedral region in which the current state lies. This procedure is called the point location problem and its computational complexity is challenging. In this paper a new flexible algorithm is proposed which enables the designer to tradeoff between time and storage complexities. Utilizing the concept of hash tables and the associate hash functions the proposed method is modified to solve an aggregated point location problem in processing complexity independent of the number of polyhedral regions while the storage needs remains tractable. The effectiveness of this approach is supported by several numerical examples.**

I. Introduction

Model predictive control (MPC) has proved its ability in practice especially to deal with constraints directly in the design procedure. Unfortunately, the online MPC suffers with a restriction which has limited the applications of MPC mainly to slow dynamic systems. This limitation refers to the inherent online optimization since the MPC needs to solve an open loop constrained finite time optimal control (CFTOC) problem at every sampling instant [1].

Recently in [2], [3] and [4] the authors have shown that utilizing multi-parametric programming, it is possible to solve the linear and quadratic CFTOC problem offline. The standard CFTOC problem can be translated to a linear/quadratic optimization problem parameterized by the current state vector. Then, this optimization problem can be solved offline by means of multi-parametric programming. In [2] it has been proved that such a solution is a piecewise affine (PWA) function defined over a polyhedral subdivision of feasible states mapping the current state measurement to the optimal control effort. Therefore, the online closed loop operation is simplified to a PWA function evaluation of a measured state. The problem of determining the polyhedral region in which the current state lies refers to as "point location" or "set membership" problem in the literature [5] and [6]. Once the desired region is found, the associated affine optimal control law is evaluated and applied to the system, and this procedure is repeated in the next sampling time.

The point location problem is one of the most critical and restrictive parts in the explicit MPC since its complexity influences the hardware requirements in fast sampling applications. In [7] a binary search tree is constructed by dividing the polyhedral partitions using auxiliary hyper planes. By exploiting the piecewise affinity of the associated value function for MPC with linear cost function, authors in [2] and then in [8] have shown that such a problem can be solved with no need to store the polyhedral partitions. In the case that the cost-function is linear the authors in [9] have shown that the point location problem can be posed as a weighted Voronoi diagram, which can be solved using an approximate nearest neighbor (ANN) algorithm (see [10] for details). In [11] reachability analysis has been used in order to solve a reduced point location problem instead of dealing with the entire feasible set. The application of bounding boxes is exploited in [12] combining with a particular search tree to solve the point location problem. The subdivision walking method has been established in [13], which uses the concept of traveling from a predefined seed point in a known region, toward the current query point by walking from one region to the next until the region of interest is found. The idea of [11] is used in [14] to simplify the reachability analysis by using some reference points. More recently, in [15] a lattice PWA expression of the explicit MPC solution obtained based on the multi-parametric programming which can save some online calculation and memory requirements. Also in [16] a procedure has been introduced to tradeoff between the warm-start and online computational effort when a combination of explicit MPC and online computation is used.

In this paper a new algorithm has been proposed to efficiently solve the point location problem. This method substantially enables the designer to tradeoff between time and storage complexities. It is also shown that using the concept of hash tables [6] and [17] the worst case complexity of this method can be simplified to a constant number which is independent of the number of total polyhedral regions. It is illustrated that even for high complexity PWA functions; the proposed method remains tractable in terms of preprocessing complexities. Approximate explicit MPC approaches that rely on uniform gridding also benefit from efficient lookup through hash tables [18] although there is no obvious way in their approach to trade memory space for processing time for a given approximation accuracy.

The paper concludes with a series of numerical examples that illustrate the features of the method.

F. Bayat is with Department of Electrical Engineering, Iran University of Science and Technology, Narmak, 1684613114 Tehran, Iran fbayat@iust.ac.ir

T.A. Johansen is with Department of Engineering Cybernetics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway Tor.Arne.Johansen@itk.ntnu.no

A.A. Jalali is with Department of Electrical Engineering, Iran University of Science and Technology, Narmak, 1684613114 Tehran, Iran ajalali@iust.ac.ir

II. Explicit Solution of MPC

Consider a constrained discrete time LTI system:

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ y_t &= Cx_t \\ x_t &\in \mathbf{X}_c, u_t \in \mathbf{U}_c \end{aligned} \quad (1)$$

Where $A \in R^{n\times n}, B \in R^{n\times m}$ and $C \in R^{p\times n}$ and $\mathbf{X_c}, \mathbf{U_c}$ are polyhedral sets containing the origin in their interiors. Consider the linear/quadratic constrained finite time optimal control (CFTOC) problem:

$$\begin{aligned} J_N^*(x_t) = \min_{U_N} \|x_{t+N}\|_\rho^{Q_f} + \sum_{k=0}^{N-1} \|x_{t+k}\|_\rho^Q + \|u_{t+k}\|_\rho^P \\ subject\ to \quad x_{t+k} \in \mathbf{X}_c \quad \forall\ k = 1, ..., N \\ u_{t+k} \in \mathbf{U}_c \quad \forall\ k = 1, ..., N-1 \\ x_{t+k+1} = Ax_{t+k} + Bu_{t+k},\ k \geq 0 \end{aligned} \quad (2)$$

Where $Q = Q^{'} \succeq 0, P = P^{'} \succ 0, Q_f^{'} \succeq 0,\ \rho \in \{1/\infty, 2\}$ and $U_N = [u_0^T u_1^T ... u_{N-1}^T]^T$ is the optimization variable. Without loss of generality, in the following the case $\rho = 2$ is employed while all obtained results are valid for all cases with no further computation.

The explicit solution to the problem (2) has been studied in [2] and [4]. Here we will summarize the results, for completeness. The CFTOC problem can be rewritten by substitution of $x_{t+k} = A^k x_t + \sum_{j=0}^{k-1} A^j B u_{t+k-1-j}$ in (2):

$$\begin{aligned} J_N^*(x_t) = x_t^{'} Y x_t + \min_{U_N} \{U_N^{'} H U_N + x_t^{'} F x_t\} \\ subject\ to \quad x_{t+k} \in \mathbf{X}_c \quad \forall\ k = 1, ..., N \\ u_{t+k} \in \mathbf{U}_c \quad \forall\ k = 1, ..., N-1 \\ x_{t+k+1} = Ax_{t+k} + Bu_{t+k},\ k \geq 0 \end{aligned} \quad (3)$$

Where Y, H, F, G, W and E are constant matrices of appropriate dimension (see [2] for details).

The model predictive control uses the optimal solution U_N^* in a receding horizon fashion in which $u_t^* = [I\ 0 ... 0] U_N^*$ is applied to the system (see [2] and references therein for details). The following results have been proved in [2].

Theorem 1: Consider the optimization problem (3) and let $H > 0$. Then the set of feasible parameters X_f is convex, the optimizer $U_N^* : X_f \to R^s$ is a continuous and PWA partition, and the optimal cost function $J_N^* : X_f \to R$ is continuous, convex and piecewise quadratic.

Corollary 1: Once the optimization problem (3) solved offline, with $H > 0$ the explicit MPC control law is $u_t^* = [I_m 0 ... 0] U_N^* = f(x_t)$, where $f : X_f \to R^m$ is a continuous, PWA function defined over polyhedral regions as:

$$u_t^* = F_r x_t + G_r,\ \forall x_t \in P_r,\ r = 1, ..., N_r \quad (4)$$

Where $P_r = \{x \in R^n | H_r x \leq K_r\}$ is the r-th polyhedral region. F_r, G_r, H_r and K_r are piecewise constant matrices which can be computed explicitly using the KKT conditions for each region [2]. Corollary 1 demonstrates that the solution to explicit MPC is reduced to a simple affine function evaluation provided that the region P_r which contains the current state measurement (x_t) is known. Determining the relevant region P_r is the so-called point location problem. In the next section we propose a method to efficiently solve the point location problem.

III. Point location problem

Problem 1: let $\mathrm{P} := \{P_1, P_2, ..., P_{N_r}\}$ be a convex polyhedral partition such that: (i) $inter\,(P_i) \cap inter\,(P_j) = \emptyset,\ i \neq j,\ (P_i, P_j) \in \mathrm{P} \times \mathrm{P}$ and (ii) $\cup_{i=1}^{N_r} P_i = \mathrm{P}$ is convex. Where $inter\,(P_i)$ is the interior of P_i. Then, given a query point $x \in R^n$, find an appropriate integer number $i\,(x) \in \{0, 1, ..., N_r\}$ such that $i(x) = 0$ if $x \notin \mathrm{P}$ and $1 \leq i\,(x) \leq N_r$ if $x \in P_{i(x)}$.

Once Problem 1 is solved, the optimal control law can be calculated as given in (4) for $r = i(x)$. Algorithm 1 summarizes the online evaluation of the explicit MPC.

Algorithm 1 : Explicit MPC online computations

Step 1: For a measured state x determine the region P_r which contains the state vector x.

Step 2: Compute the associated PWA control law using (4) and apply to the system.

Step 3: Wait until the next sampling time and measure new state vector and then go to Step1.

The simplest way to solve the point location problem for a query point x is to check $H_r x \leq K_r$ for all regions one after another until the region containing the point x is found, so called exhaustive or direct search [7]. Algorithm 2 describes the direct search method. Unambiguously the worst case computational complexity of this method is linear in the number of polyhedral regions N_r.

Algorithm 2 : Point location

Step 1: Put $i(x) \leftarrow 0$.

Step 2:

```
while r ≤ N_r do
    if H_r x ≤ K_r then
        i(x) ← r; break;
    else
        r ← r + 1;
    end if
end while
```

A. Efficient point location: Main idea

To describe the main idea consider an artificial two dimensional polyhedral partition (Fig.1).

At first consider I_j the maximum interval length relating to the j-th axis (e_j), which is equal to the projection of the feasible set X_f on to the basis vector e_j. So $I_1 = a_{max} - a_{min}$ and $I_2 = b_{max} - b_{min}$. Then $I = \{I_1, I_2\}$ is the set of all maximum interval lengths.

Now introduce an integer scaling parameter $\varepsilon_j \geq 0$ denoting the interval resolution in the j-th axis, i.e. the I_j is divided to $n_j = 2^{\varepsilon_j}$ equally spaced sub-intervals.

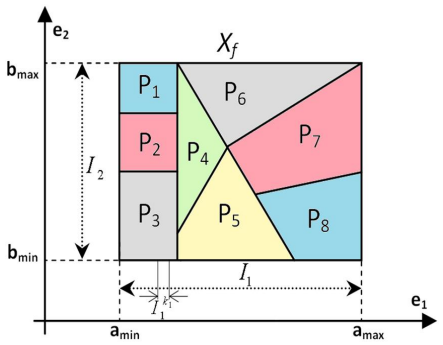

Fig. 1. Artificial 2D polyhedral partition.

remark 1: Using the parameters $\varepsilon_j,\ j = 1, ..., n_x$ one can compromise between the complexity of online computation and the complexity of required memory for storing the data structure needed for the point location.
In the following, first the general aspects of a relevant algorithm is established, then utilizing the concept of hash table the proposed algorithm is modified to solve an aggregated point location problem in time complexity independent of the number of polyhedral regions.

B. Offline computation

For brevity, consider the j-th axis e_j for which the offline procedure for generating a data structure to support online point location is stated in algorithm 3. Then the same algorithm can be used for every axis.

Algorithm 3 : Offline computation

Step 1: Choose a suitable scaling parameter $\varepsilon_j \geq 0$.
Step 2: Divide the j-th interval I_j into the $n_j = 2^{\varepsilon_j}$ sub-intervals $I_j^{k_j} = [L_j^{k_j},\ U_j^{k_j}],\ k_j = 1, ..., n_j$.
Step 3: $\forall\, r = 1, ..., N_r$ compute $\Phi_j^r = \left[\alpha_j^r\ \beta_j^r\right]$ where $\alpha_j^r = \min_i proj_{e_j}^{P_r},\ \ \beta_j^r = \max_i proj_{e_j}^{P_r}$.
Step 4: $\forall\, (k_j = 1, ..., n_j), (r = 1, ..., N_r)$: IF $(L_j^{k_j} < \beta_j^r)$ & $(U_j^{k_j} > \alpha_j^r)$ THEN $PI_j(k_j) \leftarrow r$.

Note that $PI_j(k_j)$ denotes the set containing the indices of polyhedral regions contained in the sub-interval $I_j^{k_j}$, and $proj_{e_j}^{P_r}$ denotes the orthogonal projection of P_r onto the e_j.

remark 2: Steps 3 and 4 in Alg.3 both together are equal to check whether $inter(proj_{e_j}^{P_r}) \cap I_j^{k_j} \neq \emptyset$ or not. More efficiently, it can be replaced by checking if $(L_j^{k_j} < \max(proj_{e_j}^{P_r}))$ and $(U_j^{k_j} > \min(proj_{e_j}^{P_r}))$ or not. Furthermore, when the vertex representation of polyhedral regions are available, then LPs in step 3 can be replaced by $\alpha_j^r = \min_i V_i^r(j)$ and $\beta_j^r = \max_i V_i^r(j)$, where $V_i^r(j)$ denotes the j-th element of i-th vertex of P_r.

C. Online computation

Applying the steps 1 through 3 of offline computation to all axes returns n_x polyhedral index sets $PI_j,\ j = 1, ..., n_x$. These n_x sets together with the polyhedral region descriptions (H_r, K_r) are the only data one needs to store to solve the online point location problem. The online procedure is presented in algorithm 4.

Algorithm 4 : Online computation

Step 1: At time t, for a given query point $\mathrm{x} = [x_1, ..., x_{n_x}]^T$, and for each axis e_j determine sub-interval $I_j^{k_j}$ which contains the j-th element of state x, i.e. $x_j \in I_j^{\bar{k}_j} = [L_j^{\bar{k}_j}\ U_j^{\bar{k}_j}]$.
Step 2: Let PL be a set of all common indices contained in $PI_j(\bar{k}_j),\ j = 1, ..., n_x$, that is $PL = \cap_{j=1}^{n_j} PI_j(\bar{k}_j)$.
Step 3: Using direct search through members of the set PL determine the region in which the query point x is located.

Note that the computation of the set intersection $PL = \cap_{j=1}^{n_j} PI_j(\bar{k}_j)$ is more efficient than the sequential search through all PI_j's elements. Moreover due to the incremental procedure in step 3 of algorithm 3, the elements in each index set $PI_j(\bar{k}_j)$ are ascending. Using this property a logarithmic-time solution to compute the set intersection is introduced in Algorithm 5 based on the binary search method.

Algorithm 5 : Set Intersection

Step 1: Initialize $PL \leftarrow \{PI_1(\bar{k}_1)\};\ \ j \leftarrow 2$.
Step 2:

while $j \leq n_x$ **do**
  $PL \leftarrow$ **Set_Intersect** $\left(PL, PI_j(\bar{k}_j)\right)$;
  $j \leftarrow j + 1$;
end while

FUNCTION Set_Intersect(S_1, S_2)

for $i = 1$ **to** $length(S_1)$ **do**
  $first \leftarrow 1;\ \ last \leftarrow length(S_2)$;
  $is\_common \leftarrow 0$;
  while $first \leq last$ **do**
    $mid \leftarrow [(first + last)/2]$
    if $S_2(mid) < S_1(i)$ **then**
      $first \leftarrow mid + 1$;
    else if $S_2(mid) > S_1(i)$ **then**
      $last \leftarrow mid - 1$;
    else
      $is\_common \leftarrow 1; break$;
    end if
  end while
end for

remark 3: The scaling parameters (ε_j) can be adjusted by setting $\varepsilon_j = 2$ and increasing it iteratively until a satisfactory result is met. Note that increasing the ε_js will decrease the online time complexity and increase the storage complexity simultaneously, so, the tradeoff is adjusted until both requirements are satisfied.
There are some efficient algorithms which can be used to solve the step 1 in Alg.4, e.g. in [6] and [19] authors proposed a logarithmic time solution. However, in this paper we introduce a more efficient algorithm based on the concept of the hash tables and the associated hash functions in order to solve the step1 in the time of order $O(1)$.

A hash table is a data structure that can increase the search efficiency from $O(log(n))$ (binary search) to $O(1)$ (constant time) [6]. A hash table is made up of two parts: an array (the actual table where the data to be searched is stored) and a mapping function, known as a hash function. The hash function is a mapping from the input data space (in our case, state vector) to the integer space that defines the indices of the array (in our case, interval). Perfect hashing and construction of a suitable hash function is challenging and can not be achieved always (see [6] for details).

D. Application of hash table for point location

This section utilizes the concept of hash table to solve the point location problem efficiently. To this end, remember $PI_j(k_j)$ contains all regions intersecting with $I_j^{k_j}$. Associated with the j-th axis e_j, consider the following hash function which maps any query point $\mathrm{x} = [x_1, ..., x_{n_x}]^T$, to an integer value:

$$S_j(x) = floor\left(\frac{x_j - L_j^1}{U_j^{n_j} - L_j^1} n_j\right) + 1, \ \ \forall j = 1, ..., n_x \quad (5)$$

Where $floor(.)$ is a function which maps a real number to the largest integer not greater than the number. The following theorem summarizes the basic properties of combining the proposed algorithm with the hash function in (5) and the associated hash table.

Theorem 2: If the hash function (5) is used and the associated hash table is the set PI_j, then for a given query point $x = [x_1, ..., x_{n_x}]^T \in R^n$, the point location problem is reduced to a search through the following set of polyhedra:

$$PL = \cap_{j=1}^{n_x} PI_j\left(S_j(x)\right), \ \ \forall\ j = 1, ..., n_x \quad (6)$$

Proof: First we prove that $S_j(x) \in Dom(PI_j(.))$, i.e. the $S_j(x)$ is an integer-valued function for which $S_j(x) \in \{1, 2, ..., n_j\}$ where $Dom(PI_j(.))$ denotes the domain of the set-valued function PI_j. To this aim, from the definitions of $S_j(x)$ in (5) and floor function it is easy to investigate that $S_j(x)$ is an increasing piecewise constant integer-valued function. As a consequence $S_j(L_j^1 e_j) \leq S_j(x) \leq S_j(U_j^{n_j} e_j)$, where $e_j \in R^{n_x}$ is the j-th unit vector. This result suffices to prove that $S_j(L_j^1 e_j) \in Dom(PI_j)$ and $S_j(U_j^{n_j} e_j) \in Dom(PI_j)$. This can be simply verified by direct substitution of $L_j^1 e_j$ and $U_j^{n_j} e_j$ in (5).
Now it should be demonstrated that the function $S_j(x)$ returns the index of k_j-th sub-interval $I_j^{k_j}$ that contains the j-th element of vector x, i.e. $S_j(x) = k_j$. To this end, note that $x_j \in I_j^{k_j}$ leads to $L_j^{k_j} \leq x_j \leq U_j^{k_j}$, then from $(U_j^{n_j} - U_j^1) > 0$:

$$\frac{L_j^{k_j} - L_j^1}{U_j^{n_j} - U_j^1} n_j \leq \frac{x_j - L_j^1}{U_j^{n_j} - U_j^1} n_j \leq \frac{U_j^{k_j} - L_j^1}{U_j^{n_j} - U_j^1} n_j \quad (7)$$

By using the equations $L_j^{k_j} = L_j^1 + (k_j - 1)\delta_j$, $U_j^{k_j} = U_j^1 + k_j \delta_j$ and $\delta_j = (U_j^{n_j} - L_j^1)/n_j$, (7) can be rewritten as

$$k_j - 1 \leq \frac{x_j - L_j^1}{U_j^{n_j} - U_j^1} n_j \leq k_j \quad (8)$$

Then using (5) yields $S_j(x) = k_j$. ■

remark 4: Since the hash tables PI_j have been calculated offline, the access time to the index set of the polyhedral regions is of order $O(1)$. In other words, for each specific problem the maximum possible number of common intersecting polyhedral regions, i.e $|PL|$, is fixed and can be determined after offline calculation. Note that this constant number is strongly depended on the scaling factor ε_j and how the regions are scattered, also less depended on the dimension n_x but independent of N_r. By "independent" we mean that there are several cases for which $N_{r_1} > N_{r_2}$ but $|PL_1| < |PL_2|$ and $Time_1 < Time_2$ (see table 1).

IV. Complexity analysis

A. Offline processing complexity

The preprocessing procedure in algorithm 3 consists of four steps. Step 4 is the most dominant one and determines the complexity of the offline computation. In this step, regarding the selected scaling parameters ε_j, there are $n_j = 2^{\varepsilon_j}$ subintervals in each axis.
Since step 4 is performed for all subintervals, then considering remark 2 the computational complexity of Alg.3 can be measured as sum of the complexity of $inter(proj_{e_j}^{P_r}) \cap I_j^{k_j} \neq \emptyset$ for all axis, i.e. $\sum_{j=1}^{n_x} O\left\{inter(proj_{e_j}^{P_r}) \cap I_j^{k_j} \neq \emptyset\right\}$. For each subinterval $I_j^{k_j}$, one needs to examine two inequalities in remark 2 for all polyhedrons. Therefore the overall complexity will be of order $O\left(N_r \sum_{j=1}^{n_x} n_j\right)$. This is while the performance of method in remark 2 is much more better than intersection through solving the associated LPs.

B. Storage complexity

In addition to the polyhedral regions description (H_r, K_r), $r = 1, ..., N_r$, the only data which needs to be stored for online application is hash table (or the polyhedral index sets) PI_j, $j = 1, ..., n_x$. Suppose that the $|PI_j(k_j)|$ denotes the cardinality of the set $PI_j(k_j)$, i.e. the number of polyhedral regions index contained in the set $PI_j(k_j)$. Then corresponding to each axis, $M_j = \sum_{kj=1}^{n_j} |PI_j(k_j)|$ unsigned integer numbers need to be stored. Thus the total storage complexity will be $\sum_{j=1}^{n_x} M_j$. Note that the minimum and maximum cardinality of interval sets $PI_j(k_j)$ are mainly determined by the scaling factors ε_j, as tuning parameters.

C. Online processing complexity

The online computation in algorithm 4 is composed of two main parts. In step 1, for a given queries point $x = [x_1, ..., x_{n_x}]^T$ the associated subintervals $I_j^{\bar{k}_j}$ are determined in $O(1)$ using the proposed hash function. Steps 2 and 3 in algorithm 4 are critical and determine the online processing complexity. There are n_x index sets $PI_j(\bar{k}_j)$, $j = 1, ..., n_x$ and the objective is twofold:

1) ***Set intersection:*** find the common elements through n_x given index sets $PI_j(\bar{k}_j)$, i.e. $PL = \{r \in Z_{N_r} | r \in PI_j(\bar{k}_j), j = 1, ..., n_x\}$ where $Z_{N_r} = \{1, 2, ..., N_r\}$.
2) ***Direct search:*** for each element in the common set PL, check if it is the target region or not.

The computational complexity of steps (1) and (2) can be summarized as $O(\mathrm{N}_1+\mathrm{N}_2)$. The set intersection for two arbitrary index sets $PI_{j_1}(\bar{k}_{j_1})$ and $PI_{j_2}(\bar{k}_{j_2})$ can be done in $O(\mathrm{N}_1) = O(m_{j_1}\log m_{j_2})$ using Alg.5, where $m_{j_1} = |PI_{j_1}(\bar{k}_{j_1})|$ and $m_{j_2} = |PI_{j_2}(\bar{k}_{j_2})|$. The following theorem summarizes the online processing complexity.

Theorem 3: Consider the set intersection in Alg.5, then the worst case online complexity is of order $O(\mathrm{N}_1+\mathrm{N}_2)$, $O(\mathrm{N}_1) = O\left(\sum_{j=1}^{n_x-1}\min_{i\in\{0,\dots,j\}}(\mathrm{M}_i)\log(\mathrm{M}_{j+1})\right)$ and $O(\mathrm{N}_2) = O(\max|PL|)$, where $O(\mathrm{N}_2)$ refers to the maximum possible cardinality of common set PL, $\mathrm{M}_j = \max_{k_j\in\{1,\dots,n_j\}}|PI_j(k_j)|$ and $\mathrm{M}_0 = \max_{j\in\{1,\dots,n_x\}}\mathrm{M}_j$.

Proof: Since the cardinality of all index sets are known from the preprocessing, the $O(\mathrm{N}_2)$ is determined by the maximum possible cardinality of common set PL, i.e. $O(\max|PL|)$. Note that applying the algorithm 5 to the first two index sets $PI_1(\bar{k}_1)$ and $PI_2(\bar{k}_2)$ impose a computational complexity of order $O(m_1\log m_2)$, then for the resulting common set (PL) we have $|PL|\leq\min(m_1,m_2)$. Therefore calling the algorithm 5 for PL and next index set $PI_3(\bar{k}_3)$ will similarly impose the complexity of order $O(|PL|\log m_3) \leq O(\min(m_1,m_2)\log m_3)$. Iterating this procedure shows that for i-th index set the complexity is $O\left(\min_{i\in\{0,\dots,j\}}(m_i)\log(m_{j+1})\right)$. The worst case (upper bound) can be determined by taking maximum over subintervals for each axis, i.e. $\mathrm{M}_j = \max_{k_j\in\{1,\dots,n_j\}}(m_j = |PI_j(k_j)|)$. Finally the total online processing complexity is the sum of all individual time complexities. $\mathrm{M}_0 = \max_{j\in\{1,\dots,n_x\}}\mathrm{M}_j$ is a constant number that is used to simplify the notation. ■

remark 5: Further simplification in the online computation can be done using a secondary hash table in the set intersection algorithm. This is while only a bit array H with length of N_r is needed to be added to the storage complexity. Using this bit array as a hash table it is possible to solve the set intersection problem in time of order $O(m_1)$ instead of $O(m_1\log m_2)$ where $m_1\leq m_2$. Then it is sufficient to hash the second index set to the bit array H, i.e. $H(i)=1,\ \forall\, i\in S_2$ and $H(i)=0,\ \forall\, i\notin S_2$. Then checking if each element of first set S_1 is contained in S_2 or not can be done in $O(1)$. Sorting sets in a way that $m_1\leq m_2\leq\dots\leq m_{n_x}$ takes time of order $O(n_x\log(n_x))$ while $m_j s$ are computed offline.

remark 6: considering remark 5 and the proposed algorithm, an approximation to the number of arithmetic operations (arith. ops.) including summation/subtraction, multiplication/division and comparison can be found as $\#(arith.ops.)\leq 7n_x+n_x\log n_x+N_{\mathrm{intersect}}+2n_xN_{com}N_H$ where $7n_x$ denotes the number of arithmetic operations of hash function evaluation and $n_x\log(n_x)+N_{\mathrm{intersect}}$ refers to the arithmetic operations of array sorting and set intersection using hash table introduced in remark 5. Accordingly $N_{\mathrm{intersect}}$ can be approximated by $\min_j(\max_{k_j}|PI_j^{k_j}|)$. Finally $2n_xN_{com}N_H$ denotes the arithmetic operations of direct search through the set PL where N_H is the maximum number of hyper planes describing any polyhedral region and N_{com} is the maximum possible cardinality of the set PL which can be pre-computed when the polyhedral index sets PI_j are computed offline. A conservative upper bound for N_{com} is $\max_j(\max_{k_j}|PI_j^{k_j}|)$.

According to the remark 6, the number of arithmetic operations in the online processing is mainly depended on the problem dimension n_x and the scaling parameters ε_j instead of number of polyhedral regions N_r. Because according to the proposed algorithm, adjusting ε_j changes the cardinality of $PI_j^{k_j}$ and thus changes $N_{intersect}$ and N_{com}.

remark 7: In the online computation $2n_xN_{com}N_H$ is usually the dominant part. This is while the offline computation time in the proposed method is too small that can be used to reduce N_{com} with some extra preprocessing and storage. To this aim in Alg.3, for some predefined number of axes $e_j, (j=1,...,N_e\leq n_x-1)$ and for all sub-intervals solve:

$$LR_j^{k_j} = \left\{\min_{i=1,..,n_x}\ x_i\,|\,x_j\in I_j^{k_j},\, x\in P_r,\ r\in PI_j^{k_j}\right\}$$
$$UR_j^{k_j} = \left\{\max_{i=1,..,n_x}\ x_i\,|\,x_j\in I_j^{k_j},\, x\in P_r,\ r\in PI_j^{k_j}\right\} \quad (9)$$

Where, N_e is chosen arbitrarily with considering the offline complexity. $LR_j^{k_j}$ and $UR_j^{k_j}$ indicate the minimum and maximum values of polyhedral regions P_r contained in $PI_j^{k_j}$ and restricted to the interval $I_j^{k_j} = [L_j^{k_j}\ U_j^{k_j}]$ along all axes. Using these values in the online procedure one can exclude several regions from the set PL by checking whether $(LR_j^{\bar{k}_j}\leq U_j^{\bar{k}_j})$, $(UR_j^{\bar{k}_j}\leq L_j^{\bar{k}_j})$ or not. The storage complexity imposed by $L_j^{k_j}$ and $U_j^{k_j},\ k_j=1,...,n_j$ is $2n_xM_j$. Thus the total storage complexity will be $(\sum_{j=1}^{n_x}M_j + 2n_x\sum_{j=1}^{N_e}M_j)$.

V. Numerical examples

We have applied the proposed method to several polyhedral regions to evaluate the computational saving the proposed method achieves. Polyhedral regions were generated using the Multi Parametric Toolbox for MATLAB® (MPT2.6) [20]. Several LTI systems with $n_x=2-5$ were generated and solved using $mpt\_randLTISys$ function. The algorithms 3 and 5 together with the direct search and the algorithm in [7] were applied to the test cases and the results are compared in table 1. Also time-storage tradeoff when the scaling parameters vary from 1 to 15 is shown in Fig.1. It can be seen that the optimal scaling factor for this test case (in sense of Pareto) is around 6-8 which makes an appropriate agreement between online processing and storage demands. The results in table 1 verify the effectiveness of the proposed method in online processing while the preprocessing time show the amenability of present method for applications with large number of polyhedral regions for which the optimal algorithm [7] is not applicable. Note that the simulations with big N_r in all test cases refer to the orthogonal partitions arising in the nonlinear explicit MPC [21]-[23] and the results from these experiments verify even

though the partitions contain more polyhedral regions, the present method still works very well. Thus this method can be used as an alternative to overcome the complexity of explicit MPC. Finally it can be estimated that the storage of hash indices contributes to the overall storage requirements as much as the storage of regions P_r only. To illustrate, consider the last and the worst test case. If one assumes that each critical region, in average, is described by $2n$ hyperplanes then $2n(n+1)N_r$ floating point numbers are needed to be stored. Then about 4847 Kb (single precision) and 9694 Kb (double precision) memory is required to store only critical regions.

VI. Conclusion

A new efficient and flexible method for point location problem was introduced. Some of existing methods attempt to solve a reduced point location problem instead of the original problem (e.g. [11],[14]). The proposed method in this paper has the same goal in its interior. The main distinguishing property in the present method refers to existence of some tuning parameters that enable the designer to tradeoff between time and storage complexity in a simple way. It was also shown that combining this algorithm with hashing theory improves extensively the online processing complexity with some tractable off-line and storage complexities. It is also illustrated that the online processing complexity is mainly depended on the scaling factors ε_j, the dimension n_x and also how the regions are scattered instead of the number of polyhedral region (N_r). This fact can be helpful when the number of regions is high. In terms of the storage complexity one just needs to store the index sets PI_j as the hash tables in the proposed method in addition to the polyhedral regions and sometimes it is augmented with data structures $LR_j^{k_j}$ and $UR_j^{k_j}$ discussed in remark 7.

TABLE I

Comparing Algs. 3 & 4, Direct Search (DS) and Alg. in [7]. ⋆ denotes the associated algorithm is not tractable in this test case.

Problem			Offline Time		Arith.ops.(worstcase)			Mem.
n_x	ε_j	N_r	Alg.3 (sec)	BST (sec)	DS	Alg.4	Alg. [7]	Alg.4 (Kb)
2	7	139	2.9	57	2240	173	54	60
	8	187	14.3	108	3000	92	72	296
	8	853	46.5	2515	13656	280	82	76
	8	1720	6.9	2419	27528	220	82	76
	8	2258	8.67	5300	35984	240	72	345
	8	3125	11.66	2891	50000	245	118	101
	8	15625	58.59	⋆	250000	607	⋆	280
3	8	674	4.7	141	23724	716	78	93
	8	1565	9.9	62982	57312	938	166	279
	6	2998	478	24568	105876	1962	176	1105
	8	16807	102	⋆	605052	2903	⋆	1525
4	8	596	240	359	40544	1820	156	654
	8	834	171	981	57312	1737	226	444
	8	6561	33	⋆	419904	3480	⋆	905
	8	13716	122	⋆	877824	5272	⋆	2623
5	7	242	0.67	3998	24200	606	210	26
	7	1331	279	⋆	133100	1799	⋆	469
	8	14641	60	⋆	1464100	11687	⋆	1508
	8	20680	237	⋆	2068000	14463	⋆	7294

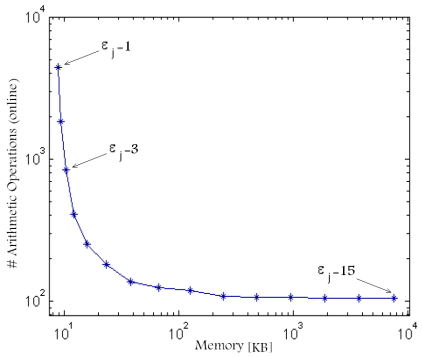

Fig. 2. Complexity tradeoff for ε_j= 1 to 15, ($n_x = 2, N_r = 2258$).

References

[1] Borrelli,F., Baotic,M., Bemporad,A. and Morari,M., "Efficient on-line computation of constrained optimal control", *40th IEEE Conf. on Decision and Cont.*, vol. 2, 2001, pp 1187-1192.

[2] Bemporad,A., Morari,M., Dua,V. and Pistikopoulos,E. N., The explicit linear quadratic regulator for constrained systems, *Automatica*, vol. 38, 2002, pp 3-20.

[3] Bemporad,A., Borrelli,F. and Morari,M., Model predictive control based on linear programming - The explicit solution, *IEEE Tran. on Auto. Cont.*, vol. 47, 2002, pp 1974-1985.

[4] Tøndel,P., Johansen,T. A. and Bemporad,A., An algorithm for multi-parametric quadratic programming and explicit MPC solutions, *Automatica*, vol. 39, 2004, pp 489-497.

[5] Snoeyink, J., *Point Location*, Handbook of Disc. and Comp. Geometry, CRC Press, J. E. Goodman and J. Órouke,559-574, 1997.

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., *Introduction to algorithms*, MIT Press, Cambridge, MA, USA, 2001.

[7] Tøndel,P., Johansen,T. A. and Bemporad,A., Evaluation of piecewise affine control via binary search tree, *Automatica*, vol. 39, 2003.

[8] Baotić,M., Borrelli,F., Bemporad,A. and Morari,M., Efficient on-line computation of constrained optimal control, *SIAM J. on Cont. and Opt.*, vol. 47, 2008, pp 2470-2489.

[9] Jones,C. N., Grieder,P. and Raković,S. V., A logarithmic-time solution to the point location problem for parametric linear programming, *Automatica*, vol. 42, 2006, pp 2215-2218.

[10] Arya,S., Mount,D. M., Netanyahu,N. S., Silverman,R. and Wu,A. Y., An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *J. of the ACM*, vol. 45, 1998, pp 891-923.

[11] Spjøtvold,J., Raković,S. V., Tøndel,P. and Johansen,T. A., "Utilizing reachability analysis in point location problems", *45th IEEE Conf. on Decision and Cont.*, 2006, pp 4568-4569.

[12] Christophersen, F.J., Kvasnica, M., Jones, C.N. and Morari, M., "Efficient evaluation of piecewise control laws defined over a large number of polyhedra", *Eur. Cont. Conf.*, 2007, pp 2360-2367.

[13] Wang, Y., Jones and C.N. and Maciejowski, J., "Efficient point location via subdivision walking with application to explicit MPC", *Eur. Cont. Conf.*, 2007, pp 447-453.

[14] Sui,D., Feng,L. and Hovd,M., "Algorithms for online implementations of explicit MPC solutions authors", *in Third SIAM Conf. on App. Lin. Alg.*, vol. 17, 2008.

[15] Wen, C. and Ma, X. and Erik Y.B., Analytical expression of explicit MPC solution via lattice piecewise-affine function, *Automatica*, vol. 45, 2009, pp 910-917.

[16] Zeilinger, M.N., Jones, C.N. and Morari, M., "Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization", *47th IEEE Conf. on Decision and Cont.*, 2008.

[17] Czech,Z. J., Havas,G. and Majewski,B. S., An optimal algorithm for generating minimal perfect hash functions, *Inf. Proc. Lett.*, vol. 43, 1992, pp 257-264.

[18] M. Canale, L. Fagiano and M. Milanese, Set Membership approximation theory for fast implementation of Model Predictive Control laws, *Automatica*, vol. 45, 2009, pp 45-54.

[19] Berg, M., Schwarzkopf, O., Kreveld, M. and Overmars, M., *Computational Geometry Algorithms and Applications*, Springer, Verlag, 2000.

[20] Kvasnica,M., Grieder,P. and Baotić,M. and Morari,M., *Multi-parametric toolbox (MPT)*, vol. 2993, pp 448-462, 2004.

[21] Johansen,T. A., "On multi-parametric nonlinear programming and explicit nonlinear model predictive control", *IEEE Conf. on Decision and Cont.*, vol. 3, 2002, pp 4718 - 4723.

[22] Johansen,T. A., Approximate explicit receding horizon control of constrained nonlinear systems, *Automatica*, vol. 40, 2004, pp 293-300.

[23] Grancharova, A. and Johansen, T. A., "Approaches to Explicit Nonlinear Model Predictive Control with Reduced Partition Complexity", *Proceedings of the Eur. Cont. Conf.*, 2009, pp 101–107.