Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 982041, 10 pages
http://dx.doi.org/10.1155/2015/982041

Research Article

Fast Model Predictive Control Combining Offline Method and Online Optimization with K-D Tree

Yi Ding, Zuhua Xu, Jun Zhao, and Zhijiang Shao

National Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China

Correspondence should be addressed to Zuhua Xu; xuzh@iipc.zju.edu.cn

Received 26 April 2015; Revised 5 July 2015; Accepted 22 July 2015

Academic Editor: Jean J. Loiseau

Computation time is the main factor that limits the application of model predictive control (MPC). This paper presents a fast model predictive control algorithm that combines offline method and online optimization to solve the MPC problem. The offline method uses a k-d tree instead of a table to implement partial enumeration, which accelerates online searching operation. Only a part of the explicit solution is stored in the k-d tree for online searching, and the k-d tree is updated in runtime to accommodate the change in the operating point. Online optimization is invoked when searching on the k-d tree fails. Numerical experiments show that the proposed algorithm is efficient on both small-scale and large-scale processes. The average speedup factor in the large-scale process is at least 6, the worst-case speedup factor is at least 2, and the performance is less than 0.05% suboptimal.

1. Introduction

Model predictive control (MPC) is widely applied in the process control area because of its ability to handle constraints and MIMO systems. However, the computation burden in every control interval [i.e., solving a quadratic programming (QP) problem] limits the application of MPC within slow processes. Thus, accelerating the computation speed of the MPC is a popular theme in the control community.

Dealing with the QP problem in the MPC mainly has two approaches. The first approach is online optimization, where two algorithms are widely used (i.e., the interior-point method and the active-set method). A simple introduction of these methods is presented in [1]. Many researchers attempted to exploit the special structure of MPC and tailor the two algorithms to speed up the solving process. Previous authors exploited the sparsity and other structure features of MPC in [2, 3] to customize the interior-point method. QPSchur, which is a QP algorithm based on the active-set method, is proposed in [4] and focuses on large-scale MPC. Another online active-set strategy is described in [5] for the fast solution of parametric QPs in MPC; this strategy uses the information of the previous QP under the assumption that the QP problem only slightly changes. Besides the interior-point method and active-set method, other online methods include the Newton method [6] and dual gradient-projection method [7].

The second method is the explicit approach proposed by Bemporad et al. [8]. They showed that the solution to the QP problem in linear MPC is a piecewise affine function of the state variables. Thus, a look-up table can be prepared offline by partitioning the feasible region of the state variables. The online work only has two steps, namely, find the critical region (CR) of a certain point and perform simple calculations with corresponding parameters. The traditional optimization computation is avoided and computation time is decreased in this manner. However, the number of candidate CRs that should be checked for optimality increases exponentially in the number of inequality constraints, which is the major drawback of this method. Thus, several researchers propounded some improvements on explicit MPC. A suboptimal solution is computed using approximation in [9]. An algorithm extended to any convex parametric optimization problem is presented in [10]. The CR number is decreased by eliminating regions in [11] if the control law from a neighbor

region can guarantee stability, which can also speed up the online searching.

Despite the developments in both online and offline methods, several problems with certain sizes and types still cannot be efficiently solved by the online method or the offline method alone. Several researchers have combined the explicit method and online optimization in recent years to implement fast MPC. The partial enumeration (PE) method is proposed in [12]. The PE method uses a table to store the recently searched CRs and combines it with the online method to achieve fast MPC for large-scale problems. However, the PE method is effective only when the disturbances change slowly and have small magnitude. This application limitation can be attributed to the table search being time consuming and the update method in PE tending to make the information in the table converge to several local areas. Thus, we propose a method that combines the explicit MPC and online optimization by using the k-d tree structure to obtain fast MPC for both small-scale and large-scale problems. The properties of our method guarantee that the proposed method is effective regardless of disturbance magnitude or setpoint changes.

The rest of this paper is organized as follows. Section 2 discusses several background theories used in our method. Section 3 presents the fast MPC algorithm based on k-d tree and online optimization. Section 4 applies the algorithm to two processes to verify the efficiency of our design. Finally, Section 5 concludes this paper.

2. Background

2.1. MPC Formulation. We consider a linear time-invariant system model in the state-space form as follows:

$$x(t+1) = Ax(t) + Bu(t), \tag{1}$$

where $x(k) \in \mathbb{R}^{n_x}$ is the state; $u(k) \in \mathbb{R}^{n_u}$ is the input; and matrices $A \in \mathbb{R}^{n_x \times n_x}$ and $B \in \mathbb{R}^{n_x \times n_u}$ are fixed matrices with appropriate dimensions. We consider the problem to regulate system (1) from the current state $x(0)$ to the origin for simplicity. Reference [8] shows that the controller for a regulation problem can be extended for a trajectory tracking problem with some linear transformation. The following input and state constraints must be fulfilled at each control interval:

$$\underline{c} \le Cx(t) \le \overline{c}, \tag{2a}$$

$$\underline{d} \le Du(t) \le \overline{d}, \tag{2b}$$

where constant matrices and vectors $C \in \mathbb{R}^{n_{oc} \times n_x}$, $D \in \mathbb{R}^{n_{ic} \times n_u}$, $\underline{c}, \overline{c} \in \mathbb{R}^{n_{oc}}$, and $\underline{d}, \overline{d} \in \mathbb{R}^{n_{ic}}$ are specified by the users. Here n_{ic} is the number constraints on inputs and n_{oc} is the number of constraints on states.

MPC computes the input $u(t)$ at each control interval by solving the following constrained QP problem:

$$\min_{\substack{u(0),\ldots,u(N-1)\\ x(0),\ldots,x(N)}} \quad x(N)^T Px(N) + \sum_{i=0}^{N-1} \left(x(i)^T Qx(i) + u(i)^T Ru(i) \right) \tag{3a}$$

$$\text{s.t.} \quad x(t+1) = Ax(t) + Bu(t), \tag{3b}$$

$$\left(x(0) \ \text{ given}\right), \tag{3c}$$

$$\underline{c} \le Cx(t) \le \overline{c}, \tag{3d}$$

$$\underline{d} \le Du(t) \le \overline{d}, \tag{3e}$$

where $P \in \mathbb{R}^{n_x \times n_x}$ and $Q \in \mathbb{R}^{n_x \times n_x}$ are positive semidefinite matrices and $R \in \mathbb{R}^{n_u \times n_u}$ is a positive-definite matrix. The prediction horizon and the control horizon is set as being equal to N for simplicity. The terminal penalty weight matrix P can be calculated by solving a Riccati equation for the unconstrained system.

Only the first control move $u^*(0)$ is applied as input to system (1) after the solution of $U^*(0) = \{u^*(0), \ldots, u^*(N-1)\}$. The optimization problem ((3a), (3b), (3c), (3d), and (3e)) is solved again in the next control interval based on the assumption that the state is obtained. Thus, the *receding horizon control* is implemented.

2.2. MPC Computation. By substituting $x(t) = A^t x(0) + \sum_{j=0}^{t-1} A^j Bu(t-1-j)$, the optimization problem ((3a), (3b), (3c), (3d), and (3e)) can be rewritten as

$$J(x(0)) = \frac{1}{2} x(0)^T Yx(0) + \min_U \frac{1}{2} U^T HU + x^T(0) FU \tag{4a}$$

$$\text{s.t.} \ GU \ge W + Ex(0), \tag{4b}$$

where $x(0)$ is the current state and Y, H, F, G, W, and E can be easily obtained [8]. The optimality (KKT) condition of the optimization problem in system ((4a), (4b)) is as follows:

$$Hu^* + F^T x(0) - G_a^T \lambda_a^* = 0, \tag{5a}$$

$$G_a u^* = W_a + E_a x(0), \tag{5b}$$

$$\lambda_i^* = 0, \tag{5c}$$

$$G_i u^* \ge W_i + E_i x(0), \tag{5d}$$

$$\lambda_a^* \ge 0, \tag{5e}$$

where a denotes the active constraint indices and i denotes the inactive constraint indices. Here G_a, W_a, and E_a represent the row subvector/submatrix that corresponds to the active set a.

The linear relation between the parameter $x(0)$ and optimal solution in the explicit MPC strategy [8] is explicitly expressed based on the equations in system ((4a), (4b)). Equations (5a) and (5b) derive the following:

$$\begin{bmatrix} H & -G_a^T \\ G_a & 0 \end{bmatrix} \begin{bmatrix} u^* \\ \lambda_a^* \end{bmatrix} = \begin{bmatrix} 0 \\ W_a \end{bmatrix} + \begin{bmatrix} -F^T \\ E_a \end{bmatrix} x(0). \tag{6}$$

At this point, we assume the linear independence of the rows in G_a. We can then derive the affine function between the parameter $x(0)$ and optimal values u^* and λ_a^* as follows:

$$u^* = K_{ax}x(0) + c_u, \tag{7a}$$

$$\lambda_a^* = L_{ax}x(0) + c_\lambda, \tag{7b}$$

where the coefficients K_{ax}, L_{ax}, c_u, and c_λ can be computed as follows:

$$c_\lambda = \left(G_a H^{-1} G_a^T\right)^{-1} W_a, \tag{8a}$$

$$L_{ax} = \left(G_a H^{-1} G_a^T\right)^{-1} \left(G_a H^{-1} F^T + E_a\right), \tag{8b}$$

$$c_u = H^{-1} G_a^T \left(G_a H^{-1} G_a^T\right)^{-1} W_a, \tag{8c}$$

$$K_{ax} = \left[H^{-1} G_a^T \left(G_a H^{-1} G_a^T\right)^{-1} \left(G_a H^{-1} F^T + E_a\right) - H^{-1} F^T\right]. \tag{8d}$$

The valid region of the formula in systems ((7a), (7b)) and ((8a), (8b), (8c), (8d)), namely, the "CR" in [8], is based on a certain active set defined by the inequality constraints in (5d) and (5e). We can derive the following inequalities by substituting from system ((7a), (7b)) to check whether a parameter instance is in the certain CR:

$$L_{ax}x(0) + c_\lambda \geq 0, \tag{9a}$$

$$G_i K_{ax} x(0) + \left(G_i c_u - W_i - E_i x(0)\right) \geq 0. \tag{9b}$$

We then define the feasible region of a specified QP problem as follows:

$$\mathbb{X} \stackrel{\text{def}}{=} \left\{x \in \mathbb{R}^{n_x} \mid GU \geq W + Ex,\ U \neq \emptyset\right\}. \tag{10}$$

Thus, $\mathbb{X}$ can be partitioned into several CRs. Figure 1 is an illustration of the partition for a QP problem, where $x(k) \in \mathbb{R}^2$.

The primary idea of the explicit MPC is to calculate and store the information with each valid CR offline. The online work is only to determine the exact CR for the current parameter and compute the optimal value using system ((7a), (7b)). References [8, 13] both proposed efficient algorithms for the offline calculation of CRs. The explicit MPC is only suitable for small-scale problems because the CR number and the time required to locate the CR increases exponentially as the problem's scale increases. Many works have been published to improve the performance of explicit MPC, which can be roughly categorized into date structure research [14, 15], approximation methods [9, 16], and combinations with online methods [12, 17]. The derivation above is based on a regulation problem and the extension to reference tracking problem can be found in [8].

3. Combined Method with K-D Tree

The PE method introduced in [12] combines explicit MPC and online optimization. This method uses a table to store the recently searched CRs and the table is sorted according to the invoke frequency of the each CR. The CR that corresponds to the state is searched in the table to solve the optimization problem for a specific state. If the appropriate CR is found, the control law can be calculated immediately; otherwise, the online optimization method is invoked and the table is updated. This approach is appealing for large-scale applications with small disturbances. However, the PE method has two limitations that cannot be ignored. First, searching an item in the table can be time consuming; the linear complexity of the searching queries on the table structure definitely limits the table length. Second, the updating strategy used in the PE method tends to make the CRs in the table converge to several local areas, which causes the PE method to be invalid when large amplitude noise or setpoint change occurs.

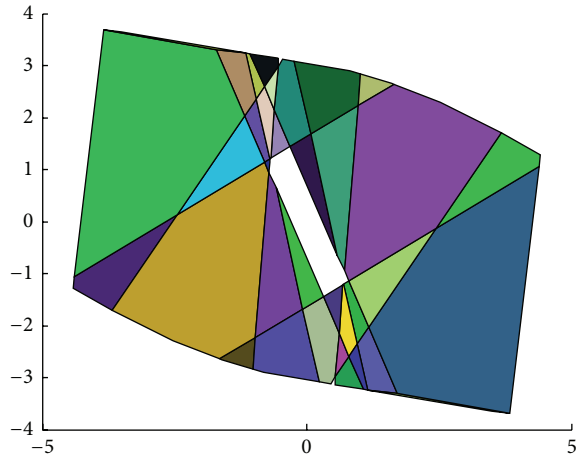

Figure 1: Partition of $\mathbb{X}$ into CRs.

The limitation talked above can be illustrated in Figures 2 and 3. Figure 2 shows a 2-D case. First, for example, the system runs in Region 1 and the information of CRs around are stored into the table. Then the setpoint changes and forces the state runs in Region 2. At this moment, the information of CRs around Region 2 is added into the table, which can be called a "learning period," and some CRs in the table will be deleted if the table is over length. In the worst case, if the system runs in Region 2 for a long time and the invoke frequency of CRs around Region 2 accumulates to a high level, thus the information of CRs around Region 1 will all be deleted and only those around Region 2 remain in the table, which tends to be a kind of "convergence." Then, if the setpoint changes again and drives the states back to Region 1, the PE method has to go through another "learning period" to add the information of CRs around Region 1. This limitation has also been referred to in [12].

We can observe the phenomenon mentioned above in Figure 3. In Figure 3 there are three subplots: the first plot is the time used for online calculation of MPC problem, the second plot is the trajectory of the output, and the third plot describes the result of PE method, in which 1 means that the corresponding CR is found in table and −1 means not found. The model comes from Example 1 in this paper and the length

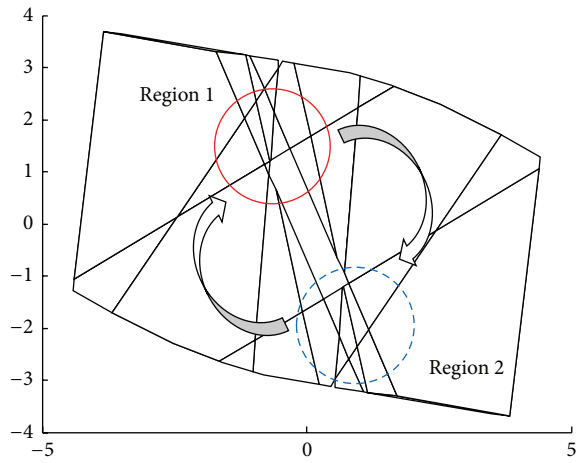

Figure 2: Limitation of PE method.

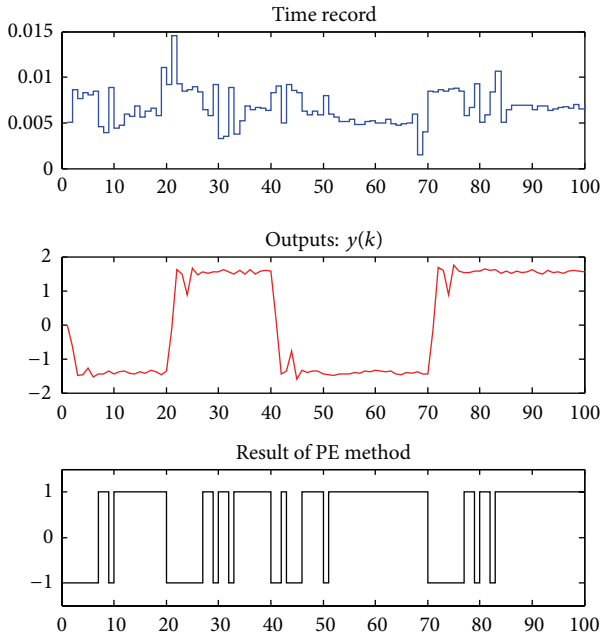

Figure 3: The simulation result of PE method with setpoint changes.

of the table is 15. We first change the setpoint from −1.5 to 1.5 at time $k = 20$; then we change the setpoint from 1.5 to −1.5 at time $k = 40$, and finally change it back to 1.5 at time $k = 70$. The "learning period" of PE method after $k = 70$ is almost as long as its first "learning period" after $k = 20$ which means that the information of CRs stored in the table in the first "learning period" has been squeezed out by information stored between $k = 40$ and $k = 70$. In fact, increasing the table length in PE method can solve this problem. However, a longer table means more time needed in searching on average and under the worst case. More detailed comparison on numerical experiments can be found in Section 4.

In order to overcome the two weaknesses of the PE method while maintaining its advantages, we propose a new method to organize the CR information in a limited k-d tree.

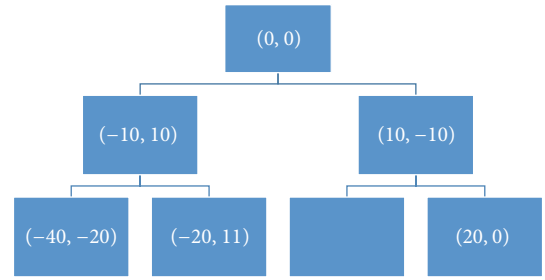

Figure 4: K-d tree with five 2-dimensional nodes.

That is, a k-d tree structure is used to implement the partial enumeration in our method. The "limited" here means that the height of the k-d tree is restricted, and the k-d tree is updated online to maintain its size.

3.1. K-D Tree. The k-d tree is a type of binary search tree that deals with multidimensional data [18, 19]. Each node in the k-d tree represents a record, which is comprised of k keys. Thus, the node can be viewed as a point in the k-dimensional space. Additionally, the node in the k-d tree can also represent a subregion of the entire space. Each node can have at most two sons or successor nodes. The region represented by each successor node is a partition of the region represented by the father node.

In regard to one-dimensional searching, each record is described by a single key. The node with a key value less than or equal to this key value lies in the left subtree, whereas the node with a larger key value lies in the right subtree. Thus, the key variable is a discriminator for assigning nodes to the two subtrees. A record is comprised by k keys in k-dimensional searching. The k-d tree uses the following strategy to determine the discriminators for the nodes based on its level in the k-d tree. The discriminator is specified for each level by cycling through the keys in order. The formula, $D = L \bmod k + 1$, is proposed in [18] to describe this relation, where D is the index of the discriminator key for level L, and the level of the root node is zeros. The complexity of inserting a node to a randomly built tree is $O(\log M)$, in which M is the total number of the records in the k-d tree.

At this point, we illustrate the above principles with a 2-dimensional example. The k-d tree in Figure 4 is constructed by inserting the following points in sequence: (0, 0), (−10, 10), (10, −10), (−40, −20), (−20, 11), and (20, 0). Figure 5 shows the plot in 2-dimensional plane that corresponds to this k-d tree.

Figure 5 shows that each node in the k-d tree represents a region in the k-dimensional space. The root node represents the entire feasible region, specifically (−50, −50, 50, 50) in the above case. At this point, we adopt $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ to describe a 2-dimensional region. The region represented by the node (−10, 10) is (−50, −50, 0, 50), whereas (−20, 11) represents the region (−50, 10, 0, 50).

A brief introduction of k-d tree and its insertion operations is discussed above. The next important operation on k-d tree is searching. Searching can be classified into four main types according to their query types [18, 19]. We used the so-called "*Region Query*" in our algorithm, particularly to

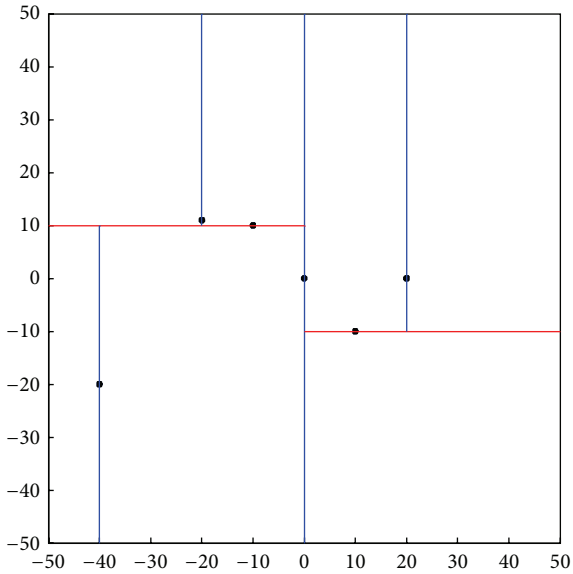

Figure 5: The 2-dimensional plot of the k-d tree.

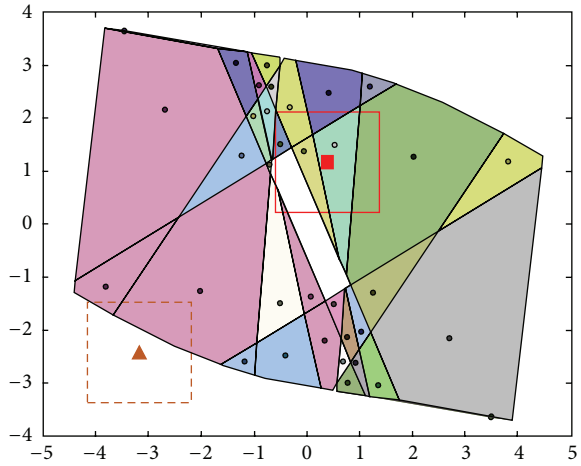

Figure 6: Searching a CR in the k-d tree.

find the records whose key vectors are located in the region specified by the query. For instance, given the query region (−45, −30, −30, −10) in the above case, the region query returns all the record(s) in it, which is the node (−40, −20). The algorithm for this query type is discussed in detail in [18]; a recursive method is used to find all the nodes whose representative regions are intersected with the query region and decide whether the node is in the region. Reference [20] proves that the complexity of the algorithm is $O(k \bullet M^{1-1/k})$ under the worst case. References [18, 19] also illustrate using many simulations in which the algorithm performance is reasonable. Additionally, the size of the query region has an effect on the performance of the searching operation.

Another important operation is deletion, which is also proposed in [18]. Notably, the deletion does not change the k-d tree property, and the complexity of deleting a node from a randomly built tree is $O(\log M)$. The last operation we used in our algorithm is to optimize an unbalanced tree, which is used in the offline preparation.

3.2. Implementation. Section 3.1 states that the key of the node in the k-d tree is a vector. Each node in our method stores the CR information, and the chebyshev center of the CR is set as the key vector of the corresponding node. At this point, the chebyshev center of the CR is a point inside the CR that is farthest from the CR exterior [21]. We selected this point to make the key vector representative of the corresponding CR.

We now assume the current stats $x(\tau)$ is obtained when $t = \tau$. We first construct a hyperrectangle $[x^1_{\min}, x^2_{\min}, \ldots, x^k_{\min}, x^1_{\max}, x^2_{\max}, \ldots, x^k_{\max}]$ with the $x(\tau)$ being its center to compute the optimal input u^*. Thus,

$$x^i_{\min} = x^i(\tau) - \varepsilon, \tag{11a}$$

$$x^i_{\max} = x^i(\tau) + \varepsilon, \tag{11b}$$

where $x^i(\tau)$ is the ith element of $x(\tau)$ and $\varepsilon > 0$ is a user-defined parameter to set the size of the hyperrectangle. The "*Region Query*" on the k-d tree is then performed with the hyperrectangle as its argument. We obtain a list of the nodes inside the region after the query. We then check these nodes to find the CR that $x(\tau)$ belongs to. The nodes are checked sequentially according to its distance from $x(\tau)$. If the exact CR is found, u^* can be calculated easily with (7a), and the invoke frequency is increased by one of this CR. The invoke frequency of each CR is checked when deleting a node form the k-d tree. If the corresponding CR is not found, the optimization problem in system ((3a), (3b), (3c), (3d), and (3e)) with a small horizon value is solved. A suboptimal solution is thus obtained, and the computation time is less than that of the origin problem. The above procedures are shown in Figure 6. The feasible region of the parameter $x \in \mathbb{R}^2$ is partitioned into CRs, and the dot in each CR is its chebyshev center.

Figure 6 shows that the square point is the current parameter $x(\tau)$ and the CR that it belongs to is a narrow quadrangle. The hyperrectangle with $x(\tau)$ being its center is plotted as a rectangle with the solid line. The three points are found in the region. The three nodes in the k-d tree are then checked; the exact CR is discovered; and u^* is computed with the corresponding information. If the current state $x(\tau)$ does not belong to any CR that is found in the region query or any of CR found, as the triangle point and its region in the dashed line showed in Figure 6, u^* it is computed using system ((3a), (3b), (3c), (3d), and (3e)) with a short horizon, and the k-d tree is updated. The update includes the following operations: the original optimization problem in system ((4a), (4b)) is solved; the related information of this newly found CR is calculated; and a node that represents this CR is inserted into the k-d tree.

If the tree height is over the threshold, the deletion operation is invoked. The deletion operation is manipulated as follows: a $\mathbb{D}$ node set is built, including a node chain from the new-insert node's father all the way up to the root node, and the chain terminates at the nodes with two child nodes.

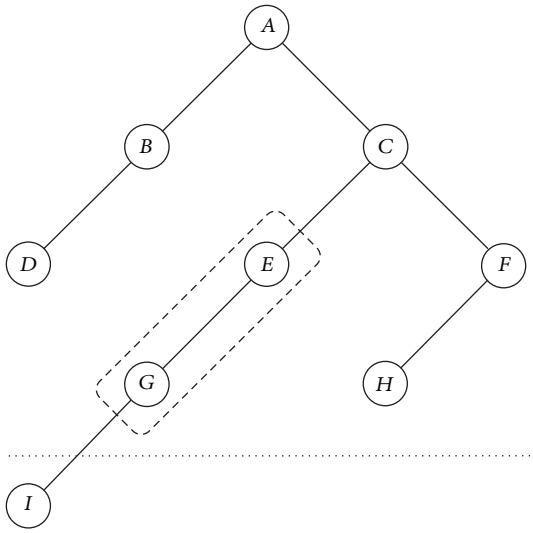

Figure 7: Set $\mathbb{D}$ in an overheight k-d tree.

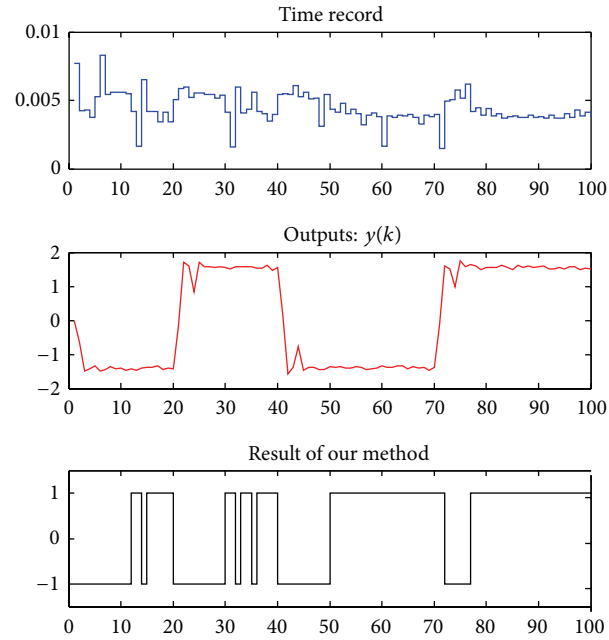

Figure 8: The simulation result of our method with setpoint changes.

However, this node with two child nodes is excluded from the set. The node chain is illustrated in Figure 7. In a k-d tree with the height threshold $h_{\text{Th}} = 4$, the new-insert node I makes the tree over height. Thus, the set $\mathbb{D}$ defined above includes nodes G and E (i.e., $\mathbb{D} = \{G, E\}$).

The node in $\mathbb{D}$ with the minimal invoke frequency is then deleted. The following theorem on the concerned deletion operation can be derived.

Theorem 1. *The node set $\mathbb{D}$ built following the above principles is nonempty; the deletion of any node in $\mathbb{D}$ lowers the height of the k-d tree, which limits the height within the threshold.*

Proof. If the new-insert node makes the tree over height, its father node must be an external node (leaf node) before its insertion. Thus, the father node of the new-insert node is exactly in $\mathbb{D}$, which guarantees that $\mathbb{D}$ has at least one element and $\mathbb{D} \neq \emptyset$. The deletion operation affects only the subtree with the node to be deleted being its root according to the properties of the deletion operation in a k-d tree [18]. The nodes in $\mathbb{D}$ together with the new-insert node are exactly the subtree with only one node in each level, like E-G-I in Figure 7. Deleting an internal node, which is a certain node in $\mathbb{D}$, in this subtree certainly lowers the subtree height. Thus, the deletion operation lowers the height of the former k-d tree and limits the height within the threshold. □

Remark 2. The updating method discussed above has the following advantages. (1) The update operation is efficient. An average logarithmic complexity can be maintained for the insertion and deletion operation. The reduction in the tree height can also be guaranteed by only one deletion operation. (2) The deletion of the internal node with only one child results in a complete binary tree, which can then benefit the searching operations. (3) The deletion operation affects only several local nodes and does not destroy the balance of the tree over the entire space. Thus some old information will be protected from being deleted and the information stored in K-D tree will not converge to some local areas as we illustrated in Figure 2. In addition, the deletion operation can be further customized to tradeoff between the local performance and overall performance. That is, if we only consider the local performance, like PE method, we can change the deletion operation and pick the node to be deleted from the whole tree but not the local areas. The structure of K-D tree makes it easier to map the real geometrical relationship into the abstract data structure. The advantages can be observed by a similar simulation compared with Figure 3. In Figure 8, we solve the same problem as in Figure 3 with our method and the height of the K-D tree is 6. The second "learning period" after $k = 70$ is significantly shorter than the first "learning period" after $k = 20$. Which means that some pieces of information stored in the K-D tree in the first "learning period" are preserved. More detailed comparison on numerical experiments can be found in Section 4.

Note that the above calculation is not completed in exactly one sample period and will continue in the next sample period until it is finished. In particular, the tree update is online without increasing the real-time computation load. Moreover, the k-d tree can be built and "trained" offline using the history data.

Remark 3. For some large-size CR, if the query parameter lies around the boundary, the node may not be found even though it is in the k-d tree. This phenomenon is illustrated in Figure 9. The star point lies in a large CR, but the chebyshev center of the CR is too far from it that no suitable CR is found by the region query. To deal with this defect, when we compute the chebyshev center of a CR and insert the node

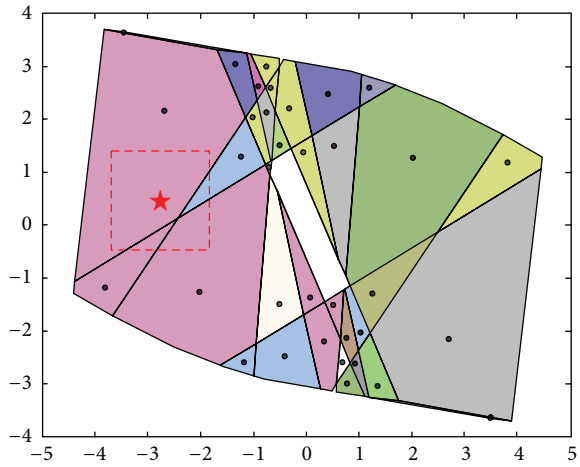

Figure 9: Failure in searching the corresponding CR around the boundary.

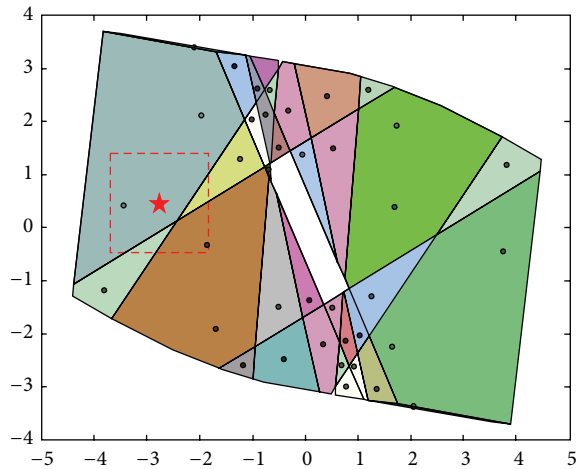

Figure 10: Successfully searching the corresponding CR around the boundary in the adjusted k-d tree.

into the k-d tree, we check the radius that corresponds to the center. If the radius is larger than a predefined threshold r_{Th}, the CR is fictitiously cut into two halves along the center at a certain dimension, and the chebyshev centers of the two halves are both inserted into the k-d tree to represent the same CR. The above operations are executed, recursively, until the radius that corresponds to each half is less than r_{Th}. If we set $r_{\text{Th}} = 1$ for the example in Figure 9, the searching on the adjusted k-d tree is illustrated in Figure 10. Figure 10 shows that several large CRs obtain more than one node. Note that the choice for the value of r_{Th} is associated with the parameter ε in system ((11a), (11b)).

Remark 4. A *constrain check* step is added before solving the optimization problem in practical implementation. In particular, we first compute the unconstrained solution of (4a), which can be easily obtained with an analytical expression. We then check whether the unconstrained solution meets the constraints in (4b). If the unconstrained solution indeed satisfies the constraints, the optimal value can be obtained immediately.

We summarize two formal algorithms on the basis of the principles introduced above.

Algorithm I (online searching). Consider the following:

(1) The unconstrained solution of the original problem is computed; that is, $U_{\text{try}} = H^{-1}x(\tau)F$.

(2) If the solution satisfies the constraints, that is, $GU_{\text{try}} \geq W + Ex(\tau)$, the solution is outputted and the algorithm is terminated.

(3) The region $\mathscr{R}e = [x^1_{\min}, x^2_{\min}, \ldots, x^k_{\min}, x^1_{\max}, x^2_{\max}, \ldots, x^k_{\max}]$ with $x^i_{\min} = x^i(\tau) - \varepsilon$ and $x^i_{\max} = x^i(\tau) + \varepsilon$ is computed.

(4) The region query of $\mathscr{R}e$ on the k-d tree is performed.

(5) If a critical region CR is found, the $u(\tau)$ with the corresponding K_{ax} and c_u is computed.

(6) Else, a small-scale problem of the original problem is solved to compute $u(\tau)$ with the short horizon, and the k-d tree with state $x(\tau)$ is updated.

Algorithm II (update k-d tree). Consider the following.

(1) The original QP problem ((3a), (3b), (3c), (3d), and (3e)) is solved, and the active set $\mathscr{A}$ is obtained.

(2) If $\mathscr{A} = \emptyset$, terminate and return.

(3) The parameters K_{ax}, L_{ax}, c_u, and c_λ are calculated.

(4) The chebyshev center x_c and radius r_c of the polyhedron $\mathscr{P}$ defined by the inequalities ((9a), (9b)) are calculated.

(5) If $r_c > r_{\text{Th}}$ and $\mathscr{P}$ is divided into two small polyhedrons $\mathscr{P}_1$ and $\mathscr{P}_2$ along x_c, go to step (4) with $\mathscr{P} = \mathscr{P}_1$ and $\mathscr{P} = \mathscr{P}_2$.

(6) The node with x_c is inserted as the key into the k-d tree.

(7) If $h > h_{\text{Th}}$, the node in $\mathbb{D}$ that is searched for minimal times is deleted.

The updated k-d tree is finally outputted.

4. Numerical Experiments

Two examples are used to verify the efficiency of the proposed algorithm. The algorithm is compared with the PE algorithm presented in [12]. Four indices from [12] are used to illustrate the algorithm performance. We describe the meaning of the indices in our implementation in this section.

Optimality Rate. $O_R \overset{\text{def}}{=} N_{\text{opt}}/N_{\text{tot}}$, where N_{opt} is the number of decision timepoints in which optimal solution is found in the k-d tree (or PE table), and N_{tot} is the total number of the decision timepoints minus the number of the decision timepoints in which the optimal value is computed by

constraint check. This index reflects the frequency of the k-d tree (or PE table) used in the simulation.

Suboptimal Index. $S_I \overset{\text{def}}{=} |\Phi - \Phi^*|/\Phi^*$, where Φ^* is the optimal closed-loop objective function value and Φ is the achieved one. This index reflects the influence brought in by the suboptimal solutions computed from the short horizon optimization problems.

Average Speedup Factor. $A_{\text{SF}} \overset{\text{def}}{=} T^*_{\text{aver}}/T_{\text{aver}}$, where T^*_{aver} is the average time required to compute the optimal solution by using the MATLAB function quadprog; T_{aver} is the average time required to compute the (sub)optimal solution by using our algorithm (or PE algorithm). This index reflects the computation speed improved by our algorithm (or PE algorithm) on average.

Worst-Case Speed Factor. $W_{\text{SF}} \overset{\text{def}}{=} T^*_{\max}/T_{\max}$, where $T^*_{\max}$ is the maximum time used to compute the optimal solution with the MATLAB function quadprog; $T_{\max}$ is the maximum time used to compute the (sub)optimal solution by using the combined method. This index reflects the computation speed improved by our algorithm (or PE algorithm) under the worst case.

Example 1. The first example is a randomly generated model from [17], which is a small-scale process with two inputs and three states. The state-space model is presented as follows:

$$\begin{aligned} x(t+1) &= \begin{bmatrix} -0.5 & 0.3 & -1.0 \\ 0.2 & -0.5 & 0.6 \\ 1.0 & 0.6 & -0.6 \end{bmatrix} x(t) \\ &\quad + \begin{bmatrix} -0.601 & -0.890 \\ 0.955 & -0.715 \\ 0.246 & -0.184 \end{bmatrix} u(t), \\ y(t) &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} x(t). \end{aligned} \tag{12}$$

In order to test our method and compare its behavior with PE method with setpoint changes, we consider a reference tracking problem and choose a state as output. The constraints on the input and state are $-2 \le u \le 2$ and $-3 \le y \le 3$. At this point, we set the horizon $N = 10$, and the control parameters are set as $R = I$ and $Q = 10 \times I$. P is calculated by solving a Riccati equation as in [8]. We set the threshold of the height of the k-d tree $h_{\text{Th}} = 5, 6, 7, 8, 9, 10$ to verify the performance of our algorithm. If the right CR is not found in the k-d tree, the horizon length of the optimization problem we turn to solve is $N = 3$. The search range in system ((11a), (11b)) is set as $\varepsilon = 0.1$. We change the setpoint 20 times randomly in the 2000-time-points simulation to test the efficiency of the algorithm. Additionally, some white noise is acting on the states. The performance of our algorithm is presented in Table 1.

Compared with the quadprog function in MATLAB, our algorithm is more than 9 times faster on average and more than 4 times faster even under the worst case. However, the degradation of the controller's performance is relatively small. The choice of the h_{Th} is a tradeoff between the computation speed and controller's performance. If we increase the h_{Th} by one, the total tree capacity is theoretically doubled. A higher tree can store more nodes, so the optimality rate is higher, but it also requires more time to search for a certain node in that tree.

For comparison, the performance of the PE algorithm is presented in Table 2. The number XX in "PEXX" in the table is the length of the table used to store CR information in the PE algorithm. The optimal rate indices of these two algorithms are basically the same, whereas our algorithm is better than the PE algorithm considering the average speed factor index and worst-case speed factor. Due to the setpoint changes in the simulation, the PE method behaves poorly when the table is short (PE1). A longer table improves the performance but consumes more time (PE25~PE200), which is overcome in our method.

Example 2. A large-scale process is tested in Example 2 to verify the performance of our algorithm on large-scale problems. It is a copolymerization reactor model for the copolymerization of methyl methacrylate (MMA) and vinyl acetate (VA) from [2]. The transfer function is as follows:

$$G(s) = \begin{bmatrix} \dfrac{0.34}{0.85s+1} & \dfrac{0.21}{0.42s+1} & \dfrac{0.50\,(0.50s+1)}{0.12s^2+0.4s+1} & 0 & \dfrac{6.46\,(0.9s+1)}{0.07s^2+0.3s+1} \\ -\dfrac{0.41}{2.41s+1} & \dfrac{0.66}{1.51s+1} & -\dfrac{0.3}{1.45s+1} & 0 & -\dfrac{3.72}{0.8s+1} \\ \dfrac{0.3}{2.54s+1} & \dfrac{0.49}{1.54s+1} & -\dfrac{0.71}{1.35s+1} & -\dfrac{0.20}{2.71s+1} & -\dfrac{4.71}{0.008s^2+0.41s+1} \\ 0 & 0 & 0 & 0 & \dfrac{1.02\,(0.23s+1)}{0.07s^2+0.31s+1} \end{bmatrix}. \tag{13}$$

The five normalized inputs of the process are the flows of the monomer MMA (u_1), monomer VA (u_2), initiator (u_3), transfer agent (u_4), and the temperature of the reactor jacket (u_5). The four normalized outputs of the system are

Table 1: Performance of our algorithm for Example 1.

	O_R	S_I	A_{SF}	W_{SF}
$h_{\mathrm{Th}} = 5$	0.7625	0.0524	17.2225	4.3900
$h_{\mathrm{Th}} = 6$	0.7965	0.0515	9.4324	4.2019
$h_{\mathrm{Th}} = 7$	0.7406	0.0541	10.0620	4.6899
$h_{\mathrm{Th}} = 8$	0.8209	0.0424	14.2288	6.3138
$h_{\mathrm{Th}} = 9$	0.9257	0.0431	13.9681	5.3240
$h_{\mathrm{Th}} = 10$	0.9027	0.0334	9.8521	5.1914

Table 2: Performance of the PE algorithm for Example 1.

	O_R	S_I	A_{SF}	W_{SF}
PE1	0.5970	0.0446	7.4264	2.1549
PE25	0.9363	0.0365	3.4468	1.0734
PE50	0.9479	0.0240	2.0919	1.1227
PE100	0.9519	0.0213	2.9028	1.5418
PE200	0.9376	0.0201	1.8311	0.3980

Table 3: Performance of our algorithm for Example 2.

	O_R	S_I	A_{SF}	W_{SF}
$h_{\mathrm{Th}} = 5$	0.0333	$4.9722e-04$	7.6372	5.6111
$h_{\mathrm{Th}} = 6$	0.0500	$4.6017e-04$	7.1483	3.3522
$h_{\mathrm{Th}} = 7$	0.1556	$4.3205e-04$	7.3093	3.0754
$h_{\mathrm{Th}} = 8$	0.0889	$3.2581e-04$	7.2294	2.8064
$h_{\mathrm{Th}} = 9$	0.1379	$2.1258e-04$	7.0314	4.3614
$h_{\mathrm{Th}} = 10$	0.2278	$3.0410e-04$	6.3084	2.1019

Table 4: Performance of the PE algorithm for Example 2.

	O_R	S_I	A_{SF}	W_{SF}
PE1	0.0167	$8.5483e-04$	7.0241	3.1902
PE25	0.0333	$3.9296e-04$	4.0917	0.2765
PE50	0.2389	$5.2056e-04$	2.8095	0.1422
PE100	0.2278	$4.8364e-04$	1.7481	0.0967
PE200	0.4389	$8.3627e-04$	1.2293	0.0431

the polymer production rate (y_1), mole fraction of MMA in the polymer (y_2), average molecular weight of the polymer (y_3), and reactor temperature (y_4).

The continuous model is discretized at a sampling rate of 1 Hz, and there are 18 states in the discretized state-space model. The feasible regions for the inputs and states are $-0.1 \leq u \leq 0.1$ and $-0.5 \leq x \leq 0.5$. The controller's parameters are set as $N = 100$, $R = 0.1I$, and $Q = \widetilde{C}^T\widetilde{C}$, where $\widetilde{C}$ is the output matrix in the state-space model, and P is calculated by solving a Riccati equation as in [8].

We consider a regulation problem in Example 2. The simulation is implemented as follows. First, we initialize the state and input with two randomly generated values in their feasible region. The task of the controller is to regulate the state to zero, and the state and input must not violate their feasible region. The simulation length is set as $N_{\mathrm{sim}} = 150$ so that all states have been driven around zero. The above simulation is repeated 30 times as a single experiment. Similar to Example 1, we change the threshold of the height of the k-d tree with $h_{\mathrm{Th}} = 5, 6, 7, 8, 9, 10$ to verify the performance of our algorithm. The horizon length with $N = 3$ of the optimization problem ((3a), (3b), (3c), (3d), and (3e)) is solved if the right CR is not found in the region query. The search range in system ((11a), (11b)) is set as $\varepsilon = 0.4$. White noise is also added on the state to mimic the real application. The performance is illustrated in Table 3.

The suboptimal index is relatively low for large-scale processes. The computation speed is accelerated by more than 6 times on average and at least twice even under the worst case, compared with the quadprog function in MATLAB.

The performance of the PE algorithm is illustrated in Table 4. The optimal rate is small if the length of the table is set to one, which means that the table information is hardly used in the simulation. This result is reasonable because the length is too short here, whereas the worst-case speed factor is below one if we build a long table to store information. This condition indicates that the PE algorithm solves the optimization problem slower than the quadprog function in MATLAB under the worst case. It is because, in the worst case, the algorithm has to check every item in the table to make the conclusion that the corresponding CR is not in the table, while, in our algorithm, the nodes which need to be checked are always only a part of all the nodes in the k-d tree.

Example 1 shows the performance of our algorithm on randomly generated small-scale processes, whereas Example 2 shows that our algorithm is also efficient on large-scale, industry-oriented processes. Both examples demonstrate the efficiency of our algorithm. The comparison with the quadprog function in MATLAB illustrates that our algorithm is remarkably faster than general online optimization method. The comparison with the PE method shows that our algorithm is competitive as a combined method based on the combination of explicit MPC and online optimization. Note that, in [12], the indices of A_{SF} and W_{SF} are much greater than one (i.e., $A_{\mathrm{SF}}, W_{\mathrm{SF}} > 1$) for the PE method, whereas the two indices are unattractive for the PE method in this study. This condition can be attributed to different reasons. First, different online optimization solvers are used for comparison. Some details in the algorithm implementation can also result in differences. In particular, the performance of our algorithm and that of the PE algorithm can be improved under further optimization of the algorithm design and implementation.

5. Conclusion

Considering combining explicit MPC and online optimization, the k-d tree is used in this study to design an algorithm that implements fast MPC. Traditional explicit MPC has been shown to be unsuitable for large-scale processes, and the online method is also incapable of fast dynamic problems. Other hybrid methods such as PE are only efficient in certain scenarios where setpoint signals change slowly and

disturbances have small magnitude. Our algorithm is hardly influenced by setpoint changes or disturbances with the introduction of the k-d tree. Numerical experiments show that the algorithm is significantly faster than the general online optimization and PE methods with small degradation in performance. A copolymerization reactor model is tested with 500 decision variables and 4600 constraints. The suboptimality is less than 0.05%; the average speedup factor is more than 6.3; and the worst-case speedup factor is at least 2.1.

Several extensions can be made based on existing achievements. The node in the k-d tree is searched through a region query in our algorithm, whereas the nearest query discussed in [19] can be used to improve the searching efficiency. Moreover, the strategy in our method is to solve a small optimization problem with a short horizon when a correct CR is not found in the k-d tree, which can be time consuming. Other techniques can accelerate the QP solution if we can utilize the searching results (i.e., CR information around the query point).

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This study is financially supported by the 863 Program of China (no. 2014AA041802) and the National Science Foundation of China (nos. 61273145 and 61273146). The authors acknowledge their support.

References

[1] S. J. Wright, "Applying new optimization algorithms to model predictive control," in *Chemical Process Control-V*, vol. 93 of *CACHE, AIChE Symposium Series no. 316*, pp. 147–155, American Institute of Chemical Engineers, New York, NY, USA, 1997.

[2] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998.

[3] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[4] R. A. Bartlett, L. T. Biegler, J. Backstrom, and V. Gopal, "Quadratic programming algorithms for large-scale model predictive control," *Journal of Process Control*, vol. 12, no. 7, pp. 775–795, 2002.

[5] H. J. Ferreau, H. G. Bock, and M. Diehl, "An online active set strategy to overcome the limitations of explicit MPC," *International Journal of Robust and Nonlinear Control*, vol. 18, no. 8, pp. 816–830, 2008.

[6] P. Patrinos, P. Sopasakis, and H. Sarimveis, "A global piecewise smooth Newton method for fast large-scale model predictive control," *Automatica*, vol. 47, no. 9, pp. 2016–2022, 2011.

[7] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2014.

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

[9] A. Bemporad and C. Filippi, "Suboptimal explicit receding horizon control via approximate multiparametric quadratic programming," *Journal of Optimization Theory and Applications*, vol. 117, no. 1, pp. 9–38, 2003.

[10] C. N. Jones and M. Morari, "Polytopic approximation of explicit model predictive controllers," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2542–2553, 2010.

[11] F. J. Christophersen, M. N. Zeilinger, C. N. Jones, and M. Morari, "Controller complexity reduction for piecewise affine systems through safe region elimination," in *Proceedings of the 46th IEEE Conference on Decision and Control (CDC '07)*, pp. 4773–4778, December 2007.

[12] G. Pannocchia, J. B. Rawlings, and S. J. Wright, "Fast, large-scale model predictive control by partial enumeration," *Automatica*, vol. 43, no. 5, pp. 852–860, 2007.

[13] P. Tøndel, T. A. Johansen, and A. Bemporad, "An algorithm for multi-parametric quadratic programming and explicit MPC solutions," *Automatica*, vol. 39, no. 3, pp. 489–497, 2003.

[14] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003.

[15] F. Bayat, T. A. Johansen, and A. A. Jalali, "Using hash tables to manage the time-storage complexity in a point location problem: application to explicit model predictive control," *Automatica*, vol. 47, no. 3, pp. 571–577, 2011.

[16] T. A. Johansen, "Approximate explicit receding horizon control of constrained nonlinear systems," *Automatica*, vol. 40, no. 2, pp. 293–300, 2004.

[17] M. N. Zeilinger, C. N. Jones, and M. Morari, "Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1524–1534, 2011.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[19] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[20] D. T. Lee and C. K. Wong, "Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees," *Acta Informatica*, vol. 9, no. 1, pp. 23–29, 1977.

[21] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, UK, 2004.